プログラミング

SAP Sybase IQ 16.0 SP03

ドキュメント ID： DC02022-01-1603-01
改訂： 2013 年 11 月

目次

パートナ認定

SAP® Sybase® IQ パートナエコシステムには、認定パートナ、データウェアハウスインフラストラクチャパートナ、分析ソリューションパートナ、およびビジネスインテリジェンスパートナのアプリケーションが含まれます。

認定レポートおよび SAP Sybase IQ パートナの一覧は、SAP Sybase IQ マーケットプレイスを参照してください。

プラットフォーム認定

*認定*とは、ある製品が特定のプラットフォーム環境で稼働すること、およびサポートされていることを意味します。SAP Sybase IQ は、特定の CPU アーキテクチャの組み合わせを持つ特定のオペレーティングシステムで認定されています。

認定された製品とプラットフォームの組み合わせについては、http://certification.sybase.com/ucr/search.do を参照してください。

クライアントアプリケーションのデータサーバとしての SAP Sybase IQ

SAP Sybase IQ は、ODBC または JDBC を介したクライアントアプリケーション接続をサポートしています。SAP Sybase IQ をクライアントアプリケーションのデータサーバとして使用できます。

一定の制限がありますが、SAP Sybase IQ は特定のクライアントアプリケーションに対して Open Server™ としても機能できます。

この章で説明する機能は、Windows システムや Sun Solaris システムを使用する IQ ユーザに対してリモートデータアクセスを提供するものではありません。 リモートデータアクセスは、Enterprise Connect™ Data Access (ECDA) の相互運用性機能の核であるコンポーネント統合サービス (CIS) を利用することで実現します。

Open Client アーキテクチャ

Sybase Open Client™ アプリケーション開発の基本のマニュアルは、SAP から入手できる Open Client マニュアルです。この項は、SAP Sybase IQ 特有の機能について説明していますが、Sybase Open Client アプリケーションプログラミングの包括的なガイドではありません。

Sybase Open Client には、2 つのコンポーネントがあります。プログラミングインタフェースとネットワークサービスです。

DB-Library と Client Library

Sybase Open Client には、クライアントアプリケーションを記述するための主要なプログラミングインタフェースが 2 つ用意されています。DB-Library™ と Client-Library です。

Open Client DB-Library は、以前の Open Client アプリケーションをサポートするもので、Client-Library とはまったく別のプログラミングインタフェースです。DB-Library については、Sybase Open Client 製品に付属する『Open Client DB-Library/C リファレンスマニュアル』を参照してください。

Client-Library プログラムも CS-Library に依存しています。CS-Library は、Client-Library アプリケーションと Server-Library アプリケーションの両方が使用するルーチンを提供します。Client-Library アプリケーションは、Bulk-Library のルーチンを使用して高速データ転送を行うこともできます。

CS-Library と Bulk-Library はどちらも Sybase Open Client に含まれていますが、別々に使用できます。

ネットワークサービス

Open Client ネットワークサービスは、TCP/IP や DECnet などの特定のネットワークプロトコルをサポートする Sybase Net-Library を含みます。Net-Library インタフェースはアプリケーション開発者からは見えません。ただしプラットフォームによっては、アプリケーションがシステムネットワーク構成ごとに別の Net-Library ドライバを必要とする場合もあります。Net-Library ドライバの指定は、ホストプラットフォームに応じて、システムの Sybase 設定で行うか、またはプログラムをコンパイルしてリンクするときに行います。

ドライバ設定の詳細については、『Open Client/Server 設定ガイド』を参照してください。

Client-Library プログラムの作成方法については、『Open Client/Server プログラマーズガイド補足』を参照してください。

Open Client および jConnect 接続

SAP Sybase IQ は TDS を通してアプリケーションからの要求を処理するとき、関連するデータベースオプションを、SAP Sybase SQL Anywhere® サーバのデフォルトの動作と互換性のある値に自動的に設定します。このオプションの設定は、その接続中だけの一時的なものです。クライアントアプリケーションはこれらのオプションをいつでも独自に設定して変更できます。

注意： SAP Sybase IQ は `ANSI_BLANKS`、`FLOAT_AS_DOUBLE`、`TSQL_HEX_CONSTANT` オプションをサポートしていません。

SAP Sybase IQ では長いユーザ名とパスワードが許可されていますが、TDS クライアントのユーザ名とパスワードは最大で 30 バイトです。パスワードまたはユーザ ID が 30 バイトを超えている場合、TDS によって (たとえば jConnect を使用して) 接続しようとすると、`Invalid user ID or password` エラーが返されます。

注意： Interactive SQL アプリケーションなどの ODBC アプリケーションは、ODBC 仕様で要求される特定のデータベースオプションの値を自動的に設定します。これにより、`LOGIN_PROCEDURE` データベースオプションによる設定が上書きされます。

login_procedure オプション

起動時の接続互換性オプションを設定するログインプロシージャを指定します。

指定可能な値
文字列

デフォルト
sp_login_environment システムプロシージャ

スコープ
個別の接続または PUBLIC に対して設定できます。このオプションを設定するには SET ANY SECURITY OPTION システム権限が必要です。

備考
このログインプロシージャは、ランタイムに sp_login_environment プロシージャを呼び出して、データベース接続設定を決定します。このログインプロシージャは、すべてのチェックが完了し、接続が有効であることが確認された後で呼び出されます。login_procedure オプションで指定されたプロシージャは、イベント接続では実行されませんが、Web サービス接続では実行されます。

新規プロシージャを作成し、その新規プロシージャを呼び出すための login_procedure を設定して、デフォルトデータベースオプション設定をカスタマイズできます。このカスタムプロシージャは、sp_login_environment を呼び出すか、TDS 接続が確立されたことを検出し (デフォルトの sp_login_environment コードを確認します)、sp_tsql_environment を直接呼び出す必要があります。この操作が失敗した場合、TDS ベースの接続は切断されることがあります。sp_login_environment も sp_tsql_environment も編集しないでください。

ユーザ定義のログインプロシージャから SQLSTATE 08WA0 のパスワード有効期限切れエラーメッセージを発生させ、パスワードの有効期限が切れていることをユーザに通知することができます。エラーの通知により、アプリケーションはエラーを確認し、有効期限の切れたパスワードを処理できます。パスワードの有効期限を実装する場合はログインポリシーを使用し、パスワードの有効期限切れのメッセージを返すログインプロシージャは使用しないでください。

NewPassword=* 接続パラメータを使用する場合は、このエラーを通知して、クライアントライブラリが新しいパスワードの入力を要求するプロンプトを表示できるようにする必要があります。プロシージャで SQLSTATE 28000 (無効なユーザ ID またはパスワード) または SQLSTATE 08WA0 (パスワードの有効期限切れ) が通知されるか、RAISERROR のエラーが発生すると、ログインは失敗し、エラーがユーザに返されます。その他のエラーを通知するか、別のエラーが発生した場合

は、ユーザログインは成功し、メッセージがデータベースサーバメッセージログに書き込まれます。

例

次に、INVALID_LOGON エラーを通知して接続を拒否するサンプルコードを示します。

```
CREATE PROCEDURE DBA.login_check( )
   BEGIN
      DECLARE INVALID_LOGON EXCEPTION FOR SQLSTATE '28000';
      // Allow a maximum of 3 concurrent connections
      IF( DB_PROPERTY( 'ConnCount' ) > 3 ) THEN
          SIGNAL INVALID_LOGON;
      ELSE
          CALL sp_login_environment;
      END IF;
   END
go

GRANT EXECUTE ON DBA.login_check TO PUBLIC
go

SET OPTION PUBLIC.login_procedure='DBA.login_check'
go
```

次の例は、ユーザの失敗した接続の数が 30 分間で 3 回よりも多くなった場合に、接続試行をブロックする方法を示しています。ブロック期間中にブロックされた試行は、すべて無効パスワードエラーを受け取り、ログに失敗として記録されます。DBA がログを解析するために、ログは十分な時間保持されます。

```
CREATE TABLE DBA.ConnectionFailure(
    pk INT PRIMARY KEY DEFAULT AUTOINCREMENT,
    user_name CHAR(128) NOT NULL,
    tm TIMESTAMP NOT NULL DEFAULT CURRENT TIMESTAMP
)
go

CREATE INDEX ConnFailTime ON DBA.ConnectionFailure(
    user_name, tm )
go

CREATE EVENT ConnFail TYPE ConnectFailed
HANDLER
BEGIN
    DECLARE usr CHAR(128);
    SET usr = event_parameter( 'User' );

    // Put a limit on the number of failures logged.
    IF (SELECT COUNT(*) FROM DBA.ConnectionFailure
        WHERE user_name = usr
        AND tm >= DATEADD( minute, -30,
            CURRENT TIMESTAMP )) < 20 THEN
        INSERT INTO DBA.ConnectionFailure( user_name )
            VALUES( usr );
```

```
        COMMIT;
        // Delete failures older than 7 days.
        DELETE DBA.ConnectionFailure
        WHERE user_name = usr
        AND tm < dateadd( day, -7, CURRENT TIMESTAMP );
        COMMIT;
    END IF;
END
go

CREATE PROCEDURE DBA.login_check( )
BEGIN
    DECLARE usr CHAR(128);
    DECLARE INVALID_LOGON EXCEPTION FOR SQLSTATE '28000';
    SET usr = CONNECTION_PROPERTY( 'Userid' );
    // Block connection attempts from this user
    // if 3 or more failed connection attempts have occurred
    // within the past 30 minutes.
    IF ( SELECT COUNT( * ) FROM DBA.ConnectionFailure
        WHERE user_name = usr
        AND tm >= DATEADD( minute, -30,
            CURRENT TIMESTAMP ) ) >= 3 THEN
        SIGNAL INVALID_LOGON;
    ELSE
        CALL sp_login_environment;
    END IF;
END
go

GRANT EXECUTE ON DBA.login_check TO PUBLIC
go

SET OPTION PUBLIC.login_procedure='DBA.login_check'
go
```

次の例は、ユーザのパスワードの有効期限が切れていることを知らせるエラーを通知する方法を示しています。パスワードの有効期限切れの通知を実装するには、ログインポリシーを使用してください。

```
CREATE PROCEDURE DBA.check_expired_login( )
BEGIN
  DECLARE PASSWORD_EXPIRED EXCEPTION FOR SQLSTATE '08WA0';

  IF( condition-to-check-for-expired-password ) THEN
      SIGNAL PASSWORD_EXPIRED;
  ELSE
      CALL sp_login_environment;
  END IF;
END;
```

複数のデータベースがあるサーバ

サーバに複数のデータベースがある場合、Open Client Library を使用して、接続するデータベースを指定できます。

- `interfaces` ファイル内にサーバのエントリを設定します。
- **start_iq** コマンドで **-n** パラメータを指定し、データベース名のショートカットを設定します。
- **isql** コマンドでデータベース名と共に **-S** *database_name* パラメータを指定します。 このパラメータは接続時に常に必要です。

プログラム自体を変更しなくても、ショートカット名をプログラムに記述し、ショートカット定義を変更するだけで、同じプログラムを複数のデータベースに対して実行できます。

たとえば、次の `live_sales` と `test_sales` の 2 つのサーバ定義は、`interfaces` ファイルから抜粋したものです。

```
live_sales

     query tcp ether myhostname 5555
    master tcp ether myhostname 5555

test_sales

     query tcp ether myhostname 7777
    master tcp ether myhostname 7777
```

サーバを起動して、所定のデータベースのエイリアスを設定します。次のコマンドは、`live_sales` を `salesbase.db` と等価に設定します。

```
start_iq -n sales_live <other parameters> -x ¥ 'tcpip{port=5555}'
salesbase.db -n live_sales
```

`live_sales` サーバに接続するには次のように記述します。

```
isql -Udba -Psql -Slive_sales
```

サーバ名は `interfaces` ファイル内に一度のみ記述します。これは、SAP Sybase IQ への接続がデータベース名に基づくようにしており、データベース名はユニークである必要があるためです。すべてのスクリプトが `salesbase` データベースで機能するように設定されている場合、`live_sales` または `test_sales` を使用して処理するように変更する必要はありません。

アプリケーションでのインデータベース分析の使用

SAP Sybase IQ には、3 とおりのインデータ分析が用意されています。ネイティブ組み込み分析、ネイティブ UDF プラグイン分析、外部 UDF プラグイン分析です。開発者は、外部 UDF として分析機能を提供することにより、ビッグデータの複雑な分析を可能にできます。

- **ネイティブ組み込み分析** – ネイティブのカーネル内分析の例として、OLAP、全文検索などがあります。**CUME_DIST** 関数は、組み込み ANSI SQL OLAP 集計関数の一例です。
- **ネイティブ UDF プラグイン分析** – プロセス外共有ライブラリを使用して、*テキスト分析*ソリューションを開発できます。プロセス外のデータベース内 UDF を開発することにより、ユーザ定義コードのプロセス内での実行に伴うセキュリティおよび堅牢性に関するリスクを最小化できます。LOB マニュアルについては、非構造化データ分析 を参照してください。
- **外部 UDF プラグイン分析** – Java UDF、テーブル UDF、およびテーブルパラメータ化関数 (TPF) を使用すると、ビッグデータのプロセス外分析ソリューションを開発できます。

参照：

- 付録： OLAP の使用 (645 ページ)

スカラ C/ C++ UDF

スカラ UDF とは、単一の値に対して処理を行う V3 または V4 の外部 C/C++ プロシージャです。

詳細と例については、ユーザ定義関数 を参照してください。外部 C および C++ プロシージャは、別途ライセンスが必要な SAP Sybase IQ オプションを必要とします。

集合 C/ C++ UDF

集合 UDF とは、複数の値に対して処理を行う V3 または V4 の外部 C/C++ プロシージャです。集合 UDF は、UDA または UDAF とも呼ばれます。集合 UDF をコーディングするためのコンテキスト構造体は、スカラ UDF をコーディングするためのコンテキスト構造体とは少し異なります。

詳細と例については、ユーザ定義関数 を参照してください。外部 C および C++ プロシージャは、別途ライセンスが必要な SAP Sybase IQ オプションを必要とします。

Java UDF

Java UDF の動作は SQL 関数と似ていますが、プロシージャや関数のコードが Java で記述され、データベースサーバ外の Java VM 環境で実行される点が異なります。Java スカラ UDF と Java テーブル UDF を定義できます。

Java UDF は、別途ライセンスが必要な SAP Sybase IQ オプションを必要としません。

Java スカラ UDF

Java コードで実装されたプロセス外の (外部環境) スカラユーザ定義関数です。

詳細と例については、ユーザ定義関数 を参照してください。

Java テーブル UDF

Java コードで実装されたプロセス外の (外部環境) テーブル UDF です。

詳細と例については、ユーザ定義関数 を参照してください。

テーブル UDF

テーブル UDF は、C、 C++、または Java による外部ユーザ定義テーブル関数です。スカラ UDF や集合 UDF と違い、テーブル UDF は出力としてローセットを生成します。SQL クエリでは、SQL 文の FROM 句にテーブルと同様に指定することで、そのローセットを利用できます。

スカラ UDF および集合 UDF では v3 または v4 extfn API が使用できますが、テーブル UDF では v4 のみが使用できます。

詳細と例については、ユーザ定義関数 を参照してください。

TPF

テーブルパラメータ化関数 (TPF) は、スカラ値またはローセットのどちらかを入力として受け取る拡張テーブル UDF です。TPF のユーザ定義パーティションを構成できます。UDF の開発者はパーティションスキームを宣言し、データセットを

より小さなクエリ処理単位に分割して、複数のマルチプレックスノードに分散させることができます。これにより、分散サーバ環境でローセットのパーティションをまたがって TPF を並列に実行することが可能になります。クエリエンジンは TPF 処理の超並列化をサポートしています。

詳細と例については、ユーザ定義関数 を参照してください。

Hadoop 統合

SAP Sybase IQ には、MapReduce コンポーネントの構築に使用できる UDF API が含まれています。これを Hadoop 統合に使用できます。SAP Sybase ソリューションストアに Hadoop 統合の例が用意されています。

MapReduce プログラミングモデルは、大規模な並列分散コンピューティング向けに設計されています。MapReduce プログラミングモデルは、主に次の 2 つのステージで構成されています。

- **Map ステージ** – リーダノードが問題を複数のサブ問題 (*Map*) に分割します。それぞれの Map は互いに独立し、並列に実行される必要があります。
- **Reduce ステージ** – リーダノードが各サブ問題の答えを集約して有意味に組み合わせ、元の問題に対する答えを得ます。

Apache Hadoop は MapReduce の実装です。Hadoop は、ジョブの Map 処理と Reduce 処理のスケジューリングを自動化する Java ソフトウェアワークフレームです。

SAP Sybase IQ では、外部ユーザ定義関数の一種であるテーブルパラメータ化関数 (TPF) により、Hadoop のような並列スケジューリングをサポートしています。TPF は、任意のテーブル値入力パラメータのローセットを受け付け、分散サーバ環境で並列に実行可能です。パーティションと順序付けに関する要件は、TPF 入力で指定できます。開発者は、SQL と TPF を使用してデータベースサーバ内から MapReduce パラダイムを活用できます。

TPF の基礎については、『ユーザ定義関数 ガイド』を参照してください。

SAP Sybase IQ と Hadoop 分散ファイルシステムの統合

Hadoop 分析から返されたデータを SAP Sybase IQ データベースに統合するには、複数の方法があります。

- **ETL 処理** – オープンソースのユーティリティ SCOOP を使用して、Hadoop データからのバルクロードデータを SAP Sybase IQ に格納します。

- **データフェデレーション** – HDFS ファイルを、SQL クエリに参加する SAP Sybase IQ データベース内のテーブルとして公開します。HDFS ファイルを SAP Sybase IQ にロードする必要はありません。
- **クエリフェデレーション** – SAP Sybase IQ 内の SQL クエリで Hadoop プロセスを実行できるようにします。Hadoop プロセスが返すデータが SQL 結果セットに組み込まれます。
- **クライアント側フェデレーション** – TOAD™ SQL ツールを使用して、クエリを SAP Sybase IQ データベースおよび Hadoop ファイル全体にフェデレートします。

Hadoop 分散ファイルシステム内のファイルをインメモリテーブルとして読み込む

SAP Sybase IQ が Hadoop 分散ファイルシステム (HDFS) 内のファイルをインメモリテーブルとして読み込むデータフェデレーション例です。

注意： このサンプルコードは、主として図解のためのもので、本稼動用のものではありません。エラー処理を妥当なものにするために努力しましたが、ここに示す例は本稼動基準のものではなく、本稼動で使用する前にさらなる安全対策とテストが必要となります。

1. Java クラスを作成します。

```
public class HDFSclient {
    public static void readFileByLine(String file, ResultSet
rset[])
throws IOException {

// Set Configuration to point to HDFS NameNode and find input dir
Configuration conf = new Configuration();
conf.addResource(new Path("/home/mymachine/hadoop/conf/core-
site.xml"));
FileSystem fileSystem = FileSystem.get(conf);
Path path = new Path(file);
if (!fileSystem.exists(path)); {
    System.out.println("File " + file + " does not exists");
    return;
}

// Create meta data for the result set
ResultSetMetaDataImpl rsmd = new ResultSetMetaDataImpl(1);
rsmd.setColumnType(1, Typs.VARCHAR);
rsmd.setColumnName(1, "c1");
rsmd.setColumnLabel(1, "c1");
rsmd.setColumnDisplaySize(1, "c1");
rsmd.setTableName(1, "MyTable");

// Create ResultSet using the meta data
ResultSetImpl rs = null;
try {
    rs = new ResultSetImpl((ResultSetMetaDataImpl)rsmd);
    rs.beforeFirst();// Make sure we are at the beginning
```

```
} catch(Exception e) {
    System.out.println(”Could not create result set.”);
    System.out.println(e.toString());
}

// Read files from input dir line by line inserting into rs
String line;
DataInputStream in = new DataInputSteam(fileSystem.open(path));
BufferedReader reader = new BufferedReader(new
InputStreamReader(in));
while ((line = reader.readline()) != null) {
try {
        rs.insertRow();// Insert a new row
        rs.updateString(1,(line));
} catch(Exception e) {
    System.out.println(”Could not insert row/data”);
    System.out.println(e.toString());
}
}
try {
rs.beforeFirst();// Make sure we are at the beginning
} catch(Exception e) {
    System.out.println(e.toString());
    }

rset[0] = rs; // Assign result set to the 1st of the passé din
array.

in.close();
reader.close();
fileSystem.close();

}
}
}
```

2. クラスまたはパッケージ化された JAR ファイルをインストールします。

```
INSTALL JAVA NEW JAR ‘myjar’ FROM FILE ‘/home/mymachine/UDFs/
myjar.jar’;
```

3. 関数を作成します。

```
CREATE or REPLACE PROCEDURE readFileByLine( IN fileName CHAR(50) )
RESULT ( c1 VARCHAR(255) )
EXTERNAL NAME 'example.HDFSclient.readFileByLine(Ljava/lang/
String;[Ljava/sql/ResultSet;)V'
LANGUAGE JAVA;
```

4. 関数を実行します。

```
SELECT c1 FROM readFileByLine('/home/mymachine/input/input.txt');
```

外部 Hadoop MapReduce ジョブの起動とクエリにおける結果の使用

<key, value> ペアで構造化されたデータを入出力する map メソッドと reduce メソッドを定義します。

あるディレクトリの中に 2 つのテキストファイルがあり、それぞれ次の内容が記述されているとします。

- `File1.txt`: Hello World Goodbye World
- `File2.txt`: Goodbye World Hadoop

1. map ステップでは、各ファイルが独立した Map ジョブとして処理され、それぞれの Map の出力は次のような <key, value> になります。
   - Job1： <Hello, one> <World, one> <Goodbye, one> <World, one>
   - Job2： <Goodbye, one> <world, one> <Hadoop, one>
2. Map ステップから出力された <key, value> を単純に追加する Reducer を呼び出します。ローカル Reducer の出力は次のとおりです。
   - Job1： <Hello, one> <World, two> <Goodbye, one>
   - Job2： <Goodbye, one> <World, one> <Hadoop, one>
3. 組み合わせて最終出力を取得します。
   `<Hello, one> <World, 3><Goodbye,2><Hadoop, 1>`

注意： このサンプルコードは、主として図解のためのもので、本稼動用のものではありません。エラー処理を妥当なものにするために努力しましたが、ここに示す例は本稼動基準のものではなく、本稼動で使用する前にさらなる安全対策とテストが必要となります。

```
     public class WordCountDriver extends Configured {
    public static void String HADOOP_ROOT_DIR = “hdfs://localhost:
9000”
private Text word = new Text();
private final IntWritable one = new IntWritable(1);

static class WordCountMapper extends Mapper<LongWritable, Text,
Text, IntWritable> {

public void map(LongWritable key Text value, Context context) throws
IOException, InterruptedException {
        String line = value.toString();
StringTokenizer itr = new StringTokenizer(line.toLowerCase());
while (itr.hasMoreTokens()){
    word.set(itr.next(Token));
    context.write(word, one);
}
};

static class WordCountReducer extends Reducer<Text, IntWritable,
Text, IntWritable > {
```

```
public void reduce (Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
    int sum = 0;
for (IntWritable value : values) {
    sum += value.get();
}
context.write(key, new IntWrtiable(sum));
}
};

Public static void run(String input, String output, ResultSet rs[])
throws Exception {
Configuration conf = new Configuration();
conf.addResource(new Path("/home/mymachine/hadoop/conf/core-
site.xml"));
conf.set("fs.default.name","hdfs://localhost:9000");
conf.set("mapred.job.tracker","localhost:9000");

// Specify output types
Job job = new Job(conf, "Word Count");
Job.setOutputKeyClass(Text.class);
Job.setOutputValueClass(IntWritable.class);

// Specify input and output locations
FileInputFormat.addInputPath(job, new Path(HADOOP_ROOT_DIR+input));)
FileOutputFormat.addInputPath(job, new Path(HADOOP_ROOT_DIR
+output));

// Specify a mapper
job.setMapperClass(WordCountDriver.WordCountMapper.class);

// Specify a reducer
job.setReducerClass(WordCountDriver.WordCountReducer.class);
job.setCombinerClass(WordCountDriver.WordCountReducer.class);
job.setJarByClass(WordCountDriver.class)

// Wait for MR job to complete
while (job.waitForCompletion(true) ? false : true) {
    // Waiting…
}
    HDFSclient hdfsc = new HDFSclient();
    hdfsc.readFileByLine(file, rs);
   }
}
}
```

a_v4_extfn の API リファレンス

a_v4_extfn の関数、メソッド、属性のリファレンス情報です。

BLOB (a_v4_extfn_blob)

a_v4_extfn_blob 構造体を使用して、独立した BLOB オブジェクトを表現します。

実装

```
typedef struct a_v4_extfn_blob {
    a_sql_uint64 (SQL_CALLBACK *blob_length)(a_v4_extfn_blob *blob);
    void (SQL_CALLBACK *open_istream)(a_v4_extfn_blob *blob,
a_v4_extfn_blob_istream **is);
    void (SQL_CALLBACK *close_istream)(a_v4_extfn_blob *blob,
a_v4_extfn_blob_istream *is);
    void (SQL_CALLBACK *release)(a_v4_extfn_blob *blob);
} a_v4_extfn_blob;
```

メソッドの概要

メソッド名	データ型	説明
blob_length	a_sql_uint64	指定の BLOB の長さをバイト数で返す。
open_istream	void	指定の BLOB からの読み込みを開始するために使用できる入力ストリームを開く。
close_istream	void	指定の BLOB の入力ストリームを閉じる。
release	void	呼び出し元がこの BLOB の処理を完了し、BLOB の所有者がリソースを解放できることを示します。 **release()** の後で BLOB を参照するとエラーになります。所有者は通常、**release()** が呼び出されたときにメモリを削除します。

説明

オブジェクト a_v4_extfn_blob を使用するのは次のときです。

- テーブル UDF がスカラ入力値から LOB または CLOB データを読み込む必要がある。
- TPF が入力テーブルのカラムから LOB または CLOB データを読み込む必要がある。

制約と制限事項

なし。

blob_length

v4 API メソッド `blob_length` を使用して、指定の BLOB の長さをバイト数で返します。

宣言

```
a_sql_uint64 blob_length(
    a_v4_extfn_blob *
)
```

使用法

指定の BLOB の長さをバイト数で返します。

パラメータ

パラメータ	説明
blob	長さを返す対象の BLOB。

戻り値

指定の BLOB の長さ。

参照：

- open_istream (19 ページ)
- close_istream (20 ページ)
- release (21 ページ)

open_istream

v4 API メソッド `open_istream` を使用して、BLOB から読み込むための入力ストリームを開きます。

宣言

```
void open_istream(
    a_v4_extfn_blob *blob,
    a_v4_extfn_blob_istream **is
    )
```

使用法

指定の BLOB からの読み込みを開始するために使用できる入力ストリームを開きます。

パラメータ

パラメータ	説明
blob	入力ストリームを開く対象の BLOB。
is	オープンして返された入力ストリームを示す出力パラメータ。

戻り値
なし。

参照：
- blob_length (19 ページ)
- close_istream (20 ページ)
- release (21 ページ)

close_istream
v4 API メソッド `close_istream` を使用して、指定の BLOB の入力ストリームを閉じます。

宣言
```
void close_istream(
    a_v4_extfn_blob *blob,
    a_v4_extfn_blob_istream *is
    )
```

使用法
`open_istream` API で開いた入力ストリームを閉じます。

パラメータ

パラメータ	説明
blob	入力ストリームを閉じる対象の BLOB。
is	閉じる入力ストリームを示すパラメータ。

戻り値
なし。

参照：
- blob_length (19 ページ)
- open_istream (19 ページ)

- release (21 ページ)

release

v4 API メソッド `release` を使用して、現在選択中の BLOB の処理を呼び出し元が完了したことを示します。release によって、所有者がメモリを解放できるようになります。

宣言

```
void release(
a_v4_extfn_blob *blob
)
```

使用法

呼び出し元がこの BLOB の処理を完了し、BLOB の所有者がリソースを解放できることを示します。 **release()** の後で BLOB を参照するとエラーになります。所有者は通常、**release()** が呼び出されたときにメモリを削除します。

パラメータ

パラメータ	説明
blob	解放する対象の BLOB。

戻り値

なし。

参照：

- blob_length (19 ページ)
- open_istream (19 ページ)
- close_istream (20 ページ)

BLOB 入力ストリーム (a_v4_extfn_blob_istream)

`a_v4_extfn_blob_istream` 構造体を使用して、LOB または CLOB のスカラ入力カラムの BLOB データや、入力テーブルの LOB または CLOB カラムを読み込みます。

実装

```
typedef struct a_v4_extfn_blob_istream {
    size_t (SQL_CALLBACK *get)( a_v4_extfn_blob_istream *is, void
*buf, size_t len );
    a_v4_extfn_blob     *blob;
    a_sql_byte          *beg;
    a_sql_byte          *ptr;
```

```
    a_sql_byte          *lim;
} a_v4_extfn_blob_istream;
```

メソッドの概要

メソッド名	データ型	説明
get	size_t	指定の分量のデータを BLOB 入力ストリームから取得する。

データメンバーとデータ型の概要

データメンバー	データ型	説明
Blob	a_v4_extfn_blob	この入力ストリームの作成の基になった BLOB 構造体。
Beg	a_sql_byte	現在のデータチャンクの先頭位置のポインタ。
Ptr	a_sql_byte	データチャンク内の現在のバイト位置のポインタ。
Lim	a_sql_byte	現在のデータチャンクの末尾の位置のポインタ。

get

v4 API メソッド `get` を使用して、指定の分量のデータを BLOB 入力ストリームから取得します。

宣言

```
size_t get(
    a_v4_extfn_blob_istream *is,
    void *buf,
    size_t len
)
```

使用法

指定の分量のデータを BLOB 入力ストリームから取得します。

パラメータ

パラメータ	説明
is	データの取得元の入力ストリーム。
buf	データを格納するバッファ。
len	取得するデータの分量。

戻り値

取得したデータの分量。

カラムデータ (a_v4_extfn_column_data)

構造体 `a_v4_extfn_column_data` は、カラム 1 つ分のデータを表します。これは、プロデューサが結果セットのデータを生成するとき、またはコンシューマが入力テーブルのカラムデータを読み込むときに使用します。

実装

```
typedef struct a_v4_extfn_column_data {
    a_sql_byte          *is_null;
    a_sql_byte          null_mask;
    a_sql_byte          null_value;

    void                *data;
    a_sql_uint32        *piece_len;
    size_t              max_piece_len;

    void                *blob_handle;
} a_v4_extfn_column_data;
```

データメンバーとデータ型の概要

データメンバー	データ型	説明
is_null	a_sql_byte *	値の NULL 情報が格納されているバイトへのポインタ。
null_mask	a_sql_byte	NULL 値を表すために使用する 1 つまたは複数のビット。
null_value	a_sql_byte	NULL を表す値。
data	void *	カラムのデータのポインタ。フェッチメカニズムの種類に応じて、コンシューマ内のアドレスか、UDF 内でデータが格納されているアドレスのどちらかを指す。
piece_len	a_sql_uint32 *	可変長データ型のデータの実際の長さ。
max_piece_len	size_t	このカラムが保持できる最大データ長。
blob_handle	void *	NULL 以外の値の場合、このカラムのデータは `blob` API を使用して読み込む必要があることを表す。

説明

`a_v4_extfn_column_data` 構造体は、特定のデータカラムのデータ値および関連する属性を表します。この構造体は、プロデューサが結果セットのデータを生成するときに使用します。*data*、*piece_len*、および *is_null* フラグの記憶領域もプロデューサが作成することになっています。

is_null、*null_mask*、*null_value* の各データメンバーは、カラム内の NULL の扱いを表し、8 カラム分の NULL ビットを 1 バイトにコード化するという状況や、各カラムに対して 1 バイトを使用するという状況に対処できます。

次の例は、NULL を表す 3 つのフィールドである *is_null*、*null_mask*、*null_value* を解釈する方法を示します。

```
is_value_null()
   return( (*is_null & null_mask) == null_value )

set_value_null()
   *is_null = ( *is_null & ~null_mask) | null_value

set_value_not_null()
   *is_null = *is_null & ~null_mask | (~null_value & null_mask)
```

カラムリスト (a_v4_extfn_column_list)

`a_v4_extfn_column_list` 構造体を使用して、describe で **PARTITION BY** を使用するときのカラムのリスト、または **TABLE_UNUSED_COLUMNS** を使用するときのカラムのリストを表します。

実装

```
typedef struct a_v4_extfn_column_list {
    a_sql_int32         number_of_columns;
    a_sql_uint32        column_indexes[1];      // there are
number_of_columns entries
} a_v4_extfn_column_list;
```

データメンバーとデータ型の概要

データメンバー	データ型	説明
number_of_columns	a_sql_uint32	リスト内のカラム数。
column_indexes	a_sql_uint32 *	カラムインデックス (1 ベース) を使用した、サイズ number_of_columns の連続した配列。

説明

カラムリストの中身の意味は、**TABLE_PARTITIONBY** と **TABLE_UNUSED_COLUMNS** のどちらで使用するリストかによって変わります。

カラム順序 (a_v4_extfn_order_el)

`a_v4_extfn_order_el` 構造体を使用して、カラム内の要素の順序を表します。

実装

```
typedef struct a_v4_extfn_order_el {
    a_sql_uint32       column_index;    // Index of the column in the
table (1-based)
    a_sql_byte       ascending;             // Nonzero if the column
is ordered "ascending".
} a_v4_extfn_order_el;
```

データメンバーとデータ型の概要

データメンバー	データ型	説明
column_index	a_sql_uint32	テーブル内のカラムのインデックス (1 ベース)。
ascending	a_sql_byte	カラム順序が昇順の場合は 0 以外。

説明

`a_v4_extfn_order_el` 構造体は、カラムについて表し、昇順と降順のどちらであるかを示します。この構造体の配列を `a_v4_extfn_orderby_list` 構造体が保持します。`a_v4_extfn_order_el` 構造体は **ORDERBY** 句の各カラムについて 1 つずつあります。

カラムサブセット (a_v4_extfn_col_subset_of_input)

`a_v4_extfn_col_subset_of_input` 構造体を使用して、出力カラムの値が常に、特定の入力カラムから UDF に取得されることを宣言します。

実装

```
typedef struct a_v4_extfn_col_subset_of_input {
    a_sql_uint32    source_table_parameter_arg_num;    // arg_num of
the source table parameter
    a_sql_uint32    source_column_number;        // source column of
the source table
} a_v4_extfn_col_subset_of_input;
```

データメンバーとデータ型の概要

データメンバー	データ型	説明
source_table_parameter_arg_num	a_sql_uint32 *	取得元 TABLE パラメータの *arg_num*
source_column_number	a_sql_uint32 *	取得元テーブルの取得元カラム。

説明
クエリオプティマイザは、入力のサブセットを使用して、出力カラムの値の論理プロパティを推測します。 たとえば、入力カラム内の重複しない値の数は出力カラム内の重複しない値の上限であり、入力カラムのローカル述部は出力カラムでも保持されます。

describe の API

_describe_extfn 関数は `a_v4_extfn_proc` のメンバーです。UDF は、`a_v4_extfn_proc_context` オブジェクトの `describe_column`、`describe_parameter`、`describe_udf` の各プロパティを使用して、論理プロパティを取得および設定します。

_describe_extfn の宣言

```
void (UDF_CALLBACK *_describe_extfn)(a_v4_extfn_proc_context
*cntxt );
)
```

使用法
_describe_extfn 関数はプロシージャの評価をサーバに対して伝えます。

`describe_column`、`describe_parameter`、`describe_udf` の各プロパティには、対応する get メソッドと set メソッド、属性タイプのセット、および各属性に対応するデータ型があります。get メソッドではサーバから情報を取得します。set メソッドでは UDF の論理プロパティ (出力カラム数や、出力カラム内の重複しない値の数) をサーバに伝えます。

*describe_column_get

`describe_column_get` v4 API メソッドは、TABLE パラメータの個別のカラムに関するプロパティを取得するために、テーブル UDF によって使用されます。

宣言

```
a_sql_int32 (SQL_CALLBACK *describe_column_get)(
    a_v4_extfn_proc_context       *cntxt,
    a_sql_uint32                     arg_num,
    a_sql_uint32                        column_num,
    a_v4_extfn_describe_parm_type describe_type,
    void                                *describe_buffer,
    size_t                              describe_buffer_len );
```

パラメータ

パラメータ	説明
cntxt	この UDF のプロシージャコンテキストオブジェクト。
arg_num	TABLE パラメータの順序数 (0 は結果テーブル、1 は最初の入力引数)。
column_num	カラムの順序数 (開始値は 1)。
describe_type	取得するプロパティを示すセレクタ。
describe_buffer	サーバから取得する指定のプロパティの describe 情報を保持する構造体。具体的な構造体またはデータ型は **describe_type** パラメータで示される。
describe_buffer_length	**describe_buffer** のバイト単位の長さ。

戻り値
成功した場合は、**describe_buffer** に書き込まれたバイト数を返します。エラーが発生した場合や、プロパティが取得されなかった場合は、汎用の `describe_column` エラーのいずれかを返します。

***describe_column_get* の属性**
v4 API メソッド `describe_column_get` の属性を示すコードです。

```
typedef enum a_v4_extfn_describe_col_type {
    EXTFNAPIV4_DESCRIBE_COL_NAME,
    EXTFNAPIV4_DESCRIBE_COL_TYPE,
    EXTFNAPIV4_DESCRIBE_COL_WIDTH,
    EXTFNAPIV4_DESCRIBE_COL_SCALE,
    EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL,
    EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES,
    EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE,
    EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT,
    EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER,
    EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT,
    } a_v4_extfn_describe_col_type;
```

EXTFNAPIV4_DESCRIBE_COL_NAME (get)
EXTFNAPIV4_DESCRIBE_COL_NAME 属性はカラム名を示します。
`describe_column_get` のシナリオで使用します。

データ型
`char[]`

説明
カラム名。このプロパティはテーブル引数に対してのみ有効です。

使用法
UDF がこのプロパティを取得すると、指定のカラムの名前が返ります。

戻り値
成功した場合は、カラム名の長さを返します。

失敗時には、汎用的な `describe_column` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – クエリ処理フェーズが初期フェーズより後でない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – バッファの長さの文字数が不十分か、または 0 の場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – パラメータが TABLE パラメータでない場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

EXTFNAPIV4_DESCRIBE_COL_TYPE (get)
`EXTFNAPIV4_DESCRIBE_COL_TYPE` 属性は、カラムのデータ型を示しています。これは、`describe_column_get` のシナリオで使用されます。

データ型
`a_sql_data_type`

説明
カラムのデータ型。このプロパティはテーブル引数に対してのみ有効です。

使用法
UDF がこのプロパティを取得すると、指定のカラムのデータ型が返ります。

戻り値
成功時には、`sizeof(a_sql_data_type)` が返されます。

失敗時には、汎用的な `describe_column` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – describe_buffer が `a_sql_data_type` のサイズでない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – クエリ処理フェーズが初期フェーズより後でない場合に返される get エラー。

クエリ処理フェーズ

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

EXTFNAPIV4_DESCRIBE_COL_WIDTH (get)
EXTFNAPIV4_DESCRIBE_COL_WIDTH 属性は、カラム幅を示します。`describe_column_get` のシナリオで使用します。

データ型
`a_sql_uint32`

説明
カラムの幅。カラム幅は、対応するデータ型の値を格納するために必要な記憶領域のサイズのバイト数です。このプロパティはテーブル引数に対してのみ有効です。

使用法
UDF がこのプロパティを取得すると、**CREATE PROCEDURE** 文で定義されているカラム幅が返ります。

戻り値
成功時には、`sizeof(a_sql_uint32)` を返します。

失敗時には、汎用的な `describe_column` エラーのいずれかを返します。または、以下のエラーを返します。

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – describe_buffer が a_sql_uint32 のサイズでない場合に返される get エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – クエリ処理フェーズが初期フェーズより後でない場合に返される get エラー。

クエリ処理フェーズ

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

EXTFNAPIV4_DESCRIBE_COL_SCALE (get)
EXTFNAPIV4_DESCRIBE_COL_SCALE 属性は、カラムの位取りを示します。describe_column_get のシナリオで使用します。

データ型
a_sql_uint32

説明
カラムの位取り。算術データ型については、パラメータの位取りは、数値の小数点の右側の桁数です。このプロパティはテーブル引数に対してのみ有効です。

使用法
UDF がこのプロパティを取得すると、**CREATE PROCEDURE** 文で定義されているカラムの位取りが返ります。このプロパティは算術データ型に対してのみ有効です。

戻り値
成功時には、値が返された場合は sizeof(a_sql_uint32) を返します。または、以下のエラーを返します。

- EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE – 指定されたカラムのデータ型で位取りを取得できない場合に返されるエラー。

失敗時には、汎用的な describe_column エラーのいずれかを返します。または、以下のエラーを返します。

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – describe_buffer が a_sql_uint32 のサイズでない場合に返される get エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – クエリ処理フェーズが初期フェーズより後でない場合に返される get エラー。

クエリ処理フェーズ

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL (get)

EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL 属性は、カラムが NULL になり得るかどうかを示します。`describe_column_get` のシナリオで使用します。

データ型
`a_sql_byte`

説明
カラムが NULL になり得る場合は true。このプロパティはテーブル引数に対してのみ有効です。このプロパティは引数 0 に対してのみ有効です。

使用法
UDF がこのプロパティを取得すると、カラムが NULL になり得る場合には 1 が、その他の場合には 0 が返ります。

戻り値
成功時には、属性が入手可能な場合は `sizeof(a_sql_byte)` を返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 属性を取得できなかった場合に返されます。これは、カラムがクエリに含まれていなかった場合に発生する可能性があります。

失敗時には、汎用的な `describe_column` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – describe_buffer が `a_sql_byte` のサイズでない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 指定の引数が入力テーブルで、クエリ処理フェーズがプラン構築フェーズより後でない場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- 実行フェーズ

EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES (get)
EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES 属性は、カラム内の重複しない値を表します。`describe_column_get` のシナリオで使用します。

データ型
`a_v4_extfn_estimate`

説明
カラムの重複しない値の推定数。このプロパティはテーブル引数に対してのみ有効です。

使用法
UDF がこのプロパティを取得すると、カラム内の重複しない値の推定数が返ります。

戻り値
成功時には、値が返される場合は `sizeof(a_v4_extfn_estimate)` を返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 属性を取得できなかった場合に返されます。これは、カラムがクエリに含まれていなかった場合に発生する可能性があります。

失敗時には、汎用的な `describe_column` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – describe_buffer が `a_v4_extfn_estimate` のサイズでない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 指定の引数が入力テーブルで、クエリ処理フェーズが最適化フェーズより後でない場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- プラン構築フェーズ
- 実行フェーズ

例
`_describe_extfn` API 関数を使用する、次のようなプロシージャ定義とコードを考えます。

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2 INT ) )
RESULTS ( r1 INT, r2 INT, r3 INT )
EXTERNAL 'my_tpf_proc@mylibrary';
```

```
CREATE TABLE T( x INT, y INT, z INT );

select * from my_tpf( 'test', TABLE( select x,y from T ) )
```

この例は、入力テーブルのカラム 1 の重複しない値の数を TPF が取得する方法を示しています。 TPF では、適切な処理アルゴリズムの選択に役立つ場合に、この値を取得できます。

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state == EXTFNAPIV4_STATE_PLAN_BUILDING ) {
           a_v4_extfn_estimate num_distinct;

           a_sql_int32 ret = 0;

         // Get the number of distinct values expected from the first
column
        // of the table input parameter 'col_table'
           ret = cntxt->describe_column_get( cntxt, 2, 1
                      EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES,
                      &num_distinct,
                      sizeof(a_v4_extfn_estimate) );

           // default algorithm is 1
           _algorithm = 1;

           if( ret > 0 ) {
               // choose the best algorithm for sample size.

               if ( num_distinct.value < 100 ) {
                   // use faster algorithm for small distinct values.
                       _algorithm = 2;
               }
           }
           else {
               if ( ret < 0 ) {
                      // Handle the error
                      //   or continue with default algorithm
               } else {
                      // Attribute was unavailable
                      // We will use the default algorithm.
               }
           }
   }
}
```

EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE (get)
EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE 属性は、カラムがテーブル内でユニークかどうかを示します。`describe_column_get` のシナリオで使用します。

データ型
`a_sql_byte`

説明
カラムがテーブル内でユニークな場合には true。このプロパティはテーブル引数に対してのみ有効です。

使用法
UDF がこのプロパティを取得すると、カラムがユニークな場合には 1 が、それ以外の場合には 0 が返ります。

戻り値
成功時には、`sizeof(a_sql_byte)` を返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 属性を取得できなかった場合。これは、カラムがクエリに含まれていなかった場合に発生する可能性があります。

失敗した場合は、汎用の `describe_column` エラーのいずれかか、次のいずれかを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – describe_buffer が `a_sql_byte` のサイズでない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – クエリ処理フェーズが初期フェーズより後でない場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT (get)
EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT 属性は、カラムが定数かどうかを示します。`describe_column_get` のシナリオで使用します。

データ型
`a_sql_byte`

説明
文の有効期間全体にわたってカラムが定数の場合には true。このプロパティは入力のテーブル引数に対してのみ有効です。

使用法
UDF がこのプロパティを取得すると、文の有効期間全体にわたってカラムが定数の場合には戻り値は 1、その他の場合には戻り値は 0 です。入力テーブルのカラムが定数となるのは、その入力テーブルの SELECT リストでそのカラムが定数式または NULL の場合です。

戻り値
成功時には、値が返された場合は `sizeof(a_sql_byte)` を返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 属性を取得できません。カラムがクエリに含まれていない場合に返されます。

失敗した場合は、汎用の `describe_column` エラーのいずれかか、次のいずれかを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – describe_buffer が `a_sql_byte` のサイズでない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – クエリ処理フェーズが初期以下の場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – 指定された引数が入力テーブルでない場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE (get)

EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE 属性は、カラムの定数を示します。`describe_column_get` のシナリオで使用します。

データ型
`an_extfn_value`

説明
文の有効期間全体で定数の場合のカラムの値。このカラムの `EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT` が true を返す場合、この値を取得できます。このプロパティはテーブル引数に対してのみ有効です。

使用法
定数値を持つ入力テーブルのカラムについては、その値を返します。値を取得できない場合は NULL を返します。

戻り値
成功時には、値が返された場合は `sizeof(a_sql_byte)` を返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 属性を取得できません。カラムがクエリに含まれていない場合、または値が定数と見なされない場合に返されます。

失敗した場合は、汎用の `describe_column` エラーのいずれかか、次のいずれかを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – describe_buffer が `a_sql_byte` のサイズでない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – クエリ処理フェーズが初期以下の場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – 指定された引数が入力テーブルでない場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER (get)

EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER 属性は、結果テーブルのカラムをコンシューマが使用しているかどうかを示します。
`describe_column_get` のシナリオで使用します。

データ型
`a_sql_byte`

説明
結果テーブルのカラムをコンシューマが使用するかどうかを判断するため、または入力内のカラムが不要であることを示すために使用します。テーブル引数に対して有効です。単一のカラムについての情報を設定または取得できます。類似の

属性 EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS では、1 回の呼び出しですべてのカラムの情報を設定および取得できます。

使用法
UDF は、このプロパティを使用して、結果テーブルのカラムをコンシューマが必要としているかどうかを判断します。これは、使用しないカラムに対する不要な処理を UDF が回避するのに役立ちます。

戻り値
成功時には、sizeof(a_sql_byte) を返します。または、以下のエラーを返します。

- EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE – 属性を取得できなかった場合。これは、カラムがクエリに含まれていなかった場合に発生する可能性があります。

失敗した場合は、汎用の describe_column エラーのいずれかか、次のいずれかを返します。

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – describe_buffer が a_v4_extfn_estimate のサイズでない場合に返される get エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – クエリ処理フェーズが初期フェーズより後でない場合に返される get エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – 指定された引数が引数 0 でない場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

_describe_extfn API 関数を使用する PROCEDURE 定義とコードです。

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2
INT ) )
      RESULTS ( r1 INT, r2 INT, r3 INT )
      EXTERNAL 'my_tpf_proc@mylibrary';

CREATE TABLE T( x INT, y INT, z INT );

select r2,r3 from my_tpf( 'test', TABLE( select x,y from T ) )
```

このTPFの実行中は、結果セットのカラム r1 をユーザが選択したかどうかがわかると有益です。ユーザが r1 を必要としていなければ、r1 の計算は不要と考えられ、サーバ用に生成する必要はありません。

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state > EXTFNAPIV4_STATE_INITIAL ) {
      a_sql_byte  col_is_used = 0;
      a_sql_int32 ret = 0;

      ret = cntxt->describe_column_get( cntxt, 0, 1,
            EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER,
            &col_is_used,
            sizeof(a_sql_byte) );

      if( ret < 0 ) {
          // Handle the error.
      }

    }
```

EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE (get)
EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE 属性は、カラムの最小値を示します。describe_column_get のシナリオで使用します。

データ型
an_extfn_value

説明
カラムの最小値 (取得可能な場合)。引数 0 およびテーブル引数に対してのみ有効です。

使用法
UDF が **EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE** プロパティを取得すると、カラムデータの最小値が **describe_buffer** に返ります。入力テーブルがベーステーブルの場合には、最小値はテーブル内のすべてのカラムデータに基づいて決まり、対象のテーブルカラムにインデックスがある場合にのみ使用できます。入力テーブルが別の UDF の結果の場合には、最小値はその UDF が設定した EXTFNAPIV4_DESCRIBE_COL_TYPE です。

このプロパティのデータ型はカラムごとに異なります。UDF は EXTFNAPIV4_DESCRIBE_COL_TYPE を使用してカラムのデータ型を判断できます。さらに UDF は、EXTFNAPIV4_DESCRIBE_COL_WIDTH を使用して、カラムに必要な記憶領域の要件を判断し、過不足のないサイズで値を保持するバッファを提供できます。

サーバは、バッファが有効かどうかを、**describe_buffer_length** を使用して判断できます。

EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE プロパティを入手できない場合、`describe_buffer` は NULL になります。

戻り値
成功時には、`describe_buffer_length` を返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 属性を取得できなかった場合。カラムがクエリに含まれていなかった場合、または要求されたカラムの最小値が入手できなかった場合に返されます。

失敗した場合は、汎用の `describe_column` エラーのいずれかか、次のいずれかを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – describe_buffer が、最小値を保持できるほど十分に大きくない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状態が初期状態より後でない場合に返される get エラー。

クエリ処理の状態
初期状態を除く、以下の状態のときに有効です。

- 注釈状態
- クエリ最適化状態
- プラン構築状態
- 実行状態

例
`_describe_extfn` API 関数を使用するプロシージャ定義とコードです。

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2 INT ) )
    RESULTS ( r1 INT, r2 INT, r3 INT )
    EXTERNAL 'my_tpf_proc@mylibrary';

CREATE TABLE T( x INT, y INT, z INT );

select * from my_tpf( 'test', TABLE( select x,y from T ) )
```

この例は、TPF が内部的な最適化を目的として入力テーブルのカラム 2 の最小値を取得する方法を示します。

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
      if( cntxt->current_state > EXTFNAPIV4_STATE_INITIAL ) {
```

```
            a_sql_int32 min_value = 0;
            a_sql_int32 ret = 0;

         // Get the minimum value of the second column of the
         // table input parameter 'col_table'

         ret = cntxt->describe_column_get( cntxt, 2, 2
                  EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE,
                  &min_value,
                  sizeof(a_sql_int32) );

     if( ret < 0 ) {
               // Handle the error.
         }

      }
}
```

EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE (get)
EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE 属性は、カラムの最大値を示します。`describe_column_get` のシナリオで使用します。

データ型
`an_extfn_value`

説明
カラムの最大値。このプロパティは引数 0 およびテーブル引数に対してのみ有効です。

使用法
UDF が `EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE` プロパティを取得した場合は、 **describe_buffer** でカラムデータの最大値が返されます。入力テーブルがベーステーブルの場合は、最大値はそのテーブル内のカラムデータのすべてに基づいており、テーブルカラムにインデックスがある場合にのみアクセスできます。入力テーブルが別の UDF の結果である場合は、最大値はその UDF によって設定された `COL_MAXIMUM_VALUE` です。

このプロパティのデータ型はカラムごとに異なります。UDF は `EXTFNAPIV4_DESCRIBE_COL_TYPE` を使用してカラムのデータ型を判断できます。さらに UDF は、`EXTFNAPIV4_DESCRIBE_COL_WIDTH` を使用して、カラムに必要な記憶領域の要件を判断し、過不足のないサイズで値を保持するバッファを提供できます。

サーバは、バッファが有効かどうかを、**describe_buffer_length** を使用して判断できます。

EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE を入手できない場合、describe_buffer は NULL になります。

戻り値
成功時には、describe_buffer_length を返します。または、以下のエラーを返します。

- EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE – 属性を取得できなかった場合。これは、カラムがクエリに含まれていなかった場合、または要求されたカラムの最大値が入手できなかった場合に発生する可能性があります。

失敗した場合は、汎用の describe_column エラーのいずれかか、次のいずれかを返します。

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – describe_buffer が、最大値を保持できるほど十分に大きくない場合に返される get エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – クエリ処理フェーズが初期フェーズより後でない場合に返される get エラー。

クエリ処理フェーズ
初期フェーズ以外のあらゆるフェーズで有効:

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

例
_describe_extfn API 関数を使用する **PROCEDURE** 定義とコードです。

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2 INT ) )
         RESULTS ( r1 INT, r2 INT, r3 INT )
         EXTERNAL 'my_tpf_proc@mylibrary';

CREATE TABLE T( x INT, y INT, z INT );

select * from my_tpf( 'test', TABLE( select x,y from T ) )
```

この例は、TPF が内部的な最適化を目的として入力テーブルのカラム 2 の最大値を取得する方法を示します。

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state > EXTFNAPIV4_STATE_INITIAL ) {
         a_sql_int32 max_value = 0;
         a_sql_int32 ret = 0;

         // Get the maximum value of the second column of the
```

```
          // table input parameter  'col_table'
          ret = cntxt->describe_column_get( cntxt, 2, 2
                   EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE,
                   &max_value,
                   sizeof(a_sql_int32) );

          if( ret < 0 ) {
              // Handle the error.
             }
        }
}
```

EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT (get)
EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT 属性では、入力カラムが示す値のサブセットを設定します。この属性を `describe_column_get` のシナリオで使用するとエラーが返ります。

データ型
`a_v4_extfn_col_subset_of_input`

説明
カラム値は入力カラムが示す値のサブセットです。

使用法
この属性は設定のみが可能です。

戻り値
エラー `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE` を返します。

クエリ処理の状態
いずれの状態であっても、エラー `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE` が返されます。

***describe_column_set**
v4 API メソッド `describe_column_set` では、UDF のカラムレベルのプロパティをサーバに対して設定します。

説明
カラムレベルのプロパティでは、結果セットまたは TPF の入力テーブルのカラムについてのさまざまな特性を記述します。たとえば、UDF からサーバに対して、結果セットのカラムは重複しない値を 10 個のみ持つということを伝えることができます。

宣言

```
a_sql_int32 (SQL_CALLBACK *describe_column_set)(
    a_v4_extfn_proc_context        *cntxt,
    a_sql_uint32        arg_num,
    a_sql_uint32        column_num,
    a_v4_extfn_describe_udf_type        describe_type,
    const void        *describe_buffer,
    size_t        describe_buffer_len );
```

パラメータ

パラメータ	説明
cntxt	この UDF のプロシージャコンテキストオブジェクト。
arg_num	TABLE パラメータの順序数 (0 は結果テーブル、1 は最初の入力引数)。
column_num	カラムの順序数 (開始値は 1)。
describe_type	設定するプロパティを示すセレクタ。
describe_buffer	サーバで設定する、指定のプロパティの describe 情報を保持する構造体。具体的な構造体またはデータ型は **describe_type** パラメータで示される。
describe_buffer_length	**describe_buffer** のバイト単位の長さ。

戻り値

成功した場合は、**describe_buffer** に書き込まれたバイト数を返します。エラーが発生した場合や、プロパティが取得されなかった場合は、汎用の `describe_column` エラーのいずれかを返します。

***describe_column_set** の属性*

`describe_column_set` の属性を示すコードです。

```
typedef enum a_v4_extfn_describe_col_type {
    EXTFNAPIV4_DESCRIBE_COL_NAME,
    EXTFNAPIV4_DESCRIBE_COL_TYPE,
    EXTFNAPIV4_DESCRIBE_COL_WIDTH,
    EXTFNAPIV4_DESCRIBE_COL_SCALE,
    EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL,
    EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES,
    EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE,
    EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT,
    EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER,
    EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE,
```

```
    EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT,
    } a_v4_extfn_describe_col_type;
```

EXTFNAPIV4_DESCRIBE_COL_NAME (set)
EXTFNAPIV4_DESCRIBE_COL_NAME 属性はカラム名を示します。
`describe_column_set` のシナリオで使用します。

データ型
`char[]`

説明
カラム名。このプロパティはテーブル引数に対してのみ有効です。

使用法
引数 0 について、UDF がこのプロパティを設定すると、サーバはその値を、**CREATE PROCEDURE** 文で指定されたカラム名と比較します。この比較によって、**CREATE PROCEDURE** 文のカラム名が UDF が予期するものと同じであることを確認できます。

戻り値
成功した場合は、カラム名の長さを返します。

失敗時には、汎用的な `describe_column` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状態が注釈フェーズでない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – パラメータが TABLE パラメータでない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_ INVALID_ATTRIBUTE_VALUE` – 入力カラム名の長さが 128 文字を超えている場合、または入力カラム名とカタログに格納されているカラム名が一致しない場合に返される set エラー。

クエリ処理のフェーズ

- 注釈フェーズ

例

```
short desc_rc = 0;
     char name[7] = 'column1';
     // Verify that the procedure was created with the second column
of the result table as an int
     if( ctx->current_state == EXTFNAPIV4_STATE_ANNOTATION ) {
```

```
          desc_rc = ctx->describe_column_set( ctx, 0, 2,
EXTFNAPIV4_DESCRIBE_COL_NAME,
                                                                  name,
                                                              sizeof(name) );
          if( desc_rc < 0 ) {
          // handle the error.
          }
     }
```

EXTFNAPIV4_DESCRIBE_COL_TYPE (set)
EXTFNAPIV4_DESCRIBE_COL_TYPE 属性は、カラムのデータ型を示しています。これは、describe_column_set のシナリオで使用されます。

データ型
a_sql_data_type

説明
カラムのデータ型。このプロパティはテーブル引数に対してのみ有効です。

使用法
引数 0 について、UDF がこのプロパティを設定すると、サーバはその値を、**CREATE PROCEDURE** 文で指定されたカラムのデータ型と比較します。これによって UDF は、**CREATE PROCEDURE** 文のデータ型が UDF が予期するものと同じであることを確認できます。

戻り値
成功時には、a_sql_data_type を返します。

失敗時には、汎用的な describe_column エラーのいずれかを返します。または、以下のエラーを返します。

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – describe_buffer が a_sql_data_type のサイズでない場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 状態が注釈状態でない場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_ INVALID_ATTRIBUTE_VALUE – 入力データ型とカタログに格納されているデータ型が一致しない場合に返される set エラー。

クエリ処理の状態

- 注釈状態

例

```
short desc_rc = 0;
a_sql_data_type type = DT_INT;
```

```
   // Verify that the procedure was created with the second column of
the result table as an int
    if( ctx->current_state == EXTFNAPIV4_STATE_ANNOTATION ) {
        desc_rc = ctx->describe_column_set( ctx, 0, 2,
EXTFNAPIV4_DESCRIBE_COL_TYPE,
        &type,
                sizeof(a_sql_data_type) );
        if( desc_rc < 0 ) {
            // handle the error.
        }
    }
```

EXTFNAPIV4_DESCRIBE_COL_WIDTH (set)
EXTFNAPIV4_DESCRIBE_COL_WIDTH 属性は、カラム幅を示します。
`describe_column_set` のシナリオで使用します。

データ型
`a_sql_uint32`

説明
カラムの幅。カラム幅は、対応するデータ型の値を格納するために必要な記憶領域のサイズのバイト数です。このプロパティはテーブル引数に対してのみ有効です。

使用法
UDF がこのプロパティを設定すると、サーバはその値を、**CREATE PROCEDURE** 文で指定されたカラム幅と比較します。これによって UDF は、**CREATE PROCEDURE** 文のカラム幅が UDF が予期するものと同じであることを確認できます。

戻り値
成功時には、`sizeof(a_sql_uint32)` を返します。

失敗時には、汎用的な `describe_column` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – describe_buffer が `a_sql_uint32` のサイズでない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – クエリ処理状態が注釈状態でない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – 入力幅とカタログに格納されている幅が一致しない場合に返される set エラー。

クエリ処理の状態
以下のときに有効です。

- 注釈状態

EXTFNAPIV4_DESCRIBE_COL_SCALE (set)
EXTFNAPIV4_DESCRIBE_COL_SCALE 属性は、カラムの位取りを示します。`describe_column_set` のシナリオで使用します。

データ型
`a_sql_uint32`

説明
カラムの位取り。算術データ型については、パラメータの位取りは、数値の小数点の右側の桁数です。このプロパティはテーブル引数に対してのみ有効です。

使用法
UDF がこのプロパティを設定すると、サーバはその値を、**CREATE PROCEDURE** 文で指定された位取りと比較します。これによって UDF は、**CREATE PROCEDURE** 文のカラム幅が UDF が予期するものと同じであることを確認できます。このプロパティは算術データ型に対してのみ有効です。

戻り値
成功時には、`sizeof(a_sql_uint32)` を返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 指定されたカラムのデータ型で位取りを取得できない場合に返される set エラー。

失敗時には、汎用的な `describe_column` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – describe_buffer が `a_sql_uint32` のサイズでない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – クエリ処理状態が注釈状態でない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – 入力位取りとカタログに格納されている位取りが一致しない場合に返される set エラー。

クエリ処理の状態
以下のときに有効です。

- 注釈状態

例

```
    short desc_rc = 0;
    a_sql_uint32 scale = 0;
```

```
           // Verify that the procedure has a scale of zero for the
second result table column.
        if( ctx->current_state == EXTFNAPIV4_STATE_ANNOTATION ) {
        desc_rc = ctx->describe_column_set( ctx, 0, 2,
EXTFNAPIV4_DESCRIBE_COL_SCALE,
            &scale,
            sizeof(a_sql_data_type) );
        if( desc_rc < 0 ) {
            // handle the error.
        }
    }
```

EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL (set)
EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL 属性は、カラムが NULL になり得るかどうかを示します。`describe_column_set` のシナリオで使用します。

データ型
`a_sql_byte`

説明
カラムが NULL になり得る場合は true。このプロパティはテーブル引数に対してのみ有効です。このプロパティは引数 0 に対してのみ有効です。

使用法
UDF は、NULL になり得る結果テーブルカラムに対してこのプロパティを設定できます。UDF がこのプロパティを明示的に設定しない場合、カラムは NULL になり得るものと見なされます。サーバはこの情報を、最適化状態のときに使用できます。

戻り値
成功時には、属性が設定されていた場合は `sizeof(a_sql_byte)` を返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 属性を設定できなかった場合に返されます。これは、カラムがクエリに含まれていなかった場合に発生する可能性があります。

失敗時には、汎用的な `describe_column` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – describe_buffer が `a_sql_byte` のサイズでない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状態が最適化状態でない場合に返される set エラー。

- EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE – UDF がこの属性を 0 でも 1 でもない値に設定しようとした場合に返される set エラー。

クエリ処理の状態
以下のときに有効です。

- 最適化状態

EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES (set)
EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES 属性は、カラム内の重複しない値を表します。describe_column_set のシナリオで使用します。

データ型
a_v4_extfn_estimate

説明
カラムの重複しない値の推定数。このプロパティはテーブル引数に対してのみ有効です。

使用法
UDF は、結果テーブル内のカラムが持つことのできる重複しない値の数を把握している場合にこのプロパティを設定できます。サーバはこの情報を、最適化状態のときに使用します。

戻り値
成功時には、値が設定されている場合は sizeof(a_v4_extfn_estimate) を返します。または、以下のエラーを返します。

- EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE – 属性を設定できなかった場合に返されます。これは、カラムがクエリに含まれていなかった場合に発生する可能性があります。

失敗時には、以下のエラーを返します。

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – describe_buffer が a_v4_extfn_estimate のサイズでない場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 状態が最適化状態でない場合に返される set エラー。

クエリ処理の状態
以下のときに有効です。

- 最適化状態

EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE (set)
EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE 属性は、カラムがテーブル内でユニークかどうかを示します。`describe_column_set` のシナリオで使用します。

データ型
`a_sql_byte`

説明
カラムがテーブル内でユニークな場合には true。このプロパティはテーブル引数に対してのみ有効です。

使用法
UDF は、結果テーブルのカラム値がユニークかどうかを把握している場合にこのプロパティを設定できます。サーバはこの情報を、最適化状態のときに使用します。UDF はこのプロパティを引数 0 に対してのみ設定できます。

戻り値
成功時には、`sizeof(a_sql_byte)` を返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 属性を設定できなかった場合。これは、カラムがクエリに含まれていなかった場合に発生する可能性があります。

失敗時には、汎用的な `describe_column` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – describe_buffer が `a_sql_byte` のサイズでない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – クエリ処理状態が最適化状態でない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – **arg_num** が 0 でない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – UDF がこの属性を 0 でも 1 でもない値に設定しようとした場合に返される set エラー。

クエリ処理の状態
以下のときに有効です。

- 最適化状態

EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT (set)
EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT 属性は、カラムが定数かどうかを示します。describe_column_set のシナリオで使用します。

データ型
a_sql_byte

説明
文の有効期間全体にわたってカラムが定数の場合には true。このプロパティは入力のテーブル引数に対してのみ有効です。

使用法
これは読み込み専用のプロパティです。設定しようとすると、常に EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE が返されます。

戻り値

- EXTFNAPIV4_DESCRIBE_INVALID ATTRIBUTE – これは読み込み専用のプロパティです。設定しようとすると、常にこのエラーが返されます。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 状態が最適化でない場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – **arg_num** が 0 でない場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE – UDF がこの属性を 0 でも 1 でもない値に設定しようとした場合に返される set エラー。

クエリ処理の状態
適用されません。

EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE (set)
EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE 属性は、カラムの定数を示します。describe_column_set のシナリオで使用します。

データ型
an_extfn_value

説明
文の有効期間全体で定数の場合のカラムの値。このカラムの EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT が true を返す場合、この値を取得できます。このプロパティはテーブル引数に対してのみ有効です。

使用法
このプロパティは読み込み専用です。

戻り値

- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE` – これは読み込み専用のプロパティです。設定しようとすると、常にこのエラーが返されます。

クエリ処理のフェーズ
適用されません。

EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER (set)
EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER 属性は、結果テーブルのカラムをコンシューマが使用しているかどうかを示します。
`describe_column_set` のシナリオで使用します。

データ型
`a_sql_byte`

説明
結果テーブルのカラムをコンシューマが使用するかどうかを判断するため、または入力内のカラムが不要であることを示すために使用します。テーブル引数に対して有効です。単一のカラムについての情報を設定または取得できます。類似の属性 `EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS` では、1 回の呼び出しですべてのカラムの情報を設定および取得できます。

使用法
UDF は、入力テーブルのカラムに
EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER を設定することによって、そのカラムの値が必要ないことをプロデューサに伝えます。

戻り値
成功時には、`sizeof(a_sql_byte)` を返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 属性を設定できなかった場合。これは、カラムがクエリに含まれていなかった場合に発生する可能性があります。

失敗した場合は、汎用の `describe_column` エラーのいずれかか、次のいずれかを返します。

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – describe_buffer が a_v4_extfn_estimate のサイズでない場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – 指定された属性が属性 0 の場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 状態が最適化状態でない場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE – UDF が設定している値が 0 でも 1 でもない場合に返される set エラー。

クエリ処理の状態
以下のときに有効です。

- 最適化状態

_describe_extfn API 関数を使用する PROCEDURE 定義とコードです。

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2
INT ) )
      RESULTS ( r1 INT, r2 INT, r3 INT )
      EXTERNAL 'my_tpf_proc@mylibrary';

CREATE TABLE T( x INT, y INT, z INT );

select r2,r3 from my_tpf( 'test', TABLE( select x,y from T ) )
```

この TPF の実行中は、カラム y がこの TPF によって使用されているかどうかをサーバで認識するようになっていると有益です。TPF が y を必要としていない場合、サーバはこの知識を最適化に使用でき、このカラム情報を TPF に送信しません。

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
      a_sql_byte  col_is_used = 0;
      a_sql_int32 ret = 0;

      ret = cntxt->describe_column_get( cntxt, 2, 2,
            EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER,
            &col_is_used,
            sizeof(a_sql_byte) );

      if( ret < 0 ) {
          // Handle the error.
      }

    }
}
```

EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE (set)
EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE 属性は、カラムの最小値を示します。`describe_column_set` のシナリオで使用します。

データ型
`an_extfn_value`

説明
カラムが持つことのできる最小値です (使用可能な場合)。引数 0 に対してのみ有効です。

使用法
UDF は、カラムのデータの最小値を把握している場合には、`EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE` を設定できます。サーバはこの情報を、最適化のときに使用できます。

UDF は、`EXTFNAPIV4_DESCRIBE_COL_TYPE` を使用してカラムのデータ型を判断することと、`EXTFNAPIV4_DESCRIBE_COL_WIDTH` を使用してカラムの記憶領域の要件を判断することによって、設定する値を過不足のないサイズで保持するバッファを提供できます。

戻り値
成功した場合は、`describe_buffer_length` を返すか、または次を返します。

- `EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE` – 属性を設定できない。 対象のカラムがクエリに含まれていない場合か、対象のカラムの最小値を使用できない場合に返る。

失敗した場合は、汎用の `describe_column` エラーのいずれかか、次のいずれかを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – describe バッファの大きさが、最小値を保持できる十分なサイズでない場合に返る set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状態が最適化状態でない場合に返る set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – **arg_num** が 0 でない場合に返る set エラー。

クエリ処理の状態
以下のときに有効です。

- 最適化状態

例
_describe_extfn コールバック API 関数を実装する **PROCEDURE** 定義と UDF コードです。

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2 INT ) )
    RESULTS ( r1 INT, r2 INT, r3 INT )
    EXTERNAL 'my_tpf_proc@mylibrary';

CREATE TABLE T( x INT, y INT, z INT );

select * from my_tpf( 'test', TABLE( select x,y from T ) )
```

この例は、結果セットのカラム 1 の最小値を把握することがサーバにとって役に立つ (または、この TPF の結果を入力として使用する別の TPF にとって役に立つ) TPF の例です。この例では、カラム 1 の最小の出力値は 27 です。

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
    a_sql_int32 min_value = 27;
    a_sql_int32 ret = 0;

// Tell the server what the minimum value of the first column
// of our result set will be.

    ret = cntxt->describe_column_set( cntxt, 0, 1
        EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE,
        &min_value,
        sizeof(a_sql_int32) );

    if( ret < 0 ) {
        // Handle the error.
    }
}
}
```

EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE (set)
EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE 属性は、カラムの最大値を示します。describe_column_set のシナリオで使用します。

データ型
an_extfn_value

説明
カラムの最大値。このプロパティは引数 0 およびテーブル引数に対してのみ有効です。

使用法
UDF は、カラムのデータの最大値を把握している場合には、EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE を設定できます。サーバはこの情報を、最適化のときに使用できます。

UDF は、EXTFNAPIV4_DESCRIBE_COL_TYPE を使用してカラムのデータ型を判断することと、EXTFNAPIV4_DESCRIBE_COL_WIDTH を使用してカラムの記憶領域の要件を判断することによって、設定する値を過不足のないサイズで保持するバッファを提供できます。

describe_buffer_length は、このバッファの sizeof() です。

戻り値
成功時には、値が設定されていた場合は describe_buffer_length を返します。または、以下のエラーを返します。

- EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE – 属性を設定できなかった場合。カラムがクエリに含まれていなかった場合、または要求されたカラムの最大値が入手できなかった場合に返されます。

失敗した場合は、汎用の describe_column エラーのいずれかか、次のいずれかを返します。

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – describe_buffer が、最大値を保持できるほど十分に大きくない場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – クエリ処理フェーズが最適化フェーズでない場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – **arg_num** が 0 でない場合に返される set エラー。

クエリ処理のフェーズ
以下のときに有効です。

- 最適化フェーズ

例
_describe_extfn コールバック API 関数を実装する PROCEDURE 定義と UDF コードです。

```
CREATE PROCEDURE my_tpf( col_char char(10), col_table TABLE( c1 INT,
c2 INT ) )
         RESULTS ( r1 INT, r2 INT, r3 INT )
         EXTERNAL 'my_tpf_proc@mylibrary';

CREATE TABLE T( x INT, y INT, z INT );
```

```
select * from my_tpf( 'test', TABLE( select x,y from T ) )
```

この例は、結果セットのカラム 1 の最大値を把握することがサーバにとって役に立つ (または、この TPF の結果を入力として使用する別の TPF にとって役に立つ) TPF の例です。この例では、カラム 1 の最大の出力値は 500000 です。

```
my_tpf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
         a_sql_int32 max_value = 500000;
         a_sql_int32 ret = 0;

        // Tell the server what the maximum value of the first column
         // of our result set will be.

      ret = cntxt->describe_column_set( cntxt, 0, 1
                  EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE,
                  &max_value,
                  sizeof(a_sql_int32) );

         if( ret < 0 ) {
             // Handle the error.
         }

   }
}
```

EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT (set)
EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT 属性では、入力カラムが示す値のサブセットを設定します。`describe_column_set` のシナリオで使用します。

データ型
`a_v4_extfn_col_subset_of_input`

説明
カラム値は入力カラムが示す値のサブセットです。

使用法
この describe 属性を設定すると、指定のカラムの値が入力カラムで指定された値のサブセットであることをクエリオプティマイザに伝えることができます。たとえば、テーブルを利用し、関数に基づいてローをフィルタリングするフィルタ TPF があるとします。この場合、出力テーブルは入力テーブルのサブセットです。このフィルタ TPF で **EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT** を設定するとクエリを最適化できます。

戻り値
成功時には、sizeof(a_v4_extfn_col_subset_of_input) を返します。

失敗した場合は、汎用の describe_column エラーのいずれかか、次のいずれかを返します。

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – バッファの長さが sizeof (a_v4_extfn_col_subset_of_input) 未満の場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE – ソーステーブルのカラムインデックスが範囲外である場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE – subset_of_input が設定されているカラムが該当しない (たとえば、このカラムが SELECT リストの中にない) 場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_INVAILD_STATE – クエリ処理状態が最適化状態でない場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – バッファの長さが 0 の場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – 結果テーブル以外のパラメータで呼び出された場合に返される set エラー。

クエリ処理の状態
以下のときに有効です。

- 最適化状態

例

```
a_v4_extfn_col_subset_of_input colMap;

    colMap.source_table_parameter_arg_num = 4;
    colMap.source_column_number = i;

    desc_rc = ctx->describe_column_set( ctx,
        0, i,
        EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT,
        &colMap, sizeof(a_v4_extfn_col_subset_of_input) );
```

***describe_parameter_get**
v4 API メソッド describe_parameter_get では、UDF のパラメータのプロパティをサーバから取得します。

宣言

```
a_sql_int32 (SQL_CALLBACK *describe_parameter_get)(
            a_v4_extfn_proc_context             *cntxt,
```

```
            a_sql_uint32                          arg_num,
            a_v4_extfn_describe_udf_type          describe_type,
            const void                            *describe_buffer,
           size_t                               describe_buffer_len );
```

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。
arg_num	TABLE パラメータの順序数 (0 は結果テーブル、1 は最初の入力引数)。
describe_type	設定するプロパティを示すセレクタ。
describe_buffer	サーバで設定する、指定のプロパティの describe 情報を保持する構造体。具体的な構造体またはデータ型は **describe_type** パラメータで示される。
describe_buffer_length	**describe_buffer** のバイト単位の長さ。

戻り値

成功した場合は、0 または **describe_buffer** に書き込まれたバイト数を返します。値 0 の場合、サーバは属性を取得できなかったが、エラー状態は発生しなかったことを示します。エラーが発生した場合や、プロパティが取得されなかった場合は、汎用の **describe_parameter** エラーのいずれかを返します。

**describe_parameter_get の属性*

`describe_parameter_get` の属性を示すコードです。

```
typedef enum a_v4_extfn_describe_parm_type {
    EXTFNAPIV4_DESCRIBE_PARM_NAME,
    EXTFNAPIV4_DESCRIBE_PARM_TYPE,
    EXTFNAPIV4_DESCRIBE_PARM_WIDTH,
    EXTFNAPIV4_DESCRIBE_PARM_SCALE,
    EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL,
    EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES,
    EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT,
    EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE,

    EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS,
```

```
} a_v4_extfn_describe_parm_type;
```

EXTFNAPIV4_DESCRIBE_PARM_NAME 属性 (get)
EXTFNAPIV4_DESCRIBE_PARM_NAME 属性は、パラメータ名を示しています。これは、describe_parameter_get のシナリオで使用されます。

データ型
char[]

説明
UDF に対するパラメータの名前。

使用法
CREATE PROCEDURE 文で定義されているパラメータ名を取得します。パラメータ 0 に対しては無効です。

戻り値
成功した場合は、パラメータ名の長さを返します。

失敗時には、汎用的な describe_parameter エラーのいずれかを返します。または、以下のエラーを返します。

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – **describe_buffer** が、名前を保持できるほど十分に大きくない場合に返される get エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – クエリ処理フェーズが初期フェーズより後でない場合に返される get エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – パラメータが結果テーブルの場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

EXTFNAPIV4_DESCRIBE_PARM_TYPE 属性 (get)
EXTFNAPIV4_DESCRIBE_PARM_TYPE 属性は、describe_parameter_get のシナリオでデータ型を返します。

データ型
a_sql_data_type

説明
UDF に対するパラメータのデータ型。

使用法
CREATE PROCEDURE 文で定義されているパラメータのデータ型を取得します。

戻り値
成功時には、`sizeof(a_sql_data_type)` を返します。

失敗した場合は、汎用の `describe_parameter` エラーのいずれかか、次のいずれかを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** が `sizeof(a_sql_data_type)` でない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – クエリ処理フェーズが初期フェーズより後でない場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

EXTFNAPIV4_DESCRIBE_PARM_WIDTH 属性 (get)

EXTFNAPIV4_DESCRIBE_PARM_WIDTH 属性は、パラメータの幅を示します。`describe_parameter_get` のシナリオで使用します。

データ型
`a_sql_uint32`

説明
UDF に対するパラメータの幅。`EXTFNAPIV4_DESCRIBE_PARM_WIDTH` はスカラパラメータにのみ該当します。パラメータ幅は、対応するデータ型のパラメータを格納するために必要な記憶領域のサイズのバイト数です。

- **固定長のデータ型** – データの格納に必要なバイト数
- **可変長のデータ型** – 最大長
- **LOB データ型** – データへのハンドルの格納に必要な記憶領域のサイズ
- **TIME データ型** – コード化した時間の格納に必要な記憶領域のサイズ

使用法
CREATE PROCEDURE 文で定義されているパラメータの幅を取得します。

戻り値
成功時には、`sizeof(a_sql_uint32)` を返します。

失敗時には、汎用的な `describe_parameter` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – クエリ処理フェーズが初期フェーズより後でない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** が `a_sql_uint32` のサイズでない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – 指定されたパラメータが TABLE パラメータの場合に返される get エラー。これには、パラメータ 0 またはパラメータ *n* が含まれます。この *n* は入力テーブルです。

クエリ処理フェーズ
以下のときに有効です。

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

例
プロシージャ定義の例です。

```
CREATE PROCEDURE my_udf(IN p1 INT, IN p2 char(100))
RESULT (x INT)
EXTERNAL NAME 'my_udf@myudflib';
```

`_describe_extfn` API 関数のコードの例です。

```
my_udf_describe(a_v4_extfn_proc_context *cntxt)
{
    if( cntxt->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
    a_sql_uint32 width = 0;
    a_sql_int32 ret = 0;

    // Get the width of parameter 1
    ret = cntxt->describe_parameter_get( cntxt, 1,
    EXTFNAPIV4_DESCRIBE_PARM_WIDTH,
    &width,
    sizeof(a_sql_uint32) );

    if( ret < 0 ) {
        // Handle the error.
```

```
    }

    //Allocate some storage based on parameter width
    a_sql_byte *p = (a_sql_byte *)cntxt->alloc( cntxt, width )

    // Get the width of parameter 2
    ret = cntxt->describe_parameter_get( cntxt, 2,
        EXTFNAPIV4_DESCRIBE_PARM_WIDTH,
        &width,
        sizeof(a_sql_uint32) );
    if( ret <= 0 ) {
        // Handle the error.
    }

    // Allocate some storage based on parameter width
    char *c = (char *)cntxt->alloc( cntxt, width )

    …

}
}
```

EXTFNAPIV4_DESCRIBE_PARM_SCALE 属性 (get)
EXTFNAPIV4_DESCRIBE_PARM_SCALE 属性は、パラメータの位取りを示します。`describe_parameter_get` のシナリオで使用します。

データ型
`a_sql_uint32`

説明
UDF に対するパラメータの位取り。算術データ型については、パラメータの位取りは、数値の小数点の右側の桁数です。

この属性は以下に対しては無効です。

- 算術データ型以外のデータ型
- TABLE パラメータ

使用法
CREATE PROCEDURE 文で定義されているパラメータの位取りを取得します。

戻り値
成功時には、`(a_sql_uint32)` のサイズを返します。

失敗時には、汎用的な `describe_parameter` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** が `a_sql_uint32` のサイズでない場合に返される get エラー。

- EXTFNAPIV4_DESCRIBE_INVALID_STATE – クエリ処理フェーズが初期フェーズより後でない場合に返される get エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – 指定されたパラメータが TABLE パラメータの場合に返される get エラー。これには、パラメータ 0 またはパラメータ *n* が含まれます。この *n* は入力テーブルです。

クエリ処理フェーズ
以下のときに有効です。

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

例
パラメータ 1 の位取りを取得する _describe_extfn API 関数のコードの例です。

```
if( cntxt->current_state > EXTFNAPIV4_STATE_ANNOTATION ) {
    a_sql_uint32 scale = 0;
    a_sql_int32 ret = 0;

    ret = ctx->describe_parameter_get( ctx, 1,
EXTFNAPIV4_DESCRIBE_PARM_SCALE,
    &scale, sizeof(a_sql_uint32) );

    if( ret <= 0 ) {
    // Handle the error.
    }
}
```

EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL 属性 (get)

EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL 属性は、パラメータが NULL かどうかを示します。describe_parameter_get のシナリオで使用します。

データ型
a_sql_byte

説明
実行時にパラメータの値を NULL にできる場合は true。TABLE パラメータまたはパラメータ 0 の場合、値は false です。

使用法
クエリの実行中に、指定のパラメータが NULL の可能性があるかどうかを取得します。

戻り値
成功時には、`sizeof(a_sql_byte)` を返します。

失敗時には、汎用的な `describe_parameter` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** が `a_sql_byte` のサイズでない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – クエリ処理フェーズがプラン構築フェーズより後でない場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- 実行フェーズ

例：EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL (get)
プロシージャ定義、`_describe_extfn` API 関数のコード、**EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL** の値を取得する SQL クエリの例です。

プロシージャ定義
この項のサンプルクエリで使用するプロシージャ定義の例を示します。

```
CREATE PROCEDURE my_udf(IN p INT)
RESULT (x INT)
EXTERNAL NAME 'my_udf@myudflib';
```

API 関数のコード
この項のサンプルクエリで使用する `_describe_extfn` API 関数のコードの例を示します。

```
my_udf_describe(a_v4_extfn_proc_context *cntxt)
{

    if( cntxt->current_state > EXTFNAPIV4_STATE_OPTIMIZATION ) {
    a_sql_byte can_be_null = 0;
    a_sql_int32 ret = 0;

    ret = cntxt->describe_parameter_get( cntxt, 1,
    EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL,
    &can_be_null,
    sizeof(a_sql_byte) );

    if( ret <= 0 ) {
        // Handle the error.
    }
}
}
```

例 1： NOT NULL なしの場合
この例では、単一の整数カラムを持つテーブルを、**NOT NULL** 変更子を指定せずに作成しています。相関サブクエリではテーブル has_nulls のカラム *c1* を渡しています。実行状態のときにプロシージャ **my_udf_describe** が呼び出されると、describe_parameter_get の呼び出しで **can_be_null** は値 1 に設定されます。

```
CREATE TABLE has_nulls ( c1 INT );
INSERT INTO has_nulls VALUES(1);
INSERT INTO has_nulls VALUES(NULL);
SELECT * from has_nulls WHERE (SELECT sum(my_udf.x) FROM
my_udf(has_nulls.c1)) > 0;
```

例 2： NOT NULL ありの場合
この例では、単一の整数カラムを持つテーブルを、**NOT NULL** 変更子を指定して作成しています。相関サブクエリではテーブル no_nulls のカラム *c1* を渡しています。実行状態のときにプロシージャ **my_udf_describe** が呼び出されると、describe_parameter_get の呼び出しで **can_be_null** は値 0 に設定されます。

```
CREATE TABLE no_nulls ( c1 INT NOT NULL);
INSERT INTO no_nulls VALUES(1);
INSERT INTO no_nulls VALUES(2);
SELECT * from no_nulls WHERE (SELECT sum(my_udf.x) FROM
my_udf(no_nulls.c1)) > 0;
```

例 3： 定数の場合
この例では、プロシージャ **my_udf** を定数で呼び出しています。実行状態のときにプロシージャ **my_udf_describe** が呼び出されると、describe_parameter_get の呼び出しで **can_be_null** は値 0 に設定されます。

```
SELECT * from my_udf(5);
```

例 4： NULL の場合
この例では、プロシージャ **my_udf** を NULL で呼び出しています。実行状態のときにプロシージャ **my_udf_describe** が呼び出されると、describe_parameter_get の呼び出しで **can_be_null** は値 1 に設定されます。

```
SELECT * from my_udf(NULL);
```

EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES 属性 (get)
EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES 属性は、重複しない値の数を返します。describe_parameter_get のシナリオで使用します。

データ型
a_v4_extfn_estimate

説明
すべての呼び出し全体での重複しない値の推定数を返します。スカラパラメータに対してのみ有効です。

使用法
この情報を使用できる場合、UDF は重複しない値の推定数を 100% の確度で返します。この情報を使用できない場合、UDF は推定数 0 を 0% の確度で返します。

戻り値
成功時には、`sizeof(a_v4_extfn_estimate)` を返します。

失敗時には、汎用的な `describe_parameter` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** が `a_v4_extfn_estimate` のサイズでない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – クエリ処理フェーズが初期フェーズより後でない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – パラメータが TABLE パラメータの場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

例
`_describe_extfn` API 関数のコードの例です。

```
if( ctx->current_state >= EXTFNAPIV4_STATE_ANNOTATION ) {
    desc_est.value = 0.0;
    desc_est.confidence = 0.0;

    desc_rc = ctx->describe_parameter_get( ctx,
        1,
        EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES,
        &desc_est, sizeof(a_v4_extfn_estimate) );
}
```

EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT 属性 (get)
EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES 属性は、パラメータが定数かどうかを返します。これは、describe_parameter_get のシナリオで使用されます。

データ型
a_sql_byte

説明
文のパラメータが定数の場合は true。スカラパラメータに対してのみ有効です。

使用法
指定のパラメータの値が定数でない場合は 0 を返します。指定のパラメータの値が定数の場合は 1 を返します。

戻り値
成功時には、sizeof(a_sql_byte) を返します。

失敗時には、汎用的な describe_parameter エラーのいずれかを返します。または、以下のエラーを返します。

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – **describe_buffer** が a_sql_byte のサイズでない場合に返される get エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – クエリ処理フェーズが初期フェーズより後でない場合に返される get エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – パラメータが TABLE パラメータの場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

例
_describe_extfn API 関数のコードの例です。

```
if( ctx->current_state >= EXTFNAPIV4_STATE_ANNOTATION ) {
    desc_rc = ctx->describe_parameter_get( ctx,
        1,
        EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT,
```

```
        &desc_byte, sizeof( a_sql_byte ) );
}
```

EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE 属性 (get)
EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE 属性は、パラメータの値を示します。`describe_parameter_get` のシナリオで使用します。

データ型
`an_extfn_value`

説明
パラメータの値 (describe の時点で把握している場合)。スカラパラメータに対してのみ有効です。

使用法
パラメータの値を返します。

戻り値
成功時には、値が入手可能な場合は `sizeof(an_extfn_value)` を返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_NOT_AVILABLE` – 値が定数でない場合に返される値。

失敗時には、汎用的な `describe_parameter` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** が `an_extfn_value` のサイズでない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – フェーズが初期フェーズより後でない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – パラメータが TABLE パラメータの場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

例
`_describe_extfn` API 関数のコードの例です。

```
if( ctx->current_state >= EXTFNAPIV4_STATE_ANNOTATION ) {
    a_sql_int32 desc_rc;
    desc_rc = ctx->describe_parameter_get( ctx,
        1,
        EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE,
        &arg,
        sizeof( an_extfn_value ) );
}
```

EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS 属性 (get)
EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS 属性は、テーブル内のカラムの数を示しています。これは、describe_parameter_get のシナリオで使用されます。

データ型
a_sql_uint32

説明
テーブル内のカラム数。引数 0 およびテーブル引数に対してのみ有効です。

使用法
指定のテーブル引数に含まれるカラム数を返します。引数 0 は、結果テーブルに含まれるカラム数を返します。

戻り値
成功時には、sizeof(a_sql_uint32) を返します。

失敗時には、汎用的な describe_parameter エラーのいずれかを返します。または、以下のエラーを返します。

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – **describe_buffer** が sizeof a_sql_uint32 のサイズでない場合に返される get エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – クエリ処理フェーズが初期フェーズより後でない場合に返される get エラー。
- EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER – パラメータが TABLE パラメータでない場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS 属性 (get)
EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS 属性は、テーブル内のローの数を示しています。これは、describe_parameter_get のシナリオで使用されます。

データ型
a_v4_extfn_estimate

説明
テーブルの推定ロー数。引数 0 およびテーブル引数に対してのみ有効です。

使用法
指定のテーブル引数または結果セットに含まれるローの推定数を 100% の確度で返します。

戻り値
成功時には、a_v4_extfn_estimate のサイズを返します。

失敗した場合は、汎用の describe_parameter エラーのいずれかか、次のいずれかを返します。

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – **describe_buffer** が a_v4_extfn_estimate のサイズでない場合に返される get エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – クエリ処理フェーズが初期フェーズより後でない場合に返される get エラー。
- EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER – パラメータが TABLE パラメータでない場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY 属性 (get)
EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY 属性は、テーブル内のローの順序を示しています。これは、describe_parameter_get のシナリオで使用されます。

データ型
a_v4_extfn_orderby_list

説明
テーブル内のローの順序。このプロパティは引数 0 およびテーブル引数に対してのみ有効です。

使用法
この属性を使用することによって、UDF コードは以下が可能になります。

- 入力の **TABLE** パラメータが順序付けされているかどうかを判断する。
- 結果セットが順序付けされていることを宣言する。

パラメータ番号が 0 の場合、この属性は出力の結果セットを表します。パラメータが 0 より大きく、パラメータの種類がテーブルの場合、この属性は入力の **TABLE** パラメータを表します。

順序付けは a_v4_extfn_orderby_list で指定されます。これは、カラムの順序数と、それに対応する昇順または降順のプロパティのリストを保持する構造体です。UDF が出力の結果セットに対してこの順序付けのプロパティを設定した場合、サーバは順序付けの最適化を実行できます。たとえば、UDF が結果セットの最初のカラムを昇順で生成した場合、サーバは同じカラムに対する重複した順序付けの要求を省略できます。

UDF が出力の結果セットに対して順序付けのプロパティを設定しない場合、サーバはデータが順序付けされていないものと見なします。

UDF が入力の **TABLE** パラメータに順序付けのプロパティを設定する場合、その入力データに対する順序付けはサーバが保証します。このシナリオでは、入力データの順序付けが必要であることを UDF がサーバに伝達します。サーバは、実行時に競合を検出した場合は SQL 例外を発生させます。たとえば、入力の **TABLE** パラメータの最初のカラムは昇順である必要があると UDF が伝達したのに対し、SQL 文に降順を指定する句が含まれている場合、サーバは SQL 例外を発生させます。

SQL に順序付けの句が含まれていない場合、サーバは順序付けを自動的に追加します。これにより、入力の **TABLE** パラメータが要件どおりに順序付けされます。

戻り値
成功した場合は、`a_v4_extfn_orderby_list` からコピーされたバイトの数を返します。

クエリ処理の状態
以下のときに有効です。

- 注釈状態
- クエリ最適化状態

EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY (get)
`EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY` 属性は、UDF が分割を必要としていることを示しています。これは、`describe_parameter_get` のシナリオで使用されます。

データ型
`a_v4_extfn_column_list`

説明
UDF 開発者は、**EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY** を使用して、呼び出しの前にパーティション分割が必要であることを UDF のコードで宣言します。

使用法
UDF は、パーティションの適用またはパーティション分割の動的な実施のための確認を実行できます。**a_v4_extfn_column_list** を確保することは UDF の役割です。そのときは、入力テーブルに含まれるカラムの総数を考慮に入れ、そのデータをサーバに送信します。

戻り値
成功時には、`a_v4_extfn_column_list` のサイズを返します。この値は次と同じです。

```
sizeof(a_v4_extfn_column_list) + sizeof(a_sql_uint32) *
number_of_partition_columns
```

失敗した場合は、汎用の `describe_parameter` エラーのいずれかか、次のいずれかを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – バッファの長さが予期したサイズ未満の場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

例

```
void UDF_CALLBACK my_tpf_proc_describe( a_v4_extfn_proc_context
*ctx )
{
    if( ctx->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
        a_sql_uint32    col_count   = 0;
        a_sql_uint32    buffer_size = 0;
        a_v4_extfn_column_list    *clist = NULL;

        col_count = 3;    // Set to the max number of possible pby
columns

       buffer_size = sizeof( a_v4_extfn_column_list ) + (col_count -
1) * sizeof( a_sql_uint32 );

        clist = (a_v4_extfn_column_list *)ctx->alloc( ctx,
buffer_size );

        clist->number_of_columns = 0;
        clist->column_indexes[0] = 0;
        clist->column_indexes[1] = 0;
        clist->column_indexes[2] = 0;

    args->r_api_rc = ctx->describe_parameter_get( ctx,
        args->p3_arg_num,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY,
    clist,
    buffer_size );
}
}
```

EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND 属性 (get)

`EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND` 属性は、コンシューマが入力テーブルのリワインドを要求していることを示します。これは、`describe_parameter_get` のシナリオで使用されます。

データ型
`a_sql_byte`

説明
コンシューマが入力テーブルのリワインドを希望していることを示します。テーブル入力引数に対してのみ有効です。デフォルトでは、このプロパティは false です。

使用法
UDF はこのプロパティを確認して true/false 値を取得します。

戻り値
成功時には、`sizeof(a_sql_byte)` を返します。

失敗時には、汎用的な `describe_parameter` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** が `a_sql_byte` のサイズでない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – フェーズが最適化フェーズとプラン構築フェーズのいずれでもない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – UDF がパラメータ 0 でこの属性を取得しようとした場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – UDF がテーブルでないパラメータでこの属性を取得しようとした場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- 最適化フェーズ
- プラン構築フェーズ

EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND 属性 (get)

`EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND` 属性は、パラメータがリワインドをサポートしているかどうかを示しています。これは、`describe_parameter_get` のシナリオで使用されます。

データ型
`a_sql_byte`

説明
プロデューサがリワインドをサポートできるかどうかを示します。テーブル引数に対してのみ有効です。

`DESCRIBE_PARM_TABLE_HAS_REWIND` を `true` に設定する予定の場合は、リワインドテーブルコールバック (**_rewind_extfn()**) の実装を提供する必要もあります。コールバックメソッドが提供されていないと、サーバは UDF の実行に失敗します。

使用法
UDF は、テーブルの入力引数がリワインドをサポートしているかどうかを確認します。このプロパティを使用する前に、前提条件として、UDF は **DESCRIBE_PARM_TABLE_REQUEST_REWIND** を使用してリワインドを要求する必要があります。

戻り値
成功時には、`sizeof(a_sql_byte)` を返します。

失敗時には、汎用的な `describe_parameter` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** が `a_sql_byte` のサイズでない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – クエリ処理フェーズが注釈フェーズより後でない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – UDF がテーブルでないパラメータでこの属性を取得しようとした場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – UDF が結果テーブルでこの属性を取得しようとした場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- 最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS 属性 (get)
`EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS` 属性は、利用しないカラムをリストします。これは、`describe_parameter_get` のシナリオで使用されます。

データ型
`a_v4_extfn_column_list`

説明
サーバまたは UDF が利用しない出力テーブルカラムのリストです。

出力 TABLE パラメータについては、UDF は通常すべてのカラムに対してデータを生成し、サーバはすべてのカラムを利用します。入力 TABLE パラメータ につ

いても同様で、サーバは通常すべてのカラムに対してデータを生成し、UDF はすべてのカラムを利用します。

しかし場合によっては、サーバまたは UDF が一部のカラムを利用しないこともあります。そのような場合の最善の方法は、UDF が describe 属性 **EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS** を使用して、出力テーブルに対する **GET** を実行することです。この処理は、出力テーブルのうちでサーバが利用しないカラムのリストをサーバに問い合わせる処理です。UDF はこのリストを、出力テーブルのカラムデータを格納するときに利用できます。つまり、使用しないカラムに対するデータの格納を省略できます。

要約すると、出力テーブルについては、使用しないカラムのリストを UDF が問い合わせます。入力テーブルについては、使用しないカラムのリストを UDF がプッシュします。

使用法
UDF は、出力テーブルのすべてのカラムを利用するかどうかをサーバに問い合わせます。UDF は出力テーブルのすべてのカラムを含めた `a_v4_extfn_column_list` を確保してサーバに渡す必要があります。これを受けてサーバは、使用しないすべてのカラムの位置を 1 とマークします。サーバから返されたリストはデータを生成するときに使用できます。

戻り値
成功時には、カラムリストのサイズを返します。
`sizeof(a_v4_extfn_column_list) + sizeof(a_sql_uint32) * number result columns.`

失敗した場合は、汎用の `describe_parameter` エラーのいずれかか、次のいずれかを返します。

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – クエリ処理フェーズがプラン構築フェーズより後でない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** が、返されたリストを保持できるほど十分に大きくない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – UDF が入力テーブルでこの属性を取得しようとした場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – UDF がテーブルでないパラメータでこの属性を取得しようとした場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- 実行フェーズ

*describe_parameter_set

v4 API メソッド describe_parameter_set では、単一のパラメータについてのプロパティを UDF に対して設定します。

宣言

```
a_sql_int32 (SQL_CALLBACK *describe_parameter_set)(
            a_v4_extfn_proc_context             *cntxt,
            a_sql_uint32                        arg_num,
            a_v4_extfn_describe_udf_type        describe_type,
            const void                          *describe_buffer,
           size_t                             describe_buffer_len );
```

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。
arg_num	TABLE パラメータの順序数 (0 は結果テーブル、1 は最初の入力引数)。
describe_type	設定するプロパティを示すセレクタ。
describe_buffer	サーバで設定する、指定のプロパティの describe 情報を保持する構造体。具体的な構造体またはデータ型は **describe_type** パラメータで示される。
describe_buffer_length	**describe_buffer** のバイト単位の長さ。

戻り値

成功した場合は、0 または **describe_buffer** に書き込まれたバイト数を返します。値 0 の場合、サーバは属性を設定できなかったが、エラー状態は発生しなかったことを示します。エラーが発生した場合や、プロパティが取得されなかった場合は、汎用の **describe_parameter** エラーのいずれかを返します。

**describe_parameter_set の属性*

describe_parameter_set の属性を示すコードです。

```
typedef enum a_v4_extfn_describe_parm_type {
    EXTFNAPIV4_DESCRIBE_PARM_NAME,
    EXTFNAPIV4_DESCRIBE_PARM_TYPE,
    EXTFNAPIV4_DESCRIBE_PARM_WIDTH,
    EXTFNAPIV4_DESCRIBE_PARM_SCALE,
    EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL,
    EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES,
```

```
    EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT,
    EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE,

    EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS,

    } a_v4_extfn_describe_parm_type;
```

EXTFNAPIV4_DESCRIBE_PARM_NAME 属性 (set)
EXTFNAPIV4_DESCRIBE_PARM_NAME 属性は、パラメータ名を示しています。これは、describe_parameter_set のシナリオで使用されます。

データ型
char[]

説明
UDF に対するパラメータの名前。

使用法
UDF がこのプロパティを設定すると、サーバはその値を、**CREATE PROCEDURE** 文で指定されたパラメータ名と比較します。2 つの値が一致しない場合、サーバはエラーを返します。これによって UDF は、**CREATE PROCEDURE** 文のパラメータ名が UDF が予期するものと同じであることを確認できます。

戻り値
成功した場合は、パラメータ名の長さを返します。

失敗時には、汎用的な describe_parameter エラーのいずれかを返します。または、以下のエラーを返します。

- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 状態が注釈フェーズでない場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – パラメータが結果テーブルの場合に返される set エラー。
- EXTFNAPI4_DESCRIBE_INVALID_ATTRIBUTE_VALUE – UDF が名前を再設定しようとした場合に返される set エラー。

クエリ処理フェーズ
以下のときに有効です。

- 注釈フェーズ

EXTFNAPIV4_DESCRIBE_PARM_TYPE 属性 (set)
The `EXTFNAPIV4_DESCRIBE_PARM_TYPE` attribute indicates the data type of the parameter.これは、`describe_parameter_set` のシナリオで使用されます。

データ型
`a_sql_data_type`

説明
UDF に対するパラメータのデータ型。

使用法
UDF がこのプロパティを設定すると、サーバはその値を、**CREATE PROCEDURE** 文で指定されたパラメータの型と比較します。2 つの値が一致しない場合、サーバはエラーを返します。このチェックによって、**CREATE PROCEDURE** 文のパラメータのデータ型が UDF が予期するものと同じであることを確認できます。

戻り値
成功時には、`sizeof(a_sql_data_type)` を返します。

失敗した場合は、汎用の `describe_parameter` エラーのいずれかか、次のいずれかを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** が `sizeof(a_sql_data_type)` でない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – クエリ処理状態が注釈状態でない場合に返される set エラー。
- `EXTFNAPI4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – UDF がパラメータのデータ型を定義済み以外のものに設定しようとした場合に返される set エラー。

クエリ処理の状態
以下のときに有効です。

- 注釈状態

EXTFNAPIV4_DESCRIBE_PARM_WIDTH 属性 (set)
EXTFNAPIV4_DESCRIBE_PARM_WIDTH 属性は、パラメータの幅を示します。`describe_parameter_set` のシナリオで使用します。

データ型
`a_sql_uint32`

説明
UDF に対するパラメータの幅。EXTFNAPIV4_DESCRIBE_PARM_WIDTH はスカラパラメータにのみ該当します。パラメータ幅は、対応するデータ型のパラメータを格納するために必要な記憶領域のサイズのバイト数です。

- **固定長のデータ型** – データの格納に必要なバイト数
- **可変長のデータ型** – 最大長
- **LOB データ型** – データへのハンドルの格納に必要な記憶領域のサイズ
- **TIME データ型** – コード化した時間の格納に必要な記憶領域のサイズ

使用法
これは読み込み専用プロパティです。幅は、対応するカラムのデータ型から取得されます。データ型を設定した後で幅を変更することはできません。

戻り値
成功時には、sizeof(a_sql_uint32) を返します。

失敗時には、汎用的な describe_parameter エラーのいずれかを返します。または、以下のエラーを返します。

- EXTFNAPIV4_DESCRIBE_INVALID_STATE – クエリ処理フェーズが注釈フェーズでない場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – **describe_buffer** が a_sql_uint32 のサイズでない場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – 指定されたパラメータが TABLE パラメータの場合に返される set エラー。これには、パラメータ 0 またはパラメータ *n* が含まれます。この *n* は入力テーブルです。
- EXTFNAPI4_DESCRIBE_INVALID_ATTRIBUTE_VALUE – UDF がパラメータの幅を再設定しようとした場合に返される set エラー。

クエリ処理フェーズ
以下のときに有効です。

- 注釈フェーズ

EXTFNAPIV4_DESCRIBE_PARM_SCALE 属性 (set)
EXTFNAPIV4_DESCRIBE_PARM_SCALE 属性は、パラメータの位取りを示します。describe_parameter_set のシナリオで使用します。

データ型
a_sql_uint32

説明
UDF に対するパラメータの位取り。算術データ型については、パラメータの位取りは、数値の小数点の右側の桁数です。

この属性は以下に対しては無効です。

- 算術データ型以外のデータ型
- TABLE パラメータ

使用法
これは読み込み専用プロパティです。位取りは、対応するカラムのデータ型から取得されます。データ型を設定した後で位取りを変更することはできません。

戻り値
成功時には、`sizeof(a_sql_uint32)` を返します。

失敗時には、汎用的な `describe_parameter` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** が `a_sql_uint32` のサイズでない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状態が注釈状態でない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – 指定されたパラメータが TABLE パラメータの場合に返される set エラー。これには、パラメータ 0 またはパラメータ *n* が含まれます。この *n* は入力テーブルです。

クエリ処理の状態
以下のときに有効です。

- 注釈状態

EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL 属性 (set)
EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL 属性は、パラメータが NULL かどうかを返します。この属性を `describe_parameter_set` のシナリオで使用するとエラーが返ります。

データ型
`a_sql_byte`

説明
実行時にパラメータの値を NULL にできる場合は true。TABLE パラメータまたはパラメータ 0 の場合、値は false です。

使用法
これは読み込み専用プロパティです。

戻り値
これは読み込み専用のプロパティです。設定しようとすると、常に EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE エラーが返されます。

クエリ処理の状態
適用されません。

EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES 属性 (set)
EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES 属性は、重複しない値の数を返します。この属性を describe_parameter_set のシナリオで使用するとエラーが返ります。

データ型
a_v4_extfn_estimate

説明
すべての呼び出し全体での重複しない値の推定数を返します。スカラパラメータに対してのみ有効です。

使用法
これは読み込み専用プロパティです。

戻り値
これは読み込み専用のプロパティです。設定しようとすると、常に EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE エラーが返されます。

クエリ処理の状態
適用されません。

EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT 属性 (set)
EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES 属性は、パラメータが定数かどうかを返します。この属性を describe_parameter_set のシナリオで使用するとエラーが返ります。

データ型
a_sql_byte

説明
文のパラメータが定数の場合は true。スカラパラメータに対してのみ有効です。

使用法
これは読み込み専用プロパティです。

戻り値
これは読み込み専用のプロパティです。設定しようとすると、常に EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE エラーが返されます。

クエリ処理フェーズ
適用されません。

EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE 属性 (set)
EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE 属性は、パラメータの値を示します。describe_parameter_set のシナリオで使用します。

データ型
an_extfn_value

説明
パラメータの値 (describe の時点で把握している場合)。スカラパラメータに対してのみ有効です。

使用法
これは読み込み専用プロパティです。

戻り値
これは読み込み専用のプロパティです。設定しようとすると、常に EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE エラーが返されます。

クエリ処理フェーズ
適用されません。

EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS 属性 (set)
EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS 属性は、テーブル内のカラムの数を示しています。これは、describe_parameter_set のシナリオで使用されます。

データ型
a_sql_uint32

説明
テーブル内のカラム数。引数 0 およびテーブル引数に対してのみ有効です。

使用法
UDF がこのプロパティを設定すると、サーバはその値を、**CREATE PROCEDURE** 文で指定されたパラメータ名と比較します。2 つの値が一致しない場合、サーバはエラーを返します。これによって UDF は、**CREATE PROCEDURE** 文のパラメータ名が UDF が予期するものと同じであることを確認できます。

戻り値
成功時には、`sizeof(a_sql_uint32)` を返します。

失敗時には、汎用的な `describe_parameter` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** が `sizeof a_sql_uint32` のサイズでない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状態が注釈状態でない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – パラメータが TABLE パラメータでない場合に返される set エラー。
- `EXTFNAPI4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – UDF が指定のテーブルのカラム数を再設定しようとした場合に返される set エラー。

クエリ処理の状態
以下のときに有効です。

- 注釈状態

EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS 属性 (set)
`EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS` 属性は、テーブル内のローの数を示しています。これは、`describe_parameter_set` のシナリオで使用されます。

データ型
`a_sql_a_v4_extfn_estimate`

説明
テーブルの推定ロー数。引数 0 およびテーブル引数に対してのみ有効です。

使用法
UDF は、結果セットに含まれるロー数を推定した場合、引数 0 に対してこのプロパティを設定します。サーバは、最適化のときにこの推定を使用してクエリ処理の判断を下します。この値を入力テーブルに対して設定することはできません。

この値を設定しない場合、サーバは **DEFAULT_TABLE_UDF_ROW_COUNT** オプションで指定されたロー数をデフォルトとして使用します。

戻り値
成功時には、`a_v4_extfn_estimate` を返します。

失敗した場合は、汎用の `describe_parameter` エラーのいずれかか、次のいずれかを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** が `a_v4_extfn_estimate` のサイズでない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状態が最適化フェーズでない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – パラメータが TABLE パラメータでない場合に返される get エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – TABLE パラメータが結果テーブルでない場合に返される get エラー。
- `EXTFNAPI4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – UDF が指定のテーブルのカラム数を再設定しようとした場合に返される get エラー。

クエリ処理フェーズ
以下のときに有効です。

- クエリ最適化フェーズ

EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY 属性 (set)
`EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY` 属性は、テーブル内のローの順序を示しています。これは、`describe_parameter_set` のシナリオで使用されます。

データ型
`a_v4_extfn_orderby_list`

説明
テーブル内のローの順序。このプロパティは引数 0 およびテーブル引数に対してのみ有効です。

使用法
この属性を使用することによって、UDF コードは以下が可能になります。

- 入力の **TABLE** パラメータが順序付けされているかどうかを判断する。
- 結果セットが順序付けされていることを宣言する。

パラメータ番号が 0 の場合、この属性は出力の結果セットを表します。パラメータが 0 より大きく、パラメータの種類がテーブルの場合、この属性は入力の **TABLE** パラメータを表します。

順序付けは `a_v4_extfn_orderby_list` で指定されます。これは、カラムの順序数と、それに対応する昇順または降順のプロパティのリストを保持する構造体です。UDF が出力の結果セットに対してこの順序付けのプロパティを設定した場合、サーバは順序付けの最適化を実行できます。たとえば、UDF が結果セットの最初のカラムを昇順で生成した場合、サーバは同じカラムに対する重複した順序付けの要求を省略できます。

UDF が出力の結果セットに対して順序付けのプロパティを設定しない場合、サーバはデータが順序付けされていないものと見なします。

UDF が入力の **TABLE** パラメータに順序付けのプロパティを設定する場合、その入力データに対する順序付けはサーバが保証します。このシナリオでは、入力データの順序付けが必要であることを UDF がサーバに伝達します。サーバは、実行時に競合を検出した場合は SQL 例外を発生させます。たとえば、入力の **TABLE** パラメータの最初のカラムは昇順である必要があると UDF が伝達したのに対し、SQL 文に降順を指定する句が含まれている場合、サーバは SQL 例外を発生させます。

SQL に順序付けの句が含まれていない場合、サーバは順序付けを自動的に追加します。これにより、入力の **TABLE** パラメータが要件どおりに順序付けされます。

戻り値
成功した場合は、`a_v4_extfn_orderby_list` からコピーされたバイトの数を返します。

クエリ処理の状態
以下のときに有効です。

- 注釈状態
- クエリ最適化状態

EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY (set)
`EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY` 属性は、UDF が分割を必要としていることを示しています。これは、`describe_parameter_set` のシナリオで使用されます。

データ型
`a_v4_extfn_column_list`

説明
UDF 開発者は、**EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY** を使用して、呼び出しの前にパーティション分割が必要であることを UDF のコードで宣言します。

使用法
UDF は、パーティションの適用またはパーティション分割の動的な実施のための確認を実行できます。UDF は **a_v4_extfn_column_list** を確保する必要があります。そのときは、入力テーブルに含まれるカラムの総数を考慮に入れ、そのデータをサーバに送信します。

戻り値
成功時には、`a_v4_extfn_column_list` のサイズを返します。この値は次と同じです。

```
sizeof(a_v4_extfn_column_list) + sizeof(a_sql_uint32) *
number_of_partition_columns
```

失敗した場合は、汎用の `describe_parameter` エラーのいずれかか、次のいずれかを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – バッファの長さが予期したサイズ未満の場合に返される set エラー。

クエリ処理の状態
以下のときに有効です。

- 注釈状態
- クエリ最適化状態

例

```
void UDF_CALLBACK my_tpf_proc_describe( a_v4_extfn_proc_context
*ctx )
{
    if( ctx->current_state == EXTFNAPIV4_STATE_ANNOTATION ) {
        a_sql_int32    rc = 0;
        a_v4_extfn_column_list pbcol =
        { 1,        // 1 column in the partition by list
        2 };    // column index 2 requires partitioning

        // Describe partitioning for argument 1 (the table)
        rc = ctx->describe_parameter_set(
        ctx, 1,
        EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY,
        &pbcol,
        sizeof(pbcol) );

        if( rc == 0 ) {
```

```
        ctx->set_error( ctx, 17000,
           "Runtime error, unable set partitioning requirements for
column." );
        }
    }
}
```

EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND 属性 (set)
EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND 属性は、コンシューマが入力テーブルのリワインドを要求していることを示します。これは、describe_parameter_set のシナリオで使用されます。

データ型
a_sql_byte

説明
コンシューマが入力テーブルのリワインドを希望していることを示します。テーブル入力引数に対してのみ有効です。デフォルトでは、このプロパティは false です。

使用法
UDF は、入力テーブルのリワインド機能を必要とする場合には、最適化フェーズのときにこのプロパティを設定する必要があります。

戻り値
成功時には、sizeof(a_sql_byte) を返します。

失敗時には、汎用的な describe_parameter エラーのいずれかを返します。または、以下のエラーを返します。

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – **describe_buffer** が a_sql_byte のサイズでない場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – 状態が最適化フェーズでない場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER – UDF がパラメータ 0 でこの属性を設定しようとした場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER – UDF がテーブルでないパラメータにこの属性を設定しようとした場合に返される set エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE – UDF がこの属性を 0 でも 1 でもない値に設定しようとした場合に返される set エラー。

クエリ処理フェーズ
以下のときに有効です。

- 最適化フェーズ

例
この例では、最適化フェーズのときに関数 **my_udf_describe** が呼び出されると、describe_parameter_set の呼び出しによって、リワインドが必要な可能性があることがテーブル入力パラメータ 1 のプロデューサに通知されます。

プロシージャ定義の例です。

```
CREATE PROCEDURE my_udf(IN t TABLE(c1 INT))
RESULT (x INT)
EXTERNAL NAME 'my_udf@myudflib';
```

_describe_extfn API 関数のコードの例です。

```
my_udf_describe(a_v4_extfn_proc_context *cntxt)
{
if( cntxt->current_state == EXTFNAPIV4_STATE_OPTIMIZATION ) {
a_sql_byte rewind_required = 1;
a_sql_int32 ret = 0;

ret = cntxt->describe_parameter_set( cntxt, 1,
EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND,
&rewind_required,
sizeof(a_sql_byte) );

if( ret <= 0 ) {
    // Handle the error.
}
}
}
```

EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND 属性 (set)
EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND 属性は、パラメータがリワインドをサポートしているかどうかを示しています。これは、describe_parameter_set のシナリオで使用されます。

データ型
a_sql_byte

説明
プロデューサがリワインドをサポートできるかどうかを示します。テーブル引数に対してのみ有効です。

DESCRIBE_PARM_TABLE_HAS_REWIND を true に設定する予定の場合は、リワインドテーブルコールバック (**_rewind_extfn()**) の実装を提供する必要もあります。コールバックメソッドを提供しないと、サーバが UDF を実行できません。

使用法
UDF は、結果テーブルに対するリワインド機能をコストなしで提供できる場合には、最適化状態のときにこのプロパティを設定します。UDF がリワインドを提供するとコストがかかる場合には、このプロパティを設定しないか、または 0 に設定します。0 に設定した場合は、サーバがリワインドのサポートを提供します。

戻り値
成功時には、`sizeof(a_sql_byte)` を返します。

失敗時には、汎用的な `describe_parameter` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – **describe_buffer** が `a_sql_byte` のサイズでない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状態が最適化状態でない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – UDF がテーブルでないパラメータにこの属性を設定しようとした場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – 指定された引数が結果テーブルでない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – UDF がこの属性を 0 でも 1 でもない値に設定しようとした場合に返される set エラー。

クエリ処理の状態
以下のときに有効です。

- 最適化状態

EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS 属性 (set)
`EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS` 属性は、利用しないカラムをリストします。これは、`describe_parameter_set` のシナリオで使用されます。

データ型
`a_v4_extfn_column_list`

説明
サーバまたは UDF が利用しない出力テーブルカラムのリストです。

出力 TABLE パラメータについては、UDF は通常すべてのカラムに対してデータを生成し、サーバはすべてのカラムを利用します。入力 TABLE パラメータ につ

いても同様で、サーバは通常すべてのカラムに対してデータを生成し、UDF はすべてのカラムを利用します。

しかし場合によっては、サーバまたは UDF が一部のカラムを利用しないこともあります。そのような場合の最善の方法は、UDF が describe 属性 **EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS** を使用して、出力テーブルに対する **GET** を実行することです。この処理は、出力テーブルのうちでサーバが利用しないカラムのリストをサーバに問い合わせる処理です。UDF はこのリストを、出力テーブルのカラムデータを格納するときに利用できます。つまり、使用しないカラムに対するデータの格納を省略できます。

要約すると、出力テーブルについては、使用しないカラムのリストを UDF が問い合わせます。入力テーブルについては、使用しないカラムのリストを UDF がプッシュします。

使用法
UDF は、入力 TABLE パラメータの一部のカラムを使用しない場合、最適化中にこのプロパティを設定します。UDF は、出力テーブルのすべてのカラムを含む `a_v4_extfn_column_list` を割り付ける必要があり、さらにサーバに渡す必要があります。サーバは次に、すべての未投影のカラムの順序数に 1 とマークします。サーバは、内部のデータ構造体にリストをコピーします。

戻り値
成功時には、カラムリストのサイズを返します。
`sizeof(a_v4_extfn_column_list) + sizeof(a_sql_uint32) * number result columns.`

失敗した場合は、汎用の `describe_parameter` エラーのいずれかか、次のいずれかを返します。

- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状態が最適化状態でない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – UDF が入力テーブルでこの属性を取得しようとした場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER` – UDF がテーブルでないパラメータにこの属性を設定しようとした場合に返される set エラー。

クエリ処理の状態
以下のときに有効です。

- 最適化状態

*describe_udf_get

v4 API メソッド describe_udf_get では、UDF のプロパティをサーバから取得します。

宣言

```
a_sql_int32 (SQL_CALLBACK *describe_udf_get)(
    a_v4_extfn_proc_context    *cntxt,
    a_v4_extfn_describe_udf_type    describe_type,
    void    *describe_buffer,
    size_t    describe_buffer_len );
```

パラメータ

パラメータ	説明
cntxt	この UDF のプロシージャコンテキストオブジェクト。
describe_type	取得するプロパティを示すセレクタ。
describe_buffer	サーバで設定する、指定のプロパティの describe 情報を保持する構造体。具体的な構造体またはデータ型は **describe_type** パラメータで示される。
describe_buffer_length	**describe_buffer** のバイト単位の長さ。

戻り値

成功した場合は、0 または **describe_buffer** に書き込まれたバイト数を返します。値 0 の場合、サーバは属性を取得できなかったが、エラー状態は発生しなかったことを示します。エラーが発生した場合や、プロパティが取得されなかった場合は、汎用の describe_udf エラーのいずれかを返します。

**describe_udf_get の属性*

describe_udf_get の属性を示すコードです。

```
typedef enum a_v4_extfn_describe_udf_type {
    EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS,
    EXTFNAPIV4_DESCRIBE_UDF_LAST
} a_v4_extGetfn_describe_udf_type;
```

EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS 属性 (get)
EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS 属性は、パラメータの数を示しています。これは、describe_udf_get のシナリオで使用されます。

データ型
a_sql_uint32

説明
UDF に渡されるパラメータの数。

使用法
CREATE PROCEDURE 文で定義されているパラメータの数を取得します。

戻り値
成功時には、sizeof(a_sql_uint32) を返します。

失敗した場合は、汎用の describe_udf エラーのいずれかか、次のいずれかを返します。

- EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH – describe_buffer が a_sql_uint32 のサイズでない場合に返される get エラー。
- EXTFNAPIV4_DESCRIBE_INVALID_STATE – フェーズが初期フェーズより後でない場合に返される get エラー。

クエリ処理フェーズ

- 注釈フェーズ
- クエリ最適化フェーズ
- プラン構築フェーズ
- 実行フェーズ

***describe_udf_set**
v4 API メソッド describe_udf_set では、UDF のプロパティをサーバに対して設定します。

宣言

```
a_sql_int32 (SQL_CALLBACK *describe_udf_set)(
    a_v4_extfn_proc_context    *cntxt,
    a_v4_extfn_describe_udf_type    describe_type,
    const void    *describe_buffer,
    size_t    describe_buffer_len );
```

パラメータ

パラメータ	説明
cntxt	この UDF のプロシージャコンテキストオブジェクト。
describe_type	設定するプロパティを示すセレクタ。
describe_buffer	サーバで設定する、指定のプロパティの describe 情報を保持する構造体。具体的な構造体またはデータ型は **describe_type** パラメータで示される。
describe_buffer_length	**describe_buffer** のバイト単位の長さ。

戻り値

成功した場合は、**describe_buffer** に書き込まれたバイト数を返します。エラーが発生した場合や、プロパティが取得されなかった場合は、汎用の `describe_udf` エラーのいずれかを返します。

エラーが発生した場合や、プロパティが取得されなかった場合は、汎用の `describe_udf` エラーのいずれかか、次のいずれかを返します。

- `EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER` – **cntxt** 引数か **describe_buffer** 引数が NULL の場合、または **describe_buffer_length** が 0 の場合に返る set エラー。
- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – 要求された属性のサイズと指定された **describe_buffer_length** に不一致がある場合に返る set エラー。

**describe_udf_set の属性*

`describe_udf_set` の属性を示すコードです。

```
typedef enum a_v4_extfn_describe_udf_type {
    EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS,
    EXTFNAPIV4_DESCRIBE_UDF_LAST
} a_v4_extGetfn_describe_udf_type;
```

EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS 属性 (set)

`EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS` 属性は、パラメータの数を示しています。これは、`describe_udf_set` のシナリオで使用されます。

データ型

`a_sql_uint32`

説明

UDF に渡されるパラメータの数。

使用法
UDF がこのプロパティを設定すると、サーバはその値を、**CREATE PROCEDURE** 文で指定されたパラメータの数と比較します。2 つの値が一致しない場合、サーバは SQL エラーを返します。これによって UDF は、**CREATE PROCEDURE** 文のパラメータ数が UDF が予期するものと同じであることを確認できます。

戻り値
成功時には、`sizeof(a_sql_uint32)` を返します。

失敗時には、汎用的な `describe_udf` エラーのいずれかを返します。または、以下のエラーを返します。

- `EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH` – describe_buffer が `a_sql_uint32` のサイズでない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_STATE` – 状態が注釈状態でない場合に返される set エラー。
- `EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE` – UDF がパラメータのデータ型を再設定しようとした場合に返る set エラー。

クエリ処理の状態

- 注釈状態

カラムタイプの記述 (a_v4_extfn_describe_col_type)

`a_v4_extfn_describe_col_type` 列挙型では、UDF が取得または設定したカラムプロパティを選択します。

実装

```
typedef enum a_v4_extfn_describe_col_type {
    EXTFNAPIV4_DESCRIBE_COL_NAME,
    EXTFNAPIV4_DESCRIBE_COL_TYPE,
    EXTFNAPIV4_DESCRIBE_COL_WIDTH,
    EXTFNAPIV4_DESCRIBE_COL_SCALE,
    EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL,
    EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES,
    EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE,
    EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT,
    EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER,
    EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE,
    EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT,
    EXTFNAPIV4_DESCRIBE_COL_LAST
} a_v4_extfn_describe_col_type;
```

メンバーの概要

メンバー	説明
EXTFNAPIV4_DESCRIBE_COL_NAME	カラム名 (有効な識別子)。
EXTFNAPIV4_DESCRIBE_COL_TYPE	カラムのデータ型。
EXTFNAPIV4_DESCRIBE_COL_WIDTH	文字列の幅 (NUMERIC の精度)。
EXTFNAPIV4_DESCRIBE_COL_SCALE	NUMERIC の位取り。
EXTFNAPIV4_DESCRIBE_COL_CAN_BE_NULL	カラムが NULL になり得る場合は true。
EXTFNAPIV4_DESCRIBE_COL_DISTINCT_VALUES	カラム内の重複しない値の推定数。
EXTFNAPIV4_DESCRIBE_COL_IS_UNIQUE	カラムがテーブル内でユニークな場合には true。
EXTFNAPIV4_DESCRIBE_COL_IS_CONSTANT	カラムが文の有効期間全体にわたって定数の場合には true。
EXTFNAPIV4_DESCRIBE_COL_CONSTANT_VALUE	パラメータの値 (describe の時点で把握している場合)。
EXTFNAPIV4_DESCRIBE_COL_IS_USED_BY_CONSUMER	テーブルのコンシューマがカラムを必要とする場合には true。
EXTFNAPIV4_DESCRIBE_COL_MINIMUM_VALUE	カラムの最小値 (把握している場合)。
EXTFNAPIV4_DESCRIBE_COL_MAXIMUM_VALUE	カラムの最大値 (把握している場合)。
EXTFNAPIV4_DESCRIBE_COL_VALUES_SUBSET_OF_INPUT	結果カラムの値は入力テーブルのカラムのサブセット。
EXTFNAPIV4_DESCRIBE_COL_LAST	v4 API に対する最初の不正な値。範囲外の値。

パラメータタイプの記述 (a_v4_extfn_describe_parm_type)

a_v4_extfn_describe_parm_type 列挙型では、UDF が取得または設定したパラメータプロパティを選択します。

実装

```
typedef enum a_v4_extfn_describe_parm_type {
    EXTFNAPIV4_DESCRIBE_PARM_NAME,
    EXTFNAPIV4_DESCRIBE_PARM_TYPE,
    EXTFNAPIV4_DESCRIBE_PARM_WIDTH,
    EXTFNAPIV4_DESCRIBE_PARM_SCALE,
    EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL,
    EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES,
    EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT,
    EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE,

    EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND,
    EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS,

    EXTFNAPIV4_DESCRIBE_PARM_LAST
} a_v4_extfn_describe_parm_type;
```

メンバーの概要

メンバー	説明
EXTFNAPIV4_DESCRIBE_PARM_NAME	パラメータ名 (有効な識別子)。
EXTFNAPIV4_DESCRIBE_PARM_TYPE	データ型。
EXTFNAPIV4_DESCRIBE_PARM_WIDTH	文字列の幅 (NUMERIC の精度)。
EXTFNAPIV4_DESCRIBE_PARM_SCALE	NUMERIC の位取り。
EXTFNAPIV4_DESCRIBE_PARM_CAN_BE_NULL	値が NULL になり得る場合は true。
EXTFNAPIV4_DESCRIBE_PARM_DISTINCT_VALUES	すべての呼び出し全体での重複しない値の推定数。

メンバー	説明
EXTFNAPIV4_DESCRIBE_PARM_IS_CONSTANT	文のパラメータが定数の場合は true。
EXTFNAPIV4_DESCRIBE_PARM_CONSTANT_VALUE	パラメータの値 (describe の時点で把握している場合)。
以下のセレクタでは、TABLE パラメータのプロパティを取得または設定できる。これらの列挙子の値はスカラパラメータでは使用できない。	
EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_COLUMNS	テーブル内のカラム数。
EXTFNAPIV4_DESCRIBE_PARM_TABLE_NUM_ROWS	テーブルの推定ロー数。
EXTFNAPIV4_DESCRIBE_PARM_TABLE_ORDERBY	テーブル内のローの順序。
EXTFNAPIV4_DESCRIBE_PARM_TABLE_PARTITIONBY	パーティション分割。ANY には *number_of_columns*=0 を使用。
EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND	コンシューマが入力テーブルのリワインド機能を希望する場合は true。
EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND	プロデューサがリワインドをサポートする場合は true を返す。
EXTFNAPIV4_DESCRIBE_PARM_TABLE_UNUSED_COLUMNS	サーバまたは UDF が利用しない出力テーブルカラムのリストです。
EXTFNAPIV4_DESCRIBE_PARM_LAST	v4 API に対する最初の不正な値。範囲外の値。

戻り値の記述 (a_v4_extfn_describe_return)

`a_v4_extfn_describe_return` 列挙型は、`a_v4_extfn_proc_context.describe_xxx_get()` または `a_v4_extfn_proc_context.describe_xxx_set()` が成功しなかった場合の戻り値を表します。

実装

```
typedef enum a_v4_extfn_describe_return {
    EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE          =  0,   // the specified operation has no
meaning either for this attribute or in

the current context.
    EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH       = -1,   // the provided buffer size
```

```
does not match the required length or the

length is insufficient.
    EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER       = -2,   // the provided parameter number
is invalid
    EXTFNAPIV4_DESCRIBE_INVALID_COLUMN          = -3,   // the column number is invalid
for this TABLE parameter
    EXTFNAPIV4_DESCRIBE_INVALID_STATE          = -4,   // the describe method call is not
valid in the present state
    EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE       = -5,   // the attribute is known but not
appropriate for this object
     EXTFNAPIV4_DESCRIBE_UNKNOWN_ATTRIBUTE       = -6,   // the identified attribute is
not known to this server version
    EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER       = -7,   // the specified parameter is
not a TABLE parameter (for describe_col_get()

or set())
    EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE       = -8,   // the specified attribute
value is illegal
     EXTFNAPIV4_DESCRIBE_LAST                = -9
} a_v4_extfn_describe_return;
```

メンバーの概要

メンバー	戻り値	説明
EXTFNAPIV4_DESCRIBE_NOT_AVAILABLE	0	指定の操作はこの属性に対して意味を持たないか、または現在のコンテキストで意味を持たない。
EXTFNAPIV4_DESCRIBE_BUFFER_SIZE_MISMATCH	-1	指定のバッファサイズが必要な長さと一致しないか、または長さが不十分である。
EXTFNAPIV4_DESCRIBE_INVALID_PARAMETER	-2	指定のパラメータ番号が無効。
EXTFNAPIV4_DESCRIBE_INVALID_COLUMN	-3	カラム番号がこの TABLE パラメータに対して無効。
EXTFNAPIV4_DESCRIBE_INVALID_STATE	-4	現在の状態では describe メソッド呼び出しは無効。
EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE	-5	既知の属性だが、このオブジェクトには該当しない。
EXTFNAPIV4_DESCRIBE_UNKNOWN_ATTRIBUTE	-6	指定の属性は、このサーババージョンでは未知の属性である。
EXTFNAPIV4_DESCRIBE_NON_TABLE_PARAMETER	-7	指定のパラメータは、TABLE パラメータではない (対象は `describe_col_get()` または `describe_col_set()`)。
EXTFNAPIV4_DESCRIBE_INVALID_ATTRIBUTE_VALUE	-8	指定の属性値は無効。

メンバー	戻り値	説明
EXTFNAPIV4_DESCRIBE_LAST	-9	v4 API に対する最初の不正な値。

説明
`a_v4_extfn_proc_context.describe_xxx_get()` と `a_v4_extfn_proc_context.describe_xxx_set()` の戻り値は符号付き整数です。 結果が正の値の場合は、操作は成功し、値はコピーされたバイト数を表します。 戻り値が 0 以下の場合は、操作は失敗し、戻り値は `a_v4_extfn_describe_return` のいずれかの値です。

UDF タイプの記述 (a_v4_extfn_describe_udf_type)

`a_v4_extfn_describe_udf_type` 列挙型では、UDF が取得または設定する論理プロパティを選択します。

実装

```
typedef enum a_v4_extfn_describe_udf_type {
    EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS,
    EXTFNAPIV4_DESCRIBE_UDF_LAST
} a_v4_extfn_describe_udf_type;
```

メンバーの概要

メンバー	説明
EXTFNAPIV4_DESCRIBE_UDF_NUM_PARMS	UDF に渡されるパラメータの数。
EXTFNAPIV4_DESCRIBE_UDF_LAST	範囲外の値。

説明
UDF は、プロパティを取得するときに `a_v4_extfn_proc_context.describe_udf_get()` メソッドを使用し、UDF 全体についてのプロパティを設定するときに `a_v4_extfn_proc_context.describe_udf_set()` メソッドを使用します。 `a_v4_extfn_describe_udf_type` 列挙子では、UDF が取得または設定する論理プロパティを選択します。

実行状態 (a_v4_extfn_state)

a_v4_extfn_state 列挙型は、UDF のクエリ処理フェーズを表します。

実装

```
typedef enum a_v4_extfn_state {
   EXTFNAPIV4_STATE_INITIAL,                  // Server initial state,
not used by UDF
    EXTFNAPIV4_STATE_ANNOTATION,                 // Annotating parse
tree with UDF reference
    EXTFNAPIV4_STATE_OPTIMIZATION,               // Optimizing
   EXTFNAPIV4_STATE_PLAN_BUILDING,             // Building execution
plan
    EXTFNAPIV4_STATE_EXECUTING,                  // Executing UDF and
fetching results from UDF
    EXTFNAPIV4_STATE_LAST
} a_v4_extfn_state;
```

メンバーの概要

メンバー	説明
EXTFNAPIV4_STATE_INITIAL	サーバの初期フェーズ。このクエリ処理フェーズ中に呼び出される UDF メソッドは _start_extfn のみ。
EXTFNAPIV4_STATE_ANNOTATION	UDF 参照を使用して解析ツリーに注釈を付けている。このフェーズ中は UDF は呼び出されない。
EXTFNAPIV4_STATE_OPTIMIZATION	最適化を実行している。サーバは UDF の _start_extfn メソッドを呼び出し、続いて _describe_extfn 関数を呼び出す。
EXTFNAPIV4_STATE_PLAN_BUILDING	クエリ実行プランを構築している。サーバは UDF の _describe_extfn 関数を呼び出す。
EXTFNAPIV4_STATE_EXECUTING	UDF を実行し、UDF から結果をフェッチしている。サーバは UDF からデータのフェッチを開始する前に _describe_extfn 関数を呼び出す。続いてサーバは _evaluate_extfn を呼び出してフェッチサイクルを開始する。フェッチサイクルの間に、サーバは a_v4_extfn_table_func で定義されている関数を呼び出す。フェッチが終了すると、サーバは UDF の _close_extfn 関数を呼び出す。
EXTFNAPIV4_STATE_LAST	v4 API に対する最初の不正な値。範囲外の値。

説明

`a_v4_extfn_state` 列挙型は、UDF 実行のどの段階にサーバがいるかを示します。あるフェーズから次のフェーズにサーバが遷移するときには、前のフェーズが終了することを UDF に通知するために、UDF の `_leave_state_extfn` 関数を呼び出します。また、新しいフェーズに入ることを UDF に通知するために、UDF の `_enter_state_extfn` 関数を 呼び出します。

UDF のクエリ処理フェーズに応じて、UDF が実行できる操作は制限されます。たとえば、注釈フェーズのときには、UDF は定数パラメータに対してのみデータ型を取得できます。

外部関数 (a_v4_extfn_proc)

サーバは `a_v4_extfn_proc` 構造体を使用して UDF のさまざまなエントリポイントを呼び出します。サーバは各関数に `a_v4_extfn_proc_context` のインスタンスを渡します。

メソッドの概要

メソッド	説明
_start_extfn	構造体を確保し、そのアドレスを `a_v4_extfn_proc_context` の **_user_data** フィールドに格納する。
_finish_extfn	`a_v4_extfn_proc_context` の **_user_data** フィールドに格納されているアドレスの構造体を解放する。
_evaluate_extfn	必須の関数ポインタ。新しい引数値のセットによる関数呼び出しごとに呼び出される。
_describe_extfn	describe の API (26 ページ) を参照。
_enter_state_extfn	UDF はこの関数を使用して構造体を確保できる。
_leave_state_extfn	UDF はこの関数を使用して、対象の状態で必要だったメモリまたはリソースを解放できる。

_start_extfn

v4 API メソッド `_start_extfn` は、初期化関数に対するオプションのポインタとして使用します。この関数の唯一の引数は `a_v4_extfn_proc_context` 構造体のポインタです。

宣言

```
_start_extfn(
a_v4_extfn_proc_context *
```

```
)
```

使用法

`_start_extfn` メソッドでは、構造体を確保し、そのアドレスを `a_v4_extfn_proc_context` の `_user_data` フィールドに格納します。初期化が不要な場合は、この関数ポインタを NULL ポインタに設定する必要があります。

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。

_finish_extfn

v4 API メソッド `_finish_extfn` は、シャットダウン関数に対するオプションのポインタとして使用します。この関数の唯一の引数は `a_v4_extfn_proc_context` のポインタです。

宣言

```
_finish_extfn(
     a_v4_extfn_proc_context *cntxt,
)
```

使用法

`_finish_extfn` API では、`a_v4_extfn_proc_context` の `_user_data` フィールドに格納されているアドレスの構造体を解放します。クリーンアップが不要な場合は、この関数ポインタを NULL ポインタに設定する必要があります。

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。

_evaluate_extfn

v4 API メソッド `_evaluate_extfn` は、必須の関数ポインタとして使用します。新しい引数値のセットによる関数呼び出しごとに呼び出されます。

宣言

```
_evaluate_extfn(
     a_v4_extfn_proc_context *cntxt,
     void *args_handle
)
```

使用法
_evaluate_extfn 関数では、a_v4_extfn_table 構造体の a_v4_extfn_table_func 部分を設定することによって結果のフェッチ方法をサーバに伝達する必要があり、コンテキストの set_value メソッドを引数 0 で呼び出してこの情報をサーバに送信します。またこの関数では、引数 0 の set_value を呼び出す前に、Ia_v4_extfn_table 構造体の a_v4_extfn_value_schema を設定することによって、出力スキーマについてサーバに通知する必要があります。入力引数値へのアクセスには get_value コールバック関数を使用します。この時点で、UDF は定数と非定数のどちらの引数も使用できます。

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。
args_handle	サーバの引数へのハンドル。

_describe_extfn
_describe_extfn は、サーバが論理プロパティを取得および設定できるようにするために、それぞれの状態の最初の時点で呼び出されます。UDF はそのために、a_v4_proc_context object で 6 つの describe メソッド (describe_parameter_get、describe_parameter_set、describe_column_get、describe_column_set、describe_udf_get、describe_udf_set) を使用します。

詳細については、describe の API (26 ページ) を参照してください。

_enter_state_extfn
v4 API メソッド _enter_state_extfn は、UDF が新しい状態に入るときに通知するオプションのエントリポイントとして UDF が実装できます。

宣言

```
_enter_state_extfn(
     a_v4_extfn_proc_context *cntxt,
)
```

使用法
UDF はこの通知を使用して構造体を確保できます。

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。

_leave_state_extfn

v4 API メソッド `_leave_state_extfn` は、UDF のクエリ処理の状態を抜けるときに通知を受信するために UDF が実装できるオプションのエントリポイントです。

宣言

```
_leave_state_extfn(
     a_v4_extfn_proc_context *cntxt,
)
```

使用法

UDF はこの通知を使用して、対象の状態で必要だったメモリまたはリソースを解放できます。

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。

外部プロシージャコンテキスト (a_v4_extfn_proc_context)

`a_v4_extfn_proc_context` 構造体では、サーバおよび UDF からのコンテキスト情報を保持します。

実装

```
typedef struct a_v4_extfn_proc_context {
.
.
.
} a_v4_extfn_proc_context;
```

メソッドの概要

戻り値の型	メソッド	説明
short	**get_value**	UDF から入力引数を取得する。
short	**get_value_is_constant**	指定された引数が定数かどうかを示す情報を UDF が要求できる。

戻り値の型	メソッド	説明
short	**set_value**	UDF が `_evaluate_extfn` 関数または `_describe_extfn` 関数で使用し、UDF の出力の形式や、UDF からの結果をフェッチする方法についてサーバに伝達する。
a_sql_uint32	**get_is_cancelled**	**get_is_cancelled** コールバックを 1 ～ 2 秒ごとに呼び出して、現在の文をユーザが中断したかどうかを確認する。
short	**set_error**	現在の文をロールバックしてエラーを生成する。
void	**log_message**	メッセージログにメッセージを書き込む。
short	**convert_value**	データ型を別のものに変換する。
short	**get_option**	設定可能なオプションの値を取得する。
void	**alloc**	長さが "len" 以上のメモリブロックを確保する。
void	**free**	**alloc()** により指定の存続期間で確保したメモリを解放する。
a_sql_uint32	**describe_column_get**	詳細については、*describe_column_get (26 ページ) を参照。
a_sql_uint32	**describe_column_set**	詳細については、*describe_column_set (42 ページ) を参照。
a_sql_uint32	**describe_parameter_get**	詳細については、*describe_parameter_get (58 ページ) を参照。
a_sql_uint32	**describe_parameter_set**	詳細については、*describe_parameter_set (78 ページ) を参照。
a_sql_uint32	**describe_udf_get**	詳細については、*describe_udf_get (93 ページ) を参照。
a_sql_uint32	**describe_udf_set**	詳細については、*describe_udf_set (94 ページ) を参照。
short	**open_result_set**	テーブル値の結果セットを開く。
short	**close_result_set**	開いた結果セットを閉じる。
short	**get_blob**	BLOB の入力パラメータを取得する。

戻り値の型	メソッド	説明
short	**set_cannot_be_distributed**	ライブラリが分散可能な場合でも、UDF レベルで分散を無効にする。

データメンバーとデータ型の概要

データメンバー	データ型	説明
_user_data	void *	このデータポインタは、外部ルーチンが必要とする任意のコンテキストデータの使用により設定できる。
_executionMode	a_sql_uint32	**External_UDF_Execution_Mode** オプションにより要求されたデバッグ／トレースレベルを示す。これは読み込み専用フィールドである。
current_state	a_sql_uint32	*current_state* 属性は、コンテキストの現在の実行モードを反映している。これは `_describe_extfn` などの関数から確認でき、次に行う処理の判断に使用できる。

説明

`a_v4_extfn_proc_context` 構造体では、サーバおよび UDF からのコンテキスト情報の保持に加えて、UDF からサーバへのコールバック呼び出しによる処理を実行できます。UDF はプライベートデータをこの構造体の `_user_data` メンバーに格納できます。この構造体のインスタンスはサーバによって `a_v4_extfn_proc` メソッドの関数に渡されます。ユーザデータは、サーバが注釈状態に達するまでは維持されません。

get_value

v4 API メソッド `get_value` を使用して、SQL クエリで UDF に渡された入力引数の値を取得します。

宣言

```
short get_value(
       void *           arg_handle,
       a_sql_uint32     arg_num,
       an_extfn_value *value
       )
```

使用法

`get_value` API は、UDF が受け取ったそれぞれの入力引数の値を evaluate メソッドで取得するために使用します。 引数のデータ型の範囲が狭い場合 (32K より大)、`get_value` の呼び出しで十分に引数値全体を取得できます。

get_value API は、arg_handle ポインタにアクセスできるどの API からでも呼び出すことができます。これには、a_v4_table_context をパラメータで受け取る API 関数が含まれます。a_v4_table_context のメンバー変数には args_handle があり、これを使用できます。

固定長データ型の場合は、どの型についても、返される値でデータを取得でき、すべてのデータを取得するために連続して呼び出す必要はありません。プロデューサは、get_value メソッドの呼び出しで返る全体の最大長を判断できます。固定長データ型はすべて、連続した単一のバッファに収まることが保証されます。可変長データの場合は、上限はプロデューサに応じて変わります。

固定長でないデータ型の場合、データの長さによっては、データを取得するために get_blob メソッドで BLOB を作成することが必要な場合があります。BLOB オブジェクトが必要かどうかは、get_value で戻る値に対してマクロ **EXTFN_IS_INCOMPLETE** を使用することで判断できます。**EXTFN_IS_INCOMPLETE** が true の場合、BLOB が必要です。

入力引数がテーブルの場合、その型は **AN_EXTFN_TABLE** です。この型の引数については、open_result_set メソッドを使用して結果セットを作成してテーブルから値を読み込む必要があります。

UDF で、_evaluate_extfn API の呼び出しより前に引数の値が必要な場合は、_describe_extfn API を実装する必要があります。_describe_extfn API からは、describe_parameter_get メソッドを使用して定数式の値を取得できます。

パラメータ

パラメータ	説明
arg_handle	コンシューマが提供するコンテキストポインタ。
arg_num	値を取得する対象の引数のインデックス。引数のインデックスの先頭の値は 1。
value	指定の引数の値。

戻り値
成功の場合は 1、それ以外の場合は 0。

an_extfn_value 構造体
an_extfn_value 構造体は、get_value API が返す入力引数の値を表します。

次のコードは、**an_extfn_value** 構造体の宣言を示します。

```
short typedef struct an_extfn_value {
       void*           data;
       a_sql_uint32    piece_len,
       an_extfn_value *value {
                              a_sql_uint32    total_len;
                              a_sql_uint32    remain_len;
       } len;
       a_sql_data_type    type;
} an_extfn_value;
```

次の表は、get_value メソッドを呼び出した後で **an_extfn_value** オブジェクトが返す値の内容を示します。

get_value API が返す値	EXTFN_IS_IN-COMPLETE	total_len	piece_len	data
NULL	FALSE	0	0	NULL
空の文字列	FALSE	0	0	非 NULL
Size < MAX_UINT32	FALSE	実際の長さ	実際の長さ	非 NULL
size < MAX_UINT32	TRUE	実際の長さ	0	非 NULL
size >= MAX_UINT32	TRUE	MAX_UINT32	0	非 NULL

an_extfn_value の type フィールドは、値のデータ型を表します。入力引数にテーブルがある UDF の場合、その引数のデータ型は **DT_EXTFN_TABLE** です。v4 のテーブル UDF では、remain_len フィールドは使用しません。

get_value_is_constant

v4 API メソッド get_value_is_constant を使用して、指定の入力引数が定数かどうかを判断します。

宣言

```
short get_value_is_constant(
       void *          arg_handle,
       a_sql_uint32    arg_num,
       an_extfn_value *value_is_constant
       )
```

使用法

UDF は、指定された引数が定数かどうかを示す情報を要求できます。この関数は、たとえば、評価関数を呼び出すたびに行うのではなく、_evaluate_extfn 関数の最初の呼び出しで 1 回行うだけで済む場合など、UDF の最適化に便利です。

パラメータ

パラメータ	説明
arg_handle	サーバの引数のハンドル。
arg_num	取得する入力引数のインデックス値。 インデックス値は 1 ～ N の間。
value_is_constant	定数かどうかを格納する出力パラメータ。

戻り値
成功の場合は 1、それ以外の場合は 0。

set_value

v4 API メソッド `set_value` を使用して、結果セットに含まれるカラムの数とデータの読み込み方法をコンシューマに伝達します。

宣言

```
short set_value(
       void *           arg_handle,
       a_sql_uint32     arg_num,
       an_extfn_value *value
       )
```

使用法
このメソッドは、UDF が `_evaluate_extfn` API で使用します。UDF は、`set_value` メソッドを呼び出して、結果セットに含まれるカラムの数と、UDF がサポートする `a_v4_extfn_table_func` の関数のセットをコンシューマに伝える必要があります。

UDF は `set_value` API に対し、`_evaluate_extfn` API を通じた適切な **arg_handle** ポインタか、または `a_v4_extfn_table_context` 構造体の **args_handle** メンバーを渡します。

v4 のテーブル UDF については、`set_value` メソッドの **value** 引数は型 *DT_EXTFN_TABLE* である必要があります。

パラメータ

パラメータ	説明
arg_handle	コンシューマが提供するコンテキストポインタ。

パラメータ	説明
arg_num	値を設定する対象の引数のインデックス。サポートされている引数は 0 のみ。
value	指定の引数の値。

戻り値
成功の場合は 1、それ以外の場合は 0。

get_is_cancelled

v4 API メソッド `get_is_cancelled` を使用して、文の実行がキャンセルされたかどうかを判断します。

宣言

```
short get_is_cancelled(
       a_v4_extfn_proc_context *          cntxt,

       )
```

使用法
UDF エントリポイントが長期間 (かなりの秒数) にわたって動作を実行する場合、可能であれば、ユーザが現在の文を中断したかどうかを確認するために、1 秒または 2 秒ごとに `get_is_cancelled` コールバックを呼び出します。 文が中断されている場合、0 以外の値が返され、UDF エントリポイントはただちに処理を戻す必要があります。 `_finish_extfn` 関数を呼び出して、必要なクリーンアップを実行します。その後は、他の UDF エントリポイントを呼び出さないでください。

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。

戻り値
文が中断されている場合は 0 以外の値。

set_error

v4 API メソッド `set_error` を使用して、サーバにエラーを返します。このエラーは最終的にはユーザに返されます。

宣言

```
void set_error(
       a_v4_extfn_proc_context *          cntxt,
       a_sql_uint32                       error_number,
       const char                         *error_desc_string
       )
```

使用法

UDF エントリポイントでエラーが発生し、ユーザにエラーメッセージを表示して現在の文を停止する必要がある場合には、`set_error` API を呼び出します。`set_error` API を呼び出すと、現在の文がロールバックされ、ユーザには `"Error raised by user-defined function: <error_desc_string>"` と表示されます。SQLCODE は指定の <error_number> の負数です。

既存のエラーとの競合を避けるため、UDF は 17000 ～ 99999 の間のエラー番号を生成する必要があります。この範囲にない値を指定した場合、文はロールバックされますが、エラーメッセージは `"Invalid error raised by user-defined function: (<error_number>) <error_desc_string>"` となり、SQLCODE は -1577 となります。**error_desc_string** の最大長は 140 文字です。

UDF エントリポイントでは、`set_error` を呼び出した後は、直ちに処理を戻す必要があります。最終的には `_finish_extfn` 関数が呼び出され、必要なクリーンアップが実行されます。その後は、他の UDF エントリポイントを呼び出さないでください。

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。
error_number	設定するエラー番号。
error_desc_string	使用するメッセージ文字列。

log_message

v4 API メソッド log_message を使用して、サーバのメッセージログにメッセージを送信します。

宣言

```
short log_message(
       const char          *msg,
       short               msg_length
       )
```

使用法

log_message メソッドでは、メッセージログにメッセージを書き込みます。メッセージ文字列は、表示可能な 255 バイト以下のテキスト文字列とする必要があります。これより長いメッセージはトランケートされます。

パラメータ

パラメータ	説明
msg	ログに記録するメッセージ文字列。
msg_length	メッセージ文字列の長さ。

convert_value

v4 API メソッド convert_value を使用して、データ型を変換します。

宣言

```
short convert_value(
       an_extfn_value *input,
       an_extfn_value *output
       )
```

使用法

convert_value API の主な用途は、DT_DATE、DT_TIME、および DT_TIMESTAMP と DT_TIMESTAMP_STRUCT の間の変換です。 入力と出力の an_extfn_value をこの関数に渡します。

入力パラメータ

パラメータ	説明
an_extfn_value.data	入力データのポインタ。
an_extfn_value.total_len	入力データの長さ。

パラメータ	説明
an_extfn_value.type	入力の DT_ データ型。

出力パラメータ

パラメータ	説明
an_extfn_value.data	UDF が指定する出力データのポインタ。
an_extfn_value.piece_len	出力データの最大長。
an_extfn_value.total_len	サーバが設定する変換済みデータの長さ。
an_extfn_value.type	希望の出力の DT_ データ型。

戻り値
成功の場合は 1、それ以外の場合は 0。

get_option

v4 API メソッド `get_option` では、設定可能なオプションの値を取得します。

宣言

```
short get_option(
a_v4_extfn_proc_context * cntxt,
char *option_name,
an_extfn_value *output
)
```

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。
option_name	取得するオプションの名前。
output	• `an_extfn_value.data` – UDF が指定する出力データのポインタ。 • `an_extfn_value.piece_len` – 出力データの最大長。 • `an_extfn_value.total_len` – サーバが設定する変換済み出力の長さ。 • `an_extfn_value.type` – サーバが設定する値のデータ型。

戻り値
成功の場合は 1、それ以外の場合は 0。

alloc

v4 API メソッド `alloc` では、メモリのブロックを確保します。

宣言

```
void*alloc(
     a_v4_extfn_proc_context *cntxt,
     size_t len
     )
```

使用法

len 以上の長さのメモリのブロックを割り当てます。返されるメモリは 8 バイト単位で整列されています。

ヒント： メモリ割当方法として `alloc()` メソッドのみを使用してください。これにより、サーバは外部ルーチンによるメモリの使用量を追跡できるようになります。サーバは他のメモリユーザを適合させたり、リークを追跡したり、診断や監視を向上させることができます。

メモリ追跡は、**external_UDF_execution_mode** の値を 1 または 2 (検証モードまたはトレーシングモード) に設定した場合にのみ有効になります。

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。
len	確保するメモリのバイト単位の長さ。

free

v4 API メソッド `free` では、確保したメモリのブロックを解放します。

宣言

```
void free(
     a_v4_extfn_proc_context *cntxt,
     void *mem
)
```

使用法

`alloc()` により指定の存続期間で確保したメモリを解放します。

メモリトラッキングは、**external_UDF_execution_mode** が値 1 または 2 (検証モードまたはトレースモード) に設定されている場合にのみ有効になります。

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。
mem	`alloc` メソッドで確保したメモリのポインタ。

open_result_set

v4 API メソッド `open_result_set` では、テーブル値の結果セットを開きます。

宣言

```
short open_result_set(
a_v4_extfn_proc_context *cntxt,
a_v4_extfn_table *table,
a_v4_extfn_table_context **result_set
)
```

使用法

`open_result_set` では、テーブル値の結果セットを開きます。UDF は、`DT_EXTFN_TABLE` 型の入力パラメータからローを読み込むための結果セットを開くことができます。サーバ (または別の UDF) は、UDF からのローを読み込むための結果セットを開くことができます。

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。
table	結果セットを開く対象のテーブルオブジェクト。
result_set	開いた結果セットに設定される出力パラメータ。

戻り値

成功の場合は 1、それ以外の場合は 0。

`open_result_set` の使用例については、v4 API メソッド `fetch_block` および `fetch_into` の説明を参照してください。

close_result_set

v4 API メソッド `close_result_set` では、開いた結果セットを閉じます。

宣言

```
short close_result_set(
    a_v4_extfn_proc_context *cntxt,
```

```
    a_v4_extfn_table_context *result_set
)
```

使用法
close_result_set は、それぞれの結果セットに対して 1 回のみ呼び出すことができます。

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。
result_set	閉じる結果セット。

戻り値
成功の場合は 1、それ以外の場合は 0。

get_blob
v4 API メソッド get_blob を使用して、入力の BLOB パラメータを取得します。

宣言

```
short get_blob(
       void             *arg_handle,
       a_sql_uint32     arg_num,
       a_v4_extfn_blob **blob
       )
```

使用法
get_blob は、get_value() を呼び出した後で使用し、BLOB 入力パラメータを取得します。get_value() で *piece_len* < *total_len* の場合に、返った値のデータの読み込みに BLOB オブジェクトが必要かどうかを判断するには、マクロ EXTFN_IS_INCOMPLETE を使用します。 BLOB オブジェクトは出力パラメータで返り、呼び出し元が所有します。

get_blob で取得する BLOB ハンドルを使用して、BLOB の内容を読み込むことができます。このメソッドは BLOB オブジェクトが格納されたカラムに対してのみ使用します。

パラメータ

パラメータ	説明
arg_handle	サーバの引数へのハンドル。
arg_num	引数を表す番号 1 ～ N。

パラメータ	説明
blob	BLOB オブジェクトが入る出力引数。

戻り値
成功の場合は 1、それ以外の場合は 0。

set_cannot_be_distributed
v4 API メソッド `set_cannot_be_distributed` では、ライブラリレベルで分散の基準を満たす場合でも、UDF レベルで分散を無効にできます。

宣言
```
void set_cannot_be_distributed( a_v4_extfn_proc_context *cntxt)
```

使用法
デフォルトの動作では、ライブラリが分散可能な場合、UDF も分散可能です。UDF では、`set_cannot_be_distributed` を使用して、分散を無効にするという判断をサーバにプッシュできます。

ライセンス情報 (a_v4_extfn_license_info)

デザインパートナーは、`a_v4_extfn_license_info` 構造体を使用して、会社名、ライブラリのバージョン情報、SAP が提供するライセンスキーなど、ライブラリレベルでの UDF のライセンス検証を定義します。

実装
```
typedef struct an_extfn_license_info {
    short     version;
} an_extfn_license_info;

typedef struct a_v4_extfn_license_info {
    an_extfn_license_info version;

    const char        name[255];
    const char        info[255];
    void *        key;
} a_v4_extfn_license_info;
```

データメンバーの概要

データメンバー	説明
version	内部でのみ使用。1 に設定する必要がある。
name	UDF で会社名として設定する値。

データメンバー	説明
info	UDF でライブラリバージョンやビルド番号などの追加的なライブラリ情報として設定する値。
key	(デザインパートナーのみ) SAP が提供するライセンスキー。キーは 26 文字の配列である。

オプティマイザの推定 (a_v4_extfn_estimate)

`a_v4_extfn_estimate` 構造体は推定を表します。推定には値と信頼レベルが含まれます。

実装

```
typedef struct a_v4_extfn_estimate {
    double    value;
    double    confidence;
} a_v4_extfn_estimate;
```

データメンバーとデータ型の概要

データメンバー	データ型	説明
value	double	推定値。
confidence	double	この推定に該当する信頼レベル。信頼レベルは 0.0 ～ 1.0 の間で変わる。0.0 は推定が無効であることを表し、1.0 は推定が確実であることを表す。

ORDER BY リスト (a_v4_extfn_orderby_list)

`a_v4_extfn_orderby_list` 構造体はテーブルの ORDER BY プロパティを表します。

実装

```
typedef struct a_v4_extfn_orderby_list {
    a_sql_uint32        number_of_elements;
    a_v4_extfn_order_el order_elements[1];      // there are
number_of_elements entries
} a_v4_extfn_orderby_list;
```

データメンバーとデータ型の概要

データメンバー	データ型	説明
number_of_elements	a_sql_uint32	エントリの数。
order_elements[1]	a_v4_extfn_order_el	要素の順序。

説明
エントリは *number_of_elements* 個あり、それぞれのエントリには、要素が昇順と降順のどちらであるかを示すフラグと、該当するテーブル内の適切なカラムを示すカラムインデックスがあります。

PARTITION BY のカラム番号 (a_v4_extfn_partitionby_col_num)

`a_v4_extfn_partitionby_col_num` 列挙型はカラム番号を表し、UDF で SQL と同様の **PARTITION BY** のサポートを表明できます。

実装

```
typedef enum a_v4_extfn_partitionby_col_num {
    EXTFNAPIV4_PARTITION_BY_COLUMN_NONE = -1,        // NO PARTITION
BY
    EXTFNAPIV4_PARTITION_BY_COLUMN_ANY  = 0,         // PARTITION BY
ANY
                                  // + INTEGER representing a specific
column ordinal
} a_v4_extfn_partitionby_col_num;
```

メンバーの概要

a_v4_extfn_partitionby_col_num 列挙型のメンバー	値	説明
EXTFNAPIV4_PARTITION_BY_COLUMN_NONE	-1	NO PARTITION BY
EXTFNAPIV4_PARTITION_BY_COLUMN_ANY	0	PARTITION BY ANY。特定のカラム順序数を表す正の整数。
Column Ordinal Number	N > 0	パーティションの対象となるテーブルカラム番号の順序数。

説明
UDF では、この構造体を使用して、パーティション分割とその対象カラムをコードで記述できます。

a_v4_extfn_column_list number_of_columns フィールドの設定には、この列挙型を使用します。UDF が PARTITION BY のサポートをサーバに伝達するときには、number_of_columns を、列挙値のいずれかか、リストのカラム順序数の番号を表す正の整数に設定します。たとえば、パーティション分割をサポートしないことをサーバに伝達する場合には、構造体を次のように作成します。

```
a_v4_extfn_column_list nopby = {
EXTFNAPIV4_PARTITION_BY_COLUMN_NONE,
0
};
```

メンバー値の *EXTFNAPIV4_PARTITION_BY_COLUMN_ANY* は、UDF が任意の形式のパーティション分割をサポートすることをサーバに伝えます。

パーティションの対象となる順序数のセットを記述するには、構造体を次のように作成します。

```
a_v4_extfn_column_list nopby = {
2,
3, 4
};
```

こうすると、順序数が 3 と 4 のカラム 2 つによるパーティション分割を表せます。

注意： この例は説明のみを目的としたものであり、正当なコードではありません。呼び出し元は整数 3 つ分の構造体を適切に確保する必要があります。

ロー (a_v4_extfn_row)

a_v4_extfn_row 構造体は、単一のローのデータを表します。

実装

```
/* a_v4_extfn_row - */
typedef struct a_v4_extfn_row {
    a_sql_uint32            *row_status;
    a_v4_extfn_column_data    *column_data;
} a_v4_extfn_row;
```

データメンバーとデータ型の概要

データメンバー	データ型	説明
row_status	a_sql_uint32 *	ローのステータス。存在するローは 1、それ以外は 0 に設定する。
column_data	a_v4_extfn_column_data *	ローのカラムデータの配列。

説明

ローの構造体は、いくつかのカラムで構成される特定のローの情報を保持します。この構造体は個別のローのステータスを定め、そのローが持つ個別のカラムのポ

インタも保持しています。ローのステータスは、ローの存在を示すフラグです。ローステータスフラグはネストしたフェッチ呼び出しで変更でき、ローブロック構造体の操作は必要ありません。

row_status フラグを 1 に設定すると、そのローが使用可能で結果セットに含めることができるということを表します。*row_status* を 0 に設定すると、そのローを無視する必要があることを表します。これは、TPF がフィルタの役割を果たしている場合に便利です。入力テーブル内のローを結果セットにパススルーするときに、ステータスを 0 に設定することによって一部のローをスキップできるからです。

ローブロック (a_v4_extfn_row_block)

`a_v4_extfn_row_block` 構造体は、複数のローで構成されるブロックのデータを表します。

実装

```
/* a_v4_extfn_row_block - */
typedef struct a_v4_extfn_row_block {
    a_sql_uint32        max_rows;
    a_sql_uint32        num_rows;
    a_v4_extfn_row  *row_data;
} a_v4_extfn_row_block;
```

データメンバーとデータ型の概要

データメンバー	データ型	説明
max_rows	a_sql_uint32	このローブロックが対応できるローの最大数。
num_rows	a_sql_uint32	ローブロックが保持できるローの最大数以下である必要がある。
row_data	a_v4_extfn_row *	ローデータのポインタ。

説明

ローブロック構造体は、`fetch_into` メソッドと `fetch_block` メソッドで使用し、プロデューサでのデータの生成とコンシューマでの利用に対応しています。ローの最大数は、ローブロックを確保した側が設定します。ローの数はプロデューサが正確に設定します。コンシューマは、プロデューサが生成したロー数を超えてデータを読み込まないようにする必要があります。

`row_block` 構造体の *max_rows* データメンバーの値は、構造体を所有する側が決定します。たとえば、テーブル UDF が `fetch_into` を実装している場合、*max_rows* の値は、128K のメモリに収まるロー数としてサーバが決定します。一方、テーブル UDF が `fetch_block` を実装している場合、*max_rows* の値は、テーブル UDF が自ら決定します。

制約と制限事項

num_rows と *max_rows* は、どちらも 0 より大きな値です。*num_rows* は *max_rows* 以下である必要があります。有効なローブロックについては、row_data フィールドは NULL 以外である必要があります。

テーブル (a_v4_extfn_table)

a_v4_extfn_table 構造体は、テーブルにデータを格納する方法と、コンシューマがそのデータをフェッチする方法を表します。

実装

```
typedef struct a_v4_extfn_table {
    a_v4_extfn_table_func       *func;
    a_sql_uint32                number_of_columns;
} a_v4_extfn_table;
```

データメンバーとデータ型の概要

データメンバー	データ型	説明
func	a_v4_extfn_table_func *	このメンバーは、コンシューマが結果データのフェッチに使用する一連の関数ポインタを保持する。
number_of_columns	a_sql_uint32 *	テーブル内のカラム数。

テーブルコンテキスト (a_v4_extfn_table_context)

a_v4_extfn_table_context 構造体は、テーブルに対して開いた結果セットを表します。

実装

```
typedef struct a_v4_extfn_table_context {

//    size_t struct_size;

    /* fetch_into() - fetch into a specified row_block. This entry point
        is used when the consumer has a transfer area with a specific format.
       The fetch_into() function will write the fetched rows into the provided row block.
    */
    short (UDF_CALLBACK *fetch_into)(a_v4_extfn_table_context *cntxt,
a_v4_extfn_row_block *);

    /* fetch_block() - fetch a block of rows. This entry point is used
        when the consumer does not need the data in a particular format. For example,
        if the consumer is reading a result set and formatting it as HTML, the consumer
       does not care how the transfer area is layed out. The fetch_block() entry point is
        more efficient if the consumer does not need a specific layout.

        The row_block parameter is in/out. The first call should point to a NULL row
```

```
block.
        The fetch_block() call sets row_block to a block that can be consumed, and this
block
         should be passed on the next fetch_block() call.
    */
    short (UDF_CALLBACK *fetch_block)(a_v4_extfn_table_context *cntxt,
a_v4_extfn_row_block **row_block);

    /*  rewind() - this is an optional entry point. If NULL, rewind is not supported.
Otherwise,
         the rewind() entry point restarts the result set at the beginning of the table.
    */
    short (UDF_CALLBACK *rewind)(a_v4_extfn_table_context *);

    /* get_blob() - If the specified column has a blob object, return it.  The blob
         is returned as an out parameter and is owned by the caller.  This method should
         only be called on a column that contains a blob.  The helper macro
EXTFN_COL_IS_BLOB can
         be used to determine whether a column contains a blob.
    */
    short (UDF_CALLBACK *get_blob)(a_v4_extfn_table_context *cntxt,
                    a_v4_extfn_column_data *col,
                    a_v4_extfn_blob **blob);

   /* The following fields are reserved for future use and must be initialized to NULL.
*/
    void  *reserved1_must_be_null;
    void  *reserved2_must_be_null;
    void  *reserved3_must_be_null;
    void  *reserved4_must_be_null;
    void  *reserved5_must_be_null;

    a_v4_extfn_proc_context     *proc_context;
    void                        *args_handle;           // use in
a_v4_extfn_proc_context::get_value() etc.
    a_v4_extfn_table            *table;
    void                        *user_data;
    void                        *server_internal_use;

   /* The following fields are reserved for future use and must be initialized to NULL.
*/
    void  *reserved6_must_be_null;
    void  *reserved7_must_be_null;
    void  *reserved8_must_be_null;
    void  *reserved9_must_be_null;
    void  *reserved10_must_be_null;

} a_v4_extfn_table_context;
```

メソッドの概要

データ型	メソッド	説明
short	**fetch_into**	指定の row_block にフェッチする。
short	**fetch_block**	ローのブロックをフェッチする。
short	**rewind**	テーブルの先頭から結果セットを再度開始する。
short	**get_blob**	指定のカラムに BLOB オブジェクトが格納されている場合、BLOB オブジェクトを返す。

データメンバーとデータ型の概要

データメンバー	データ型	説明
proc_context	a_v4_extfn_proc_context *	プロシージャコンテキストオブジェクトへのポインタ。UDF はこれを使用して、エラーの設定、ログメッセージの記録、キャンセルなどを実行できる。
args_handle	void *	サーバが指定する引数のハンドル。
table	a_v4_extfn_table *	開いた結果セットテーブルのポインタ。`a_v4_extfn_proc_context open_result_set` の呼び出し後に設定される。
_user_data	void *	このデータポインタは、外部ルーチンが必要とする任意のコンテキストデータの使用により設定できる。
server_internal_use	void *	内部でのみ使用。

説明

`a_v4_extfn_table_context` 構造体は、プロデューサとコンシューマの仲立ちとなる中間層の役割を果たし、コンシューマとプロデューサで別のフォーマットが必要な場合のデータの管理に役立ちます。

UDF は、`a_v4_extfn_table_context` を使用して、入力 TABLE パラメータからローを読み込むことができます。サーバまたは別の UDF は、`a_v4_extfn_table_context` を使用して、UDF の結果テーブルのローを読み込むことができます。

`a_v4_extfn_table_context` のメソッドはサーバが実装します。サーバはこれを生かすことで、障害となる不一致を解決できます。

fetch_into

v4 API メソッド `fetch_into` では、指定のローブロックにデータをフェッチします。

宣言

```
short fetch_into(
a_v4_extfn_table_context *cntxt,
a_v4_extfn_row_block *)
```

使用法
`fetch_into` メソッドは、メモリ内でのデータの配置方法をプロデューサが関知していない場合に便利です。このメソッドは、コンシューマが転送領域を特定のフォーマットで保持している場合にエントリポイントとして使用します。`fetch_into()` 関数はフェッチしたローを指定のローブロックに書き込みます。このメソッドは `a_v4_extfn_table_context` 構造体に含まれています。

データ転送領域のメモリをコンシューマが保持し、その領域を使用するようプロデューサに要求する場合には、`fetch_into` を使用します。`fetch_into` を使用する場合、データ転送領域のセットアップはコンシューマが担い、その領域に対して必要なデータをコピーする処理はプロデューサが担います。

パラメータ

パラメータ	説明
cntxt	`open_result_set` API で取得したテーブルコンテキストオブジェクト。
row_block	フェッチ先のローブロックオブジェクト。

戻り値
成功の場合は 1、それ以外の場合は 0。

UDF が値 1 を返した場合は、まだローが残っていることをコンシューマが認識し、`fetch_into` メソッドを再度呼び出す必要があります。一方、UDF が値 0 を返した場合は、他にローが残っていないことを表します。`fetch_into` メソッドを再度呼び出す必要はありません。

次のプロシージャ定義は、入力パラメータのテーブルを利用し、それを結果テーブルとして生成する TPF 関数の例です。両テーブルは、それぞれ v4 API メソッドの `get_value` と `set_value` で取得および返される SQL 値のインスタンスです。

```
CREATE PROCEDURE FETCH_EX( IN a INT, INT b TABLE( c1 INT ) )
    RESULT SET ( rc INT )
```

このプロシージャ定義には次の 2 つのテーブルオブジェクトが含まれています。

- `b` という名前の入力 TABLE パラメータ
- 返される結果セットテーブル

次の例は、呼び出し元 (ここではサーバ) が出力テーブルをフェッチする方法を示しています。サーバは `fetch_into` メソッドの使用を決定できます。入力テーブルは、呼び出された側 (ここでは TPF) がフェッチします。使用するフェッチ API は TPF が決定します。

```
SELECT rc from FETCH_EX( 1, TABLE( SELECT c1 from TABLE ) )
```

次の例が示すように、入力テーブルのフェッチや利用の前には、a_v4_extfn_proc 構造体の open_result_set API を使用してテーブルコンテキストを確立する必要があります。open_result_set にはテーブルオブ ジェクトが必要で、これは get_value API で取得できます。

```
    an_extfn_value    arg;
    ctx->get_value( args_handle, 3, &arg );

    if( arg.type != DT_EXTFN_TABLE ) {
        // handle error
    }

    a_v4_extfn_table_context    *rs = NULL;
    a_v4_extfn_table         *inTable = arg.data;
    ctx->open_result_set( ctx, inTable, &rs );
```

テーブルコンテキストを作成したら、rs 構造体が fetch_into API を実行してローをフェッチします。

```
a_v4_extfn_row_block    *rb = // get a row block to hold a series of
INT values.
rs->fetch_into( rs, &rb )  // fetch the rows.
```

結果テーブルにローを生成する前には、テーブルオブジェクトを作成して呼び出し元に返す必要があります。それには、a_v4_extfn_proc_context 構造体の set_value API を使用します。

この例が示すように、テーブル UDF は a_v4_extfn_table 構造体のインスタンスを作成する必要があります。テーブル UDF のそれぞれの呼び出しに対し、a_v4_extfn_table 構造体の別個のインスタンスを返します。テーブルには状態を示すフィールドがあり、現在のローと生成するロー数を保持できます。テーブルの状態はインスタンスのフィールドとして格納できます。

```
    typedef struct rg_table : a_v4_extfn_table {
        a_sql_uint32        rows_to_generate;
        a_sql_uint32        current_row;
    } my_table;
```

次の例では、ローを生成するごとに **current_row** をインクリメントして、生成するロー数に達するまで繰り返しています。生成するロー数に達すると、fetch_into は false を返し、ファイルの末尾であることを通知します。コンシューマは、テーブル UDF が実装する fetch_into API を呼び出します。fetch_into メソッドを呼び出すときに、コンシューマはテーブルコンテキストとフェッチ先のローブロックを渡します。

```
     rs->fetch_into( rs, &rb )

short UDF_CALLBACK my_table_func_fetch_into(
```

```
    a_v4_extfn_table_context *tctx,
    a_v4_extfn_row_block *rb)
/*****************************************/
{
    my_table *myTable = tctx->table;

    if( rgTable->current_row < rgTable->rows_to_generate ) {
            // Produce the row...
    rgTable->current_row++;
    return 1;
    }

    return 0;
}
```

fetch_block

v4 API メソッド `fetch_block` では、ローのブロックをフェッチします。

宣言

```
short fetch_block(
a_v4_extfn_table_context *cntxt,
a_v4_extfn_row_block **row_block)
```

使用法

`fetch_block` メソッドは、コンシューマがデータについて特定のフォーマットを必要としていない場合にエントリポイントとして使用します。`fetch_block` ではプロデューサに対し、データ転送領域を作成してその領域のポインタを渡すよう要求します。コンシューマはこのメモリを保持し、この領域からデータをコピーする必要があります。

コンシューマが特定のレイアウトを必要としない場合には、`fetch_block` の方が効率的です。 `fetch_block` の呼び出しでは、データを格納できるローブロックを `fetch_block` に設定し、次の `fetch_block` 呼び出しでもそのブロックを渡します。このメソッドは `a_v4_extfn_table_context` 構造体に含まれています。

パラメータ

パラメータ	説明
cntxt	テーブルコンテキストオブジェクト。
row_block	入力／出力パラメータ。最初の呼び出しでは、必ず NULL の **row_block** を指す。

`fetch_block` が呼び出され、**row_block** が NULL を指している場合には、UDF は `a_v4_extfn_row_block` 構造体を割り付ける必要があります。

戻り値
成功の場合は 1、それ以外の場合は 0。

UDF が値 1 を返した場合は、まだローが残っていることをコンシューマが認識し、fetch_block メソッドを再度呼び出します。一方、UDF が値 0 を返した場合は、他にローが残っていないことを表します。fetch_block メソッドを再度呼び出す必要はありません。

次のプロシージャ定義は、入力パラメータのテーブルを利用し、それを結果テーブルとして生成する TPF 関数の例です。両テーブルは、それぞれ v4 API メソッドの get_value と set_value で取得および返される SQL 値のインスタンスです。

```
CREATE PROCEDURE FETCH_EX( IN a INT, INT b TABLE( c1 INT ) )
    RESULT SET ( rc INT )
```

このプロシージャ定義には次の 2 つのテーブルオブジェクトが含まれています。

- b という名前の入力 TABLE パラメータ
- 返される結果セットテーブル

次の例は、呼び出し元 (ここではサーバ) が出力テーブルをフェッチする方法を示しています。サーバは fetch_block メソッドの使用を決定できます。入力テーブルは、呼び出された側 (ここでは TPF) がフェッチします。使用するフェッチ API は TPF が決定します。

```
SELECT rc from FETCH_EX( 1, TABLE( SELECT c1 from TABLE ) )
```

次の例が示すように、入力テーブルのフェッチや利用の前には、a_v4_extfn_proc 構造体の open_result_set API を使用してテーブルコンテキストを確立する必要があります。open_result_set にはテーブルオブ ジェクトが必要で、これは get_value API で取得できます。

```
    an_extfn_value    arg;
    ctx->get_value( args_handle, 3, &arg );

    if( arg.type != DT_EXTFN_TABLE ) {
        // handle error
    }

    a_v4_extfn_table_context    *rs = NULL;
    a_v4_extfn_table                    *inTable = arg.data;
    ctx->open_result_set( ctx, inTable, &rs );
```

テーブルコンテキストを作成したら、rs 構造体が fetch_block API を実行してローをフェッチします。

```
a_v4_extfn_row_block    *rb =  // get a row block to hold a series of
INT values.
rs->fetch_block( rs, &rb )  // fetch the rows.
```

結果テーブルにローを生成する前には、テーブルオブジェクトを作成して呼び出し元に返す必要があります。それには、`a_v4_extfn_proc_context` 構造体の `set_value` API を使用します。

この例が示すように、テーブル UDF は `a_v4_extfn_table` 構造体のインスタンスを作成する必要があります。テーブル UDF のそれぞれの呼び出しに対し、`a_v4_extfn_table` 構造体の別個のインスタンスを返します。テーブルには状態を示すフィールドがあり、現在のローと生成するロー数を保持できます。テーブルの状態はインスタンスのフィールドとして格納できます。

```
    typedef struct rg_table : a_v4_extfn_table {
        a_sql_uint32        rows_to_generate;
        a_sql_uint32        current_row;
    } my_table;
```

rewind

v4 API メソッド `rewind` を使用して、テーブルの先頭から結果セットを再度開始します。

宣言

```
short rewind(
       a_v4_extfn_table_context           *cntxt,
       )
```

使用法

開いている結果セットの `rewind` メソッドを呼び出して、テーブルを先頭までリワインドします。UDF が入力テーブルをリワインドするときには、**EXTFNAPIV4_STATE_OPTIMIZATION** 状態の間に **EXTFNAPIV4_DESCRIBE_PARM_TABLE_REQUEST_REWIND** パラメータを使用してプロデューサに伝える必要があります。

`rewind()` はオプションのエントリポイントです。NULL の場合、リワインドはサポートされていません。それ以外の場合、`rewind()` エントリポイントによって、結果セットはテーブルの先頭から再度開始されます。

パラメータ

パラメータ	説明
cntxt	テーブルコンテキストオブジェクト。

戻り値

成功の場合は 1、それ以外の場合は 0。

get_blob

v4 API メソッド get_blob を使用して、指定のカラムの BLOB オブジェクトを取得します。

宣言

```
 short get_blob(
a_v4_extfn_table_context *cntxt,
a_v4_extfn_column_data *col,
a_v4_extfn_blob **blob
)
```

使用法

BLOB は出力パラメータで返り、呼び出し元が所有します。 このメソッドは BLOB が格納されたカラムに対してのみ使用します。

カラムに BLOB が格納されているかどうかは、ヘルパーマクロ EXTFN_COL_IS_BLOB を使用して判断します。ヘッダファイル extfnapiv4.h での EXTFN_COL_IS_BLOB の宣言は次のとおりです。

```
 #define EXTFN_COL_IS_BLOB(c, n)         (c[n].blob_handle != NULL)
```

パラメータ

パラメータ	説明
cntxt	テーブルコンテキストオブジェクト。
col	BLOB を取得する対象のカラムデータポインタ。
blob	成功の場合は、指定のカラムに該当する BLOB オブジェクトが格納される。

戻り値

成功の場合は 1、それ以外の場合は 0。

テーブル関数 (a_v4_extfn_table_func)

コンシューマは a_v4_extfn_table_func 構造体を使用してプロデューサから結果を取得します。

実装

```
typedef struct a_v4_extfn_table_func {
//    size_t struct_size;

    /*  Open a result set. The UDF can allocate any resources needed
for the result set.
    */
```

```
    short (UDF_CALLBACK *_open_extfn)(a_v4_extfn_table_context *);

   /*  Fetch rows into a provided row block. The UDF should implement
this method if it does
        not have a preferred layout for its transfer area.
    */
   short (UDF_CALLBACK *_fetch_into_extfn)(a_v4_extfn_table_context
*, a_v4_extfn_row_block

*row_block);

   /*  Fetch a block that is allocated and configured by the UDF. The
UDF should implement this
        method if it has a preferred layout of the transfer area.
    */
    short (UDF_CALLBACK *_fetch_block_extfn)
(a_v4_extfn_table_context *, a_v4_extfn_row_block

**row_block);

   /*  Restart a result set at the beginning of the table. This is an
optional entry point.
    */
    short (UDF_CALLBACK *_rewind_extfn)(a_v4_extfn_table_context *);

    /*  Close a result set. The UDF can release any resources
allocated for the result set.
    */
    short (UDF_CALLBACK *_close_extfn)(a_v4_extfn_table_context *);

    /* The following fields are reserved for future use and must be
initialized to NULL. */
    void *_reserved1_must_be_null;
    void *_reserved2_must_be_null;

} a_v4_extfn_table_func;
```

メソッドの概要

メソッド	データ型	説明
_open_extfn	void	結果セットを開いてローのフェッチを開始するために、サーバが呼び出す。UDF は結果セットに必要なリソースを確保できる。
_fetch_into_extfn	short	指定のローブロックにローをフェッチする。UDF は、転送領域のレイアウトについての意向が特にない場合にこのメソッドを実装する。

メソッド	データ型	説明
_fetch_block_extfn	short	UDF が確保および設定したブロックにフェッチする。UDF は、転送領域のレイアウトについて意向がある場合にこのメソッドを実装する。
_rewind_extfn	void	フェッチをテーブルの先頭から再度開始するためにサーバが呼び出すオプションの関数。
_close_extfn	void	結果セットを閉じてローのフェッチを終了するために、サーバが呼び出す。UDF は結果セット用として確保したリソースを解放できる。
_reserved1_must_be_null	void	今後のために予約済み。NULL に初期化する必要がある。
_reserved1_must_be_null	void	今後のために予約済み。NULL に初期化する必要がある。

説明

a_v4_extfn_table_func 構造体では、テーブルから結果をフェッチするために使用するメソッドを定義します。

_open_extfn

サーバは v4 API メソッド _open_extfn を呼び出してローのフェッチを開始します。

宣言

```
void _open_extfn(
     a_v4_extfn_table_context *cntxt,
)
```

使用法

UDF はこのメソッドを使用して、結果セットを開き、サーバに結果を送信するために必要なリソース (たとえばストリーム) を確保します。

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。

_fetch_into_extfn

v4 API メソッド `_fetch_into_extfn` では、指定のローブロックにローをフェッチします。

宣言

```
short _fetch_into_extfn(
    a_v4_extfn_table_context *cntxt,
    a_v4_extfn_row_block *row_block
    )
```

使用法

UDF は、転送領域のレイアウトについての意向が特にない場合に、このメソッドを実装する必要があります。

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。
row_block	フェッチ先のローブロックオブジェクト。

戻り値

成功の場合は 1、それ以外の場合は 0。

_fetch_block_extfn

v4 API メソッド `_fetch_block_extfn` では、UDF が確保および設定したブロックにフェッチします。

宣言

```
short _fetch_block_extfn(
a_v4_extfn_table_context *cntxt,
a_v4_extfn_row_block **
)
```

使用法

UDF は、転送領域のレイアウトについての意向がある場合に、このメソッドを実装する必要があります。

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。

パラメータ	説明
row_block	フェッチ先のローブロックオブジェクト。

戻り値
成功の場合は 1、それ以外の場合は 0。

_rewind_extfn
v4 API メソッド _rewind_extfn では、結果セットをテーブルの先頭から再度開始します。

宣言
```
void _rewind_extfn(
a_v4_extfn_table_context *cntxt,
)
```

使用法
この関数はオプションのエントリポイントです。UDF は、結果テーブルが先頭までリワインドされる場合に `_rewind_extfn` メソッドを実装します。UDF は、コストがかからず効率のよい方法でリワインド機能を提供できる場合にのみ、このメソッドの実装を検討します。

`_rewind_extfn` メソッドを実装することを選択した UDF は、そのことをコンシューマに通知するために、**EXTFNAPIV4_STATE_OPTIMIZATION** 状態のときに **EXTFNAPIV4_DESCRIBE_PARM_TABLE_HAS_REWIND** パラメータを引数 0 に対し設定する必要があります。

UDF は、リワインド機能を提供しないことも決定できます。その場合は、サーバがこれを補い、機能を提供します。

注意： サーバは、`_rewind_extfn` メソッドの呼び出しによるリワインドの実行を行わないことも選択できます。

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。

戻り値
戻り値はありません。

_close_extfn

サーバは v4 API メソッド `_close_extfn` を呼び出してローのフェッチを終了します。

宣言

```
void _close_extfn(
     a_v4_extfn_table_context *cntxt,
)
```

使用法

UDF は、フェッチが完了したときにこのメソッドを使用します。これによって結果セットを閉じ、結果セット用に確保したリソースを解放します。

パラメータ

パラメータ	説明
cntxt	プロシージャコンテキストオブジェクト。

アプリケーションでの SQL の使用

この項では、アプリケーションでの SQL の使用について説明します。

アプリケーションでの SQL 文の実行

アプリケーションに SQL 文をインクルードする方法は、使用するアプリケーション開発ツールとプログラミングインタフェースによって異なります。

- **ADO.NET –** さまざまな ADO.NET オブジェクトを使用して、SQL 文を実行できます。SACommand オブジェクトはその 1 つの例です。

```
SACommand cmd = new SACommand(
     "DELETE FROM Employees WHERE EmployeeID = 105", conn );
cmd.ExecuteNonQuery();
```

- **ODBC –** ODBC プログラミングインタフェースに直接書き込む場合、関数呼び出し部分に SQL 文を記述します。たとえば、次の C 言語の関数呼び出しは DELETE 文を実行します。

```
SQLExecDirect( stmt,
    "DELETE FROM Employees
     WHERE EmployeeID = 105",
    SQL_NTS );
```

- **JDBC –** JDBC プログラミングインタフェースを使っている場合、statement オブジェクトのメソッドを呼び出して SQL 文を実行できます。次に例を示します。

```
stmt.executeUpdate(
    "DELETE FROM Employees
     WHERE EmployeeID = 105" );
```

- **Embedded SQL –** Embedded SQL を使っている場合、キーワード EXEC SQL を C 言語の SQL 文の前に置きます。次にコードをプリプロセッサに通してから、コンパイルします。次に例を示します。

```
EXEC SQL EXECUTE IMMEDIATE
 'DELETE FROM Employees
  WHERE EmployeeID = 105';
```

- **Sybase Open Client –** Sybase Open Client インタフェースを使っている場合、関数呼び出し部分に SQL 文を記述します。たとえば、次の一組の呼び出しは DELETE 文を実行します。

```
ret = ct_command( cmd, CS_LANG_CMD,
                   "DELETE FROM Employees
                    WHERE EmployeeID=105"
                  CS_NULLTERM,
                  CS_UNUSED);
ret = ct_send(cmd);
```

アプリケーションに SQL をインクルードする方法の詳細については、使用している開発ツールのマニュアルを参照してください。ODBC または JDBC を使っている場合、そのインタフェース用ソフトウェア開発キットを調べてください。

データベースサーバ内のアプリケーション
ストアドプロシージャとトリガは、データベースサーバ内で実行するアプリケーションまたはその一部として、さまざまな方法で動作します。この場合、ストアドプロシージャの多くのテクニックも使用できます。

データベース内の Java クラスはサーバ外部の Java アプリケーションとまったく同じ方法で JDBC インタフェースを使用できます。この項では JDBC についても一部説明します。

準備文

文がデータベースへ送信されるたびに、データベースサーバは次の手順を実行する必要があります。

- 文を解析し、内部フォームに変換する。このプロセスは文の*準備*とも呼ばれます。
- データベースオブジェクトへの参照がすべて正確であるかどうかを確認する。たとえば、クエリで指定されたカラムが実際に存在するかどうかをチェックします。
- 文にジョインまたはサブクエリが含まれている場合、クエリオプティマイザがアクセスプランを生成する。
- これらすべての手順を実行してから文を実行する。

準備文の再使用によるパフォーマンスの改善
同じ文を繰り返し使用する (たとえば、1 つのテーブルに多くのローを挿入する) 場合、文の準備を繰り返し行うことにより著しいオーバヘッドが生じます。このオーバヘッドを解消するため、データベースプログラミングインタフェースによっては、準備文の使用方法を提示するものもあります。*準備文*とは、一連のプレースホルダを含む文です。文を実行するときには、文全体を何度も準備せずに、プレースホルダに値を割り当てます。

たくさんのローを挿入するときなど、同じ動作を何度も繰り返す場合は、準備文を使用すると便利です。

通常、準備文を使用するには次の手順が必要です。

- **文を準備する** – ここでは通常、値の代わりにプレースホルダを文に入力します。
- **準備文を繰り返し実行する** – ここでは、文を実行するたびに、使用する値を入力します。実行するたびに文を準備する必要ありません。

- **文を削除する –** ここでは、準備文に関連付けられたリソースを解放します。この手順を自動的に処理するプログラミングインタフェースもあります。

一度だけ使用する文は準備しない
通常、一度だけの実行には文を準備しません。準備と実行を別々に行うと、わずかではあってもパフォーマンスペナルティが生じ、アプリケーションに不要な煩雑さを招きます。

ただし、インタフェースによっては、カーソルに関連付けするためだけに文を準備する必要があります。

文の準備と実行命令の呼び出しは SQL の一部ではなく、インタフェースによって異なります。SAP Sybase IQ の各プログラミングインタフェースによって、準備文を使用する方法が示されます。

準備文の概要

この項では準備文の使用法についての簡単な概要を説明します。一般的な手順は同じですが、詳細はインタフェースによって異なります。異なるインタフェースで準備文の使い方を比較すると、違いがはっきりします。

準備文を使用するには、一般的に次のタスクを実行します。

1. 文を準備します。
2. 文中の値を保持するパラメータをバインドします。
3. 文中のバウンドパラメータに値を割り当てます。
4. 文を実行します。
5. 必要に応じて手順 3 と 4 を繰り返します。
6. 終了したら、文を削除します。JDBC では、Java ガーベジコレクションメカニズムにより文が削除されます。

ADO.NET での準備文の使用
ADO.NET で準備文を使用するには、一般的に次のタスクを実行します。

1. 文を保持する SACommand オブジェクトを作成します。
```
SACommand cmd = new SACommand(
     "SELECT * FROM Employees WHERE Surname = ?", conn );
```
2. 文中のパラメータのデータ型を宣言します。
   SACommand.CreateParameter メソッドを使用します。
```
SAParameter param = cmd.CreateParameter();
param.SADbType = SADbType.Char;
param.Direction = ParameterDirection.Input;
param.Value = "Smith";
cmd.Parameters.Add(param);
```
3. Prepare メソッドを使って文を準備します。

4. 次の文を実行します。
```
SADataReader reader = cmd.ExecuteReader();
```

ADO.NET を使用して文を準備する例については、%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥ADO.NET¥SimpleWin32 にあるソースコードを参照してください。

ODBC での準備文の使用
ODBC で準備文を使用するには、一般的に次のタスクを実行します。

1. SQLPrepare を使って文を準備します。
2. SQLBindParameter を使って文のパラメータをバインドします。
3. SQLExecute を使って文を実行します。
4. SQLFreeStmt を使って文を削除します。

ODBC を使用して文を準備する例については、%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥ODBCPrepare にあるソースコードを参照してください。

JDBC での準備文の使用
JDBC で準備文を使用するには、一般的に次のタスクを実行します。

1. 接続オブジェクトの prepareStatement メソッドを使って文を準備します。これによって準備文オブジェクトが返されます。
2. 準備文オブジェクトの適切な set*Type* メソッドを使って文パラメータを設定します。*Type* は割り当てられるデータ型です。
3. 準備文オブジェクトの適切なメソッドを使って文を実行します。挿入、更新、削除には、executeUpdate メソッドを使います。

JDBC を使用して文を準備する例については、ソースコードファイル %ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥JDBC¥JDBCExample.java を参照してください。

Embedded SQL での準備文の使用
Embedded SQL で準備文を使用するには、一般的に次のタスクを実行します。

1. EXEC SQL PREPARE 文を使用して文を準備します。
2. 文中のパラメータに値を割り当てます。
3. EXEC SQL EXECUTE 文を使用して文を実行します。
4. EXEC SQL DROP 文を使用して、その文に関連するリソースを解放します。

Open Client での準備文の使用
Open Client で準備文を使用するには、一般的に次のタスクを実行します。

1. CS_PREPARE 型パラメータで ct_dynamic 関数を使用して文を準備します。
2. ct_param を使用して文のパラメータを設定します。
3. CS_EXECUTE 型パラメータで ct_dynamic を使用して文を実行します。
4. CS_DEALLOC 型パラメータで ct_dynamic を使用して文に関連付けられたリソースを解放します。

カーソルの使用法

アプリケーションでクエリを実行すると、結果セットが複数のローで構成されます。通常は、アプリケーションが受け取るローの数は、クエリを実行するまでわかりません。カーソルを使うと、アプリケーションでクエリの結果セットを処理する方法が提供されます。

カーソルを使用する方法と使用可能なカーソルの種類は、使用するプログラミングインタフェースによって異なります。

SAP Sybase IQ には、接続に使用するカーソルとその内容を判断するのに役立つシステムプロシージャがいくつかあります。

sa_list_cursors システムプロシージャ

sa_describe_cursor システムプロシージャ

sa_copy_cursor_to_temp_table システムプロシージャ

カーソルを使うと、プログラミングインタフェースで次のようなタスクを実行できます。

- クエリの結果をループする。
- 結果セット内の任意の場所で基本となるデータの挿入、更新、削除を実行する。

プログラミングインタフェースによっては、特別な機能を使用して、結果セットを自分のアプリケーションに返す方法をチューニングできるものもあります。この結果、アプリケーションのパフォーマンスは大きく向上します。

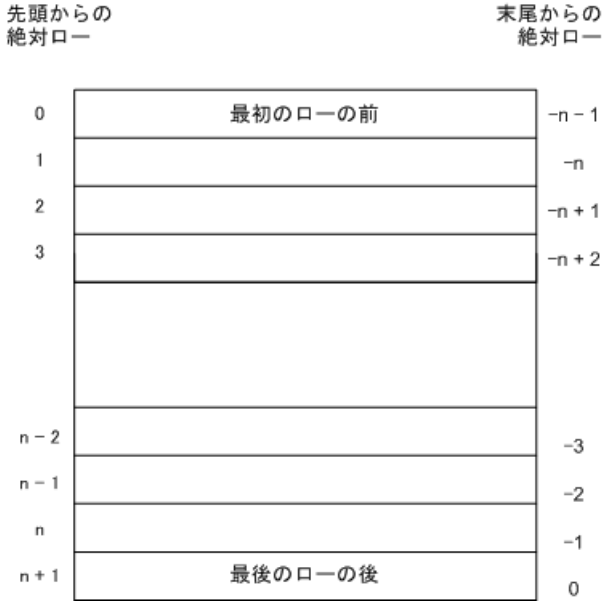
カーソル

カーソルとは、結果セットに関連付けられた名前です。結果セットは、SELECT 文かストアドプロシージャ呼び出しによって取得されます。

カーソルは、結果セットのハンドルです。カーソルには、結果セット内の適切に定義された位置が必ず指定されています。カーソルを使うと 1 回につき 1 つのローのデータを調べて操作できます。SAP Sybase IQ のカーソルは、クエリ結果内で前方や後方への移動をサポートします。

カーソル位置
カーソルは、次の場所に置くことができます。

- 結果セットの最初のローの前
- 結果セット内の 1 つのロー上
- 結果セットの最後のローの後



カーソル位置と結果セットは、データベースサーバで管理されます。ローは、クライアントによって *フェッチ*されて、1 回に 1 つまたは複数のローを表示して処理できます。結果セット全体がクライアントに配信される必要はありません。

カーソルを使用する利点

サーバ側のカーソルはデータベースアプリケーションでは必要ありませんが、いくつかの利点があります。次の理由により、サーバ側のカーソルはクライアント側のカーソルより推奨されます。

- **応答時間** – サーバ側のカーソルでは、最初のローがクライアントによってフェッチされる前に、結果セット全体をアセンブルする必要がありません。クライアント側のカーソルでは、最初のローがクライアントによってフェッチされる前に、結果セット全体を取得して転送する必要があります。
- **クライアント側メモリ** – 結果セットのサイズが大きい場合、結果セット全体をクライアント側で取得すると、クライアントに必要なメモリ容量が増えることがあります。

- **同時実行性の制御** – アプリケーションでデータを更新する場合にサーバ側のカーソルを使用しない場合、変更を適用するために UPDATE、INSERT、DELETE などの別の SQL 文をデータベースサーバに送信します。この方法では、結果セットがクライアントに送信されたためにデータベースの対応するローが変わった場合には、同時実行性の問題が生じる可能性があります。結果として、他のクライアントによる更新が失われる可能性があります。

  サーバ側のカーソルは基本となるデータへのポインタとして機能します。したがって、適切な独立性レベルを設定することによって、クライアントが加えた変更に適切な同時性制約を課すことができます。

カーソルの原則

ADO.NET、ODBC、JDBC、または Open Client でカーソルを使用するには、これらの一般的な手順に従います。

1. 文を準備して実行します。
   インタフェースの通常の方法を使用して文を実行します。文を準備して実行するか、文を直接実行します。
   ADO.NET の場合、SACommand.ExecuteReader メソッドのみがカーソルを返します。このコマンドは、読み込み専用、前方専用のカーソルを提供します。
2. 文が結果セットを返すかどうかを確認するためにテストします。
   結果セットを作成する文を実行する場合、カーソルは暗黙的に開きます。カーソルが開かれると、結果セットの第 1 ローの前に配置されます。
3. 結果をフェッチします。
   簡単なフェッチを行うと、結果セット内の次のローへカーソルが移動しますが、SAP Sybase IQ では結果セットでより複雑な移動が可能です。
4. カーソルを閉じます。
   カーソルでの作業が終了したら、閉じて関連するリソースを解放します。
5. 文を解放します。
   準備文を使った場合は、それを解放してメモリを再利用します。

Embedded SQL でカーソルを使用するためのアプローチは、他のインタフェースで使用されるアプローチと異なります。Embedded SQL でカーソルを使用するには、これらの一般的な手順に従います。

1. 文を準備します。
   通常、カーソルでは文字列ではなくステートメントハンドルが使用されます。ハンドルを使用可能にするために、文を準備する必要があります。
2. カーソルを宣言します。
   各カーソルは、単一の SELECT 文か CALL 文を参照します。カーソルを宣言するとき、カーソル名と参照した文を入力します。

3. カーソルを開きます。
   CALL 文の場合、カーソルを開くと、1 番目のローが取得されるポイントまでプロシージャが実行されます。
4. 結果をフェッチします。
   簡単なフェッチを行うと、結果セット内の次のローへカーソルが移動しますが、SAP Sybase IQ では結果セットでより複雑な移動が可能です。どのフェッチが実行可能であるかは、カーソルの宣言方法によって決定されます。
5. カーソルを閉じます。
   カーソルでの作業が終わったら、カーソルを閉じます。これにより、カーソルに関連付けられているリソースが解放されます。
6. 文を削除します。
   文に関連付けられているメモリを解放するには、文を削除する必要があります。

カーソル位置

カーソルを開くと最初のローの前に置かれます。カーソル位置は、クエリ結果の最初か最後を基準とした絶対位置、または現在のカーソル位置を基準とした相対位置に移動できます。カーソル位置の変更方法とカーソルで可能な操作は、プログラミングインタフェースが制御します。

カーソルでフェッチできるローの位置番号は、integer 型のサイズによって管理されます。integer に格納できる値より 1 小さい 2147483646 までの番号が付けられたローをフェッチできます。ローの位置番号に、クエリ結果の最後を基準として負の数を使用している場合、integer に格納できる負の最大値より 1 大きい数までの番号のローをフェッチできます。

現在のカーソル位置でローを更新または削除するには、位置付け更新と位置付け削除という特別な操作を使用できます。先頭のローの前か、末尾のローの後にカーソルがある場合、対応するカーソルローがないことを示すエラーが返されます。

注意： asensitive カーソルに挿入や更新をいくつか行うと、カーソル位置に問題が生じます。SELECT 文に ORDER BY 句を指定しないかぎり、SAP Sybase IQ はカーソル内の予測可能な位置にローを挿入しません。場合によっては、カーソルを閉じてもう一度開かないと、挿入したローがまったく表示されないことがあります。SAP Sybase IQ では、カーソルを開くためにワークテーブルを作成する必要がある場合にこうしたことが起こります。

UPDATE 文によって、カーソル内のローが移動することがあります。これは、既存のインデックスを使用する ORDER BY 句がカーソルに指定されている場合に発生します (ワークテーブルは作成されません)。STATIC SCROLL カーソルを使うとこの問題はなくなりますが、より資源を消費します。

カーソルを開くときのカーソルの動作

カーソルを開くとき、カーソルの動作について次のように設定できます。

- **独立性レベル –** カーソルに操作の独立性レベルを明示的に設定して、トランザクションの現在の独立性レベルと区別できます。これを行うには、isolation_level オプションを設定します。
- **保持 –** デフォルトでは、Embedded SQL のカーソルはトランザクションの終了時に閉じます。WITH HOLD でカーソルを開くと、接続終了まで、または明示的に閉じるまでカーソルを開いたままにできます。ADO.NET、ODBC、JDBC、Open Client はデフォルトでトランザクションの終了時までカーソルを開いたままにします。

カーソルによるローのフェッチ

カーソルを使用してクエリの結果セットをもっとも簡単に処理するには、ローがなくなるまで結果セットのすべてのローをループします。これらの手順を実行すると、このタスクを実行できます。

1. カーソル (Embedded SQL) を宣言して開くか、結果セット (ODBC、JDBC、Open Client) または SADataReader オブジェクト (ADO.NET) を返す文を実行します。
2. 「`Row Not Found`」 (ローが見つかりません) というエラーが表示されるまで、次のローをフェッチし続けます。
3. カーソルを閉じます。

次のローをフェッチするために使用する方法は、使用するインタフェースによって異なります。次に例を示します。

- **ADO.NET –** SADataReader.Read メソッドを使用します。
- **ODBC –** SQLFetch、SQLExtendedFetch、または SQLFetchScroll が次のローにカーソルを進め、データを返します。
- **JDBC –** ResultSet オブジェクトの next メソッドがカーソルを進め、データを返します。
- **Embedded SQL –** FETCH 文が同じ操作を実行します。
- **Open Client –** ct_fetch 関数が次のローにカーソルを進め、データを返します。

複数ローのフェッチ

複数ローのフェッチとローのプリフェッチを混同しないでください。複数のローのフェッチはアプリケーションによって実行されます。一方、プリフェッチはアプリケーションに対して透過的で、同様にパフォーマンスが向上します。一度に複数のローをフェッチすると、パフォーマンスを向上させることができます。

複数ローのフェッチ
インタフェースによっては、配列内の次のいくつかのフィールドへ複数のローを一度にフェッチすることができます。一般的に、実行する個々のフェッチ操作が少なければ少ないほど、サーバが応答する個々の要求が少なくなり、パフォーマンスが向上します。複数のローを取り出すように修正された FETCH 文を*ワイドフェッチ*と呼ぶこともあります。複数のローのフェッチを使うカーソルは*ブロックカーソル*または*ファットカーソル*とも呼びます。

複数ローフェッチの使用

- ODBC では、SQL_ATTR_ROW_ARRAY_SIZE または SQL_ROWSET_SIZE 属性を設定して、SQLFetchScroll または SQLExtendedFetch をそれぞれ呼び出したときに返されるローの数を設定できます。
- Embedded SQL では、FETCH 文で ARRAY 句を使用して、一度にフェッチされるローの数を制御します。
- Open Client と JDBC は複数のローのフェッチをサポートしません。これらのインタフェースではプリフェッチを使用します。

スクロール可能なカーソル

ODBC と Embedded SQL では、スクロール可能なカーソルと、スクロール可能で動的なカーソルを使う方法があります。この方法だと、結果セット内で一度にローをいくつか前方または後方へ移動できます。

JDBC または Open Client インタフェースではスクロール可能なカーソルはサポートされていません。

プリフェッチはスクロール可能な操作には適用されません。たとえば、逆方向へのローのフェッチにより、前のローがいくつかプリフェッチされることはありません。

ローの修正に使用するカーソル

カーソルには、クエリから結果セットを読み込む以外にも可能なことがあります。カーソルの処理中に、データベース内のデータ修正もできます。この操作は一般に*位置付け*挿入、更新、削除の操作と呼ばれます。また、挿入操作の場合は、これを PUT 操作ともいいます。

すべてのクエリの結果セットで、位置付け更新と削除ができるわけではありません。更新不可のビューにクエリを実行すると、基本となるテーブルへの変更は行われません。また、クエリがジョインを含む場合、ローの削除を行うテーブルまたは更新するカラムを、操作の実行時に指定してください。

テーブル内の任意の挿入されていないカラムに NULL を入力できるかデフォルト値が指定されている場合だけ、カーソルを使った挿入を実行できます。

複数のローが value-sensitive (キーセット駆動型) カーソルに挿入される場合、これらのローはカーソル結果セットの最後に表示されます。これらのローは、クエリの WHERE 句と一致しない場合や、ORDER BY 句が通常、これらを結果セットの別の場所に配置した場合でも、カーソル結果セットの最後に表示されます。この動作はプログラミングインタフェースとは関係ありません。たとえば、この動作は、Embedded SQL の PUT 文または ODBC SQLBulkOperations 関数を使用するときに適用されます。挿入された最新のローの AUTOINCREMENT カラムの値は、カーソルの最後のローを選択することによって確認できます。たとえば、Embedded SQL の場合、`FETCH ABSOLUTE -1 cursor-name` を使用することで値を取得できます。この動作のため、value-sensitive カーソルに対する最初の複数のローの挿入は負荷が大きくなる可能性があります。

ODBC、JDBC、Embedded SQL、Open Client では、カーソルを使ったデータ操作が許可されますが、ADO.NET では許可されません。Open Client の場合、ローの削除と更新はできますが、ローの挿入は単一テーブルのクエリだけです。

どのテーブルからローを削除するか
カーソルを使って位置付け削除を試行する場合、ローを削除するテーブルは次のように決定されます。

1. DELETE 文に FROM 句が含まれない場合、カーソルは単一テーブルだけにあります。
2. カーソルがジョインされたクエリ用の場合 (ジョインがあるビューの使用を含めて)、FROM 句が使われます。指定したテーブルの現在のローだけが削除されます。ジョインに含まれた他のテーブルは影響を受けません。
3. FROM 句が含まれ、テーブル所有者が指定されない場合、table-spec 値がどの相関名に対しても最初に一致します。
4. 相関名がある場合、table-spec 値は相関名で識別されます。
5. 相関名がない場合、table-spec 値はカーソルのテーブル名として明確に識別可能にします。
6. FROM 句が含まれ、テーブル所有者が指定されている場合、table-spec 値はカーソルのテーブル名として明確に指定可能にします。
7. 位置付け DELETE 文はビューでカーソルを開くときに使用できます。ただし、ビューが更新可能である場合にかぎられます。

更新可能な文

この項では、SELECT 文の句が更新可能な文とカーソルに与える影響について説明します。

読み込み専用文の更新可能性
カーソル宣言で FOR READ ONLY を指定するか、FOR READ ONLY 句を文に含めると、文は読み込み専用になります。FOR READ ONLY 句を指定するか、クライ

アント API を使用している場合に読み込み専用カーソルを宣言すると、その他の更新可能性の指定は上書きされます。

SELECT 文の最も外側のブロックに ORDER BY 句が含まれている場合、文で FOR UPDATE を指定しないと、カーソルは読み込み専用になります。SQL の SELECT 文で FOR XML を指定すると、カーソルは読み込み専用になります。それ以外の場合、カーソルは更新可能です。

更新可能な文と同時実行性制御
更新可能な文の場合、SAP Sybase IQ にはカーソルに対してオプティミスティックとペシミスティックの両方の同時実行性制御メカニズムがあり、スクロール操作中の結果セットの一貫性が保たれます。これらのメカニズムは、セマンティックとトレードオフは異なりますが、INSENSITIVE カーソルやスナップショットアイソレーションに代わる方法です。

FOR UPDATE の指定は、カーソルの更新可能性に影響する場合があります。ただし SAP Sybase IQ では、FOR UPDATE 構文は同時実行性制御に対するその他の影響はありません。FOR UPDATE で追加のパラメータを指定すると、SAP Sybase IQ では次の 2 つの同時実行性制御オプションのいずれかを組み込むように文の処理が変更されます。

- **ペシミスティック** – カーソルの結果セットにフェッチされたすべてのローに対して、データベースサーバは意図的ローロックを取得して、別のトランザクションによってローが更新されないようにします。
- **オプティミスティック** – データベースサーバで使用されるカーソルのタイプがキーセット駆動型カーソル (insensitive ローメンバーシップ、value-sensitive) に変えられ、結果内のローが任意のトランザクションによって変更または削除されると、アプリケーションに通知されるようになります。

ペシミスティックまたはオプティミスティック同時実行性は、DECLARE CURSOR 文または FOR 文のオプション、または特定のプログラミングインタフェースの同時実行性設定 API を使用して、カーソルレベルで指定します。文が更新可能でカーソルに同時実行性制御メカニズムが指定されていない場合は、文の仕様が使用されます。構文は次のとおりです。

- **FOR UPDATE BY LOCK** – データベースサーバは、結果セットのフェッチされたローに対する意図的ローロックを取得します。これは、トランザクションが COMMIT または ROLLBACK されるまで保持される長時間のロックです。
- **FOR UPDATE BY { VALUES | TIMESTAMP }** – データベースサーバは、キーセット駆動型カーソルを使用して、結果セットをスクロールしているときにローが変更または削除された場合にアプリケーションが通知されるようにします。

更新可能な文の制限
FOR UPDATE (*column-list*) を指定すると、後続の UPDATE WHERE CURRENT OF 文では指定された結果セットの属性のみ変更できるよう制限されます。

キャンセルされるカーソル操作

インタフェース機能で要求をキャンセルできます。カーソル操作実行要求をキャンセルした場合は、カーソルの位置は確定されません。要求をキャンセルしたら、カーソルを絶対位置によって見つけるか、カーソルを閉じます。

カーソルタイプ

この項では、SAP Sybase IQ のカーソルと、SAP Sybase IQ がサポートしているプログラミングインタフェースから利用できるオプションの間で行うマッピングについて説明します。

カーソルの可用性

すべてのインタフェースがすべてカーソルタイプをサポートするわけではありません。

- ADO.NET は、読み込み専用、前方専用のカーソルのみを提供します。
- ADO/OLE DB と ODBC では、すべてのカーソルタイプがサポートされています。
- Embedded SQL™ ではすべてのカーソルタイプがサポートされています。
- JDBC の場合：
  - SQL Anywhere JDBC ドライバでは、JDBC 4.0 仕様がサポートされており、insensitive、sensitive、forward-only asensitive カーソルの宣言が許可されています。
  - jConnect では、SQL Anywhere JDBC ドライバと同じく insensitive、sensitive、forward-only asensitive カーソルの宣言がサポートされています。ただし、jConnect の基本的な実装では、asensitive カーソルのセマンティックのみサポートされています。
- Sybase Open Client でサポートされているのは asensitive カーソルだけです。また、ユニークではない更新可能なカーソルを使用すると、パフォーマンスが著しく低下します。

カーソルのプロパティ

カーソルタイプは、プログラミングインタフェースから明示的または暗黙的に要求します。インタフェースライブラリが異なれば、使用できるカーソルタイプは

異なります。たとえば、JDBC と ODBC では使用できるカーソルタイプは異なります。

各カーソルタイプは、複数の特性によって定義されます。

- **一意性** – カーソルがユニークであることを宣言すると、クエリは、各ローをユニークに識別するために必要なすべてのカラムを返すように設定されます。これは、プライマリキー内にあるすべてのカラムを返すということをしばしば意味します。必要だが指定されないすべてのカラムは結果セットに追加されます。デフォルトでは、カーソルタイプは非ユニークです。
- **更新可能性** – 読み込み専用として宣言されたカーソルは、位置付け更新と位置付け削除のどちらの操作でも使用されません。デフォルトでは、更新可能のカーソルタイプに設定されています。
- **スクロール動作** – 結果セットを移動するときにカーソルが異なる動作をするように宣言できます。カーソルによっては、現在のローまたはその次のローしかフェッチできません。結果セットを後方に移動したり、前方に移動したりできるカーソルもあります。
- **感知性** – データベースに加えた変更を、カーソルを使用して表示／非表示にすることができます。

これらの特性に応じて、パフォーマンスやデータベースサーバでのメモリ使用量にかなりの影響をもたらすことがあります。

SAP Sybase IQ では、さまざまな特性を持つカーソルを使用できます。特定のタイプのカーソルを要求すると、SAP Sybase IQ は、その特性を一致させるよう試みます。

特性を全部指定できない場合もあります。たとえば、SAP Sybase IQ の insensitive カーソルは読み込み専用です。それは、更新可能な insensitive カーソルをアプリケーションが要求すると、代わりに、別のカーソルタイプ (value-sensitive カーソル) が指定されるからです。

ブックマークとカーソル

ODBC には*ブックマーク*があります。これはカーソル内のローの識別に使う値です。SAP Sybase IQ は、value-sensitive と insensitive カーソルにブックマークをサポートします。たとえば、ODBC カーソルタイプの SQL_CURSOR_STATIC と SQL_CURSOR_KEYSET_DRIVEN ではブックマークをサポートしますが、SQL_CURSOR_DYNAMIC と SQL_CURSOR_FORWARD_ONLY ではブックマークをサポートしていないということです。

ブロックカーソル

ODBC にはブロックカーソルと呼ばれるカーソルタイプがあります。ブロックカーソルを使うと、SQLFetchScroll または SQLExtendedFetch を使って単一のロー

ではなく、ローのブロックをフェッチできます。ブロックカーソルは Embedded SQL ARRAY フェッチと同じ動作をします。

SAP Sybase IQ カタログストアカーソル

SAP Sybase IQ カタログストアカーソルが開くと結果セットに関連付けられます。一度開いたカーソルは一定時間開いたままになります。カーソルが開いている間、カーソルに関連付けられた結果セットは変更される可能性があります。変更は、カーソル自体を使用して行われるか、独立性レベルの稼働条件に基づいて他のトランザクションで行われます。カーソルには、基本となるデータを表示できるように変更できるものと、変更を反映しないものがあります。基本となるデータの変更に対する感知性によって、カーソルの動作 (*カーソルの感知性*) は変わります。

SAP Sybase IQ カタログストアでは、感知性に関するさまざまな特性をカーソルに定義しています。この項では、まず感知性について説明し、次にカーソル感知性の特性について説明します。

メンバーシップ、順序、値の変更
基本となるデータに加えた変更は、カーソルの結果セットの次の部分に影響を及ぼします。

- **メンバーシップ** – 結果セットのローのセットです。プライマリキー値で指定されています。
- **順序** – 結果セットにあるローの順序です。
- **値** – 結果セットにあるローの値です。

たとえば、次のような従業員情報を記載した簡単なテーブルで考えてみます (EmployeeID はプライマリキーカラムです)。

EmployeeID	Surname
1	Whitney
2	Cobb
3	Chin

以下のクエリのカーソルは、プライマリキーの順序でテーブルからすべての結果を返します。

```
SELECT EmployeeID, Surname
FROM Employees
ORDER BY EmployeeID;
```

結果セットのメンバーシップは、ローを追加するか削除すると変更されます。値を変更するには、テーブル内の名前をどれか変更します。ある従業員のプライマリキー値を変更すると順序が変更される場合があります。

表示できる変更、表示できない変更
カーソルを開いた後、独立性レベルの稼働条件に基づいて、カーソルの結果セットのメンバーシップ、順序、値を変更できます。使用するカーソルタイプに応じて、これらの変更を反映するために、アプリケーションが表示する結果セットが変更されることも変更されないこともあります。

基本となるデータに加えた変更は、カーソルを使って*表示*または*非表示*にできます。表示できる変更とは、カーソルの結果セットに反映されている変更のことです。基本となるデータに加えた変更が、カーソルが表示する結果セットに反映されない場合は、非表示です。

カタログストアカーソルの感知性

SAP Sybase IQ のカーソルは、基本となるデータの変更に対する感知性に基づいて分類されています。いいかえると、カーソル感知性は、表示される変更内容によって定義されるものです。

- **insensitive カーソル –** カーソルが開いているとき、結果セットは固定です。基本となるデータに加えられた変更はすべて非表示です。
- **sensitive カーソル –** カーソルが開いた後に結果セットを変更できます。基本となるデータに加えられた変更内容はすべて表示されます。
- **asensitive カーソル –** 変更は、カーソルを使用して表示される結果セットのメンバーシップ、順序、または値に反映されます。
- **value-sensitive カーソル –** 基本となるデータの順序または値の変更は参照可能です。カーソルが開いているとき、結果セットのメンバーシップは固定です。

カーソルの稼働条件は異なるため、実行にさまざまな制約があり、そのため、パフォーマンスに影響します。

カーソルの感知性の例： 削除されたロー

この例では、簡単なクエリを使って、異なるカーソルが、削除中の結果セットのローに対してどのように応答するかを見ていきます。

次の一連のイベントを考えてみます。

1. アプリケーションが、次のようなサンプルデータベースに対するクエリについてカーソルを開く。

```
SELECT EmployeeID, Surname
FROM Employees
ORDER BY EmployeeID;
```

EmployeeID	Surname
102	Whitney

EmployeeID	Surname
105	Cobb
160	Breault
...	...

2. アプリケーションがカーソルを使って最初のローをフェッチする (102)。
3. アプリケーションがカーソルを使ってその次のローをフェッチする (105)。
4. 別のトランザクションが、employee 102 (Whitney) を削除して変更をコミットする。

この場合、カーソルアクションの結果は、カーソルの感知性によって異なります。

- **insensitive カーソル** – DELETE は、カーソルを使用して表示される結果セットのメンバーシップにも値にも反映されません。

アクション	結果
前のローをフェッチする	ローのオリジナルコピーを返す (102)
最初のローをフェッチする (絶対フェッチ)	ローのオリジナルコピーを返す (102)
2 番目のローをフェッチする (絶対フェッチ)	未変更のローを返す (105)

- **sensitive カーソル** – 結果セットのメンバーシップが変更されたため、ロー (105) は結果セットの最初のローになります。

アクション	結果
前のローをフェッチする	`Row Not Found` を返す。前のローが存在しない。
最初のローをフェッチする (絶対フェッチ)	ロー 105 を返す
2 番目のローをフェッチする (絶対フェッチ)	ロー 160 を返す

- **value-sensitive カーソル** – 結果セットのメンバーシップは固定であり、ロー 105 は、結果セットの 2 番目のローのままです。DELETE はカーソルの値に反映され、結果セットに有効なホールを作成します。

アクション	結果
前のローをフェッチする	`No current row of cursor` を返す。最初のローが以前存在したカーソルにホールがある。
最初のローをフェッチする (絶対フェッチ)	`No current row of cursor` を返す。最初のローが以前存在したカーソルにホールがある。

アクション	結果
2 番目のローをフェッチする (絶対フェッチ)	ロー 105 を返す

- **asensitive カーソル** – 変更に対して、結果セットのメンバーシップおよび値は確定されません。前のロー、最初のロー、または 2 番目のローのフェッチに対する応答は、特定のクエリ最適化方法によって異なります。また、その方法にワークテーブル構成が含まれているかどうか、フェッチ中のローがクライアントからプリフェッチされたものかどうかによっても異なります。

  多くのアプリケーションで感知性の重要度は高くはなく、その場合、asensitive カーソルは利点をもたらします。特に、前方専用や読み取り専用のカーソルを使用している場合は、基本となる変更は表示されません。また、高い独立性レベルで実行している場合は、基本となる変更は禁止されます。

カーソルの感知性の例： 更新されるロー

この例では、簡単なクエリを使って、順序が変更されるように現在更新されている結果セット内のローに対して、カーソルがどのように応答するかを見ていきます。

次の一連のイベントを考えてみます。

1. アプリケーションが、次のようなサンプルデータベースに対するクエリについてカーソルを開く。

```
SELECT EmployeeID, Surname
FROM Employees;
```

EmployeeID	Surname
102	Whitney
105	Cobb
160	Breault
...	...

2. アプリケーションがカーソルを使って最初のローをフェッチする (102)。
3. アプリケーションがカーソルを使ってその次のローをフェッチする (105)。
4. 別のトランザクションが employee 102 (Whitney) の従業員 ID を 165 に更新して変更をコミットする。

この場合、カーソルアクションの結果は、カーソルの感知性によって異なります。

- **insensitive カーソル** – UPDATE は、カーソルを使用して表示される結果セットのメンバーシップと値のどちらにも反映されません。

アクション	結果
前のローをフェッチする	ローのオリジナルコピーを返す (102)
最初のローをフェッチする (絶対フェッチ)	ローのオリジナルコピーを返す (102)
2 番目のローをフェッチする (絶対フェッチ)	未変更のローを返す (105)

- **sensitive カーソル** – 結果セットのメンバーシップが変更されたため、ロー (105) は結果セットの最初のローになります。

アクション	結果
前のローをフェッチする	SQLCODE 100 を返す。結果セットのメンバーシップは変更されたため、105 が最初のローになる。カーソルが最初のローの前の位置に移動する。
最初のローをフェッチする (絶対フェッチ)	ロー 105 を返す
2 番目のローをフェッチする (絶対フェッチ)	ロー 160 を返す

  また、sensitive カーソルでフェッチすると、ローが前回読み取られてから変更されている場合、SQLE_ROW_UPDATED_WARNING 警告が返されます。警告が出されるのは 1 回だけです。同じローを再びフェッチしても警告は発生しません。

  同様に、前回フェッチした後で、カーソルを使ってローを更新したり削除した場合には、SQLE_ROW_UPDATED_SINCE_READ エラーが返されます。sensitive カーソルで更新や削除を行うには、修正されたローをアプリケーションでもう一度フェッチします。

  カーソルによってカラムが参照されなくても、任意のカラムを更新すると警告やエラーの原因となります。たとえば、Surname を返すクエリにあるカーソルは、Salary カラムだけが修正されていても、更新をレポートします。

- **value-sensitive カーソル** – 結果セットのメンバーシップは固定であり、ロー 105 は、結果セットの 2 番目のローのままです。UPDATE はカーソルの値に反映され、結果セットに有効な「ホール」を作成します。

アクション	結果
前のローをフェッチする	SQLCODE 100 を返す。結果セットのメンバーシップは変更されたため、105 が最初のローになる。カーソルはホール上にある。つまり、ロー 105 の前にある。

アクション	結果
最初のローをフェッチする (絶対フェッチ)	SQLCODE -197 を返す。結果セットのメンバーシップは変更されたため、105 が最初のローになる。カーソルはホール上にある。つまり、ロー 105 の前にある。
2 番目のローをフェッチする (絶対フェッチ)	ロー 105 を返す

- **asensitive カーソル –** 変更に対して、結果セットのメンバーシップおよび値は確定されません。前のロー、最初のロー、または 2 番目のローのフェッチに対する応答は、特定のクエリ最適化方法によって異なります。また、その方法にワークテーブル構成が含まれているかどうか、フェッチ中のローがクライアントからプリフェッチされたものかどうかによっても異なります。

注意： 更新警告とエラーの状態はバルクオペレーションモード (-b データベースサーバオプション) では発生しません。

カタログストア insensitive カーソル

insensitive カーソルには、insensitive メンバーシップ、順序、値が指定されています。カーソルが開かれた後の変更は表示されません。

insensitive カーソルは、読み込み専用のカーソルタイプだけで使用されます。

規格
insensitive カーソルは、ISO/ANSI 規格の insensitive カーソル定義と ODBC の静的カーソルに対応しています。

プログラミングインタフェース

インタフェース	カーソルタイプ	コメント
ODBC、ADO/OLE DB	静的	更新可能な静的カーソルが要求された場合は、代わりに value-sensitive カーソルが使用される
Embedded SQL	INSENSITIVE	
JDBC	INSENSITIVE	insensitive セマンティックは、SQL Anywhere JDBC ドライバでのみサポートされる
Open Client	サポート対象外	

説明
insensitive カーソルは常に、クエリの選択基準に合ったローを、ORDER BY 句が指定した順序で返します。

カーソルが開かれている場合は、insensitive カーソルの結果セットがワークテーブルとして完全に実体化されます。その結果は次のようになります。

- 結果セットのサイズが大きい場合は、それを管理するためディスクスペースとメモリの要件が重要になる。
- 結果セットがワークテーブルとしてアセンブルされるより前にアプリケーションに返されるローはない。このため、複雑なクエリでは、最初のローがアプリケーションに返される前に遅れが生じることがある。
- 後続のローはワークテーブルから直接フェッチできるため、処理が早くなる。クライアントライブラリは 1 回に複数のローをプリフェッチできるため、パフォーマンスはさらに向上する。
- insensitive カーソルは、ROLLBACK または ROLLBACK TO SAVEPOINT には影響を受けない。

カタログストア sensitive カーソル

sensitive カーソルは、読み取り専用か更新可能なカーソルタイプで使用されます。

このカーソルには、sensitive なメンバーシップ、順序、値が指定されています。

規格
sensitive カーソルは、ISO/ANSI 規格の sensitive カーソル定義と ODBC の動的カーソルに対応しています。

プログラミングインタフェース

インタフェース	カーソルタイプ	コメント
ODBC、ADO/OLE DB	動的	
Embedded SQL	SENSITIVE	要求されているワークテーブルがなく、prefetch オプションが Off に設定されている場合は、DYNAMIC SCROLL カーソルの要求にも応じて提供される
JDBC	SENSITIVE	sensitive セマンティックは、SQL Anywhere JDBC ドライバで完全にサポートされる

説明
sensitive カーソルでのプリフェッチは無効です。カーソルを使用した変更や他のトランザクションからの変更など、変更はどれもカーソルを使用して表示できます。上位の独立性レベルでは、ロックを実行しなければならないという理由から、他のトランザクションで行われた変更のうち、一部が非表示になっている場合もあります。

カーソルのメンバーシップ、順序、すべてのカラム値に対して加えられた変更は、すべて表示されます。たとえば、sensitive カーソルにジョインが含まれており、基本となるテーブルの 1 つにある値がどれか 1 つでも修正されると、その基本のローで構成されたすべての結果ローには新しい値が表示されます。結果セットのメンバーシップと順序はフェッチのたびに変更できます。

sensitive カーソルは常に、クエリの選択基準に合ったローを、ORDER BY 句が指定した順序で返します。更新は、結果セットのメンバーシップ、順序、値に影響する場合があります。

sensitive カーソルを実装するときには、sensitive カーソルの稼働条件によって、次のような制限が加えられます。

- ローのプリフェッチはできない。プリフェッチされたローに加えた変更は、カーソルを介して表示されないからです。これは、パフォーマンスに影響を与えます。
- sensitive カーソルを実装する場合は、作成中のワークテーブルを使用しない。ワークテーブルとして保管されたローに加えた変更はカーソルを介して表示されないからです。
- ワークテーブルの制限事項では、オプティマイザによるジョインメソッドの選択を制限しない。これは、パフォーマンスに影響を及ぼす可能性があります。
- クエリによっては、カーソルを sensitive にするワークテーブルを含まないプランをオプティマイザが構成できない。
  通常、ワークテーブルは、中間結果をソートしたりグループ分けしたりするときに使用されます。インデックスからローにアクセスできる場合、ソートにワークテーブルは不要です。どのクエリがワークテーブルを使用するかを正確に述べることはできませんが、次のようなクエリでは必ずワークテーブルを使用します。
  - UNION クエリ。ただし、UNION ALL クエリでは必ずしもワークテーブルは使用されません。
  - ORDER BY 句を持つ文。ただし、ORDER BY カラムにはインデックスが存在しません。
  - ハッシュジョインを使って最適化されたクエリ全般。
  - DISTINCT 句または GROUP BY 句を必要とする多くのクエリ。

  この場合、SAP Sybase IQ は、アプリケーションにエラーを返すか、カーソルタイプを asensitive に変更して警告を返します。

カタログストア asensitive カーソル

asensitive カーソルには、メンバーシップ、順序、値に対する明確に定義された感知性はありません。感知性の持つ柔軟性によって、asensitive カーソルのパフォーマンスは最適化されます。

asensitive カーソルは、読み取り専用のカーソルタイプだけに使用されます。

規格
asensitive カーソルは、ISO/ANSI 規格で定めた asensitive カーソルの定義と、感知性について特別な指定のない ODBC カーソルに対応しています。

プログラミングインタフェース

インタフェース	カーソルタイプ
ODBC、ADO/OLE DB	感知性は未指定
Embedded SQL	DYNAMIC SCROLL

説明
SAP Sybase IQ がクエリを最適化してアプリケーションにローを返すときに使用する方法に対して、asensitive カーソルの要求では制約がほとんどありません。このため、asensitive カーソルを使うと最高のパフォーマンスを得られます。特に、オプティマイザは中間結果をワークテーブルとして実体化するというような措置をとる必要はありません。また、クライアントはローをプリフェッチできます。

SAP Sybase IQ では、基本のローに加えた変更の表示については保証されません。表示されるものと、されないものがあります。メンバーシップと順序はフェッチのたびに変わります。特に、基本のローを更新しても、カーソルの結果には、更新されたカラムの一部しか反映されないことがあります。

asensitive カーソルでは、クエリの選択内容と順序に一致するローを返すことは保証されません。ローのメンバーシップはカーソルが開いたときは固定ですが、その後加えられる基本の値への変更は結果に反映されます。

asensitive カーソルは常に、カーソルのメンバーシップが確立された時点で顧客の WHERE 句と ORDER BY 句に一致したローを返します。カーソルが開かれた後でカラム値が変わると、WHERE 句や ORDER BY 句に一致しないローは返される場合があります。

カタログストア value-sensitive カーソル

value-sensitive カーソルは、メンバーシップに対しては感知せず、結果セットの順序と値に対しては感知します。

value-sensitive カーソルは、読み取り専用か更新可能なカーソルタイプで使用されます。

規格
value-sensitive カーソルは、ISO/ANSI 規格の定義に対応していません。このカーソルは、ODBC キーセット駆動型カーソルに対応します。

プログラミングインタフェース

インタフェース	カーソルタイプ	コメント
ODBC、ADO/ OLE DB	キーセット駆動型	
Embedded SQL	SCROLL	
JDBC	INSENSITIVE と CONCUR_UPDATABLE	SQL Anywhere JDBC ドライバでは、更新可能な INSENSITIVE カーソルの要求は value-sensitive カーソルで応答される
Open Client と jConnect	サポートされていない	

説明
変更した基本のローで構成されているローをアプリケーションがフェッチすると、そのアプリケーションは更新された値を表示します。また、SQL_ROW_UPDATED ステータスがアプリケーションに発行されます。削除された基本のローで構成されているローをアプリケーションがフェッチした場合は、SQL_ROW_DELETED ステータスがアプリケーションに発行されます。

プライマリキー値に加えられた変更によって、結果セットからローが削除されます (削除として処理され、その後、挿入が続きます)。カーソルまたは外部から結果セットのローが削除されると、特別のケースが発生し、同じキー値を持つ新しいキーが挿入されます。この結果、新しいローと、それが表示されていた古いローが置き換えられます。

結果セットのローが、クエリの選択内容や順序指定に一致するという保証はありません。ローのメンバーシップは開かれた時に固定であるため、ローが変更されて WHERE 句または ORDER BY 句と一致しなくなっても、ローのメンバーシップと位置はいずれも変更されません。

どの値にも、カーソルを使用して行われた変更に対する感知性があります。カーソルを使用して行われた変更に対するメンバーシップの感知性は、ODBC オプションの SQL_STATIC_SENSITIVITY によって制御されます。このオプションが ON になっている場合は、カーソルを使った挿入によってそのカーソルにローが追加されます。それ以外の場合は、結果セットに挿入は含まれません。カーソルを使って削除すると、結果セットからローが削除され、SQL_ROW_DELETED ステータスを返すホールは回避されます。

value-sensitive カーソルは*キーセットテーブル*を使用します。カーソルが開かれている場合は、SAP Sybase IQ が、結果セットを構成する各ローの識別情報をワーク

テーブルに入力します。結果セットをスクロールする場合、結果セットのメンバーシップを識別するためにキーセットテーブルが使用されますが、値は必要に応じて基本のテーブルから取得されます。

value-sensitive カーソルのメンバーシッププロパティは固定であるため、アプリケーションはカーソル内のローの位置を記憶でき、これらの位置が変更されないことが保証されます。

- ローが更新されたか、カーソルが開かれた後に更新された可能性がある場合、SAP Sybase IQ は、ローがフェッチされた時点で SQLE_ROW_UPDATED_WARNING を返します。警告が出されるのは 1 回だけです。同じローを再びフェッチしても警告は発生しません。
  更新されたカラムがカーソルによって参照されていなくても、任意のカラムを更新すると警告の原因となります。たとえば、Surname と GivenName に対するカーソルは、Birthdate カラムだけが修正された場合でも更新の内容をレポートします。これらの更新警告とエラー条件は、バルクオペレーションモード (-b データベースサーバオプション) でローのロックが解除されている場合は発生しません。
- 前回フェッチした後に修正されたローで位置付け UPDATE 文または DELETE 文の実行を試みると、SQLE_ROW_UPDATED_SINCE_READ エラーが返されて、その文はキャンセルされます。アプリケーションでもう一度ローをフェッチすると UPDATE または DELETE が許可されます。
  更新されたカラムがカーソルによって参照されていなくても、任意のカラムを更新するとエラーの原因となります。バルクオペレーションモードでは、エラーは発生しません。
- カーソルが開かれた後にカーソルまたは別のトランザクションからローを削除した場合は、カーソルに*ホール*が作成されます。カーソルのメンバーシップは固定なので、ローの位置は予約されています。ただし、DELETE オペレーションは、変更されたローの値に反映されます。このホールでローをフェッチすると、現在のローがないことを示す -197 SQLCODE エラーが返され、カーソルはホールの上に配置されたままになります。sensitive カーソルを使用するとホールを回避できます。sensitive カーソルのメンバーシップは値とともに変化するからです。

value-sensitive カーソル用にローをプリフェッチすることはできません。この要件は、パフォーマンスに影響を及ぼすことがあります。

複数ローの挿入
複数のローを value-sensitive カーソルを介して挿入する場合、新しいローは結果セットの最後に表示されます。

カタログストアカーソルの感知性とパフォーマンス

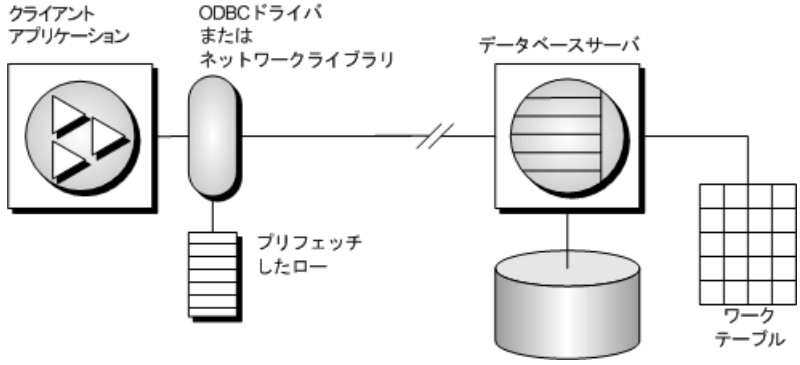
カーソルのパフォーマンスとその他のプロパティの間には、トレードオフ関係があります。特に、カーソルを更新できるようにした場合は、カーソルによるクエ

リの処理と配信で、パフォーマンスを制約する制限事項が課されます。また、カーソル感知性に稼働条件を設けると、カーソルのパフォーマンスが制約されることがあります。

カーソルの更新可能性と感知性がパフォーマンスに影響を与える仕組みを理解するには、カーソルによって表示される結果がどのようにしてデータベースからクライアントアプリケーションまで送信されるかを理解する必要があります。

特に、パフォーマンス上の理由から、結果が中間の 2 つのロケーションに格納されることを理解する必要があります。

- **ワークテーブル** – 中間結果または最終結果はワークテーブルとして保管されます。value-sensitive カーソルは、プライマリキー値のワークテーブルを使用します。また、クエリの特性によって、オプティマイザが選択した実行プランでワークテーブルを使用するようになります。
- **プリフェッチ** – クライアント側の通信はローを取り出してクライアント側のバッファに格納することで、データベースサーバに対するローごとの個別の要求を回避します。



感知性と更新可能性は中間のロケーションの使用を制限します。

プリフェッチ

プリフェッチは複数ローのフェッチとは異なります。プリフェッチはクライアントアプリケーションから明確な命令がなくても実行できます。プリフェッチはサーバからローを取り出し、クライアント側のバッファに格納しますが、クライアントアプリケーションがそれらのローを使用できるのは、アプリケーションが適切なローをフェッチしてからになります。

デフォルトでは、単一ローがアプリケーションによってフェッチされるたびに、SAP Sybase IQ のクライアントライブラリが複数のローをプリフェッチします。SAP Sybase IQ のクライアントライブラリは余分なローをバッファに格納します。

プリフェッチはクライアント／サーバのラウンドトリップを削減してパフォーマンスを高め、1 つのローやローのブロックごとにサーバへ個別に要求しないで多数のローを使用可能にすることによってスループットを高めます。

アプリケーションからのプリフェッチの制御

- prefetch オプションを使って、プリフェッチするかどうか制御できます。単一の接続では prefetch オプションを Always、Conditional、または Off に設定できます。デフォルトでは Conditional に設定されています。
- Embedded SQL では、BLOCK 句を使用して、カーソルを開くときにカーソルベースで、または個別の FETCH オペレーションで、プリフェッチを制御できます。
  アプリケーションでは、サーバから 1 つのフェッチに含められるローの最大数を、BLOCK 句で指定できます。たとえば、一度に 5 つのローをフェッチして表示する場合、BLOCK 5 を使用します。BLOCK 0 を指定すると、一度に 1 つのレコードがフェッチされ、常に FETCH RELATIVE 0 が同じローを再度フェッチするようになります。
  アプリケーションの接続パラメータを設定してフェッチをオフにすることもできますが、prefetch オプションを Off に設定するよりは、BLOCK 0 と指定する方が効果的です。
- value-sensitive カーソルタイプでは、プリフェッチはデフォルトで無効です。
- Open Client では、カーソルが宣言されてから開かれるまでの間に CS_CURSOR_ROWS で ct_cursor を使ってプリフェッチの動作を制御できます。

パフォーマンスが向上する可能性が高い場合に、プリフェッチするロー数が動的に増えます。これには、次の条件を満たすカーソルが含まれます。

- カーソルがサポートされるカーソルタイプのいずれかを使用している。
  - **ODBC と OLE DB –** 前方専用および読み込み専用 (デフォルト) のカーソル
  - **Embedded SQL –** DYNAMIC SCROLL (デフォルト)、NO SCROLL、および INSENSITIVE のカーソル
  - **ADO.NET –** すべてのカーソル
- カーソルが FETCH NEXT 操作のみ実行する (絶対フェッチ、相対フェッチ、後方フェッチは実行しない)。
- アプリケーションが、フェッチの間にホスト変数の型を変更したり、GET DATA 文を使用してチャンク単位でカラムのデータ取得を行ったりしない (GET DATA 文を 1 つ使用して、値を取得することはできる)。

更新内容の消失

更新可能なカーソルを使用する場合は、更新内容の消失から保護する必要があります。更新内容の消失は、2 つ以上のトランザクションが同じローを更新して、どのトランザクションも別のトランザクションによって変更されたことに気付かず、2 番目の変更が最初の変更内容を上書きしてしまう場合に生じます。このような問題について、次の例で説明します。

1. アプリケーションが、次のようなサンプルデータベースに対するクエリについてカーソルを開く。
```
SELECT ID, Quantity
FROM Products;
```

ID	Quantity
300	28
301	54
302	75
...	...

2. アプリケーションが、カーソルを介して ID = 300 のローをフェッチする。
3. 次の文を使用して別のトランザクションがローを更新する。
```
UPDATE Products
SET Quantity = Quantity - 10
WHERE ID = 300;
```
4. アプリケーションが、カーソルを使用してローを `(Quantity - 5)` の値に更新する。
5. 最終的な正しいロー値は 13 になる。カーソルによってローがプリフェッチされていた場合は、そのローの新しい値は 23 になる。別のトランザクションが更新した内容は失われる。

データベースアプリケーションでは、前もって値の検証を行わずにローの内容を変更すると、どの独立性レベルにおいても更新内容が消失する可能性があります。より高い独立性レベル (2 と 3) では、ロック (読み込み、意図的、書き込みロック) を使用して、アプリケーションでいったん読み込まれたローの内容を別のトランザクションが変更できないように設定できます。一方、独立性レベル 0 と 1 では、更新内容が消失する可能性が高くなります。独立性レベルが 0 の場合、データがその後変更されることを防ぐための読み込みロックは取得されません。独立性レベルが 1 の場合は、現在のローだけがロックされます。スナップショットアイソレーションを使用している場合、更新内容の消失は起こりません。これは、古い値を変更しようとすると必ず更新の競合が発生するからです。さらに、独立性レベル 1 でプリフェッチを使用した場合も更新内容が消失する可能性があります。これは、アプリケーションが位置設定されている結果セットロー (クライアントのプリフェッチバッファ内) は、サーバがカーソル内で位置設定されている現在のローとは異なる場合があるためです。

独立性レベルが 1 の場合にカーソルで更新内容が消失されるのを防ぐため、データベースサーバは、アプリケーションで指定可能な 3 種類の同時実行性制御メカニズムをサポートしています。

1. ローをフェッチするときに、カーソルの各ローに対する意図的ローロックの取得。意図的ロックを取得することで、他のトランザクションが同じローに対し

て意図的ロックや書き込みロックを取得できないようにし、同時更新の発生を防ぎます。ただし、意図的ロックでは読み込みローロックをブロックしないため、読み込み専用文の同時実行性には影響しません。

2. value-sensitive カーソルの使用。value-sensitive カーソルを使用して、基本となるローに対する変更や削除を追跡できるため、アプリケーションはそれに応じて応答できます。
3. FETCH FOR UPDATE の使用。特定のローに対する意図的ローロックを取得します。

これらのメカニズムの指定方法は、アプリケーションで使用されるインタフェースによって異なります。SELECT 文に関する最初の 2 つのメカニズムについては、次のようになります。

- ODBC では、アプリケーションで更新可能なカーソルを宣言するときに SQLSetStmtAttr 関数でカーソル同時実行性パラメータを指定する必要があるため、更新内容の消失は発生しません。このパラメータは、SQL_CONCUR_LOCK、SQL_CONCUR_VALUES、SQL_CONCUR_READ_ONLY、SQL_CONCUR_TIMESTAMP のいずれかです。SQL_CONCUR_LOCK を指定すると、データベースサーバはローに対する意図的ロックを取得します。SQL_CONCUR_VALUES と SQL_CONCUR_TIMESTAMP の場合は、value-sensitive カーソルが使用されます。SQL_CONCUR_READ_ONLY はデフォルトのパラメータで、読み込み専用カーソルに使用されます。
- JDBC では、文の同時実行性設定は ODBC の場合と似ています。SQL Anywhere JDBC ドライバでは、JDBC 同時実行性の値として RESULTSET_CONCUR_READ_ONLY と RESULTSET_CONCUR_UPDATABLE がサポートされています。最初の値は ODBC の同時実行性設定 SQL_CONCUR_READ_ONLY に対応し、読み込み専用文を指定します。2 番目の値は、ODBC の SQL_CONCUR_LOCK 設定に対応し、更新内容の消失を防ぐためにローの意図的ロックが使用されます。value-sensitive カーソルは、JDBC 4.0 仕様では直接指定できません。
- jConnect では、更新可能なカーソルは API レベルではサポートされますが、(TDS を使用する) 基本の実装ではカーソルを使用した更新はサポートされていません。その代わりに、jConnect では個別の UPDATE 文をデータベースサーバに送信して、特定のローを更新します。更新内容が失われないようにするには、アプリケーションを独立性レベル 2 以上で実行してください。アプリケーションはカーソルから個別の UPDATE 文を発行できますが、UPDATE 文の WHERE 句で条件を指定してローを読み込んだ後でロー値が変更されていないことを UPDATE 文で必ず確認するようにしてください。
- Embedded SQL では、同時実行性の指定は SELECT 文自体またはカーソル宣言に構文を含めることで設定できます。SELECT 文で構文 SELECT...FOR

UPDATE BY LOCK を使用すると、データベースサーバは結果セットに対する意図的ローロックを取得します。
または、SELECT...FOR UPDATE BY [VALUES | TIMESTAMP] を使用すると、データベースサーバはカーソルタイプを value-sensitive に変更するため、そのカーソルを使用して特定のローを最後に読み込んだ後でローが変更された場合、アプリケーションには FETCH 文に対する警告 (SQLE_ROW_UPDATED_WARNING)、または UPDATE WHERE CURRENT OF 文に対するエラー (SQLE_ROW_UPDATED_SINCE_READ) のいずれかが返されます。ローが削除されている場合も、アプリケーションにはエラー (SQLE_NO_CURRENT_ROW) が返されます。

FETCH FOR UPDATE 機能は Embedded SQL と ODBC インタフェースでもサポートされていますが、詳細は使用している API によって異なります。

Embedded SQL の場合、アプリケーションは FETCH の代わりに FETCH FOR UPDATE を使用してローに対する意図的ロックを取得します。ODBC の場合、アプリケーションは API 呼び出しの SQLSetPos を使用し、オペレーション引数 SQL_POSITION または SQL_REFRESH とロックタイプ引数 SQL_LOCK_EXCLUSIVE を指定して、ローに対する意図的ロックを取得します。SAP Sybase IQ の場合、このロックは、トランザクションがコミットまたはロールバックされるまで保持される長時間のロックです。

カタログストアカーソルの感知性と独立性レベル

カーソルの感知性と独立性レベルはどちらも同時実行性制御の問題を処理しますが、それぞれ方法や使用するトレードオフのセットが異なります。

トランザクションの独立性レベルを選択することで (通常は接続レベルで選択)、データベースのローに対するロックの種類とタイミングを設定します。ロックすると、他のトランザクションはデータベースのローにアクセスしたり修正したりできなくなります。通常、保持するロックの数が多くなるほど、同時に実行されているトランザクションにおける同時実行レベルは低くなると予期されます。

ただし、ローをロックしても、同じトランザクションの別の部分では更新が行われます。したがって、更新可能な複数のカーソルを保持する 1 つのトランザクションでは、ローをロックしたとしても、更新内容の消失などの現象が起こらないとは保証されません。

スナップショットアイソレーションは、各トランザクションでデータベースの一貫したビューを表示することで、読み込みロックの必要性を排除します。完全に直列化可能なトランザクション (独立性レベル 3) に依存せず、独立性レベル 3 を使用することで同時実行性を失うことなく、データベースの一貫したビューを問い合わせできるというのは大きな利点です。ただし、スナップショットアイソレーションの場合は、すでに実行中の同時実行のスナップショットトランザクションとまだ開始していないスナップショットトランザクションの両方の要件を満たす

ために修正されたローのコピーを保持する必要があるため、多大なコストがかかります。このようにコピーを保持する必要があるため、スナップショットアイソレーションの使用は更新を頻繁に行う負荷の高いトランザクションには適していない場合があります。

これに対して、カーソルの感知性は、カーソルの結果に対してどの変更を表示するか (または表示しないか) を決定します。カーソルの感知性はカーソルベースで指定するため、他のトランザクションと同じトランザクションの更新アクティビティの両方に影響しますが、影響度は指定されたカーソルタイプによって異なります。カーソルの感知性を設定しても、データベースのローをロックするタイミングを直接指定することにはなりません。ただし、カーソルの感知性と独立性レベルを組み合わせて使用することで、特定のアプリケーションで発生する可能性のある各種の同時実行シナリオを制御できます。

SAP Sybase IQ のカタログストアカーソルの要求

クライアントアプリケーションでカーソルタイプを要求すると、SAP Sybase IQ はカーソルを 1 つ返します。SAP Sybase IQ のカーソルは、プログラミングインタフェースで指定したカーソルタイプではなく、基本となるデータの変更に対する結果セットの感知性によって定義されます。SAP Sybase IQ は、要求されたカーソルタイプに基づいて、そのカーソルタイプに合う動作をカーソルに指定します。

クライアントがカーソルタイプを要求すると、SAP Sybase IQ はそれに答えてカーソル感知性を設定します。

ADO.NET

前方専用、読み込み専用のカーソルは、SACommand.ExecuteReader を利用して使用できます。SADataAdapter オブジェクトは、カーソルの代わりにクライアント側の結果セットを使用します。

ADO/OLE DB と ODBC

次の表は、スクロール可能な各種の ODBC カーソルタイプに応じて設定されるカーソル感知性を示します。

ODBC のスクロール可能なカーソルタイプ	SAP Sybase IQ のカーソル
STATIC	Insensitive
KEYSET-DRIVEN	Value-sensitive
DYNAMIC	Sensitive
MIXED	Value-sensitive

MIXED カーソルを取得するには、カーソルタイプを SQL_CURSOR_KEYSET_DRIVEN に指定し、SQL_ATTR_KEYSET_SIZE でキーセット駆動型カーソルのキーセット内のロー数を指定します。キーセットサイズ

が 0 (デフォルト) の場合、カーソルは完全にキーセット駆動型になります。キーセットサイズが 0 より大きい場合、カーソルは mixed (キーセット内はキーセット駆動型で、キーセット以外では動的) になります。デフォルトのキーセットサイズは 0 です。キーセットサイズが 0 より大きく、ローセットサイズ (SQL_ATTR_ROW_ARRAY_SIZE) より小さいとエラーになります。

例外
STATIC カーソルが更新可能なカーソルとして要求された場合は、代わりに value-sensitive カーソルが提供され、警告メッセージが発行されます。

DYNAMIC カーソルまたは MIXED カーソルが要求され、ワークテーブルを使用しなければクエリを実行できない場合、警告メッセージが発行され、代わりに asensitive カーソルが提供されます。

JDBC
JDBC 4.0 仕様では、insensitive、sensitive、forward-only asensitive の 3 つのカーソルタイプがサポートされています。SQL Anywhere JDBC ドライバはこれらの JDBC 仕様に準拠しており、JDBC ResultSet オブジェクトに対してこの 3 種類のカーソルタイプがサポートされています。ただし、データベースサーバが指定されたカーソルタイプに必要なセマンティックに基づいてアクセスプランを構築できない場合もあります。このような場合、データベースサーバはエラーを返すか、別のカーソルタイプに置き換えます。

jConnect の場合は、JDBC 2.0 仕様に従って別のタイプのカーソルを作成する場合は API をサポートしていますが、基本のプロトコル (TDS) ではデータベースサーバ上でサポートしているのは forward-only と read-only asensitive カーソルのみです。TDS プロトコルでは文の結果セットをブロック単位でバッファに格納するため、すべての jConnect カーソルは asensitive です。バッファに格納された結果のブロックは、スクロール動作がサポートされている insensitive または sensitive カーソルタイプを使用してアプリケーションでスクロールする必要がある場合に、スクロールされます。アプリケーションがキャッシュされた結果セットの先頭を越えて後方にスクロールすると、文は再実行されます。この場合、次の実行までにデータが変更されていると、データに矛盾が生じる可能性があります。

Embedded SQL
Embedded SQL アプリケーションからカーソルを要求するには、DECLARE 文にカーソルタイプを指定します。次の表は、各要求に応じて設定されるカーソル感知性を示しています。

カーソルタイプ	SAP Sybase IQ のカーソル
NO SCROLL	Asensitive
DYNAMIC SCROLL	Asensitive

カーソルタイプ	SAP Sybase IQ のカーソル
SCROLL	Value-sensitive
INSENSITIVE	Insensitive
SENSITIVE	Sensitive

例外
DYNAMIC SCROLL カーソルまたは NO SCROLL カーソルを UPDATABLE カーソルとして要求すると、sensitive または value-sensitive カーソルが返されます。どちらのカーソルが返されるかは保証されません。こうした不確定さは、asensitive の動作定義と矛盾しません。

INSENSITIVE カーソルが UPDATABLE (更新可能) として要求された場合は、value-sensitive カーソルが返されます。

DYNAMIC SCROLL カーソルが要求された場合、prefetch データベースオプションが Off に設定されている場合、クエリの実行プランにワークテーブルが使われない場合には、sensitive カーソルが返されます。ここでも、こうした不確定性は、asensitive の動作定義と矛盾しません。

Open Client
jConnect の場合と同様、Open Client の基本のプロトコル (TDS) は、forward-only、read-only、asensitive カーソルのみサポートしています。

結果セット記述子

アプリケーションによっては、アプリケーション内で完全に指定できない SQL 文を構築するものがあります。ユーザが表示するカラムを選択できるレポートアプリケーションなど、文がユーザからの応答に依存していて、ユーザの応答がないと、アプリケーションが検索する情報を正確に把握できない場合があります。

そのような場合、アプリケーションは、*結果セット*の性質と結果セットの内容の両方についての情報を検索する方法を必要とします。結果セットの性質についての情報を*記述子*と呼びます。記述子を用いて、返されるカラムの数や型を含むデータ構造体を識別します。アプリケーションが結果セットの性質を認識していると、内容の検索が簡単に行えます。

この*結果セットメタデータ* (データの性質と内容に関する情報) は記述子を使用して操作します。結果セットのメタデータを取得し、管理することを*記述*と呼びます。

通常はカーソルが結果セットを生成するので、記述子とカーソルは密接にリンクしています。ただし、記述子の使用をユーザに見えないように隠しているインタ

フェースもあります。通常、記述子を必要とする文は SELECT 文か、結果セットを返すストアドプロシージャのどちらかです。

カーソルベースの操作で記述子を使う手順は次のとおりです。

1. 記述子を割り付けます。インタフェースによっては明示的割り付けが認められているものもありますが、ここでは暗黙的に行います。
2. 文を準備します。
3. 文を記述します。文がストアドプロシージャの呼び出しかバッチであり、結果セットがプロシージャ定義において RESULT 句によって定義されていない場合、カーソルを開いてから記述を行います。
4. 文 (Embedded SQL) に対してカーソルを宣言して開くか、文を実行します。
5. 必要に応じて記述子を取得し、割り付けられた領域を修正します。多くの場合これは暗黙的に実行されます。
6. 文の結果をフェッチし、処理します。
7. 記述子の割り付けを解除します。
8. カーソルを閉じます。
9. 文を削除します。これはインタフェースによっては自動的に行われます。

実装の注意

- Embedded SQL では、SQLDA (SQL Descriptor Area) 構造体に記述子の情報があります。
- ODBC では、SQLAllocHandle を使って割り付けられた記述子ハンドルで記述子のフィールドへアクセスできます。SQLSetDescRec、SQLSetDescField、SQLGetDescRec、SQLGetDescField を使ってこのフィールドを操作できます。または、SQLDescribeCol と SQLColAttributes を使ってカラムの情報を取得することもできます。
- Open Client では、ct_dynamic を使って文を準備し、ct_describe を使って文の結果セットを記述します。ただし、ct_command を使って、SQL 文を最初に準備しないで送信し、ct_results を使って返されたローを 1 つずつ処理することもできます。これは Open Client アプリケーション開発を操作する場合に一般的な方法です。
- JDBC では、java.sql.ResultSetMetaData クラスが結果セットについての情報を提供します。
- INSERT 文などでは、記述子を使用してデータベースサーバにデータを送信することもできます。ただし、これは結果セットの記述子とは種類が異なります。

アプリケーション内のトランザクション

トランザクションはアトミックな SQL 文をまとめたものです。トランザクション内の文はすべて実行されるか、どれも実行されないかのどちらかです。この項ではアプリケーションのトランザクションの一面について説明します。

オートコミットまたは手動コミットモード

データベースプログラミングインタフェースは、*手動コミット*モードまたは*オートコミット*モードで操作できます。

- **手動コミットモード** – オペレーションがコミットされるのは、アプリケーションが明示的なコミットオペレーションを実行した場合、または ALTER TABLE 文やその他のデータ定義文を実行する場合などのように、データベースサーバがオートコミットを実行した場合だけです。手動コミットモードを*連鎖モード*とも呼びます。

  ネストされたトランザクションやセーブポイントなどのトランザクションをアプリケーションで使用するには、手動コミットモードで操作します。
- **オートコミットモード** – 文はそれぞれ、個別のトランザクションとして処理されます。これは、各 SQL 文の最後に COMMIT 文を付加して実行するのと同じ効果があります。オートコミットモードを*非連鎖モード*とも呼びます。

オートコミットモードは、使用中のアプリケーションのパフォーマンスや動作に影響することがあります。使用するアプリケーションでトランザクションの整合性が必要な場合は、オートコミットを使用しないでください。

オートコミットの動作を制御する方法

アプリケーションのコミット動作を制御する方法は、使用しているプログラミングインタフェースによって異なります。オートコミットの実装は、インタフェースに応じて、クライアント側またはサーバ側で行うことができます。

オートコミットモードの制御 (ADO.NET)

デフォルトでは、ADO.NET プロバイダはオートコミットモードで動作します。明示的トランザクションを使用するには、SAConnection.BeginTransaction メソッドを使用します。

オートコミットモードの制御 (OLE DB)

デフォルトでは、OLE DB プロバイダはオートコミットモードで動作します。明示的トランザクションを使用するには、ITransactionLocal::StartTransaction、ITransaction::Commit、ITransaction::Abort メソッドを使用します。

オートコミットモードの制御 (ODBC)
デフォルトでは、ODBC はオートコミットモードで動作します。オートコミットを OFF にする方法は、ODBC を直接使用しているか、アプリケーション開発ツールを使用しているかによって異なります。ODBC インタフェースに直接プログラミングしている場合には、SQL_ATTR_AUTOCOMMIT 接続属性を設定してください。

オートコミットモードの制御 (JDBC)
デフォルトでは、JDBC はオートコミットモードで動作します。オートコミットモードを OFF にするには、次に示すように、接続オブジェクトの setAutoCommit メソッドを使用します。

```
conn.setAutoCommit( false );
```

オートコミットモードの制御 (Embedded SQL)
デフォルトでは、Embedded SQL アプリケーションは手動コミットモードで動作します。オートコミットを ON にするには、次の文を実行して chained データベースオプション (サーバ側オプション) を Off に設定します。

```
SET OPTION chained='Off';
```

オートコミットモードの制御 (Open Client)
デフォルトでは、Open Client 経由で行われた接続はオートコミットモードで動作します。この動作を変更するには、次の文を使用して、作業中のアプリケーションで chained データベースオプション (サーバ側オプション) を On に設定します。

```
SET OPTION chained='On';
```

オートコミットモードの制御 (PHP)
デフォルトでは、PHP はオートコミットモードで動作します。オートコミットモードを OFF にするには、sasql_set_option 関数を使用します。

```
$result = sasql_set_option( $conn, "auto_commit", "Off" );
```

オートコミットモードの制御 (サーバの場合)
デフォルトでは、データベースサーバは手動コミットモードで動作します。オートコミットを ON にするには、次の文を実行して chained データベースオプション (サーバ側オプション) を Off に設定します。

```
SET OPTION chained='Off';
```

クライアント側でコミットを制御するインタフェースを使用している場合、chained データベースオプション (サーバ側オプション) がアプリケーションのパフォーマンスや動作に影響する場合があります。サーバの連鎖モードを設定することはおすすめしません。

オートコミット実装の詳細

オートコミットモードでは、使用するインタフェースやプロバイダ、またはオートコミット動作の制御方法に応じて、やや動作が異なります。

オートコミットモードは、次のいずれかの方法で実装できます。

- **クライアント側オートコミット** – アプリケーションがオートコミットを使用すると、各 SQL 文の実行後、クライアントライブラリが COMMIT 文を送信します。

  ADO.NET、ADO/OLE DB、ODBC、PHP、SQL Anywhere JDBC のドライバアプリケーションでは、クライアント側からコミットの動作を制御します。
- **サーバ側オートコミット** – アプリケーションで連鎖モードを OFF にすると、データベースサーバは各 SQL 文の結果をコミットします。Sybase jConnect JDBC ドライバの場合、この動作は chained データベースオプションによって制御されます。

  Embedded SQL、jConnect ドライバ、Open Client のアプリケーションでは、サーバ側でのコミット動作を操作します (たとえば、chained オプションを設定します)。

ストアドプロシージャやトリガなどの複雑な文では、クライアント側オートコミットとサーバ側オートコミットには違いがあります。クライアント側では、ストアドプロシージャは単一文であるため、オートコミットはプロシージャがすべて実行された後に単一のコミット文を送信します。データベースサーバ側から見た場合、ストアドプロシージャは複数の SQL 文で構成されているため、サーバ側オートコミットはプロシージャ内の各 SQL 文の結果をコミットします。

注意： クライアント側の実装とサーバ側の実装を混在させないでください。SAP Sybase IQ ADO.NET、OLE DB、ODBC、PHP、JDBC のアプリケーションでは、chained オプションとオートコミットオプションの設定を併用しないでください。

独立性レベルの設定

isolation_level データベースオプションを使って、現在の接続の独立性レベルを設定できます。

ODBC など、インタフェースによっては、接続時に接続の独立性レベルを設定できます。このレベルは isolation_level データベースオプションを使って、後でリセットできます。

INSERT、UPDATE、DELETE、SELECT、UNION の各文に OPTION 句を含めることによって、各文で指定したオプション設定を isolation_level データベースオプションに対するテンポラリ設定やパブリック設定よりも優先させることができます。

カーソルとトランザクション

一般的に、COMMIT が実行されると、カーソルは閉じられます。この動作には、2 つの例外があります。

- close_on_endtrans データベースオプションが Off に設定されている。
- カーソルが WITH HOLD で開かれている。Open Client と JDBC ではデフォルト。

この 2 つのどちらかが真の場合、カーソルは COMMIT 時に開いたままです。

ROLLBACK とカーソル

トランザクションがロールバックされた場合、WITH HOLD でオープンされたカーソルを除いて、カーソルは閉じられます。しかし、ロールバック後のカーソルの内容は、信頼性が高くありません。

ISO SQL3 標準の草案には、ロールバックについて、すべてのカーソルは (WITH HOLD でオープンされたカーソルも) 閉じられるべきだと述べられています。この動作は ansi_close_cursors_on_rollback オプションを On に設定して得られます。

セーブポイント

トランザクションがセーブポイントへロールバックされ、ansi_close_cursors_on_rollback オプションが On に設定されていると、SAVEPOINT 後に開かれたすべてのカーソルは (WITH HOLD でオープンされたカーソルも) 閉じられます。

カーソルと独立性レベル

トランザクションが SET OPTION 文を使って isolation_level オプションを変更する間、接続の独立性レベルを変更できます。ただし、この変更は開いているカーソルには反映されません。

WITH HOLD 句が snapshot、statement-snapshot、および readonly-statement-snapshot の各独立性レベルで使用されている場合、スナップショットの開始時にコミットされたすべてのローのスナップショットが表示されます。カーソルが開かれたトランザクションの開始以降の、現在の接続で完了された変更もすべて表示されます。

.NET アプリケーションプログラミング

この項では、.NET で SAP Sybase IQ を使用し、SAP Sybase IQ .NET データプロバイダの API を含める方法について説明します。

SAP Sybase IQ .NET データプロバイダ

この項では、Visual Studio プロジェクトでの SAP Sybase IQ .NET データプロバイダの使用、データベースとの接続、データベーステーブルからのローのフェッチ、挿入、更新、削除、ストアドプロシージャの呼び出し、トランザクションの使用、基本的なエラー処理に関するヒントなど、.NET サポートについて説明します。

SAP Sybase IQ .NET サポート

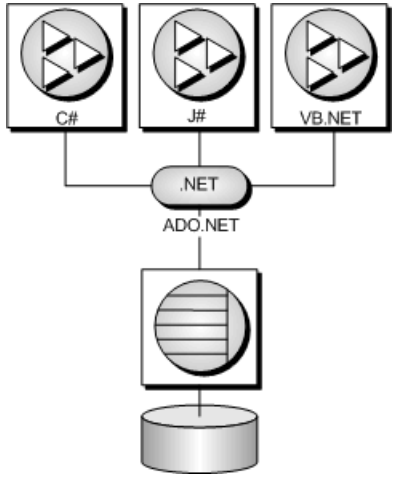
ADO.NET は、ODBC、OLE DB、ADO について Microsoft の最新のデータアクセス API です。ADO.NET は、Microsoft .NET Framework に適したデータアクセスコンポーネントであり、リレーショナルデータベースシステムにアクセスできます。

SAP Sybase IQ .NET データプロバイダは、iAnywhere.Data.SQLAnywhere ネームスペースを実装しており、.NET でサポートされている任意の言語 (C# や Visual Basic .NET など) でプログラムを作成したり、SAP Sybase IQ データベースからデータにアクセスしたりできます。

.NET データアクセスの概要については、Microsoft の「.NET データ アクセス アーキテクチャ ガイド」 (http://msdn.microsoft.com/en-us/library/Ee817654%28pandp.10%29.aspx) を参照してください。

ADO.NET アプリケーション
オブジェクト指向型言語を使用してインターネットおよびイントラネットのアプリケーションを開発し、ADO.NET データプロバイダを使用してアプリケーションを SAP Sybase IQ に接続できます。

ADO.NET データプロバイダに、組み込みの XML と Web サービス機能、MobiLink™ 同期用の .NET スクリプト機能、ハンドヘルドデータベースアプリケーション開発用の UltraLite.NET™ コンポーネントを組み合わせることで、SAP Sybase IQ を .NET フレームワークに統合できるようになります。



SAP Sybase IQ .NET データプロバイダの機能

SAP Sybase IQ は、3 つの異なるネームスペースを使用して Microsoft .NET Framework バージョン 2.0、3.0、3.5、4.0、4.5 をサポートしています。

- **iAnywhere.Data.SQLAnywhere –** ADO.NET オブジェクトモデルは、万能型のデータアクセスオブジェクトモデルです。ADO.NET コンポーネントは、データ操作によるデータアクセスを要素として組み込むよう設計されました。そのため、ADO.NET には DataSet と .NET Framework データプロバイダという 2 つの中心的なコンポーネントがあります。.NET Framework データプロバイダは、Connection、Command、DataReader、DataAdapter オブジェクトからなるコンポーネントのセットです。SAP Sybase IQ には、OLE DB または ODBC のオーバヘッドを加えずに SAP Sybase IQ データベースサーバと直接通信する .NET Entity Framework データプロバイダが含まれています。SAP Sybase IQ .NET データプロバイダは、.NET ネームスペースでは iAnywhere.Data.SQLAnywhere として表現されます。

  Microsoft .NET Compact Framework は、Microsoft .NET 用のスマートデバイス開発フレームワークです。SAP Sybase IQ .NET Compact Framework データプロバイダは、Windows Mobile が稼働しているデバイスをサポートしています。Compact Framework 2.0 および 3.5 がサポートされています。

  SAP Sybase IQ .NET データプロバイダのネームスペースについては、このマニュアルで説明します。

  ADO.NET オブジェクトモデルを使用して、特に、エンティティ手法への統合言語クエリ (LINQ) を介して、SAP Sybase IQ データベースの内部に格納されたデータにアクセスする方法の詳細については、www.sybase.com/detail?

id=1060541 の「SQL Anywhere ADO.NET Entity Framework のチュートリアル」を参照してください。

- **System.Data.Oledb –** このネームスペースは、OLE DB データソースをサポートしています。これは、Microsoft .NET Framework 固有の部分です。System.Data.Oledb を SQL Anywhere OLE DB プロバイダの SAOLEDB とともに使用して、SAP Sybase IQ データベースにアクセスできます。
- **System.Data.Odbc –** このネームスペースは、ODBC データソースをサポートしています。これは、Microsoft .NET Framework 固有の部分です。System.Data.Odbc を SQL Anywhere ODBC ドライバとともに使用して、SAP Sybase IQ データベースにアクセスできます。

SAP Sybase IQ .NET データプロバイダを使用する場合、次のような主な利点がいくつかあります。

- .NET 環境では、SAP Sybase IQ .NET データプロバイダは、SAP Sybase IQ データベースに対するネイティブアクセスを提供します。サポートされている他のプロバイダとは異なり、このデータプロバイダは SAP Sybase IQ サーバと直接通信を行うため、ブリッジテクノロジを必要としません。
- そのため、SAP Sybase IQ .NET データプロバイダは、OLE DB や ODBC のデータプロバイダより処理速度が高速です。SQL Anywhere データベースへのアクセスには SAP Sybase IQ .NET データプロバイダを使用することをおすすめします。

.NET サンプルプロジェクト

SAP Sybase IQ .NET データプロバイダには、複数のサンプルプロジェクトが用意されています。

- **LinqSample –** SAP Sybase IQ .NET データプロバイダと C# を使用して、統合言語クエリ、セット、変換操作を示す、Windows 用の .NET Framework サンプルプロジェクト。
- **SimpleWin32 –** [接続] をクリックしたときに Employees テーブルの名前が設定された簡単なリストボックスを示す、Windows 用の .NET Framework サンプルプロジェクト。
- **SimpleXML –** ADO.NET. を介して SAP Sybase IQ から XML データを取得する方法を示す、Windows 用の .NET Framework サンプルプロジェクト。C#、Visual Basic、Visual C++ のサンプルが用意されています。
- **SimpleViewer –** Windows 用の .NET Framework サンプルプロジェクト。
- **TableViewer –** SQL 文を入力および実行可能な、Windows 用の .NET Framework サンプルプロジェクト。

Visual Studio プロジェクトでの .NET データプロバイダの使用

Visual Studio で SAP Sybase IQ .NET データプロバイダを使用して .NET アプリケーションを開発するには、SAP Sybase IQ .NET データプロバイダへの参照と、ソースコードで SAP Sybase IQ .NET データプロバイダクラスを参照する行の両方を追加します。

前提条件

この作業を実行するための前提条件はありません。

手順

1. Visual Studio を起動し、プロジェクトを開きます。
2. [ソリューション エクスプローラー] ウィンドウで、[参照設定] を右クリックし、[参照の追加] をクリックします。

   参照によって、インクルードする必要があるプロバイダが示され、SAP Sybase IQ .NET データプロバイダのコードが検索されます。
3. [.NET] タブをクリックし、リストをスクロールして、次のいずれかを見つけます。

   iAnywhere.Data.SQLAnywhere for .NET 2
   iAnywhere.Data.SQLAnywhere for .NET 3.5
   iAnywhere.Data.SQLAnywhere for .NET 4
4. 必要なプロバイダをクリックし、[OK] をクリックします。

   プロジェクトの [ソリューション エクスプローラー] ウィンドウの [参照設定] フォルダにプロバイダが追加されます。
5. ソースコードにディレクティブを追加し、SAP Sybase IQ .NET データプロバイダのネームスペースと定義済みの型を簡単に使用できるようにします。

   次の行をプロジェクトに追加します。
   - C# を使用している場合、ソースコードの先頭にある `using` ディレクティブのリストに次の行を追加します。
     ```
     using iAnywhere.Data.SQLAnywhere;
     ```
   - Visual Basic を使用している場合、ソースコードの先頭に次の行を追加します。
     ```
     Imports iAnywhere.Data.SQLAnywhere
     ```

SAP Sybase IQ .NET データプロバイダが、.NET アプリケーションで使用できるように設定されます。

.NET データベースの接続例

データベースに接続するには、SAConnection オブジェクトを作成する必要があります。接続文字列はオブジェクトを作成する場合に指定するか、ConnectionString プロパティを設定して後で確立できます。

設計が良くできているアプリケーションは、データベースに接続しようとするときに発生するエラーを処理します。

データベースとの接続は、接続が開かれると作成され、接続が閉じられると解放されます。

C# の SAConnection の例

次の C# コードは、SAP Sybase IQ のサンプルデータベースとの接続を開いて閉じるボタンクリックハンドラを作成します。例外ハンドラが含まれています。

```
private void button1_Click(object sender, EventArgs e)
{
    SAConnection conn = new SAConnection("Data Source=Sybase IQ
Demo");
    try
    {
        conn.Open();

        conn.Close();
    }
    catch (SAException ex)
    {
        MessageBox.Show(ex.Errors[0].Source + " : " +
             ex.Errors[0].Message + " (" +
             ex.Errors[0].NativeError.ToString() + ")",
             "Failed to connect");
    }
}
```

Visual Basic SAConnection の例

次の Visual Basic コードは、SAP Sybase IQ のサンプルデータベースとの接続を開いて閉じるボタンクリックハンドラを作成します。例外ハンドラが含まれています。

```
Private Sub Button1_Click(ByVal sender As System.Object, _
     ByVal e As System.EventArgs) Handles Button1.Click
    Dim conn As New SAConnection("Data Source=Sybase IQ Demo")
    Try
        conn.Open()

        conn.Close()
    Catch ex As SAException
        MessageBox.Show(ex.Errors(0).Source & " : " & _
                 ex.Errors(0).Message & " (" & _
                 ex.Errors(0).NativeError.ToString() & ")", _
                 "Failed to connect")
```

```
    End Try
End Sub
```

接続プーリング

SAP Sybase IQ .NET データプロバイダは、ネイティブ .NET 接続プーリングをサポートしています。接続プーリングを使用すると、アプリケーションは、データベースへの新しい接続を繰り返し作成しなくても、接続ハンドルをプールに保存して再使用できるようにして、既存の接続を再使用できます。デフォルトでは、接続プーリングは有効になっています。

接続プーリングは、`Pooling` オプションを使用して有効または無効にします。最大プールサイズは、`Max Pool Size` オプションを使用して接続文字列に設定します。最小または初期プールサイズは、`Min Pool Size` オプションを使用して接続文字列に設定します。デフォルトの最大プールサイズは 100 で、デフォルトの最小プールサイズは 0 です。

```
"Data Source=Sybase IQ Demo;Pooling=true;Max Pool Size=50;Min Pool
Size=5"
```

アプリケーションは、最初にデータベースに接続しようとするときに、指定したものと同じ接続パラメータを使用する既存の接続があるかどうかプールを調べます。一致する接続がある場合は、その接続が使用されます。ない場合は、新しい接続が使用されます。接続を切断すると、接続がプールに戻されて再使用できるようになります。

SAP Sybase IQ データベースサーバも接続プーリングをサポートしています。この機能は、ConnectionPool (CPOOL) 接続パラメータを使用して制御します。ただし、SAP Sybase IQ .NET データプロバイダでは、このサーバ機能は使用されず、無効になります (CPOOL=NO)。代わりに、接続プーリングはすべて .NET クライアントアプリケーションで実行されます (クライアント側接続プーリング)。

接続状態

アプリケーションからデータベースへの接続が確立したら、接続がまだ開かれているかについて接続状態を確認してから、要求をデータベースサーバに送信できます。接続が閉じている場合、ユーザに適切なメッセージを返信するか、接続を開き直すように試みることができます。

SAConnection クラスには、接続状態の確認に使用できる State プロパティがあります。取り得る状態値は ConnectionState.Open と ConnectionState.Closed です。

次のコードは、SAConnection オブジェクトが初期化されているかどうかを確認し、初期化されている場合は、接続が開かれていることを確認します。接続が開かれていない場合は、ユーザにメッセージが返されます。

```
if ( conn == null || conn.State != ConnectionState.Open )
{
    MessageBox.Show( "Connect to a database first", "Not
```

```
connected" );
    return;
}
```

データへのアクセスとデータの操作

SAP Sybase IQ .NET データプロバイダでは、次の 2 つの方法でデータにアクセスできます。

- **SACommand オブジェクト** – .NET のデータにアクセスして操作する場合、SACommand オブジェクトを使用する方法をおすすめします。

  SACommand オブジェクトを使用して、データベースからデータを直接取得または修正する SQL 文を実行できます。SACommand オブジェクトを使用すると、データベースに対して直接 SQL 文を発行し、ストアドプロシージャを呼び出すことができます。

  SACommand オブジェクトでは、SADataReader を使用してクエリまたはストアドプロシージャから読み込み専用結果セットが返されます。SADataReader は 1 回に 1 つのローのみを返しますが、SAP Sybase IQ クライアント側のライブラリはプリフェッチバッファリングを使用して 1 回に複数のローをプリフェッチするため、これによってパフォーマンスが低下することはありません。

  SACommand オブジェクトを使用すると、オートコミットモードで操作しなくても、変更をトランザクションにグループ化できます。SATransaction オブジェクトを使用する場合、ローがロックされるため、他のユーザがこれらのローを修正できなくなります。
- **SADataAdapter オブジェクト** – SADataAdapter オブジェクトは、結果セット全体を DataSet に取り出します。DataSet は、データベースから取り出されたデータの、切断されたストアです。DataSet のデータは編集できます。編集が終了すると、SADataAdapter オブジェクトは、DataSet の変更内容に応じてデータベースを更新します。SADataAdapter を使用する場合、他のユーザによる DataSet 内のローの修正を禁止する方法はありません。このため、発生する可能性がある競合を解消するための論理をアプリケーションに構築する必要があります。

SADataAdapter オブジェクトとは異なり、SACommand オブジェクト内で SADataReader を使用してデータベースからローをフェッチする方法の場合、パフォーマンス上の影響はありません。

SACommand： ExecuteReader と ExecuteScalar を使用したデータのフェッチ

SACommand オブジェクトを使用して、SAP Sybase IQ データベースに対して SQL 文を実行したりストアドプロシージャを呼び出したりできます。次のいずれかのメソッドを使用して、データベースからデータを取り出すことができます。

- **ExecuteReader –** 結果セットを返す SQL クエリを発行します。このメソッドは、前方専用、読み込み専用のカーソルを使用します。結果セット内のローを 1 方向で簡単にループできます。
- **ExecuteScalar –** 単一の値を返す SQL クエリを発行します。これは、結果セットの最初のローの最初のカラムの場合や、COUNT または AVG などの集約値を返す SQL 文の場合があります。このメソッドは、前方専用、読み込み専用のカーソルを使用します。

SACommand オブジェクトを使用する場合、SADataReader を使用して、ジョインに基づく結果セットを取り出すことができます。ただし、変更 (挿入、更新、または削除) を行うことができるのは、単一テーブルのデータのみです。ジョインに基づく結果セットは更新できません。

SADataReader を使用する場合、指定したデータ型で結果を返すための Get メソッドが複数あります。

C# の ExecuteReader の例

次の C# コードは、SAP Sybase IQ サンプルデータベースへの接続を開き、ExecuteReader メソッドを使用して Employees テーブルの従業員の姓を含む結果セットを作成します。

```
SAConnection conn = new SAConnection("Data Source=Sybase IQ Demo");
conn.Open();
SACommand cmd = new SACommand("SELECT Surname FROM Employees", conn);
SADataReader reader = cmd.ExecuteReader();
listEmployees.BeginUpdate();
while (reader.Read())
{
    listEmployees.Items.Add(reader.GetString(0));
}
listEmployees.EndUpdate();
reader.Close();
conn.Close();
```

Visual Basic の ExecuteReader の例

次の Visual Basic コードは、SAP Sybase IQ サンプルデータベースへの接続を開き、ExecuteReader メソッドを使用して Employees テーブルの従業員の姓を含む結果セットを作成します。

```
Dim conn As New SAConnection("Data Source=Sybase IQ Demo")
Dim cmd As New SACommand("SELECT Surname FROM Employees", conn)
Dim reader As SADataReader
conn.Open()
```

```
reader = cmd.ExecuteReader()
ListEmployees.BeginUpdate()
Do While (reader.Read())
    ListEmployees.Items.Add(reader.GetString(0))
Loop
ListEmployees.EndUpdate()
conn.Close()
```

C# の ExecuteScalar の例

次の C# コードは、SAP Sybase IQ サンプルデータベースへの接続を開き、ExecuteScalar メソッドを使用して Employees テーブルの男性従業員の人数を取得します。

```
SAConnection conn = new SAConnection("Data Source=Sybase IQ Demo");
conn.Open();
SACommand cmd = new SACommand(
    "SELECT COUNT(*) FROM Employees WHERE Sex = 'M'", conn );
int count = (int) cmd.ExecuteScalar();
textBox1.Text = count.ToString();
conn.Close();
```

SACommand： GetSchemaTable を使用した結果セットのスキーマのフェッチ

結果セット内のカラムに関するスキーマ情報を取得できます。

SADataReader クラスの GetSchemaTable メソッドは、現在の結果セットに関する情報を取得します。GetSchemaTable メソッドは、標準 .NET DataTable オブジェクトを返します。このオブジェクトは、結果セット内のすべてのカラムに関する情報 (カラムプロパティを含む) を提供します。

C# のスキーマ情報の例

次の例は、GetSchemaTable メソッドを使用して結果セットに関する情報を取得し、DataTable オブジェクトを画面上のデータグリッドにバインドします。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SACommand cmd = new SACommand("SELECT * FROM Employees", conn);
SADataReader reader = cmd.ExecuteReader();
DataTable schema = reader.GetSchemaTable();
reader.Close();
conn.Close();
dataGridView1.DataSource = schema;
```

SACommand： ExecuteNonQuery を使用したローの挿入、削除、更新

SACommand オブジェクトを使用してローを挿入、更新、削除するには、ExecuteNonQuery 関数を使用します。ExecuteNonQuery 関数は、結果セットを返さないクエリ (SQL 文またはストアドプロシージャ) を発行します。

変更 (挿入、更新、または削除) を行うことができるのは、単一テーブルのデータのみです。ジョインに基づく結果セットは更新できません。SACommand オブジェクトを使用するには、データベースに接続してください。

SQL 文の独立性レベルを設定するには、SACommand オブジェクトを SATransaction オブジェクトの一部として使用します。SATransaction オブジェクトを使用しないでデータを修正すると、プロバイダはオートコミットモードで動作し、実行した変更内容は即座に適用されます。

C# の ExecuteNonQuery DELETE および INSERT の例

次の例は、SAP Sybase IQ サンプルデータベースへの接続を開き、ExecuteNonQuery メソッドを使用して、ID が 600 以上の部署をすべて削除し、2 つの新しいローを Departments テーブルに追加します。そして、更新されたテーブルをデータグリッドに表示します。

```
SAConnection conn = new SAConnection("Data Source=Sybase IQ Demo");
conn.Open();

SACommand deleteCmd = new SACommand(
    "DELETE FROM Departments WHERE DepartmentID >= 600",
    conn);
deleteCmd.ExecuteNonQuery();

SACommand insertCmd = new SACommand(
    "INSERT INTO Departments(DepartmentID, DepartmentName)
VALUES( ?, ? )",
    conn );
SAParameter parm = new SAParameter();
parm.SADbType = SADbType.Integer;
insertCmd.Parameters.Add( parm );
parm = new SAParameter();
parm.SADbType = SADbType.Char;
insertCmd.Parameters.Add( parm );

insertCmd.Parameters[0].Value = 600;
insertCmd.Parameters[1].Value = "Eastern Sales";
int recordsAffected = insertCmd.ExecuteNonQuery();

insertCmd.Parameters[0].Value = 700;
insertCmd.Parameters[1].Value = "Western Sales";
recordsAffected = insertCmd.ExecuteNonQuery();

SACommand selectCmd = new SACommand(
    "SELECT * FROM Departments", conn );
```

```
SADataReader dr = selectCmd.ExecuteReader();

System.Windows.Forms.DataGrid dataGrid;
dataGrid = new System.Windows.Forms.DataGrid();
dataGrid.Location = new Point(15, 50);
dataGrid.Size = new Size(275, 200);
dataGrid.CaptionText = "SACommand Example";
this.Controls.Add(dataGrid);

dataGrid.DataSource = dr;
dr.Close();
conn.Close();
```

C# の ExecuteNonQuery UPDATE の例

次の例は、SAP Sybase IQ サンプルデータベースへの接続を開き、ExecuteNonQuery メソッドを使用して、DepartmentID が 100 である Departments テーブルのすべてのローで DepartmentName カラムを "Engineering" に更新します。そして、更新されたテーブルをデータグリッドに表示します。

```
SAConnection conn = new SAConnection("Data Source=Sybase IQ Demo");
conn.Open();

SACommand updateCmd = new SACommand(
    "UPDATE Departments SET DepartmentName = 'Engineering' " +
    "WHERE DepartmentID = 100", conn );
int recordsAffected = updateCmd.ExecuteNonQuery();

SACommand selectCmd = new SACommand(
    "SELECT * FROM Departments", conn );
SADataReader dr = selectCmd.ExecuteReader();

System.Windows.Forms.DataGrid dataGrid;
dataGrid = new System.Windows.Forms.DataGrid();
dataGrid.Location = new Point(15, 50);
dataGrid.Size = new Size(275, 200);
dataGrid.CaptionText = "SACommand Example";
this.Controls.Add(dataGrid);

dataGrid.DataSource = dr;
dr.Close();
conn.Close();
```

SACommand： 新しく挿入したローのプライマリキー値の取得

更新するテーブルにオートインクリメントプライマリキーがある場合は、UUID を使用します。また、プライマリキーがプライマリキープールのものである場合は、ストアドプロシージャを使用して、データソースによって生成されたプライマリキー値を取得できます。

C# の SACommand プライマリキーの例

次の例は、新しく挿入されたローに対して生成されるプライマリキーの取得方法を示します。この例では、SACommand オブジェクトを使用して SQL ストアドプ

ロシージャを呼び出し、返されるプライマリキーを SAParameter オブジェクトを使用して取得します。デモンストレーションのため、この例ではサンプルテーブル (adodotnet_primarykey) とストアドプロシージャ (sp_adodotnet_primarykey) を作成して、ローの挿入とプライマリ値の取得に使用します。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();

SACommand cmd = conn.CreateCommand();

cmd.CommandText = "DROP TABLE adodotnet_primarykey";
cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE TABLE IF NOT EXISTS adodotnet_primarykey ("
+
    "ID INTEGER DEFAULT AUTOINCREMENT, " +
    "Name CHAR(40) )";
cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE or REPLACE PROCEDURE
sp_adodotnet_primarykey(" +
    "out p_id int, in p_name char(40) )" +
    "BEGIN " +
    "INSERT INTO adodotnet_primarykey( name ) VALUES( p_name );" +
    "SELECT @@IDENTITY INTO p_id;" +
    "END";
cmd.ExecuteNonQuery();

cmd.CommandText = "sp_adodotnet_primarykey";
cmd.CommandType = CommandType.StoredProcedure;

SAParameter parmId = new SAParameter();
parmId.SADbType = SADbType.Integer;
parmId.Direction = ParameterDirection.Output;
cmd.Parameters.Add(parmId);

SAParameter parmName = new SAParameter();
parmName.SADbType = SADbType.Char;
parmName.Direction = ParameterDirection.Input;
cmd.Parameters.Add(parmName);

parmName.Value = "R & D --- Command";
cmd.ExecuteNonQuery();
int id1 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id1);

parmName.Value = "Marketing --- Command";
cmd.ExecuteNonQuery();
int id2 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id2);

parmName.Value = "Sales --- Command";
cmd.ExecuteNonQuery();
int id3 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id3);
```

```
parmName.Value = "Shipping --- Command";
cmd.ExecuteNonQuery();
int id4 = (int)parmId.Value;
System.Console.WriteLine("Primary key=" + id4);

cmd.CommandText = "SELECT * FROM adodotnet_primarykey";
cmd.CommandType = CommandType.Text;
SADataReader dr = cmd.ExecuteReader();
conn.Close();
dataGridView1.DataSource = dr;
```

SADataAdapter： 概要

SADataAdapter オブジェクトは、結果セットを DataTable に取り出します。DataSet は、テーブルのコレクション (DataTables) と、これらのテーブル間の関係と制約です。DataSet は、.NET Framework に組み込まれており、データベースへの接続に使用されるデータプロバイダとは関係ありません。

SADataAdapter を使用する場合、DataTable を設定し、DataTable の変更内容を使用してデータベースを更新するためにデータベースに接続されている必要があります。ただし、DataTable を一度設定すれば、データベースと切断されていても DataTable を修正できます。

変更内容をデータベースに即座に適用したくない場合、WriteXml メソッドを使用して DataSet (データかスキーマまたはその両方を含む) を XML ファイルに書き込むことができます。これによって、後で ReadXml メソッドを使用して DataSet をロードして変更を適用できるようになります。以下では 2 つの例を示します。

```
ds.WriteXml("Employees.xml");
ds.WriteXml("EmployeesWithSchema.xml", XmlWriteMode.WriteSchema);
```

詳細については、.NET Framework のマニュアルの WriteXml と ReadXml を参照してください。

Update メソッドを呼び出して変更を DataSet からデータベースに適用すると、SADataAdapter は、実行された変更を分析してから、必要に応じて適切な文 (INSERT、UPDATE、または DELETE) を呼び出します。DataSet を使用する場合、変更 (挿入、更新、または削除) を行うことができるのは、単一テーブルのデータのみです。ジョインに基づく結果セットは更新できません。更新しようとしているローを別のユーザがロックしている場合、例外がスローされます。

警告！ DataSet を変更できるのは、接続が切断されている場合のみです。データベース内のこれらのローはアプリケーションによってロックされません。DataSet の変更がデータベースに適用されるときに発生する可能性がある競合を解消できるようアプリケーションを設計してください。これは、自分の変更がデータベースに適用される前に自分が修正しているデータを別のユーザが変更しようとするような場合です。

SADataAdapter を使用するときの競合の解消

SADataAdapter オブジェクトを使用する場合、データベース内のローはロックされません。つまり、DataSet からデータベースに変更を適用するときに競合が発生する可能性があります。このため、アプリケーションには、発生する競合を解消または記録する論理を採用する必要があります。

アプリケーション論理が対応すべき競合には、次のようなものがあります。

- **ユニークなプライマリキー** – 2 人のユーザが新しいローをテーブルに挿入する場合、ローごとにユニークなプライマリキーが必要です。AUTOINCREMENT プライマリキーがあるテーブルの場合、DataSet の値とデータソースの値の同期がとれなくなる可能性があります。
- **同じ値に対して行われた更新** – 2 人のユーザが同じ値を修正する場合、どちらの値が正しいかを確認する論理をアプリケーションに採用する必要があります。
- **スキーマの変更** – DataSet で更新したテーブルのスキーマを別のユーザが修正する場合、データベースに変更を適用するとこの更新が失敗します。
- **データの同時実行性** – 同時実行アプリケーションは、一連の一貫性のあるデータを参照する必要があります。SADataAdapter はフェッチするローをロックしないため、いったん DataSet を取り出してからオフラインで処理する場合、別のユーザがデータベース内の値を更新できます。

これらの潜在的な問題の多くは、SACommand、SADataReader、SATransaction オブジェクトを使用して変更をデータベースに適用することによって回避できます。このうち、SATransaction オブジェクトを使用することをおすすめします。これは、SATransaction オブジェクトを使用すると、トランザクションに独立性レベルを設定できるほか、他のユーザが修正できないようにローをロックできるためです。

競合の解消プロセスを簡素化するために、INSERT、UPDATE、DELETE 文をストアドプロシージャ呼び出しとして設定できます。INSERT、UPDATE、DELETE 文をストアドプロシージャに入れることによって、オペレーションが失敗したときのエラーを取得できます。文のほかにも、エラー処理論理をストアドプロシージャに追加することによって、オペレーションが失敗したときに、エラーをログファイルに記録したりオペレーションを再試行したりするなど、適切なアクションが行われるようにすることができます。

SADataAdapter： Fill を使用したデータの DataTable へのフェッチ

SADataAdapter を使用すると、Fill メソッドを使用して DataTable を表示グリッドにバインドすることによってクエリの結果を DataTable に設定し、結果セットを表示できます。

SADataAdapter を設定する場合、結果セットを返す SQL 文を指定できます。Fill を呼び出して DataTable を設定する場合、前方専用、読み込み専用のカーソルを使用

してすべてのローが 1 回のオペレーションでフェッチされます。結果セット内のすべてのローが読み込まれると、カーソルは閉じます。DataTable の行に行われた変更は、Update メソッドを使用してデータベースに反映できます。

SADataAdapter オブジェクトを使用して、ジョインに基づく結果セットを取り出すことができます。ただし、変更 (挿入、更新、または削除) を行うことができるのは、単一テーブルのデータのみです。ジョインに基づく結果セットは更新できません。

警告！ DataTable に対して行う変更は、元のデータベーステーブルとは別に行われます。データベース内のこれらのローはアプリケーションによってロックされません。DataTable の変更がデータベースに適用されるときに発生する可能性がある競合を解消できるようアプリケーションを設計してください。これは、自分の変更がデータベースに適用される前に自分が修正しているデータを別のユーザが変更しようとするような場合です。

DataTable を使用した C# の SADataAdapter Fill の例

次の例は、SADataAdapter を使用して DataTable を設定する方法を示します。Results という新しい DataTable オブジェクトと、新しい SADataAdapter オブジェクトを作成します。SADataAdapter の Fill メソッドを使用して、クエリの結果を DataTable に設定します。そして、DataTable を画面上のグリッドにバインドします。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
DataTable dt = new DataTable("Results");
SADataAdapter da = new SADataAdapter("SELECT * FROM Employees",
conn);
da.Fill(dt);
conn.Close();
dataGridView1.DataSource = dt;
```

DataSet を使用した C# の SADataAdapter Fill の例

次の例は、SADataAdapter を使用して DataTable を設定する方法を示します。新しい DataSet オブジェクトと、新しい SADataAdapter オブジェクトを作成します。SADataAdapter の Fill メソッドを使用して、Results という DataTable テーブルを DataSet 内に作成した後、クエリの結果を設定します。そして、Results DataTable を画面上のグリッドにバインドします。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
DataSet ds = new DataSet();
SADataAdapter da = new SADataAdapter("SELECT * FROM Employees",
conn);
da.Fill(ds, "Results");
conn.Close();
dataGridView1.DataSource = ds.Tables["Results"];
```

SADataAdapter： FillSchema を使用した DataTable のフォーマット

SADataAdapter を使用すると、FillSchema メソッドを使用して、DataTable のスキーマが特定のクエリのスキーマと一致するように設定できます。DataTable のカラムの属性は、SADataAdapter オブジェクトの SelectCommand と一致します。Fill メソッドとは異なり、DataTable に行は保存されません。

DataTable を使用した C# の SADataAdapter FillSchema の例

次の例は、FillSchema メソッドを使用して結果セットと同じスキーマを持つ新しい DataTable オブジェクトを設定する方法を示します。そして、Additions DataTable を画面上のグリッドにバインドします。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SADataAdapter da = new SADataAdapter("SELECT * FROM Employees",
conn);
DataTable dt = new DataTable("Additions");
da.FillSchema(dt, SchemaType.Source);
conn.Close();
dataGridView1.DataSource = dt;
```

DataSet を使用した C# の SADataAdapter FillSchema の例

次の例は、FillSchema メソッドを使用して結果セットと同じスキーマを持つ新しい DataTable オブジェクトを設定する方法を示します。Merge メソッドを使用して DataTable を DataSet に追加します。そして、Additions DataTable を画面上のグリッドにバインドします。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SADataAdapter da = new SADataAdapter("SELECT * FROM Employees",
conn);
DataTable dt = new DataTable("Additions");
da.FillSchema(dt, SchemaType.Source);
DataSet ds = new DataSet();
ds.Merge(dt);
conn.Close();
dataGridView1.DataSource = ds.Tables["Additions"];
```

SADataAdapter： Update を使用したローの挿入

SADataAdapter の Update メソッドを使用してテーブルにローを追加する方法の例です。

C# の SADataAdapter Insert の例

この例では、SADataAdapter の SelectCommand プロパティと Fill メソッドを使用して、Departments テーブルを DataTable にフェッチします。次に、2 つの新しいローを DataTable に追加し、SADataAdapter の InsertCommand プロパティと Update メソッドを使用して、DataTable からの Departments テーブルを更新します。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SACommand deleteCmd = new SACommand(
    "DELETE FROM Departments WHERE DepartmentID >= 600", conn);
deleteCmd.ExecuteNonQuery();

SADataAdapter da = new SADataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.Add;
da.SelectCommand = new SACommand(
    "SELECT * FROM Departments", conn );
da.InsertCommand = new SACommand(
    "INSERT INTO Departments( DepartmentID, DepartmentName ) " +
    "VALUES( ?, ? )", conn );
da.InsertCommand.UpdatedRowSource =
    UpdateRowSource.None;

SAParameter parm = new SAParameter();
parm.SADbType = SADbType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add( parm );

parm = new SAParameter();
parm.SADbType = SADbType.Char;
parm.SourceColumn = "DepartmentName";
parm.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add( parm );

DataTable dataTable = new DataTable( "Departments" );
int rowCount = da.Fill( dataTable );

DataRow row1 = dataTable.NewRow();
row1[0] = 600;
row1[1] = "Eastern Sales";
dataTable.Rows.Add( row1 );

DataRow row2 = dataTable.NewRow();
row2[0] = 700;
row2[1] = "Western Sales";
dataTable.Rows.Add( row2 );

rowCount = da.Update( dataTable );

dataTable.Clear();
rowCount = da.Fill( dataTable );
conn.Close();
dataGridView1.DataSource = dataTable;
```

SADataAdapter： Update を使用したローの削除

SADataAdapter の Update メソッドを使用してテーブルからローを削除する方法の例です。

C# の SADataAdapter Delete の例

この例では、2 つの新しいローを Departments テーブルに追加した後、SADataAdapter の SelectCommand プロパティと Fill メソッドを使用して、Departments テーブルを DataTable にフェッチします。次に、DataTable からいくつかのローを削除し、SADataAdapter の DeleteCommand プロパティと Update メソッドを使用して、DataTable からの Departments テーブルを更新します。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SACommand prepCmd = new SACommand("", conn);
prepCmd.CommandText =
    "DELETE FROM Departments WHERE DepartmentID >= 600";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO Departments VALUES (600, 'Eastern Sales', 902)";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO Departments VALUES (700, 'Western Sales', 902)";
prepCmd.ExecuteNonQuery();

SADataAdapter da = new SADataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.AddWithKey;
da.SelectCommand = new SACommand(
    "SELECT * FROM Departments", conn);
da.DeleteCommand = new SACommand(
    "DELETE FROM Departments WHERE DepartmentID = ?",
    conn);
da.DeleteCommand.UpdatedRowSource = UpdateRowSource.None;

SAParameter parm = new SAParameter();
parm.SADbType = SADbType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Original;
da.DeleteCommand.Parameters.Add(parm);

DataTable dataTable = new DataTable("Departments");
int rowCount = da.Fill(dataTable);

foreach (DataRow row in dataTable.Rows)
{
    if (Int32.Parse(row[0].ToString()) > 500)
    {
        row.Delete();
    }
}
rowCount = da.Update(dataTable);
```

```
dataTable.Clear();
rowCount = da.Fill(dataTable);
conn.Close();
dataGridView1.DataSource = dataTable;
```

SADataAdapter： Update を使用したローの更新

SADataAdapter の Update メソッドを使用してテーブル内のローを更新する方法の例です。

C# の SADataAdapter Update の例

この例では、2 つの新しいローを Departments テーブルに追加した後、SADataAdapter の SelectCommand プロパティと Fill メソッドを使用して、Departments テーブルを DataTable にフェッチします。次に、DataTable の値の一部を変更し、SADataAdapter の UpdateCommand プロパティと Update メソッドを使用して、DataTable からの Departments テーブルを更新します。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SACommand prepCmd = new SACommand("", conn);
prepCmd.CommandText =
    "DELETE FROM Departments WHERE DepartmentID >= 600";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO Departments VALUES (600, 'Eastern Sales', 902)";
prepCmd.ExecuteNonQuery();
prepCmd.CommandText =
    "INSERT INTO Departments VALUES (700, 'Western Sales', 902)";
prepCmd.ExecuteNonQuery();

SADataAdapter da = new SADataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.Add;
da.SelectCommand = new SACommand(
    "SELECT * FROM Departments", conn );
da.UpdateCommand = new SACommand(
    "UPDATE Departments SET DepartmentName = ? " +
    "WHERE DepartmentID = ?",
    conn );
da.UpdateCommand.UpdatedRowSource = UpdateRowSource.None;

SAParameter parm = new SAParameter();
parm.SADbType = SADbType.Char;
parm.SourceColumn = "DepartmentName";
parm.SourceVersion = DataRowVersion.Current;
da.UpdateCommand.Parameters.Add( parm );

parm = new SAParameter();
parm.SADbType = SADbType.Integer;
parm.SourceColumn = "DepartmentID";
parm.SourceVersion = DataRowVersion.Original;
da.UpdateCommand.Parameters.Add( parm );
```

```
DataTable dataTable = new DataTable( "Departments" );
int rowCount = da.Fill( dataTable );

foreach ( DataRow row in dataTable.Rows )
{
    if (Int32.Parse(row[0].ToString()) > 500)
    {
        row[1] = (string)row[1] + "_Updated";
    }
}
rowCount = da.Update( dataTable );

dataTable.Clear();
rowCount = da.Fill( dataTable );
conn.Close();
dataGridView1.DataSource = dataTable;
```

SADataAdapter： 新しく挿入したローのプライマリキー値の取得

更新するテーブルにオートインクリメントプライマリキーがある場合は、UUID を使用します。また、プライマリキーがプライマリキープールのものである場合は、ストアドプロシージャを使用して、データソースによって生成されたプライマリキー値を取得できます。

C# の SADataAdapter プライマリキーの例

次の例は、新しく挿入されたローに対して生成されるプライマリキーの取得方法を示します。この例では、SADataAdapter オブジェクトを使用して SQL ストアドプロシージャを呼び出し、返されるプライマリキーを SAParameter オブジェクトを使用して取得します。デモンストレーションのため、この例ではサンプルテーブル (adodotnet_primarykey) とストアドプロシージャ (sp_adodotnet_primarykey) を作成して、ローの挿入とプライマリ値の取得に使用します。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();

SACommand cmd = conn.CreateCommand();

cmd.CommandText = "DROP TABLE adodotnet_primarykey";
cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE TABLE IF NOT EXISTS adodotnet_primarykey ("
+
    "ID INTEGER DEFAULT AUTOINCREMENT, " +
    "Name CHAR(40) )";
cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE or REPLACE PROCEDURE
sp_adodotnet_primarykey(" +
    "out p_id int, in p_name char(40) )" +
    "BEGIN " +
    "INSERT INTO adodotnet_primarykey( name ) VALUES( p_name );" +
    "SELECT @@IDENTITY INTO p_id;" +
```

```
    "END";
cmd.ExecuteNonQuery();

SADataAdapter da = new SADataAdapter();
da.MissingMappingAction = MissingMappingAction.Passthrough;
da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

da.SelectCommand = new SACommand(
    "SELECT * FROM adodotnet_primarykey", conn);

da.InsertCommand = new SACommand(
    "sp_adodotnet_primarykey", conn);
da.InsertCommand.CommandType = CommandType.StoredProcedure;
da.InsertCommand.UpdatedRowSource =
UpdateRowSource.OutputParameters;

SAParameter parmId = new SAParameter();
parmId.SADbType = SADbType.Integer;
parmId.Direction = ParameterDirection.Output;
parmId.SourceColumn = "ID";
parmId.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add(parmId);

SAParameter parmName = new SAParameter();
parmName.SADbType = SADbType.Char;
parmName.Direction = ParameterDirection.Input;
parmName.SourceColumn = "Name";
parmName.SourceVersion = DataRowVersion.Current;
da.InsertCommand.Parameters.Add(parmName);

DataTable dataTable = new DataTable("Departments");
da.FillSchema(dataTable, SchemaType.Source);

DataRow row = dataTable.NewRow();
row[0] = -1;
row[1] = "R & D --- Adapter";
dataTable.Rows.Add(row);

row = dataTable.NewRow();
row[0] = -2;
row[1] = "Marketing --- Adapter";
dataTable.Rows.Add(row);

row = dataTable.NewRow();
row[0] = -3;
row[1] = "Sales --- Adapter";
dataTable.Rows.Add(row);

row = dataTable.NewRow();
row[0] = -4;
row[1] = "Shipping --- Adapter";
dataTable.Rows.Add(row);

DataSet ds = new DataSet();
ds.Merge(dataTable);
da.Update(ds, "Departments");
```

```
conn.Close();
dataGridView1.DataSource = ds.Tables["Departments"];
```

BLOB

長い文字列値またはバイナリデータをフェッチする場合、データを分割してフェッチするメソッドがいくつかあります。バイナリデータの場合は GetBytes メソッド、文字列データの場合は GetChars メソッドを使用します。それ以外の場合、データベースからフェッチする他のデータと同じ方法で BLOB データが処理されます。

C# の GetChars BLOB の例

次の例は、結果セットから 3 つのカラムを読み込みます。最初の 2 つのカラムは整数で、3 番目のカラムは LONG VARCHAR です。GetChars メソッドを使用して 3 番目のカラムを 100 文字のチャンクに読み込み、このカラムの長さを計算します。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SACommand cmd = new SACommand("SELECT * FROM MarketingInformation",
conn);
SADataReader reader = cmd.ExecuteReader();

int idValue;
int productIdValue;
int length = 100;
char[] buf = new char[length];
while (reader.Read())
{
    idValue = reader.GetInt32(0);
    productIdValue = reader.GetInt32(1);
    long blobLength = 0;
    long charsRead;
    while ((charsRead = reader.GetChars(2, blobLength, buf, 0,
length))
            == (long)length)
    {
        blobLength += charsRead;
    }
    blobLength += charsRead;
}
reader.Close();
conn.Close();
```

時間値

.NET Framework には Time 構造体はありません。SAP Sybase IQ から時間値をフェッチするには、GetTimeSpan メソッドを使用します。このメソッドは、データを .NET Framework TimeSpan オブジェクトとして返します。

C# の TimeSpan の例

次の例は、時間を TimeSpan として返す GetTimeSpan メソッドを使用します。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
SACommand cmd = new SACommand("SELECT 123, CURRENT TIME", conn);
SADataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
    int ID = reader.GetInt32(0);
    TimeSpan time = reader.GetTimeSpan(1);
}
reader.Close();
conn.Close();
```

ストアドプロシージャ

SAP Sybase IQ .NET データプロバイダでは SQL ストアドプロシージャを使用できます。

ExecuteReader メソッドを使用して、結果セットを返すストアドプロシージャを呼び出します。また、ExecuteNonQuery メソッドを使用して、結果セットを返さないストアドプロシージャを呼び出します。ExecuteScalar メソッドを使用して、単一値のみを返すストアドプロシージャを呼び出します。

SAParameter オブジェクトを追加してパラメータをストアドプロシージャに渡すことができます。

C# のパラメータ付きストアドプロシージャ呼び出しの例

次の例は、ストアドプロシージャを呼び出してパラメータを渡す 2 つの方法を示します。この例では、ストアドプロシージャから返される結果セットを SADataReader を使用してフェッチします。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
bool method1 = true;

SACommand cmd = new SACommand("", conn);
if (method1)
{
    cmd.CommandText = "ShowProductInfo";
    cmd.CommandType = CommandType.StoredProcedure;
}
else
```

```
{
    cmd.CommandText = "call ShowProductInfo(?)";
    cmd.CommandType = CommandType.Text;
}

SAParameter param = cmd.CreateParameter();
param.SADbType = SADbType.Integer;
param.Direction = ParameterDirection.Input;
param.Value = 301;
cmd.Parameters.Add(param);

SADataReader reader = cmd.ExecuteReader();
reader.Read();
int ID = reader.GetInt32(0);
string name = reader.GetString(1);
string description = reader.GetString(2);
decimal price = reader.GetDecimal(6);
reader.Close();

listBox1.BeginUpdate();
listBox1.Items.Add("Name=" + name +
    " Description=" + description + " Price=" + price);
listBox1.EndUpdate();

conn.Close();
```

トランザクション処理

SAP Sybase IQ .NET データプロバイダでは、SATransaction オブジェクトを使用して文をグループ化できます。各トランザクションは COMMIT または ROLLBACK で終了します。これらは、データベースの変更内容を確定したり、トランザクションのすべてのオペレーションをキャンセルしたりします。トランザクションが完了したら、さらに変更を行うための SATransaction オブジェクトを新しく作成する必要があります。この動作は、COMMIT または ROLLBACK を実行した後もトランザクションが閉じられるまで持続する ODBC や Embedded SQL とは異なります。

トランザクションを作成しない場合、デフォルトでは、SAP Sybase IQ .NET データプロバイダはオートコミットモードで動作します。挿入、更新、または削除の各処理後には COMMIT が暗黙的に実行され、オペレーションが完了すると、データベースが変更されます。この場合、変更はロールバックできません。

トランザクションの独立性レベルの設定

デフォルトでは、トランザクションに対してデータベースの独立性レベルが使用されます。トランザクションを開始するときに IsolationLevel プロパティを使用してトランザクションに対して独立性レベルを指定できます。独立性レベルは、トランザクション内で実行されるすべての文に対して適用されます。SQL Anywhere .NET データプロバイダは、スナップショットアイソレーションをサポートしています。

SQL 文を実行するときに SAP Sybase IQ で使用されるロックは、トランザクションの独立性レベルによって異なります。

分散トランザクション処理
.NET 2.0 フレームワークで、トランザクションアプリケーションを記述するためのクラスが含まれる新しいネームスペース System.Transactions が導入されました。クライアントアプリケーションで 1 つまたは複数の参加者が存在する分散トランザクションを作成し、そのトランザクションに参加できます。クライアントアプリケーションでは、TransactionScope クラスを使用して、暗黙的にトランザクションを作成できます。接続オブジェクトでは、TransactionScope によって作成されたアンビエントトランザクションの存在を検出し、自動的にエンリストできます。クライアントアプリケーションでは、CommittableTransaction を作成し、EnlistTransaction メソッドを呼び出してエンリストすることもできます。この機能は SAP Sybase IQ .NET データプロバイダでサポートされています。分散トランザクションには、大きなパフォーマンスのオーバヘッドがあります。非分散トランザクションにデータベーストランザクションを使用することをおすすめします。

C# の SATransaction の例
次の例は、コミットやロールバックができるように INSERT をトランザクションにラップする方法を示します。SATransaction オブジェクトを使用してトランザクションを作成し、SACommand オブジェクトを使用して SQL 文の実行にリンクします。独立性レベル 2 (RepeatableRead) を指定して、他のデータベースユーザがローを更新できないようにします。トランザクションがコミットまたはロールバックされると、ローのロックは解放されます。トランザクションを使用しない場合、SAP Sybase IQ .NET データプロバイダはオートコミットモードで動作し、データベースの変更内容をロールバックできません。

```
SAConnection conn = new SAConnection( "Data Source=Sybase IQ Demo" );
conn.Open();
string stmt = "UPDATE Products SET UnitPrice = 2000.00 " +
    "WHERE Name = 'Tee shirt'";
bool goAhead = false;

SATransaction trans =
conn.BeginTransaction(SAIsolationLevel.RepeatableRead);
SACommand cmd = new SACommand(stmt, conn, trans);
int rowsAffected = cmd.ExecuteNonQuery();
if (goAhead)
    trans.Commit();
else
    trans.Rollback();
conn.Close();
```

エラー処理

アプリケーションは発生するあらゆるエラーを処理するように設計する必要があります。

SAP Sybase IQ .NET データプロバイダは、実行時にエラーが発生した場合はいつでも SAException オブジェクトを作成して例外をスローします。各 SAException オブジェクトは SAError オブジェクトのリストから成り、これらのエラーオブジェクトにはエラーメッセージとコードが含まれます。

エラーは競合とは異なります。競合は、データベースに変更が適用されたときに発生します。このため、アプリケーションには、競合が発生したときに正しい値を計算したり競合のログを取ったりするプロセスを採用する必要があります。

C# のエラー処理の例

次の C# コードは、SAP Sybase IQ のサンプルデータベースとの接続を開くボタンクリックハンドラを作成します。接続を確立できない場合、例外ハンドラにより 1 つ以上のメッセージが表示されます。

```
private void button1_Click(object sender, EventArgs e)
{
    SAConnection conn = new SAConnection("Data Source=Sybase IQ
Demo");
    try
    {
        conn.Open();
    }
    catch (SAException ex)
    {
        for (int i = 0; i < ex.Errors.Count; i++)
        {
            MessageBox.Show(ex.Errors[i].Source + " : " +
             ex.Errors[i].Message + " (" +
             ex.Errors[i].NativeError.ToString() + ")",
             "Failed to connect");
        }
    }
}
```

Visual Basic のエラー処理の例

次の Visual Basic コードは、SAP Sybase IQ のサンプルデータベースとの接続を開くボタンクリックハンドラを作成します。接続を確立できない場合、例外ハンドラにより 1 つ以上のメッセージが表示されます。

```
Private Sub Button1_Click(ByVal sender As System.Object, _
     ByVal e As System.EventArgs) Handles Button1.Click
    Dim conn As New SAConnection("Data Source=Sybase IQ Demo")
    Try
        conn.Open()
    Catch ex As SAException
```

```
        For i = 0 To ex.Errors.Count - 1
            MessageBox.Show(ex.Errors(i).Source & " : " & _
              ex.Errors(i).Message & " (" & _
              ex.Errors(i).NativeError.ToString() & ")", _
              "Failed to connect")
        Next i
    End Try
End Sub
```

Entity Framework のサポート

SAP Sybase IQ .NET データプロバイダでは Entity Framework 4.3 がサポートされています。Entity Framework は Microsoft から別個のパッケージとして入手できます。Entity Framework 4.3 を使用するには、Microsoft の NuGet Package Manager を使用して Visual Studio に追加する必要があります。

Entity Framework の新機能の 1 つに Code First があります。この機能では異なる開発ワークフローが可能です。C# または VB.NET のクラスを記述するだけでデータモデルが定義され、データベースオブジェクトにマッピングされます。デザイナを開く必要も XML マッピングファイルを定義する必要もありません。または、データの注釈や Fluent API を使用して追加の設定を実行することもできます。モデルを使用して、データベーススキーマを生成したり既存のデータベースにマッピングしたりできます。

次の例は、このモデルを使用して新しいデータベースオブジェクトを作成します。

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using iAnywhere.Data.SQLAnywhere;

namespace CodeFirstExample
{
    [Table( "EdmCategories", Schema = "DBA" )]
    public class Category
    {
        public string CategoryID { get; set; }
        [MaxLength( 64 )]
        public string Name { get; set; }

        public virtual ICollection<Product> Products { get; set; }
    }

    [Table( "EdmProducts", Schema = "DBA" )]
    public class Product
    {
        public int ProductId { get; set; }
        [MaxLength( 64 )]
        public string Name { get; set; }
```

```
        public string CategoryID { get; set; }

        public virtual Category Category { get; set; }
    }

    [Table( "EdmSuppliers", Schema = "DBA" )]
    public class Supplier
    {
        [Key]
        public string SupplierCode { get; set; }
        [MaxLength( 64 )]
        public string Name { get; set; }
    }

    public class Context : DbContext
    {
        public Context() : base() { }
        public Context( string connStr ) : base( connStr ) { }

        public DbSet<Category> Categories { get; set; }
        public DbSet<Product> Products { get; set; }
        public DbSet<Supplier> Suppliers { get; set; }

        protected override void OnModelCreating( DbModelBuilder
modelBuilder )
        {
            modelBuilder.Entity<Supplier>().Property( s =>
s.Name ).IsRequired();
        }
    }

    class Program
    {
        static void Main( string[] args )
        {
            Database.DefaultConnectionFactory = new
SAConnectionFactory();
            Database.SetInitializer<Context>( new
DropCreateDatabaseAlways<Context>() );

            using ( var db = new Context( "DSN=Sybase IQ Demo" ) )
            {
                var query = db.Products.ToList();
            }
        }
    }
}
```

この例を構築して実行するには、次のアセンブリ参照を追加する必要があります。

```
EntityFramework
iAnywhere.Data.SQLAnywhere.v4.0
System.ComponentModel.DataAnnotations
System.Data.Entity
```

次の例は、既存のデータベースにマッピングします。

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using iAnywhere.Data.SQLAnywhere;

namespace CodeFirstExample
{
    [Table( "Customers", Schema = "GROUPO" )]
    public class Customer
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string CompanyName { get; set; }

        public virtual ICollection<Contact> Contacts { get; set; }
    }

    [Table( "Contacts", Schema = "GROUPO" )]
    public class Contact
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Title { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string Fax { get; set; }

        [ForeignKey( "Customer" )]
        public int CustomerID { get; set; }
        public virtual Customer Customer { get; set; }
    }

    public class Context : DbContext
    {
        public Context() : base() { }
        public Context( string connStr ) : base( connStr ) { }

        public DbSet<Contact> Contacts { get; set; }
        public DbSet<Customer> Customers { get; set; }
```

```
    }

    class Program
    {
        static void Main( string[] args )
        {
            Database.DefaultConnectionFactory = new
SAConnectionFactory();
            Database.SetInitializer<Context>( null );

            using ( var db = new Context( "DSN=SAP Sybase IQ 16
Demo" ) )
            {
                foreach ( var customer in db.Customers.ToList() )
                {
                  Console.WriteLine( "Customer - " + string.Format(
                      "{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8},
{9}",
                        customer.ID, customer.SurName,
customer.GivenName,
                    customer.Street, customer.City, customer.State,
                        customer.Country, customer.PostalCode,
                        customer.Phone, customer.CompanyName ) );

                    foreach ( var contact in customer.Contacts )
                    {
                        Console.WriteLine( "    Contact - " +
string.Format(
                       "{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8},
{9}, {10}",
                            contact.ID, contact.SurName,
contact.GivenName,
                            contact.Title,
                            contact.Street, contact.City,
contact.State,
                            contact.Country, contact.PostalCode,
                            contact.Phone, contact.Fax ) );
                    }
                }
            }
        }
    }
}
```

Microsoft .NET Framework Data Provider for SQL Server (SqlClient) と SAP Sybase IQ .NET データプロバイダとでは、実装の詳細部分でいくつかの違いがあり、それらを理解しておく必要があります。

1. 新しいクラス SAConnectionFactory (IDbConnectionFactory を実装) が追加されています。次のように、Database.DefaultConnectionFactory に SAConnectionFactory のインスタンスを設定してから、データモデルを作成します。
   ```
   Database.DefaultConnectionFactory = new SAConnectionFactory();
   ```

2. Entity Framework の Code First では、原則として規則によるコーディングを行います。Entity Framework では、規則をコーディングすることによってデータモデルが推測されます。また、自動的な処理が多数行われます。場合によっては、開発者が Entity Framework のすべての規則を把握できないこともあります。しかし、SAP Sybase IQ で意味を持たないコード規則もあります。SQL Server と SAP Sybase IQ の間には大きな違いがあります。SQL Server のインスタンスはすべて複数のデータベースを保持しますが、SAP Sybase IQ のデータベースはすべて単一のファイルです。
   - ユーザがパラメータのないコンストラクタを使用してユーザ定義の DbContext を作成した場合、SqlClient では統合セキュリティを使用してローカルコンピュータの SQL Server Express に接続します。SAP Sybase IQ プロバイダでは、ユーザがすでにログインマッピングを作成している場合、統合ログインを使用してデフォルトのサーバに接続します。
   - SqlClient では、Entity Framework から DbDeleteDatabase または DbCreateDatabase を呼び出したときに、既存のデータベースを削除して新しいデータベースを作成します (SQL Server Express Edition のみ)。SAP Sybase IQ プロバイダではデータベースの削除や作成をすることは決してなく、データベースオブジェクト (テーブル、関係、制約など) を作成または削除します。ユーザは最初にデータベースを作成する必要があります。
   - IDbConnectionFactory.CreateConnection メソッドでは、文字列パラメータ "nameOrConnectionString" がデータベース名 (SQL Server では初期カタログ) または接続文字列として扱われます。ユーザが DbContext の接続文字列を指定しない場合、SqlClient では、ユーザ定義の DbContext クラスのネームスペースを初期カタログとして使用し、ローカルコンピュータの SQL Express サーバに自動的に接続します。SAP Sybase IQ では、このパラメータに接続文字列のみ含めることができます。データベース名は無視され、代わりに統合ログインが使用されます。
3. SQL Server SqlClient API では、データ注釈属性 TimeStamp を持つカラムが、SQL Server のデータ型 timestamp/rowversion にマッピングされます。SQL Server の timestamp/rowversion については開発者の間で誤解されている面があります。SQL Server の timestamp/rowversion データ型は、SAP Sybase IQ や他のほとんどのデータベースベンダーとは異なります。
   - SQL Server の timestamp/rowversion はバイナリ (8) です。日付と時刻の組み合わせ値はサポートされていません。SAP Sybase IQ でサポートされている timestamp というデータ型は、SQL Server の datetime データ型と同等です。
   - SQL Server の timestamp/rowversion 値はユニークであることが保証されています。SAP Sybase IQ の timestamp 値はユニークではありません。
   - SQL Server の timestamp/rowversion 値は、ローが更新されるたびに変更されます。

TimeStamp データ注釈属性は SAP Sybase IQ プロバイダではサポートされていません。

4. Entity Framework 4.1 のデフォルトでは、スキーマ名または所有者名が常に dbo に設定されます。これは SQL Server のデフォルトのスキーマです。ただし、dbo は SAP Sybase IQ では適切ではありません。SAP Sybase IQ では、データ注釈か Fluent API のどちらかを使用して、テーブル名を持つスキーマ名 (GROUPO など) を指定する必要があります。次に例を示します。

```
namespace CodeFirstTest
{
    public class Customer
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string CompanyName { get; set; }

       public virtual ICollection<Contact> Contacts { get; set; }
    }

    public class Contact
    {
        [Key()]
        public int ID { get; set; }
        public string SurName { get; set; }
        public string GivenName { get; set; }
        public string Title { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
        public string Phone { get; set; }
        public string Fax { get; set; }

        [ForeignKey( "Customer" )]
        public int CustomerID { get; set; }
        public virtual Customer Customer { get; set; }
    }

    [Table( "Departments", Schema = "GROUPO" )]
    public class Department
    {
        [Key()]
        public int DepartmentID { get; set; }
        public string DepartmentName { get; set; }
```

```
        public int DepartmentHeadID { get; set; }
    }

    public class Context : DbContext
    {
        public Context() : base() { }
        public Context( string connStr ) : base( connStr ) { }

        public DbSet<Contact> Contacts { get; set; }
        public DbSet<Customer> Customers { get; set; }
        public DbSet<Department> Departments { get; set; }

        protected override void OnModelCreating( DbModelBuilder
modelBuilder )
        {
            modelBuilder.Entity<Contact>().ToTable( "Contacts",
"GROUPO" );
            modelBuilder.Entity<Customer>().ToTable( "Customers",
"GROUPO" );
        }
    }
}
```

SAP Sybase IQ .NET データプロバイダの配備

次の各項では、SAP Sybase IQ .NET データプロバイダを配備する方法について説明します。

SAP Sybase IQ .NET データプロバイダシステムの稼働条件

SAP Sybase IQ .NET データプロバイダを使用するには、コンピュータまたはハンドヘルドデバイスに以下をインストールしてください。

- .NET Framework および .NET Compact Framework バージョン 2.0 以降 (あるいはそのいずれか)
- Visual Studio 2005 以降、または C# などの .NET 言語コンパイラ (開発用としてのみ必要)

SAP Sybase IQ .NET データプロバイダに必要なファイル

SAP Sybase IQ .NET データプロバイダのコードは、各プラットフォームの DLL にあります。

Windows に必要なファイル

Windows の場合、次のいずれかの DLL が必要です。

- `%IQDIR16%¥V2¥Assembly¥V2¥iAnywhere.Data.SQLAnywhere.dll`
- `%IQDIR16%¥V2¥Assembly ¥V3.5¥iAnywhere.Data.SQLAnywhere.v3.5.dll`

- `%IQDIR16%¥V2¥Assembly ¥V4¥iAnywhere.Data.SQLAnywhere.v4.0.dll`

DLL の選択は、対象とする .NET のバージョンによって異なります。

Windows バージョンのプロバイダには、次の DLL も必要です。

- **`policy.16.0.iAnywhere.Data.SQLAnywhere.dll`** – ポリシーファイルを使用すると、アプリケーションが構築されたプロバイダのバージョンを上書きできます。プロバイダへの更新がリリースされるたびに、ポリシーファイルは Sybase によって更新されます。また、バージョン 3.5 プロバイダ用 (`policy.16.0.iAnywhere.Data.SQLAnywhere.v3.5.dll`) とバージョン 4.0 プロバイダ用 (`policy.16.0.iAnywhere.Data.SQLAnywhere.v4.0.dll`) のポリシーファイルもあります。
- **`dblgen16.dll`** – この言語 DLL には、プロバイダによって発行された英語 (en) のメッセージが含まれています。この言語 DLL は、中国語 (zh)、フランス語 (fr)、ドイツ語 (de)、日本語 (jp) などの他の多くの言語で使用できます。
- **`dbcon16.dll`** – この DLL には、[SQL Anywhere への接続] ウィンドウのサポートコードが含まれています。

Visual Studio は、.NET データプロバイダ DLL (`iAnywhere.Data.SQLAnywhere.dll` または `iAnywhere.Data.SQLAnywhere.v3.5.dll`) をプログラムとともにデバイスに配備します。Visual Studio を使用していない場合は、データプロバイダ DLL をアプリケーションとともにデバイスにコピーする必要があります。この DLL は、アプリケーションと同じディレクトリまたは `¥Windows` ディレクトリに配置できます。

SAP Sybase IQ .NET データプロバイダ dbdata DLL

SAP Sybase IQ .NET データプロバイダが .NET アプリケーションによって最初にロードされると (通常、SAConnection を使用したデータベース接続の作成時)、プロバイダのアンマネージコードを含む DLL がアンパックされます。プロバイダによって、次の方式を使用して識別されたディレクトリのサブディレクトリに `dbdata16.dll` ファイルが格納されます。

1. プロバイダがアンロードで最初に使用しようとするディレクトリは、次のものによって返されたうちの最初のディレクトリです。

   `TMP` 環境変数によって識別されたパス。
   `TEMP` 環境変数によって識別されたパス。
   `USERPROFILE` 環境変数によって識別されたパス。
   Windows ディレクトリ。

2. 識別されたディレクトリにアクセスできない場合、プロバイダは現在の作業ディレクトリの使用を試みます。
3. 現在の作業ディレクトリにアクセスできない場合、プロバイダはアプリケーション自体のロード元ディレクトリの使用を試みます。

サブディレクトリ名は GUID の形式をとり、サフィックスが付きます。このサフィックスには、バージョン番号、DLL のビット数、一意性の保証に使用されるインデックス番号が含まれます。次は、サブディレクトリ名の例です。

```
{16AA8FB8-4A98-4757-B7A5-0FF22C0A6E33}_1601.x64_1
```

SAP Sybase IQ .NET データプロバイダ DLL の登録

Windows バージョンの SAP Sybase IQ .NET データプロバイダ DLL (`%IQDIR%¥Assembly¥V2¥iAnywhere.Data.SQLAnywhere.dll`) は、SAP Sybase IQ ソフトウェアをインストールするとき、グローバルアセンブリキャッシュに登録されます。Windows Mobile の場合、この DLL を登録する必要はありません。

SAP Sybase IQ .NET データプロバイダを配備する場合は、Microsoft SDK に含まれている `gacutil` ユーティリティを使用して、この DLL を登録できます。

SAP Sybase IQ .NET データプロバイダを配備するときに、プロバイダを DbProviderFactory インスタンスとして登録するには、.NET `machine.config` ファイルにエントリを追加する必要があります。次に似たエントリを `<DbProviderFactories>` セクションに配置する必要があります。

```
<add invariant="iAnywhere.Data.SQLAnywhere"
name="SAP Sybase IQ 16 Data Provider"
description=".Net Framework Data Provider for SAP Sybase IQ 16"
type="iAnywhere.Data.SQLAnywhere.SAFactory,
iAnywhere.Data.SQLAnywhere.v3.5,
    Version=16.0.0.36003, Culture=neutral,
PublicKeyToken=f222fc4333e0d400"/>
```

バージョン番号は、インストールしているプロバイダのバージョンに一致する必要があります。設定ファイルは `¥WINDOWS¥Microsoft.NET¥Framework¥v2.0.50727¥CONFIG` にあります。64 ビット Windows システムの場合、同様に変更する必要のある `Framework64` ツリーの下に 2 番目の設定ファイルがあります。

.NET でのトレースのサポート

SAP Sybase IQ .NET データプロバイダでは、.NET のトレーシング機能を使用したトレースをサポートしています。

デフォルトでは、トレースは無効です。トレースを有効にするには、アプリケーションの設定ファイルでトレースソースを指定します。

次は、設定ファイルの例です。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<system.diagnostics>
<sources>
 <source name="iAnywhere.Data.SQLAnywhere"
         switchName="SASourceSwitch"
         switchType="System.Diagnostics.SourceSwitch">
  <listeners>
   <add name="ConsoleListener"
        type="System.Diagnostics.ConsoleTraceListener"/>
   <add name="EventListener"
        type="System.Diagnostics.EventLogTraceListener"
        initializeData="MyEventLog"/>
   <add name="TraceLogListener"
        type="System.Diagnostics.TextWriterTraceListener"
        initializeData="myTrace.log"
        traceOutputOptions="ProcessId, ThreadId, Timestamp"/>
   <remove name="Default"/>
  </listeners>
 </source>
</sources>
<switches>
 <add name="SASourceSwitch" value="All"/>
 <add name="SATraceAllSwitch" value="1" />
 <add name="SATraceExceptionSwitch" value="1" />
 <add name="SATraceFunctionSwitch" value="1" />
 <add name="SATracePoolingSwitch" value="1" />
 <add name="SATracePropertySwitch" value="1" />
</switches>
</system.diagnostics>
</configuration>
```

上記の設定ファイルでは、4 種類のトレースリスナが参照されています。

- **ConsoleTraceListener –** トレース出力またはデバッグ出力は、標準出力か標準エラーストリームのどちらかに送られます。Microsoft Visual Studio を使用する場合、出力は [出力] ウィンドウに表示されます。
- **DefaultTraceListener –** このリスナは、名前 "Default" を使用して、自動的に Debug.Listeners および Trace.Listeners コレクションに追加されます。トレース出力またはデバッグ出力は、標準出力か標準エラーストリームのどちらかに送られます。Microsoft Visual Studio を使用する場合、出力は [出力] ウィンドウに表示されます。`ConsoleTraceListener` により生成される出力の重複を避けるため、このリスナは削除されます。
- **EventLogTraceListener –** トレース出力またはデバッグ出力は、`initializeData` オプションで識別される `EventLog` に送られます。この例では、イベントログの名前は `MyEventLog` です。システムイベントログに書き込むには管理者権限が必要であり、この方法でアプリケーションをデバッグすることはおすすめしません。

- **TextWriterTraceListener –** トレース出力またはデバッグ出力は `TextWriter` に送られ、`initializeData` オプションで識別されるファイルにストリームが書き込まれます。

上記のトレースリスナへのトレース出力を無効にするには、`<listeners>` にある対応する `add` エントリを削除します。

トレースの設定情報は、`App.config` というファイル名で、アプリケーションのプロジェクトフォルダに置かれます。このファイルが存在しない場合は、Visual Studio を使用して作成し、プロジェクトに追加できます。それには、[追加]»[新しい項目] を選択し、[アプリケーション構成ファイル] を選択します。

どのリスナでも `traceOutputOptions` を指定できます。次のオプションがあります。

- **Callstack –** コールスタックを書き込みます。コールスタックは、Environment.StackTrace プロパティの戻り値で表されます。
- **DateTime –** 日付と時刻を書き込みます。
- **LogicalOperationStack –** 論理演算スタックを書き込みます。論理演算スタックは、CorrelationManager.LogicalOperationStack プロパティの戻り値で表されます。
- **None –** 要素を書き込みません。
- **ProcessId –** プロセス ID を書き込みます。プロセス ID は、Process.Id プロパティの戻り値で表されます。
- **ThreadId –** スレッド ID を書き込みます。スレッド ID は、現在のスレッドの Thread.ManagedThreadId プロパティの戻り値で表されます。
- **Timestamp –** タイムスタンプを書き込みます。タイムスタンプは、System.Diagnostics.Stopwatch.GetTimeStamp メソッドの戻り値で表されます。

前述のサンプルの設定ファイルでは、`TextWriterTraceListener` に対してのみトレース出力オプションを指定しています。

特定のトレースオプションを設定することで、トレース対象を限定できます。デフォルトでは、数値のトレースオプション設定はすべて 0 です。設定できるトレースオプションには次の項目などがあります。

- **SASourceSwitch –** SASourceSwitch は、次の値のいずれかを取ることができます。`Off` の場合、トレーシングはありません。

  Off – イベントを許可しません。

  Critical – 重大なイベントのみを許可します。

  Error – 重大なイベントとエラーイベントを許可します。

  Warning – 重大なイベント、エラーイベント、警告イベントを許可します。

Information – 重大なイベント、エラーイベント、警告イベント、情報イベントを許可します。

Verbose – 重大なイベント、エラーイベント、警告イベント、情報イベント、冗長イベントを許可します。

ActivityTracing – 停止、開始、中断、転送、再開イベントを許可します。

All – すべてのイベントを許可します。

次に設定例を示します。

```
<add name="SASourceSwitch" value="Error"/>
```

- **SATraceAllSwitch –** すべてのトレースオプションが有効になります。すべてのオプションが選択されるため、その他のオプションを設定する必要はありません。このオプションを選択した場合は、個々のオプションを無効にできません。たとえば、次のようにしても、例外トレースは無効になりません。

  ```
  <add name="SATraceAllSwitch" value="1" />
  <add name="SATraceExceptionSwitch" value="0" />
  ```

- **SATraceExceptionSwitch –** すべての例外が記録されます。トレースメッセージの形式は次のとおりです。

  ```
  <Type|ERR> message='message_text'[ nativeError=error_number]
  ```

  nativeError=error_number は、SAException オブジェクトが存在する場合に限り表示されます。

- **SATraceFunctionSwitch –** すべての関数スコープの開始と終了が記録されます。トレースメッセージの形式は次のいずれかです。

  ```
  enter_nnn <sa.class_name.method_name|API> [object_id#]
  [parameter_names]
  leave_nnn
  ```

  nnn は、スコープのネストレベル 1、2、3 などを表す整数です。省略可能な parameter_names は、スペースで区切られたパラメータ名のリストです。

- **SATracePoolingSwitch –** すべての接続プーリングが記録されます。トレースメッセージの形式は次のいずれかです。

  ```
  <sa.ConnectionPool.AllocateConnection|CPOOL>
  connectionString='connection_text'
  <sa.ConnectionPool.RemoveConnection|CPOOL>
  connectionString='connection_text'
  <sa.ConnectionPool.ReturnConnection|CPOOL>
  connectionString='connection_text'
  <sa.ConnectionPool.ReuseConnection|CPOOL>
  connectionString='connection_text'
  ```

- **SATracePropertySwitch –** すべてのプロパティの設定と取得が記録されます。トレースメッセージの形式は次のいずれかです。

  ```
  <sa.class_name.get_property_name|API> object_id#
  <sa.class_name.set_property_name|API> object_id#
  ```

詳細については、http://msdn.microsoft.com/ja-jp/library/ms971550.aspx の「データアクセスのトレース」を参照してください。

Windows アプリケーションのトレース設定

TableViewer サンプルアプリケーションでトレースを有効にするには、作成する設定ファイルの中で、ConsoleTraceListener および TextWriterTraceListener リスナを参照し、デフォルトのリスナを削除し、本来は 0 に設定されるスイッチをすべて有効にします。

前提条件

Visual Studio がインストールされている必要があります。

手順

1. Visual Studio で TableViewer サンプルを開きます。

   Visual Studio を起動し、`%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥ADO.NET¥TableViewer¥TableViewer.sln` を開きます。
2. `App.config` という名前のアプリケーションファイルを作成し、次の設定内容をコピーします。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<system.diagnostics>
<sources>
 <source name="iAnywhere.Data.SQLAnywhere"
         switchName="SASourceSwitch"
         switchType="System.Diagnostics.SourceSwitch">
  <listeners>
   <add name="ConsoleListener"
        type="System.Diagnostics.ConsoleTraceListener"/>
   <add name="TraceLogListener"
        type="System.Diagnostics.TextWriterTraceListener"
        initializeData="myTrace.log"
        traceOutputOptions="ProcessId, ThreadId, Timestamp"/>
   <remove name="Default"/>
  </listeners>
 </source>
</sources>
<switches>
 <add name="SASourceSwitch" value="All"/>
 <add name="SATraceAllSwitch" value="1" />
 <add name="SATraceExceptionSwitch" value="1" />
 <add name="SATraceFunctionSwitch" value="1" />
 <add name="SATracePoolingSwitch" value="1" />
 <add name="SATracePropertySwitch" value="1" />
</switches>
</system.diagnostics>
</configuration>
```

3. アプリケーションを再構築します。
4. [デバッグ] » [デバッグの開始] をクリックします。

アプリケーションの実行が完了すると、トレース出力が `bin¥Debug ¥myTrace.log` ファイルに記録されます。

次のステップ

Visual Studio の [出力] ウィンドウでトレースログを表示します。

.NET データプロバイダチュートリアル

.NET データプロバイダには、Simple サンプルプロジェクトと Table Viewer サンプルプロジェクトが付属しています。

サンプルプロジェクトは Visual Studio 2005 以降のバージョンで使用できます。サンプルプロジェクトは Visual Studio 2005 を使用して開発されています。2005 よりも新しいバージョンをお使いの場合は、Visual Studio の[アップグレードウィザード]を実行する必要がある場合があります。また、この項のチュートリアルでは、Visual Studio を使用して Simple Viewer .NET データベースアプリケーションを構築する手順をひととおり説明します。

チュートリアル： Simple コードサンプルの使用

Simple プロジェクトでは、.NET データプロバイダを使用してデータベースサーバから結果セットを取得します。

前提条件

コンピュータに Visual Studio および .NET Framework がインストールされている必要があります。

SELECT ANY TABLE システム権限が必要です。

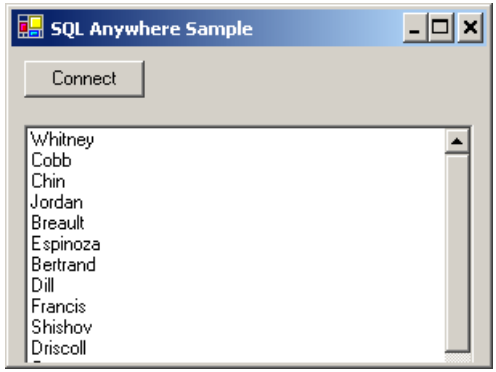
手順

Simple プロジェクトは SAP Sybase IQ のサンプルに付属しています。このプロジェクトでは、Employees テーブルからの名前を設定する簡単なリストボックスの例を示します。

1. Visual Studio を起動します。
2. [ファイル] » [開く] » [プロジェクト] をクリックします。

3. %ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥ADO.NET¥SimpleWin32 を参照し、Simple.sln プロジェクトを開きます。
4. プロジェクトで SAP Sybase IQ .NET データプロバイダを使用するには、データプロバイダへの参照を追加する必要があります。これはすでに Simple コードサンプルで行われています。データプロバイダ (iAnywhere.Data.SQLAnywhere) への参照を表示するには、[ソリューション エクスプローラー] ウィンドウで [参照設定] フォルダを開きます。
5. また、データプロバイダクラスを参照する using ディレクティブもソースコードに追加してください。これはすでに Simple コードサンプルで行われています。using ディレクティブを表示するには、次の手順に従います。
   - プロジェクトのソースコードを開きます。[ソリューション エクスプローラー] ウィンドウで Form1.cs を右クリックし、[コードの表示] をクリックします。
     上部セクションの using ディレクティブに次の行が表示されます。
     ```
     using iAnywhere.Data.SQLAnywhere;
     ```
     この行は C# プロジェクトに必要です。Visual Basic .NET を使用している場合、ソースコードに Imports 行を追加する必要があります。
6. [デバッグ] » [デバッグなしで開始] をクリックするか、[Ctrl + F5] を押して、Simple サンプルを実行します。
7. [SQL Anywhere Sample] ウィンドウで [Connect] をクリックします。

   アプリケーションが SAP Sybase IQ サンプルデータベースに接続され、次のように各従業員の姓がウィンドウに表示されます。



8. [SQL Anywhere Sample] ウィンドウを閉じ、アプリケーションを終了してサンプルデータベースとの接続を切断します。これによって、データベースサーバも停止します。

これで、SAP Sybase IQ .NET データプロバイダを使用して SAP Sybase IQ データベースサーバから結果セットを取得するシンプルな .NET アプリケーションを構築して実行しました。

チュートリアル： Table Viewer コードサンプルの使用

TableViewer プロジェクトでは、.NET データプロバイダを使用してデータベースに接続し、SQL 文を実行し、DataGrid オブジェクトを使用して結果を表示します。

前提条件

コンピュータに Visual Studio および .NET Framework がインストールされている必要があります。

SELECT ANY TABLE システム権限が必要です。

手順

TableViewer プロジェクトは SAP Sybase IQ のサンプルに付属しています。Table Viewer プロジェクトは Simple プロジェクトよりも複雑です。これを使用して、データベースへの接続、テーブルの選択、データベースでの SQL 文の実行ができます。

1. Visual Studio を起動します。
2. [ファイル] » [開く] » [プロジェクト] をクリックします。
3. `%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥ADO.NET¥TableViewer` を参照し、`TableViewer.sln` プロジェクトを開きます。
4. プロジェクトで SAP Sybase IQ .NET データプロバイダを使用するには、データプロバイダ DLL への参照を追加する必要があります。これはすでに Table Viewer コードサンプルで行われています。データプロバイダ (`iAnywhere.Data.SQLAnywhere`) への参照を表示するには、[ソリューション エクスプローラー] ウィンドウで [参照設定] フォルダを開きます。
5. また、データプロバイダクラスを参照する `using` ディレクティブもソースコードに追加してください。これはすでに Table Viewer コードサンプルで行われています。`using` ディレクティブを表示するには、次の手順に従います。
   - プロジェクトのソースコードを開きます。[ソリューション エクスプローラー] ウィンドウで `TableViewer.cs` を右クリックし、[コードの表示] をクリックします。
   - 上部セクションの `using` ディレクティブに次の行が表示されます。

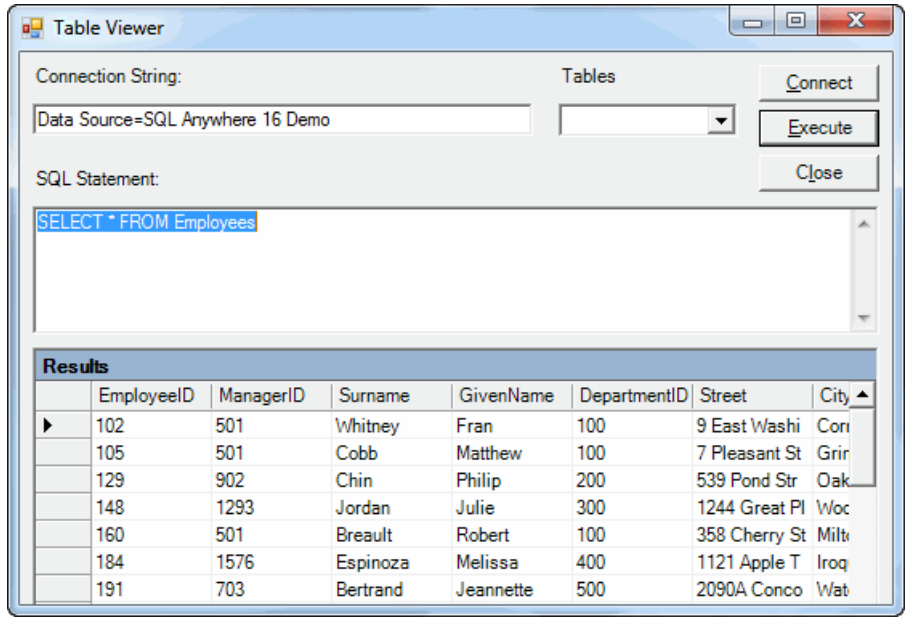
     ```
     using iAnywhere.Data.SQLAnywhere;
     ```

この行は C# プロジェクトに必要です。Visual Basic を使用している場合、ソースコードに `Imports` 行を追加する必要があります。

6. [デバッグ] » [デバッグなしで開始] をクリックするか、[Ctrl + F5] を押して、Table Viewer サンプルを実行します。

   アプリケーションが SAP Sybase IQ サンプルデータベースに接続します。
7. [Table Viewer] ウィンドウで [Connect] をクリックします。
8. [Table Viewer] ウィンドウで [Execute] をクリックします。

   アプリケーションは、サンプルデータベースの Employees テーブルからデータを取り出し、次のようにクエリ結果を [Results] データグリッドに表示します。



   このアプリケーションから別の SQL 文を実行することもできます。これを行うには、[SQL Statement] ウィンドウ枠に SQL 文を入力し、[Execute] をクリックします。
9. [Table Viewer] ウィンドウを閉じ、アプリケーションを終了してサンプルデータベースとの接続を切断します。これによって、データベースサーバも停止します。

これで、.NET データプロバイダを使用してデータベースに接続し、SQL 文を実行し、DataGrid オブジェクトを使用して結果を表示する .NET アプリケーションを構築して実行しました。

チュートリアル： Visual Studio を使用したシンプルな .NET データベースアプリケーションの開発

この項のチュートリアルでは、Visual Studio を使用して Simple Viewer .NET データベースアプリケーションを構築する手順をひととおり説明します。

前提条件

SELECT ANY TABLE システム権限が必要です。

レッスン 1： テーブルビューアの作成

このレッスンでは、Microsoft Visual Studio、サーバーエクスプローラ、および SAP Sybase IQ .NET データプロバイダを使用して、SAP Sybase IQ サンプルデータベース内の 1 つのテーブルにアクセスするアプリケーションを作成します。このアプリケーションを使用して、テーブルのローを調べたり更新を実行することができます。

前提条件

コンピュータに Visual Studio および .NET Framework がインストールされている必要があります。

このレッスンでは、このチュートリアル「チュートリアル： Visual Studio を使用したシンプルな .NET データベースアプリケーションの開発」の冒頭の「権限」セクションに列挙されているロールと権限を持っていることを前提としています。

手順

このチュートリアルは、Visual Studio と .NET Framework に基づきます。完全なアプリケーションは、ADO.NET プロジェクトの `%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥ADO.NET¥SimpleViewer¥SimpleViewer.sln` にあります。

1. Visual Studio を起動します。
2. [ファイル] » [新規作成] » [プロジェクト] をクリックします。

   [新しいプロジェクト] ウィンドウが表示されます。

   a. [新しいプロジェクト] ウィンドウの左ウィンドウ枠で、プログラミング言語として [Visual Basic] または [Visual C#] をクリックします。
   b. [Windows] サブカテゴリで、[Windows アプリケーション] (VS 2005 の場合) または [Windows フォーム アプリケーション] (VS 2008/2010 の場合) をクリックします。

c. プロジェクトの [名前] フィールドに、`MySimpleViewer` と入力します。

d. [OK] をクリックし、新規プロジェクトを作成します。

3. [表示] » [サーバー エクスプローラー] をクリックします。

4. [サーバー エクスプローラー] ウィンドウで [データ接続] を右クリックし、[接続の追加] をクリックします。

5. [接続の追加] ウィンドウで次の作業を実行します。

a. 他のプロジェクトで [接続の追加] を一度も使用したことがない場合は、データソースのリストが表示されます。データソースのリストから [SQL Anywhere] をクリックします。

[接続の追加] を以前に使用したことがある場合は、[変更] をクリックしてデータソースを [SQL Anywhere] に変更します。

b. [データソース] で、[ODBC データソース名] をクリックし、`Sybase IQ Demo` と入力します。

注意： Visual Studio の接続の追加ウィザードを 64 ビット Windows で使用する場合は、64 ビットのシステムデータソース名 (DSN) のみがユーザデータソース名と一緒に含まれます。32 ビットのシステムデータソース名は表示されません。Visual Studio の 32 ビット設計環境で [テスト接続] ボタンを使用すると、64 ビットのシステム DSN と等価な 32 ビットのシステム DSN を使用して接続の確立が試みられます。32 ビットのシステム DSN が存在しない場合、テストは失敗します。

c. [テスト接続] をクリックして、サンプルデータベースに接続できることを確認します。

d. [OK] をクリックします。

`Sybase IQ.demo` という新しい接続が [サーバー エクスプローラー] ウィンドウに表示されます。

6. [サーバー エクスプローラー] ウィンドウで、テーブル名が表示されるまで `Sybase IQ.demo` 接続を展開します。

(Visual Studio 2005 のみ) 次の手順を試してみます。

a. Products テーブルを右クリックし、[テーブル データの表示] をクリックします。

Products テーブルのローとカラムがウィンドウに表示されます。

b. テーブルデータのウィンドウを閉じます。

7. [データ] » [新しいデータ ソースの追加] をクリックします。

8. [データ ソース構成ウィザード]で、次の作業を実行します。

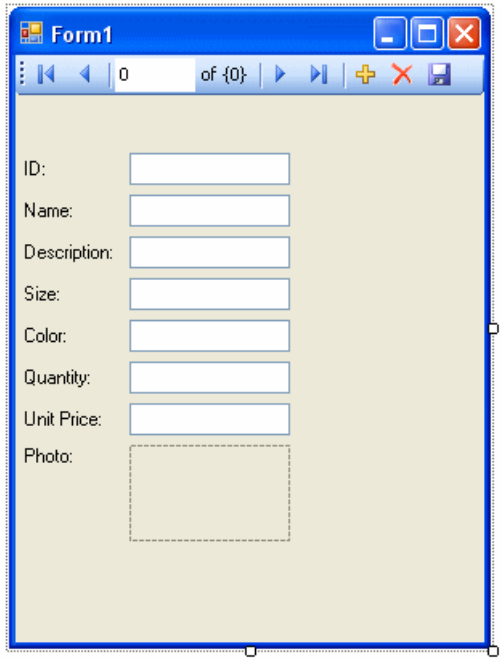
a. [データ ソースの種類を選択] ページで [データベース] をクリックし、[次へ] をクリックします。
b. (Visual Studio 2010 のみ) [データベースモデル] ページで [データセット] をクリックし、[次へ] をクリックします。
c. [データ接続の選択] ページで `Sybase IQ.demo` をクリックし、[次へ] をクリックします。
d. [接続文字列をアプリケーション構成ファイルに保存する] ページで、[次の名前で接続を保存する] が選択されていることを確認して [次へ] をクリックします。
e. [データベース オブジェクトの選択] ページで [テーブル] をクリックし、[完了] をクリックします。

9. [データ] » [データ ソースの表示] をクリックします。

[データソース] ウィンドウが表示されます。

[データソース] ウィンドウで Products テーブルを展開します。

a. [Products] をクリックし、ドロップダウンリストから [詳細] をクリックします。
b. [Photo] をクリックし、ドロップダウンリストから [Picture Box] をクリックします。
c. [Products] をクリックして、フォーム (Form1) にドラッグします。



データセットコントロールと複数のラベル付きテキストフィールドがフォームに表示されます。

10. フォームで、[Photo] の横にあるピクチャボックスをクリックします。
    a. ボックスの形を四角形に変更します。
    b. ピクチャボックスの右上の右矢印をクリックします。
       [Picture Box タスク] ウィンドウが開きます。
    c. [サイズ モード] ドロップダウンリストから、[ズーム] をクリックします。
    d. [Picture Box タスク] ウィンドウを閉じるには、ウィンドウの外側で任意の場所をクリックします。
11. プロジェクトをビルドし、実行します。
    a. [ビルド] » [ソリューションのビルド] をクリックします。
    b. [デバッグ] » [デバッグの開始] をクリックします。
       アプリケーションが SAP Sybase IQ サンプルデータベースに接続されて、Products テーブルの最初のローがテキストボックスとピクチャボックスに表示されます。

    c. コントロールのボタンを使用して、結果セットのローをスクロールできます。
    d. ロー番号をスクロールコントロールに入力すると、結果セットのローに直接アクセスできます。

e. テキストボックスを使用して結果セットの値を更新し、[Save Data] ボタンをクリックして値を保存できます。

12. アプリケーションを終了してプロジェクトを保存します。

これで、Visual Studio、サーバーエクスプローラ、SAP Sybase IQ .NET データプロバイダを使用して、シンプルでありながら強力な .NET アプリケーションが作成されました。

次のステップ

次のレッスンでは、このレッスンで作成したフォームにデータグリッドコントロールを追加します。

レッスン 2： 同期データコントロールの追加

このレッスンでは、前のレッスンで作成したフォームにデータグリッドコントロールを追加します。このコントロールは、結果セット内のナビゲーションに合わせて内容を自動的に更新します。

前提条件

このレッスンでは、このチュートリアル「チュートリアル: データベースでの Java の使用」および「チュートリアル: Visual Studio を使用したシンプルな .NET データベースアプリケーションの開発」の冒頭の「権限」セクションに列挙されているロールと権限を持っていることを前提としています。

手順

完全なアプリケーションは、ADO.NET プロジェクトの `%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥ADO.NET¥SimpleViewer¥SimpleViewer.sln` にあります。

1. Visual Studio を起動し、MySimpleViewer プロジェクトをロードします。
2. [データ ソース] ウィンドウで [DataSet1] を右クリックし、[デザイナで DataSet を編集] をクリックします。
3. [データセット デザイナ] ウィンドウの空白領域を右クリックし、[追加] » [TableAdapter] をクリックします。
4. [TableAdapter 構成ウィザード] で次の作業を実行します。
   a. [データ接続の選択] ページで、[次へ] をクリックします。
   b. [コマンドの種類を選択します] ページで、[SQL ステートメントを使用する] をクリックし、[次へ] をクリックします。
   c. [SQL ステートメントの入力] ページで、[クエリ ビルダ] をクリックします。

d. [テーブルの追加] ウィンドウの [ビュー] タブをクリックし、[ViewSalesOrders] をクリックして [追加] をクリックします。

e. [閉じる] をクリックして [テーブルの追加] ウィンドウを閉じます。

5. ウィンドウのすべてのセクションが表示されるように、[クエリ ビルダ] ウィンドウを拡大します。

a. すべてのチェックボックスが表示されるように、[ViewSalesOrders] ウィンドウを拡大します。

b. [Region] をクリックします。

c. [Quantity] をクリックします。

d. [ProductID] をクリックします。

e. [ViewSalesOrders] ウィンドウの下のグリッドで、[ProductID] カラムの [出力] の下にあるチェックボックスをオフにします。

f. [ProductID] カラムで、[フィルタ] セルに疑問符 (?) を入力します。これで ProductID の WHERE 句が生成されます。

次のような SQL クエリが作成されました。

```
SELECT   Region, Quantity
FROM     GROUPO.ViewSalesOrders
WHERE    (ProductID = :Param1)
```

6. この SQL クエリを次のように変更します。

a. `Quantity` を `SUM(Quantity) AS TotalSales` に変更します。

b. `GROUP BY Region` を WHERE 句の後に続くクエリの末尾に追加します。

変更した SQL クエリは次のようになります。

```
SELECT   Region, SUM(Quantity) as TotalSales
FROM     GROUPO.ViewSalesOrders
WHERE    (ProductID = :Param1)
GROUP BY Region
```

7. [OK] をクリックします。

8. [完了] をクリックします。

[ViewSalesOrders] という新しい [TableAdapter] が [データセット デザイナ] ウィンドウに追加されました。

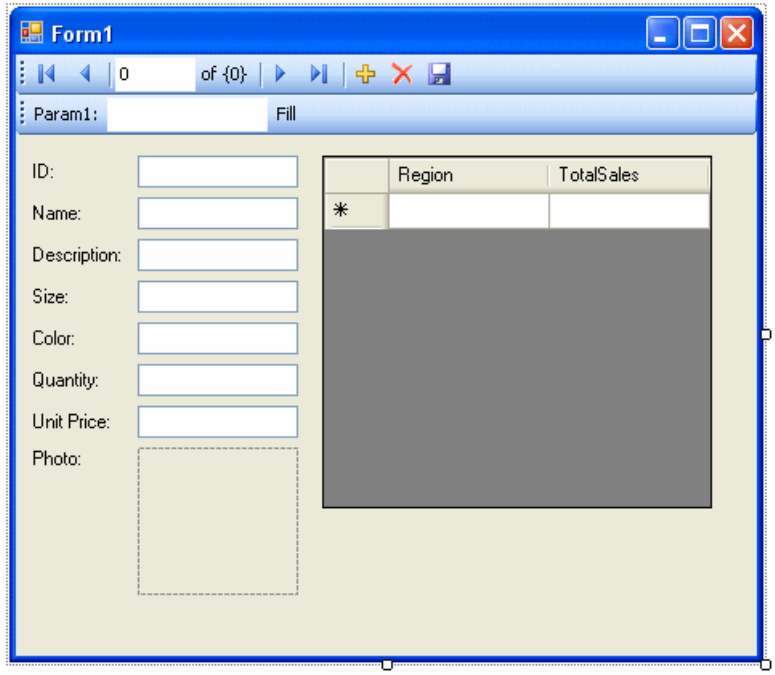
9. フォームデザインタブ (Form1) をクリックします。

- フォームを右方向にドラッグして拡大し、新しいコントロールの領域を確保します。

10. [データ ソース] ウィンドウで [ViewSalesOrders] を展開します。

a. [ViewSalesOrders] をクリックし、ドロップダウンリストから [DataGridView] をクリックします。

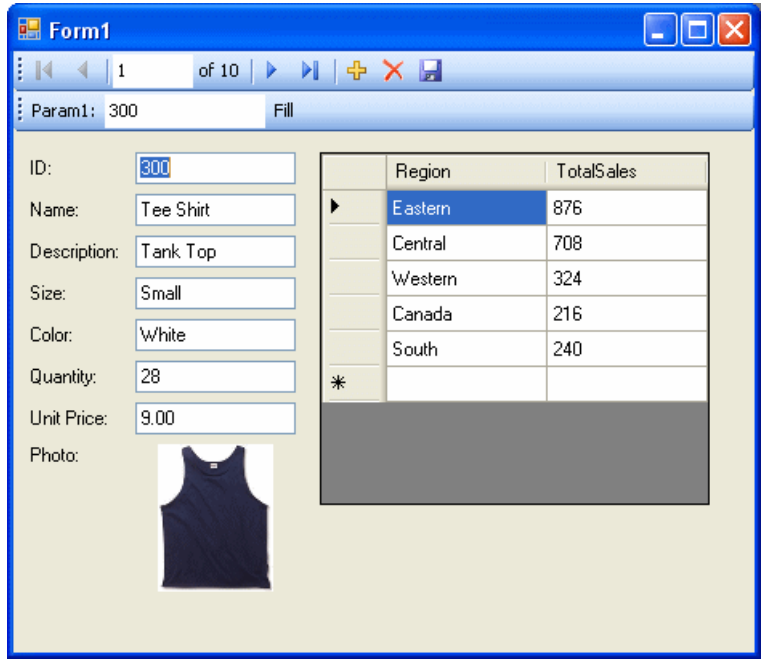
b. [ViewSalesOrders] をクリックして、フォーム (Form1) にドラッグします。



データグリッドビューのコントロールがフォームに表示されます。

11. プロジェクトをビルドし、実行します。

- [ビルド] » [ソリューションのビルド] をクリックします。
- [デバッグ] » [デバッグの開始] をクリックします。
- [Param1] または [ProductID] (VS 2010) テキストボックスに、300 などの製品 ID を入力して [Fill] をクリックします。
  入力した製品 ID の販売概要が、データグリッドビューに地域別に表示されます。



フォームのもう一方のコントロールを使用して、結果セットのローを移動することもできます。

ただし、2 つのコントロールが互いに同期された状態になっていることが理想的です。これを実現するための手順を次に示します。

12. アプリケーションを終了してプロジェクトを保存します。
13. Fill ストリップは不要なのでフォームから削除します。
    - デザインフォーム (Form1) で、[Fill] の右側にある Fill ストリップを右クリックして、[削除] をクリックします。

      Fill ストリップがフォームから削除されます。
14. 2 つのコントロールを次のようにして同期します。
    a. デザインフォーム (Form1) で [ID] テキストボックスを右クリックし、[プロパティ] をクリックします。
    b. [イベント] ボタン (電球で表示される) をクリックします。
    c. [TextChanged] イベントが見つかるまでスクロールダウンします。
    d. [TextChanged] をクリックし、ドロップダウンリストから [fillToolStripButton_Click] をクリックします。Visual Basic を使用している場合は、[FillToolStripButton_Click] というイベントです。

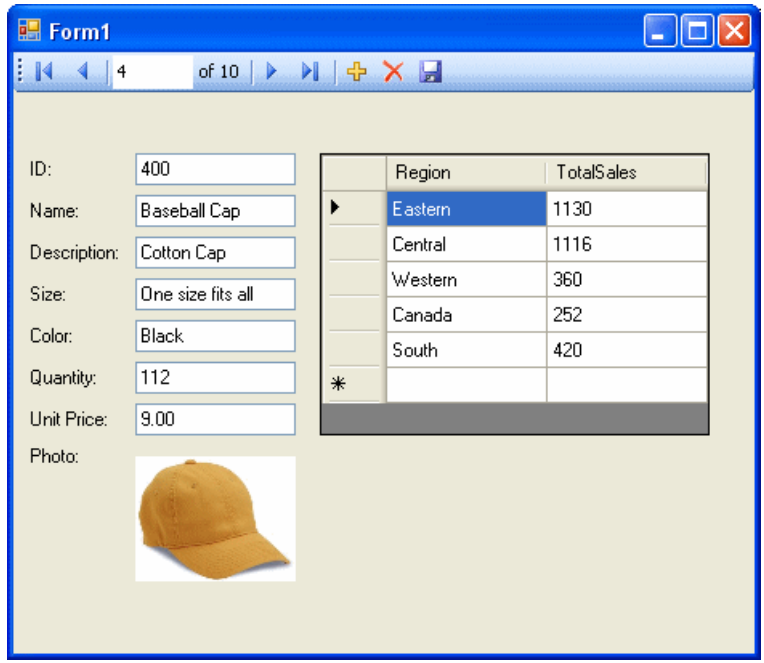
e. [fillToolStripButton_Click] をダブルクリックして、`fillToolStripButton_Click` イベントハンドラに関するフォームのコードウィンドウを開きます。

f. `param1ToolStripTextBox` または `productIDToolStripTextBox` (VS 2010) への参照を検索し、これを `iDTextBox` に変更します。Visual Basic を使用している場合は、`IDTextBox` というテキストボックスです。

g. プロジェクトを再度ビルドし、実行します。

15. アプリケーションフォームに表示されるナビゲーションコントロールが 1 つだけになりました。

- 結果セットを移動すると、それに合わせて現在の製品の販売概要が更新され、データグリッドビューに地域ごとに表示されます。



16. アプリケーションを終了してプロジェクトを保存します。

これで、結果セットの移動に合わせて内容を自動的に更新するコントロールが追加されました。

このチュートリアルでは、強力なツールである Microsoft Visual Studio、サーバーエクスプローラ、および SAP Sybase IQ .NET データプロバイダを組み合わせて使用して、データベースアプリケーションを作成する方法を学習しました。

.NET API リファレンス

ネームスペースは iAnywhere.Data.SQLAnywhere です。

OLE DB と ADO の開発

SAP Sybase IQ には、OLE DB と ADO の OLE DB プロバイダが含まれています。

OLE DB は、Microsoft が開発した一連のコンポーネントオブジェクトモデル (COM: Component Object Model) インタフェースです。さまざまな情報ソースに格納されているデータに対して複数のアプリケーションから同じ方法でアクセスしたり、追加のデータベースサービスを実装したりできるようにします。これらのインタフェースは、データストアに適した多数の DBMS 機能をサポートし、データを共有できるようにします。

ADO は、OLE DB システムインタフェースを通じてさまざまなデータソースに対してプログラムからアクセス、編集、更新するためのオブジェクトモデルです。ADO も Microsoft が開発したものです。OLE DB プログラミングインタフェースを使用しているほとんどの開発者は、OLE DB API に直接記述するのではなく、ADO API に記述しています。

ADO インタフェースと ADO.NET を混同しないでください。ADO.NET は別のインタフェースです。

OLE DB と ADO のプログラミングに関するマニュアルについては、Microsoft Developer Network を参照してください。OLE DB と ADO の開発に関する SAP Sybase IQ に固有の情報については、このマニュアルを使用してください。

OLE DB

OLE DB は Microsoft が提供するデータアクセスモデルです。OLE DB は、Component Object Model (COM) インタフェースを使用します。ODBC と違って、OLE DB は、データソースが SQL クエリプロセッサを使用することを仮定していません。

SAP Sybase IQ には `SAOLEDB` という名前の *OLE DB プロバイダ*が含まれています。このプロバイダは現在の Windows プラットフォームで使用できます。このプロバイダは Windows Mobile プラットフォームでは使用できません。

また、Microsoft OLE DB Provider for ODBC (MSDASQL) を使用すると、SQL Anywhere の ODBC ドライバで SAP Sybase IQ にアクセスすることもできます。

SAP Sybase IQ の OLE DB プロバイダを使用すると、いくつかの利点が得られます。

- カーソルによる更新など、OLE DB/ODBC ブリッジを使用している場合には利用できない機能がいくつかあります。

- SAP Sybase IQ OLE DB プロバイダを使用する場合は、配備時に ODBC は必要ありません。
- MSDASQL によって、OLE DB クライアントはどの ODBC ドライバでも動作しますが、各 ODBC ドライバが備えている機能のすべてを利用できるかどうかは、保証されていません。SAP Sybase IQ プロバイダを使用すると、OLE DB プログラミング環境から SAP Sybase IQ のすべての機能を利用できます。

OLE DB を使用した接続

SAP Sybase IQ には、ODBC の代替として OLE DB プロバイダが用意されています。OLE DB は、Microsoft から提供されているデータアクセスモデルであり、COM (Component Object Mode) インタフェースを使用します。OLE DB は、データソースでの SQL クエリプロセッサの使用を前提としない点で、ODBC とは異なります。OLE DB には Windows クライアントが必要ですが、OLE DB を使用すると、Windows サーバと UNIX サーバにアクセスできます。

SAP Sybase IQ OLE DB のサポートは SQL Anywhere のサポートとは異なります。SAP Sybase IQ では、動的 (動的スクロール) カーソル、静的 (無反応) カーソル、前方スクロールのみ (スクロールなし) カーソルがサポートされますが、キーセット (スクロール) カーソルはサポートされません。SAP Sybase IQ では、独立性レベルは何を指定しても必ず 3 になります。

SAP Sybase IQ では、動的 (動的スクロール) カーソル、静的 (無反応) カーソル、前方スクロールのみ (スクロールなし) カーソルがサポートされますが、キーセット (スクロール) カーソルはサポートされません。SAP Sybase IQ では、独立性レベルは何を指定しても必ず 3 になります。

SAP Sybase IQ では、Windows CE はサポートされません。また、カーソルを通じたリモートアップデートもサポートされません。

その他の情報
『プログラミング』>「OLE DB と ADO の開発」>「OLE DB 接続パラメータ」

サポートするプラットフォーム

SAP Sybase IQ の OLE DB プロバイダは、Microsoft Data Access Components (MDAC) 2.8 以降のバージョンで動作するように設計されています。

OLE DB での分散トランザクション

OLE DB ドライバを、分散トランザクション環境のリソースマネージャとして使用できます。

SAP Sybase IQ を使用した ADO プログラミング

ADO (ActiveX Data Objects) は Automation インタフェースを通じて公開されているデータアクセスオブジェクトモデルで、オブジェクトに関する予備知識がなくても、クライアントアプリケーションが実行時にオブジェクトのメソッドとプロパティを発見できるようにします。Automation によって、Visual Basic のようなスクリプト記述言語は標準のデータアクセスオブジェクトモデルを使用できるようになります。ADO は OLE DB を使用してデータアクセスを提供します。

SAP Sybase IQ OLE DB プロバイダを使用して、ADO プログラミング環境から SAP Sybase IQ のすべての機能を利用できます。

この項では、Visual Basic から ADO を使用するときに基本的な作業を実行する方法について説明します。ADO を使用したプログラミングに関する完全なガイドではありません。

この項のコードサンプルは、`%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥VBSampler¥vbsampler.sln` プロジェクトファイルにあります。

ADO によるプログラミングについては、開発ツールのマニュアルを参照してください。

Connection オブジェクトを使用してデータベースに接続する方法

この項では、データベースに接続する簡単な Visual Basic ルーチンについて説明します。

サンプルコード

このルーチンは、フォームに cmdTestConnection というコマンドボタンを配置し、その Click イベントに次のルーチンをペーストすることで試用できます。プログラムを実行し、ボタンをクリックして接続と切断を行います。

```
Private Sub cmdTestConnection_Click( _
        ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) _
        Handles cmdTestConnection.Click

    ' Declare variables
    Dim myConn As New ADODB.Connection
    Dim myCommand As New ADODB.Command
    Dim cAffected As Integer

    On Error GoTo HandleError

    ' Establish the connection
```

```
    myConn.Provider = "SAOLEDB"
    myConn.ConnectionString = _
        "Data Source=Sybase IQ Demo"
    myConn.Open()
    MsgBox("Connection succeeded")
    myConn.Close()
    Exit Sub

HandleError:
    MsgBox(ErrorToString(Err.Number))
    Exit Sub
End Sub
```

説明
この例は、次の作業を行います。

- ルーチンで使われる変数を宣言します。
- SAP Sybase IQ の OLE DB プロバイダを使用して、サンプルデータベースへの接続を確立します。
- Command オブジェクトを使用して簡単な文を実行し、データベースサーバメッセージウィンドウにメッセージを表示します。
- 接続を閉じます。

Command オブジェクトを使用して文を実行する方法

この項では、データベースに簡単な SQL 文を送る簡単なルーチンについて説明します。

サンプルコード
このルーチンは、フォームに cmdUpdate というコマンドボタンを配置し、その Click イベントに次のルーチンをペーストすることで試用できます。プログラムを実行し、ボタンをクリックして接続、データベースサーバメッセージウィンドウへのメッセージの表示、切断を行います。

```
Private Sub cmdUpdate_Click( _
        ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) _
        Handles cmdUpdate.Click

    ' Declare variables
    Dim myConn As New ADODB.Connection
    Dim myCommand As New ADODB.Command
    Dim cAffected As Integer

    On Error GoTo HandleError

    ' Establish the connection
    myConn.Provider = "SAOLEDB"
    myConn.ConnectionString = _
        "Data Source=Sybase IQ Demo"
    myConn.Open()
```

```
    'Execute a command
    myCommand.CommandText = _
        "UPDATE Customers SET GivenName='Liz' WHERE ID=102"
    myCommand.ActiveConnection = myConn
    myCommand.Execute(cAffected)
    MsgBox(CStr(cAffected) & " rows affected.", _
        MsgBoxStyle.Information)

    myConn.Close()
    Exit Sub

HandleError:
    MsgBox(ErrorToString(Err.Number))
    Exit Sub
End Sub
```

説明
サンプルコードは、接続を確立した後、Command オブジェクトを作成し、CommandText プロパティを update 文に、ActiveConnection プロパティを現在の接続に設定します。次に update 文を実行し、この更新で影響を受けるローの数をウィンドウに表示します。

この例では、更新はデータベースに送られ、実行と同時にコミットされます。

カーソルを使用して更新を実行することもできます。

Recordset オブジェクトを使用して結果セットを取得する方法

ADO の Recordset オブジェクトは、クエリの結果セットを表します。これを使用して、データベースのデータを参照できます。

サンプルコード
このルーチンは、フォームに cmdQuery というコマンドボタンを配置し、その Click イベントに次のルーチンをペーストすることで試用できます。プログラムを実行し、ボタンをクリックして接続、データベースサーバメッセージウィンドウへのメッセージの表示を行います。次にクエリの実行、最初の 2、3 のローのウィンドウへの表示、切断を行います。

```
Private Sub cmdQuery_Click( _
        ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) _
        Handles cmdQuery.Click

    ' Declare variables
    Dim i As Integer
    Dim myConn As New ADODB.Connection
    Dim myCommand As New ADODB.Command
    Dim myRS As New ADODB.Recordset

    On Error GoTo ErrorHandler
```

```
    ' Establish the connection
    myConn.Provider = "SAOLEDB"
    myConn.ConnectionString = _
        "Data Source=Sybase IQ Demo"
    myConn.CursorLocation = _
        ADODB.CursorLocationEnum.adUseServer
    myConn.Mode = _
        ADODB.ConnectModeEnum.adModeReadWrite
    myConn.IsolationLevel = _
        ADODB.IsolationLevelEnum.adXactCursorStability
    myConn.Open()

    'Execute a query
    myRS = New ADODB.Recordset
    myRS.CacheSize = 50
    myRS.let_Source("SELECT * FROM Customers")
    myRS.let_ActiveConnection(myConn)
    myRS.CursorType = ADODB.CursorTypeEnum.adOpenKeyset
    myRS.LockType = ADODB.LockTypeEnum.adLockOptimistic
    myRS.Open()

    'Scroll through the first few results
    myRS.MoveFirst()
    For i = 1 To 5
        MsgBox(myRS.Fields("CompanyName").Value, _
            MsgBoxStyle.Information)
        myRS.MoveNext()
    Next

    myRS.Close()
    myConn.Close()
    Exit Sub

ErrorHandler:
    MsgBox(ErrorToString(Err.Number))
    Exit Sub
End Sub
```

説明
この例の Recordset オブジェクトは、Customers テーブルに対するクエリの結果を保持します。For ループは最初にあるいくつかのローをスクロールして、各ローに対する CompanyName の値を表示します。

これは、ADO のカーソルを使用した簡単な例です。

Recordset オブジェクト

ADO の Recordset は、SAP Sybase IQ で処理する場合、カーソルを表します。Recordset オブジェクトの CursorType プロパティを宣言することでカーソルのタイプを選択してから、Recordset を開きます。カーソルタイプの選択は、Recordset で行える操作を制御し、パフォーマンスを左右します。

カーソルタイプ
ADO には、カーソルタイプに対する固有の命名規則があります。

使用できるカーソルタイプと、対応するカーソルタイプの定数と、それらと同等の SQL Anywhere のタイプは、次のとおりです。

ADO カーソルタイプ	ADO 定数	SAP Sybase IQ のタイプ
動的カーソル	adOpenDynamic	動的スクロールカーソル
キーセットカーソル	adOpenKeyset	スクロールカーソル
静的カーソル	adOpenStatic	insensitive カーソル
前方向カーソル	adOpenForwardOnly	非スクロールカーソル

サンプルコード
次のコードは、ADO の Recordset オブジェクトに対してカーソルタイプを設定します。

```
Dim myRS As New ADODB.Recordset
myRS.CursorType = ADODB.CursorTypeEnum.adOpenDynamic
```

Recordset オブジェクトを使用したカーソルによるローの更新

SAP Sybase IQ の OLE DB プロバイダで、カーソルによる結果セットの更新ができます。この機能は、MSDASQL プロバイダでは使用できません。

レコードセットの更新
データベースは Recordset を通じて更新できます。

```
Private Sub cmdUpdateThroughCursor_Click( _
        ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) _
        Handles cmdUpdateThroughCursor.Click

    ' Declare variables
    Dim i As Integer
    Dim myConn As New ADODB.Connection
    Dim myRS As New ADODB.Recordset
    Dim SQLString As String

    On Error GoTo HandleError

    ' Connect
    myConn.Provider = "SAOLEDB"
    myConn.ConnectionString = _
        "Data Source=Sybase IQ Demo"
    myConn.Open()
    myConn.BeginTrans()
    SQLString = "SELECT * FROM Customers"
```

```
    myRS.Open(SQLString, myConn, _
        ADODB.CursorTypeEnum.adOpenDynamic, _
        ADODB.LockTypeEnum.adLockBatchOptimistic)

    If myRS.BOF And myRS.EOF Then
        MsgBox("Recordset is empty!", 16, "Empty Recordset")
    Else
        MsgBox("Cursor type: " & CStr(myRS.CursorType), _
            MsgBoxStyle.Information)
        myRS.MoveFirst()
        For i = 1 To 3
            MsgBox("Row: " & CStr(myRS.Fields("ID").Value), _
                MsgBoxStyle.Information)
            If i = 2 Then
                myRS.Update("City", "Toronto")
                myRS.UpdateBatch()
            End If
            myRS.MoveNext()
        Next i
        myRS.Close()
    End If
    myConn.CommitTrans()
    myConn.Close()
    Exit Sub

HandleError:
    MsgBox(ErrorToString(Err.Number))
    Exit Sub

End Sub
```

説明

Recordset で adLockBatchOptimistic 設定を使用すると、myRS.Update メソッドはデータベース自体には何も変更を加えません。代わりに、Recordset のローカルコピーを更新します。

myRS.UpdateBatch メソッドはデータベースサーバに対して更新を実行しますが、コミットはしません。このメソッドは、トランザクションの内部で実行されるためです。トランザクションの外部で UpdateBatch メソッドを呼び出した場合、変更はコミットされます。

myConn.CommitTrans メソッドは、変更をコミットします。Recordset オブジェクトはこのときまでに閉じられているため、データのローカルコピーが変更されたかどうかが問題になることはありません。

ADO のトランザクション

デフォルトでは、ADO を使用したデータベースの変更は実行と同時にコミットされます。これには、明示的な更新、および Recordset の UpdateBatch メソッドも含まれます。しかし、前の項では、トランザクションを使用するために、

Connection オブジェクトで BeginTrans メソッドと RollbackTrans メソッドまたは CommitTrans メソッドを使用できると説明しました。

トランザクションの独立性レベルは、Connection オブジェクトのプロパティとして設定されます。IsolationLevel プロパティは、次の値のいずれかを取ることができます。

ADO 独立性レベル	定数	SAP Sybase IQ のレベル
未指定	adXactUnspecified	不適用。0 に設定する。
混沌	adXactChaos	サポートされない。0 に設定する。
参照	adXactBrowse	0
コミットされない読み出し	adXactReadUncommitted	0
カーソル安定性	adXactCursorStability	1
コミットされた読み出し	adXactReadCommitted	1
繰り返し可能読み出し	adXactRepeatableRead	2
独立	adXactIsolated	3
直列化可能	adXactSerializable	3
スナップショット	2097152	4
文のスナップショット	4194304	5
読み込み専用文のスナップショット	8388608	6

OLE DB 接続パラメータ

OLE DB 接続パラメータは、Microsoft が定義しています。SAP Sybase IQ OLE DB プロバイダは、これらの接続パラメータのサブセットをサポートしています。一般的な接続文字列は、次のようなものです。

```
"Provider=SAOLEDB;Data Source=myDsn;Initial Catalog=myDbn;
    User ID=myUid;Password=myPwd"
```

次に示すのは、プロバイダがサポートする OLE DB 接続パラメータです。ときには、OLE DB 接続パラメータが SAP Sybase IQ 接続パラメータと同一だったり (Password など)、類似していたり (User ID など) することもあります。これらの接続パラメータの多くでのスペースの使用に注意してください。

- **Provider –** このパラメータは、SQL Anywhere OLE DB プロバイダ (SAOLEDB) を特定するために使用されます。
- **User ID –** この接続パラメータは、SAP Sybase IQ の UserID (UID) 接続パラメータに直接マップします
- **Password –** この接続パラメータは、SAP Sybase IQ の Password (PWD) 接続パラメータに直接マップします
- **Data Source –** この接続パラメータは、SAP Sybase IQ の DataSourceName (DSN) 接続パラメータに直接マップします(例： `Data Source=Sybase IQ Demo`)。
- **Initial Catalog –** この接続パラメータは、SAP Sybase IQ の DatabaseName (DBN) 接続パラメータに直接マップします(例： `Initial Catalog=demo`)。
- **Location –** この接続パラメータは、SAP Sybase IQ の Host 接続パラメータに直接マップしますパラメータ値は Host パラメータ値と同じ形式になります(例： `Location=localhost:4444`)。
- **Extended Properties –** この接続パラメータは、SAP Sybase IQ に固有のすべての接続パラメータに引き渡すために、OLE DB によって使用されます(例： `Extended Properties="UserID=DBA;DBKEY=V3moj3952B;DBF=demo.db"`)。

  ADO は、この接続パラメータを使用して、認識しないすべての接続パラメータを収集して渡します。

  Microsoft の接続ウィンドウの中には、[Prov String] または [Provider String] というフィールドがあるものがあります。このフィールドの内容が、Extended Properties への値として渡されます。
- **OLE DB Services –** この接続パラメータは、SAP Sybase IQ OLE DB プロバイダからは直接処理されません。これは、ADO で接続プーリングを制御します。
- **Prompt –** この接続パラメータは、接続がエラーを処理する方法を管理します。可能なプロンプト値は、1、2、3、4 のいずれかです。その意味は、DBPROMPT_PROMPT (1)、DBPROMPT_COMPLETE (2)、DBPROMPT_COMPLETEREQUIRED (3)、DBPROMPT_NOPROMPT (4) です。

  デフォルトのプロンプト値は 4 であり、これは、プロバイダが接続ウィンドウに存在しないことを意味します。プロンプト値を 1 に設定すると、接続ウィンドウが常に表示されるようになります。プロンプト値を 2 に設定すると、初期接続試行が失敗した場合に接続ウィンドウが表示されるようになります。プロンプト値を 3 に設定すると、初期接続試行が失敗した場合に接続ウィンドウが表示されるようになりますが、プロバイダはデータソースへの接続には必要ない情報に対する制御を無効にします。
- **Window Handle –** アプリケーションは、適用できる場合、親ウィンドウのハンドルを渡すことができ、また、ウィンドウハンドルが適用できないか、プロバ

イダがどのウィンドウも表示しない場合、NULL ポインタを渡すことができます。ウィンドウハンドル値は通常は 0 (NULL) です。

その他の OLE DB 接続パラメータも指定できますが、OLE DB プロバイダからは無視されます。

SAP Sybase IQ OLE DB プロバイダは、起動されるときに、OLE DB 接続パラメータ用のプロパティ値を取得します。次に示すのは、Microsoft の RowsetViewer アプリケーションから取得した典型的なプロパティ値のセットです。

```
User ID '<user_id>'
Password '<password>'
Location 'localhost:4444'
Initial Catalog 'demo'
Data Source 'testds'
Extended Properties 'appinfo=api=oledb'
Prompt 2
Window Handle 0
```

このパラメータ値のセットからプロバイダが構築する接続文字列は、次のとおりです。

```
'DSN=testds;HOST=localhost:
4444;DBN=demo;UID=<user_id>;PWD=<password>;appinfo=api=oledb'
```

SAP Sybase IQ OLE DB プロバイダは、接続文字列、Window Handle、Prompt 値を、作成するデータベースサーバ接続呼び出しへのパラメータとして使用します。

これは、簡単な ADO 接続文字列の例です。

```
connection.Open
"Provider=SAOLEDB;UserID=<user_id>;Location=localhost:
4444;Pwd=<password>"
```

ADO は、接続文字列を解析して、認識できない接続パラメータすべてを Extended Properties で渡します。SAP Sybase IQ OLE DB プロバイダは、起動されるときに、OLE DB 接続パラメータ用のプロパティ値を取得します。次に示すのは、前述の接続文字列で使用された ADO アプリケーションから取得されたプロパティ値のセットです。

```
User ID ''
Password ''
Location 'localhost:4444'
Initial Catalog ''
Data Source ''
Extended Properties 'UserID=<user_id>;Pwd=<password>'
Prompt 4
Window Handle 0
```

このパラメータ値のセットからプロバイダが構築する接続文字列は、次のとおりです。

```
'HOST=localhost:4444; UserID=<user_id>;Pwd=<password>'
```

プロバイダは、接続文字列、Window Handle、Prompt 値を、作成するデータベースサーバ接続呼び出しへのパラメータとして使用します。

OLE DB 接続プーリング

OLE DB の .NET Framework データプロバイダは、OLE DB セッションプーリングを使用して接続を自動的にプールします。

アプリケーションが接続を閉じても、実際には接続は閉じません。代わりに、接続は一定時間保持されます。アプリケーションが接続を再度開くと、ADO/OLE DB では、アプリケーションが同じ接続文字列を使用することを認識し、開いている接続を再使用します。たとえば、アプリケーションで Open/Execute/Close を 100 回実行しても、実際には 1 回開いて、1 回閉じるのみです。最後の閉じる操作は、約 1 分のアイドル時間が経過した後で行われます。

接続が外的手段 (管理ツールを使用した強制切断など) で終了した場合は、サーバとの次の対話まで、この状態が発生したことが ADO/OLE DB によって把握されません。強制的に切断する場合は、十分に注意してください。

接続プーリングを制御するフラグは DBPROPVAL_OS_RESOURCEPOOLING (1) です。このフラグは、接続文字列の接続パラメータを使用してオフにできます。

接続文字列で `OLE DB Services=-2` と指定すると、接続プーリングが無効になります。サンプル接続文字列を以下に示します。

```
Provider=SAOLEDB;OLE DB Services=-2;...
```

接続文字列で `OLE DB Services=-4` と指定すると、接続プーリングとトランザクションのエンリストが無効になります。サンプル接続文字列を以下に示します。

```
Provider=SAOLEDB;OLE DB Services=-4;...
```

接続プーリングを無効にして、アプリケーションで同じ接続文字列を使用して接続を頻繁に開いたり閉じたりすると、パフォーマンスが低下します。

Microsoft リンクサーバ

SAP Sybase IQ OLE DB プロバイダを使用して SAP Sybase IQ データベースへのアクセスを取得する Microsoft リンクサーバを作成することができます。SQL クエリの発行には、Microsoft の 4 部構成のテーブル参照構文か Microsoft の OPENQUERY SQL 関数を使用できます。4 部構成構文の例を次に示します。

```
SELECT * FROM SADATABASE.demo.GROUPO.Customers
```

この例では、SADATABASE はリンクサーバの名前であり、demo はカタログまたはデータベース名であり、GROUPO は SAP Sybase IQ データベースのテーブル所有者であり、また、Customers は SAP Sybase IQ データベースのテーブル名です。

もう 1 つの例では、Microsoft の OPENQUERY 関数を使用しています。

```
SELECT * FROM OPENQUERY( SADATABASE, 'SELECT * FROM Customers' )
```

OPENQUERY 構文では、2 番目の SELECT 文 ('SELECT * FROM Customers') が SAP Sybase IQ サーバに渡され、実行されます。

複雑なクエリの場合、クエリ全体が SAP Sybase IQ サーバで評価されるため、OPENQUERY を使用する方が適しています。4 部構成の構文では、SQL Server は、クエリによって参照されるすべてのテーブルの内容を取得してから評価できるようになります (たとえば、WHERE、JOIN、ネストされたクエリによるクエリなど)。非常に大きいテーブルを必要とするクエリの場合、4 部構成の構文を使用すると処理時間が非常に長くなる可能性があります。次の 4 部構成のクエリの例では、SQL Server は OLE DB プロバイダ経由で、テーブル全体での単純な SELECT (WHERE 句なし) を SAP Sybase IQ データベースサーバに渡してから、WHERE 条件自体を評価します。

```
SELECT ID, Surname, GivenName FROM [SADATABASE].[demo].[GROUPO].
[Customers]
WHERE Surname = 'Elkins'
```

結果セットの 1 つのローを SQL Server に返す代わりに、すべてのローが返され、この結果セットは SQL Server によって 1 つのローに減らされます。次の例では同じ結果が生成されますが、1 つのローのみが SQL Server に返されます。

```
SELECT * FROM OPENQUERY( SADATABASE,
    'SELECT ID, Surname, GivenName FROM [GROUPO].[Customers]
    WHERE Surname = ''Elkins''' )
```

Microsoft SQL Server 対話型アプリケーションまたは SQL Server スクリプトを使用して、SAP Sybase IQ OLE DB プロバイダを使用するリンクサーバを設定できます。

注意： リンクサーバの設定の前に、Windows Vista 以降の Windows を使用するときに、2、3 考慮すべきことがあります。SQL Server は、システムのサービスとして実行します。Windows Vista 以降のバージョンでサービスが設定される方法に従い、サービスが共有メモリ接続を使用できない可能性、サーバを起動できない可能性、およびユーザデータソース定義にアクセスできない可能性があります。たとえば、[ネットワーク サービス] としてログインするサービスがサーバを起動できない、共有メモリ経由で接続できない、ユーザデータソースをアクセスできないといった場合があります。これらの状況において、SAP Sybase IQ サーバは、事前に起動させる必要があり、TCPIP 接続プロトコルを使用する必要があります。また、データソースが使用される予定の場合、システムデータソースにする必要があります。

インタラクティブなアプリケーションを使用したリンクサーバの設定

Microsoft SQL Server のインタラクティブなアプリケーションを使用して、SAP Sybase IQ OLE DB プロバイダを使用して SAP Sybase IQ データベースへのアクセスを取得する Microsoft リンクサーバを作成します。

前提条件

SQL Server 2000 以降。

手順

1. Microsoft SQL Server 2005/2008 の場合は、SQL Server Management Studio を起動します。他のバージョンの SQL Server の場合は、このアプリケーション名とリンクサーバの設定手順が異なることがあります。

   [オブジェクト エクスプローラー] ウィンドウ枠で、[サーバ オブジェクト] » [リンク サーバ] を展開します。[リンクサーバ] を右クリックし、[新しいリンクサーバ] をクリックします。
2. [全般] ページに必要な情報を入力します。

   [全般] ページの [リンク サーバ] フィールドに[リンクサーバ]名を指定します (上記の例では SADATABASE)。

   [その他のデータ ソース] オプションを選択して、[プロバイダ] リストから [SQL Anywhere OLE DB Provider 16] を選択します。

   [プロダクト名] フィールドには、任意の内容を入力できます (SAP Sybase IQ またはアプリケーション名など)。

   [データ ソース] フィールドには、ODBC データソース名を指定できます (DSN)。これは便利なオプションであり、また、データソース名は必須ではありません。システム DSN を使用する場合、SQL Server の 32 ビットバージョン用の 32 ビット DSN であるか、SQL Server の 64 ビットバージョンの 64 ビット DSN である必要があります。

   ```
   Data Source: SAP Sybase IQ 16 Demo
   ```

   [プロバイダ文字列] フィールドには、UserID (UID)、ServerName (Server)、DatabaseFile (DBF) などの追加の接続パラメータを含めることができます。

   ```
   Provider string: Server=myserver;DBF=sample.db
   ```

   [場所] フィールドは、SAP Sybase IQ Host 接続パラメータと同等のものを指定できます (localhost:4444 や 10.25.99.253:2638 など)。

   ```
   Location: AppServer-pc:2639
   ```

[初期カタログ] フィールドには、データベースの名前や接続先 (demo など) を含むことができます。データベースは、事前に開始しておく必要があります。

```
Initial Catalog: demo
```

これら最後の 4 つのフィールドと [セキュリティ] ページからのユーザ ID とパスワードの組み合わせは、データベースサーバへの接続に成功するための十分な情報を含む必要があります。

3. データベースのユーザ ID とパスワードは、プレーンテキストとして公開されてしまう [プロバイダ文字列] に接続文字列として指定する代わりに、[セキュリティ] ページで入力できます。

   SQL Server 2005/2008 では、[このセキュリティコンテキストを使用する] オプションをクリックし、[リモート ログイン] と [リモート パスワード] フィールドに値を入力します (パスワードはアスタリスクで表示されます)。

4. [サーバ オプション] ページに移動します。

   [RPC] および [RPC 出力] オプションを有効にします。

   選択方法は、Microsoft SQL Server のバージョンによって異なります。SQL Server 2000 の場合、この 2 つのオプションを指定するために 2 つのチェックボックスをオンにする必要があります。SQL Server 2005/2008 の場合、このオプションは True/False で設定します。すべて True に設定されていることを確認してください。ストアドプロシージャまたはファンクションの呼び出しを SAP Sybase IQ データベースで実行し、出入力パラメータの受け渡しを正常に行うには、[リモートプロシージャコール] ([RPC]) のオプションを選択する必要があります。

5. [InProcess 許可] プロバイダオプションを選択します。

   選択方法は、Microsoft SQL Server のバージョンによって異なります。SQL Server 2000 の場合、[プロバイダ オプション] ボタンをクリックすると、このオプションを選択できるページに移動します。SQL Server 2005/2008 の場合、[リンク サーバ] » [プロバイダ] の下で SAOLEDB.16 というプロバイダ名を右クリックし、[プロパティ] をクリックします。[Inprocess 許可] チェックボックスがオンになっていることを確認します。この [Inprocess] オプションがオンになっていないと、クエリが失敗します。

6. 他のプロバイダオプションは無視できます。これらのオプションのいくつかは SQL Server の下位互換性に関連しており、SQL Server が SAP Sybase IQ OLE DB プロバイダと対話する方法に影響を与えません。例は[ネストされたクエリ]と[LIKE 演算子のサポート]です。他のオプションを選択した場合、構文エラーまたはパフォーマンスの低下が発生することがあります。

Microsoft リンクサーバが設定されます。

スクリプトを使用したリンクサーバの設定

リンクサーバの定義は、SQL Server スクリプトを使用して設定できます。

前提条件

SQL Server 2005 以降。

手順

次の手順を使用して次のスクリプトに必要な変更を加えた後、SQL Server で実行してください。

```
USE [master]
GO
EXEC master.dbo.sp_addlinkedserver @server=N'SADATABASE',
     @srvproduct=N'SAP Sybase IQ', @provider=N'SAOLEDB.16',
     @datasrc=datasrc=N'Sybase IQ Demo',
     @provstr=N'host=localhost:4444;server=myserver;dbn=demo'
GO
EXEC master.dbo.sp_serveroption @server=N'SADATABASE',
     @optname=N'rpc', @optvalue=N'true'
GO
EXEC master.dbo.sp_serveroption @server=N'SADATABASE',
     @optname=N'rpc out', @optvalue=N'true'
GO
-- Set remote login
EXEC master.dbo.sp_addlinkedsrvlogin @rmtsrvname = N'SADATABASE',
     @locallogin = NULL , @useself = N'False',
     @rmtuser = N'DBA', @rmtpassword = N'sql'
GO
-- Set global provider "allow in process" flag
EXEC master.dbo.sp_MSset_oledb_prop N'SAOLEDB.16',
N'AllowInProcess', 1
```

1. 新しいリンクサーバ名を選択します (例では SADATABASE が使用されています)。
2. オプションのデータソース名を選択します (例では SAP Sybase IQ 16 Demo が使用されています)。
3. オプションのプロバイダ文字列を選択します (例では `N'host=localhost:4444;server=myserver;dbn=demo'` が使用されています)。
4. リモートユーザ ID とパスワードを選択します (例では N'DBA' と N'sql' が使用されています)。

変更されたスクリプトを Microsoft SQL Server で実行し、新しいリンクサーバを作成できます。

サポートされる OLE DB インタフェース

OLE DB API はインタフェースのセットで構成されています。次の表は SQL Anywhere の OLE DB ドライバにある各インタフェースのサポートを示します。

インタフェース	目的	制限事項
IAccessor	クライアントのメモリとデータストアの値のバインドを定義する。	DBACCESSOR_PASSBYREF はサポートされない。DBACCESSOR_OPTIMIZED はサポートされない。
IAlterIndex IAlterTable	テーブル、インデックス、カラムを変更する。	サポートされない。
IChapteredRowset	区分化されたローセットで、ローセットのローを別々の区分でアクセスできる。	サポートされない。SAP Sybase IQ では、区分化されたローセットはサポートされない。
IColumnsInfo	ローセットのカラムについての簡単な情報を得る。	サポートされる。
IColumnsRowset	ローセットにあるオプションのメタデータカラムについての情報を得て、カラムメタデータのローセットを取得する。	サポートされる。
ICommand	SQL 文を実行する。	設定できなかったプロパティを見つけるための、DBPROPSET_PROPERTIESINERROR による ICommandProperties::GetProperties の呼び出しは、サポートされない。

インタフェース	目的	制限事項
ICommandPersist	command オブジェクトの状態を保持する (アクティブなローセットは保持しない)。保持されているこれらの command オブジェクトは、PROCEDURES か VIEWS ローセットを使用すると、続けて列挙できる。	サポートされる。
ICommandPrepare	コマンドを準備する。	サポートされる。
ICommandProperties	コマンドが作成したローセットに、Rowset プロパティを設定する。ローセットがサポートするインタフェースを指定するのに、最も一般的に使用される。	サポートされる。
ICommandText	ICommand に SQL 文を設定する。	DBGUID_DEFAULT SQL ダイアレクトのみサポートされる。
ICommandWithParameters	コマンドに関するパラメータ情報を、設定または取得する。	スカラ値のベクトルとして格納されているパラメータは、サポートされない。 BLOB パラメータはサポートされない。
IConvertType		サポートされる。
IDBAsynchNotify IDBAsynchStatus	非同期処理。 データソース初期化の非同期処理、ローセットの移植などにおいて、クライアントにイベントを通知する。	サポートされない。
IDBCreateCommand	セッションからコマンドを作成する。	サポートされる。
IDBCreateSession	データソースオブジェクトからセッションを作成する。	サポートされる。

インタフェース	目的	制限事項
IDBDataSourceAd-min	データソースオブジェクトを作成／破壊／修正する。このオブジェクトはクライアントによって使用される COM オブジェクトである。このインタフェースは、データストア (データベース) の管理には使用されない。	サポートされない。
IDBInfo	このプロバイダにとってユニークなキーワードについての情報を検索する (非標準の SQL キーワードを検索する)。 また、テキスト一致クエリで使用されるリテラルや特定の文字、その他のリテラル情報についての情報を検索する。	サポートされる。
IDBInitialize	データソースオブジェクトと列挙子を初期化する。	サポートされる。
IDBProperties	データソースオブジェクトまたは列挙子のプロパティを管理する。	サポートされる。
IDBSchemaRowset	標準フォーム (ローセット) にあるシステムテーブルの情報を取得する。	サポートされる。
IErrorInfo IErrorLookup IErrorRecords	ActiveX エラーオブジェクトサポート。	サポートされる。

インタフェース	目的	制限事項
IGetDataSource	インタフェースポインタを、セッションのデータソースオブジェクトに返す。	サポートされる。
IIndexDefinition	データストアにインデックスを作成または削除する。	サポートされない。
IMultipleResults	コマンドから複数の結果 (ローセットやローカウント) を取り出す。	サポートされる。
IOpenRowset	名前でデータベーステーブルにアクセスする非 SQL 的な方法。	サポートされる。 名前でテーブルを開くことはサポートされるが、GUID で開くことはサポートされない。
IParentRowset	区分化／階層ローセットにアクセスする。	サポートされない。
IRowset	ローセットにアクセスする。	サポートされる。
IRowsetChange	ローセットデータへの変更を許し、変更をデータストアに反映させる。 BLOB に対する InsertRow/SetData は実装されていない。	サポートされる。
IRowsetChapterMember	区分化／階層ローセットにアクセスする。	サポートされない。
IRowsetCurrentIndex	ローセットのインデックスを動的に変更する。	サポートされない。
IRowsetFind	指定された値と一致するローを、ローセットの中から検索する。	サポートされない。
IRowsetIdentity	ローのハンドルを比較する。	サポートされない。

インタフェース	目的	制限事項
IRowsetIndex	データベースインデックスにアクセスする。	サポートされない。
IRowsetInfo	ローセットプロパティについての情報を検索する、または、ローセットを作成したオブジェクトを検索する。	サポートされる。
IRowsetLocate	ブックマークを使用して、ローセットのローを検索する。	サポートされる。
IRowsetNotify	ローセットのイベントに COM コールバックインタフェースを提供する。	サポートされる。
IRowsetRefresh	トランザクションで参照可能な最後のデータの値を取得する。	サポートされない。
IRowsetResynch	以前の OLE DB 1.x のインタフェースで、IRowsetRefresh に変更された。	サポートされない。
IRowsetScroll	ローセットをスクロールして、ローデータをフェッチする。	サポートされない。
IRowsetUpdate	Update が呼ばれるまで、ローセットデータの変更を遅らせる。	サポートされる。
IRowsetView	既存のローセットにビューを使用する。	サポートされない。
ISequentialStream	BLOB カラムを取り出す。	読み出しのみのサポート。 このインタフェースを使用した SetData はサポートされない。
ISessionProperties	セッションプロパティ情報を取得する。	サポートされる。

インタフェース	目的	制限事項
ISourcesRowset	データソースオブジェクトと列挙子のローセットを取得する。	サポートされる。
ISQLErrorInfo ISupportErrorInfo	ActiveX エラーオブジェクトサポート。	サポートされる。
ITableDefinition ITableDefinitionWith-Constraints	制約を使用して、テーブルを作成、削除、変更する。	サポートされる。
ITransaction	トランザクションをコミットまたはアボートする。	すべてのフラグがサポートされているわけではない。
ITransactionJoin	分散トランザクションをサポートする。	すべてのフラグがサポートされているわけではない。
ITransactionLocal	セッションでトランザクションを処理する。 すべてのフラグがサポートされているわけではない。	サポートされる。
ITransactionOptions	トランザクションでオプションを取得または設定する。	サポートされる。
IViewChapter	既存のローセットでビューを使用する。特に、後処理フィルタやローのソートを適用するために利用される。	サポートされない。
IViewFilter	ローセットの内容を、一連の条件と一致するローに制限する。	サポートされない。
IViewRowset	ローセットを開くときに、ローセットの内容を、一連の条件と一致するローに制限する。	サポートされない。
IViewSort	ソート順をビューに適用する。	サポートされない。

OLE DB プロバイダの登録

SAP Sybase IQ のインストーラを使用して SAOLEDB プロバイダがインストールされるときに、SAOLEDB プロバイダはそれ自体を登録します。この登録プロセスには、レジストリの COM セクションにレジストリエントリを作成することも含まれます。このエントリによって、ADO は SAOLEDB プロバイダが呼び出されたときに DLL を見つけることができます。DLL のロケーションを変更した場合は、それを再度登録する必要があります。

例

プロバイダがインストールされているディレクトリから次のコマンドを実行すると、SAP Sybase IQ OLE DB プロバイダが登録されます。

```
regsvr32 dboledb16.dll
regsvr32 dboledba16.dll
```

ODBC CLI

ODBC (Open Database Connectivity) は、Microsoft が開発した標準 CLI (コールレベルインタフェース) です。SQL Access Group CLI 仕様に基づいています。ODBC アプリケーションは、ODBC ドライバを提供するあらゆるデータソースに使用できます。ODBC ドライバを持つ他のデータソースにアプリケーションを移植できるようにしたい場合は、プログラミングインタフェースとして ODBC を使用することをおすすめします。

ODBC 準拠

SQL Anywhere は、Microsoft Data Access Kit 2.7 の一部として提供されている ODBC 3.5 をサポートしています。

ODBC のサポートレベル

ODBC の機能は、準拠のレベルによって異なります。機能は**コア**、**レベル 1**、または**レベル 2** のいずれかです。レベル 2 は ODBC を完全にサポートします。これらの機能は、http://msdn.microsoft.com/ja-jp/library/ms714177.aspx にある Microsoft の『ODBC Programmer's Reference』にリストされています。

SQL Anywhere がサポートする機能

SQL Anywhere での ODBC 3.5 仕様のサポートは、次のとおりです。

- **コア準拠 –** SQL Anywhere は、コアレベルの機能をすべてサポートします。
- **レベル 1 準拠 –** SQL Anywhere は、ODBC 関数の非同期実行を除いてレベル 1 の機能をすべてサポートします。

  SQL Anywhere は、単一の接続を共有するマルチスレッドをサポートします。各スレッドからの要求は、SQL Anywhere によって直列化されます。
- **レベル 2 準拠 –** SQL Anywhere は、次の機能を除いてレベル 2 の機能をすべてサポートします。
  - 3 語で構成されるビュー名とテーブル名。この名前は SQL Anywhere では使用できません。
  - 特定の独立した文についての ODBC 関数の非同期実行。
  - ログイン要求と SQL クエリをタイムアウトする機能。

ODBC アプリケーションの開発

ODBC 関数を呼び出す C/C++ ソースファイルには、プラットフォーム固有の ODBC ヘッダファイルが必要です。各プラットフォーム固有のヘッダファイルは、ODBC のメインヘッダファイル `odbc.h` を含みます。このヘッダファイルには、ODBC プログラムの作成に必要な関数、データ型、定数定義がすべて含まれています。

C/C++ ソースファイルに ODBC ヘッダファイルをインクルードするには、次のタスクを実行します。

1. ソースファイルに、該当するプラットフォーム固有のヘッダファイルを参照するインクルード行を追加します。使用する行は次のとおりです。

オペレーティングシステム	インクルード行
Windows	#include "ntodbc.h"
Unix	#include "unixodbc.h"
Windows Mobile	#include "ntodbc.h"

2. ヘッダファイルがあるディレクトリを、コンパイラのインクルードパスに追加します。
   プラットフォーム固有のヘッダファイルと `odbc.h` は、どちらも SQL Anywhere インストールディレクトリの `SDK¥Include` サブフォルダにインストールされます。
3. UNIX 用の ODBC アプリケーションを構築するときは、正しいデータアラインメントとサイズを取得するために、32 ビットのアプリケーションの場合はマクロ "UNIX"、64 ビットのアプリケーションの場合は "UNIX64" を定義する必要があることがあります。ただし、次に示すサポートされるコンパイラのいずれかを使用している場合は、マクロの定義は不要です。

   サポートされるプラットフォームにインストールされている GNU C/C++ コンパイラ
   Linux 用の Intel C/C++ コンパイラ (icc)
   Linux または Solaris 用の SunPro C/C++ コンパイラ
   AIX 用の VisualAge C/C++ コンパイラ
   HP-UX 用の C/C++ コンパイラ (cc/aCC)

ソースコードが書き込まれると、アプリケーションをコンパイルしてリンクできます。次の項では、実行アプリケーションを作成する方法について説明します。

Windows での ODBC アプリケーション

アプリケーションをリンクする場合は、ODBC の関数にアクセスできるように、適切なインポートライブラリファイルにリンクします。

インポートライブラリでは、ODBC ドライバマネージャ `odbc32.dll` のエントリポイントが定義されます。このドライバマネージャは、SAP Sybase IQ の ODBC ドライバ `dbodbc16.dll` をロードします。

通常、インポートライブラリは Microsoft プラットフォーム SDK の `Lib` ディレクトリ構造下に格納されています。

オペレーティングシステム	インポートライブラリ
Windows (32 ビット)	`¥Lib¥X86¥odbc32.lib`
Windows (64 ビット)	`¥Lib¥X86¥odbc32.lib`

例

次のコマンドは、プラットフォーム固有のインポートライブラリがあるディレクトリを、LIB 環境変数内のライブラリディレクトリのリストに追加する方法を示します。

```
set LIB=%LIB%;c:¥mssdk¥v7.0¥lib
```

次のコマンドは、Microsoft のコンパイルおよびリンクツールを使用して、`odbc.c` に保存されたアプリケーションをコンパイルしてリンクする方法を示します。

```
cl odbc.c /I"%IQDIR16%¥SDK¥Lib¥X86¥Include" odbc32.lib
```

UNIX での ODBC アプリケーション

UNIX 用の ODBC ドライバマネージャは SQL Anywhere に同梱されており、サードパーティ製のドライバマネージャも利用できます。この項では、ODBC ドライバマネージャを使用しない ODBC アプリケーションの構築方法について説明します。

ODBC ドライバ

ODBC ドライバは、共有オブジェクトまたは共有ライブラリです。シングルスレッドアプリケーションとマルチスレッドアプリケーションには、SQL Anywhere ODBC ドライバの別々のバージョンが用意されています。汎用の SQL Anywhere ODBC ドライバは、使用中のスレッドモデルを検出し、シングルスレッドまたはマルチスレッドのライブラリを直接呼び出します。

ODBC ドライバのファイルは、次のとおりです。

オペレーティングシステム	スレッドモデル	ODBC ドライバ
(HP-UX 以外のすべての UNIX)	汎用	`libdbodbc16.so (libdbodbc16.so.1)`
(HP-UX 以外のすべての UNIX)	シングルスレッド	`libdbodbc16_n.so (libdbodbc16_n.so.1)`
(HP-UX 以外のすべての UNIX)	マルチスレッド	`libdbodbc16_r.so (libdbodbc16_r.so.1)`
HP-UX	汎用	`libdbodbc16.sl (libdbodbc16.sl.1)`
HP-UX	シングルスレッド	`libdbodbc16_n.sl (libdbodbc16_n.sl.1)`
HP-UX	マルチスレッド	`libdbodbc16_r.sl (libdbodbc16_r.sl.1)`

ライブラリは、バージョン番号 (カッコ内に表示) を使用して共有ライブラリへのシンボリックリンクとしてインストールされます。

UNIX で ODBC アプリケーションをリンクする場合、アプリケーションを汎用 ODBC ドライバ `libdbodbc16` に対してリンクします。アプリケーションを配備するときに、適切な (またはすべての) ODBC ドライババージョン (非スレッドまたはスレッド) がユーザのライブラリパスに含まれていることを確認します。

データソース情報
SQL Anywhere で ODBC ドライバマネージャの存在が検出されない場合は、データソース情報にシステム情報ファイルを使用します。

unixODBC ドライバマネージャ

バージョン 2.2.14 より前の unixODBC では、64 ビット ODBC 仕様の一部が Microsoft の規定とは異なって実装されています。この違いにより、unixODBC ドライバマネージャを SQL Anywhere 64 ビット ODBC ドライバで使用すると問題が生じます。

このような問題を回避するためには、両者の違いについて認識する必要があります。相違点の 1 つとして挙げられるのが、SQLLEN と SQLULEN の定義です。SQLLEN と SQLULEN は、Microsoft 64 ビット ODBC 仕様では 64 ビット型であり、SQL Anywhere 64 ビット ODBC ドライバでも 64 ビット数として処理されることを想定しています。unixODBC の一部の実装では、これらの 2 つを 32 ビット数とし

て定義しているため、SQL Anywhere 64 ビット ODBC ドライバとインタフェースする際に問題が生じます。

64 ビットのプラットフォームで問題を回避するには、次の 3 つの処理が必要になります。

1. `sql.h` や `sqlext.h` などの unixODBC ヘッダをインクルードする代わりに、SQL Anywhere ODBC ヘッダファイルの `unixodbc.h` をインクルードします。これにより、SQLLEN および SQLULEN は正しく定義されます。unixODBC 2.2.14 以降のバージョンのヘッダファイルでは、この問題が修正されています。
2. すべてのパラメータで正しい型を使用していることを確認する必要があります。正しいヘッダファイルや C/C++ コンパイラの強力な型チェックを使用すると便利です。また、SQL Anywhere ドライバがポインタを介して間接的に設定したすべての変数についても、正しい型を使用していることを確認する必要があります。
3. リリース 2.2.14 より前の unixODBC ドライバマネージャは使用せずに、SQL Anywhere ODBC ドライバに直接リンクしてください。たとえば、libodbc 共有オブジェクトを SQL Anywhere ドライバにリンクします。

   ```
   libodbc.so.1 -> libdbodbc16_r.so.1
   ```

   プラットフォームによっては、SQL Anywhere ドライバマネージャを代わりに使用できます。

UNIX 用 UTF-32ODBC ドライバマネージャ

ワイド呼び出しをサポートする SQL Anywhere ODBC ドライバは 16 ビットの SQLWCHAR 用に構築されているため、SQLWCHAR を 32 ビット (UTF-32) の数量として定義する ODBC ドライバマネージャのバージョンは、SQL Anywhere ODBC ドライバで使用できません。このような場合に備えて ANSI 専用バージョンの SQL Anywhere ODBC ドライバが用意されています。このバージョンの ODBC ドライバでは、ワイド呼び出しインタフェース (SQLConnectW など) はサポートされていません。

ドライバの共有オブジェクト名は `libdbodbcansi16_r` です。ドライバのスレッド変形のみが提供されています。Real Basic などの特定のフレームワークは dylib で動作しないため、バンドルが必要になります。

通常の ODBC ドライバでは、SQLWCHAR 文字列が UTF-16 文字列として処理されます。このドライバは、iODBC などの、SQLWCHAR 文字列を UTF-32 文字列として処理する一部の ODBC ドライバマネージャでは使用できません。Unicode 対応のドライバを扱う場合、これらのドライバマネージャでは、アプリケーションからのナロー呼び出しがドライバへのワイド呼び出しに変換されます。ANSI 専用ドライバではこの動作が回避されます。このため、アプリケーションがワイド呼び

出しを行わないかぎり、ANSI 専用ドライバをこのようなドライバマネージャで使用することができます。iODBC を介したワイド呼び出しや、同様のセマンティックを持つ他のドライバマネージャは、引き続きサポートされません。

ODBC のサンプル

SAP Sybase IQ には、ODBC のサンプルがいくつか用意されています。サンプルは %ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥C ディレクトリ (Windows) または $SYBASE/IQ-16_0/samples/sqlanywhere/c ディレクトリ (UNIX) にあります。

ディレクトリ内の ODBC で始まるサンプルは、データベースへの接続や文の実行など、簡単な ODBC 作業をそれぞれ示します。完全なサンプル ODBC プログラムは、odbc.c ファイルにあります。このプログラムの動作は、同じディレクトリにある Embedded SQL 動的 CURSOR のサンプルプログラムと同じです。

Windows 用の ODBC サンプルプログラムの構築

サンプル ODBC プログラムを構築することでプログラムを実行でき、データベースへの接続や文の実行など、ODBC タスクがどのように実行されるのかを確認できます。

前提条件

x64 プラットフォームのビルドでは、コンパイルとリンクに適した環境を設定する必要があります。x64 プラットフォーム用のサンプルプログラムを構築するコマンド例を次に示します。

```
set mssdk=c:¥mssdk¥v7.0
build64
```

手順

%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥C ディレクトリにあるバッチファイルを使用して、すべてのサンプルアプリケーションをコンパイルしてリンクできます。

1. コマンドプロンプトを開き、%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥C ディレクトリに移動します。
2. build.bat または build64.bat バッチファイルを実行します。

サンプル ODBC プログラムが構築されます。

UNIX 用の ODBC サンプルプログラムの構築

サンプル ODBC プログラムを構築することでプログラムを実行でき、データベースへの接続や文の実行など、ODBC タスクがどのように実行されるのかを確認できます。

前提条件

この作業を実行するための前提条件はありません。

手順

`$SYBASE/IQ-16_0/samples/sqlanywhere/c` ディレクトリにあるシェルスクリプトを使用して、すべてのサンプルアプリケーションをコンパイルしてリンクできます。

1. コマンドシェルを開き、`$SYBASE/IQ-16_0/samples/sqlanywhere/c` ディレクトリに移動します。
2. `build.sh` シェルスクリプトを実行します。

サンプル ODBC プログラムが構築されます。

ODBC サンプルプログラム

適切なプラットフォームでファイルを実行し、サンプル ODBC プログラムをロードできます。

- 32 ビットの Windows では、`%ALLUSERSPROFILE%¥SybaseIQ¥samples¥sqlanywhere¥C¥odbcwin.exe` を実行します。
- 64 ビットの Windows では、`%ALLUSERSPROFILE%¥SybaseIQ¥samples¥sqlanywhere¥C¥odbcx64.exe` を実行します。
- UNIX では、`$SYBASE/IQ-16_0/samples/sqlanywhere/C/odbc` を実行します。

ファイルを実行後、サンプルデータベースのいずれかのテーブルを選択します。たとえば、`Customers` または `Employees` を入力します。

ODBC ハンドル

ODBC アプリケーションは、小さい**ハンドル**セットを使用して、データベース接続や SQL 文などの基本的な機能を定義します。ハンドルは、32 ビット値です。

次のハンドルは、事実上すべての ODBC アプリケーションで使用されます。

- **環境** – 環境ハンドルは、データにアクセスするグローバルコンテキストを提供します。すべての ODBC アプリケーションは、起動時に環境ハンドルを 1 つだけ割り付け、アプリケーションの終了時にそれを解放します。

  次のコードは、環境ハンドルを割り付ける方法を示します。

  ```
  SQLRETURN rc;
  SQLHENV env;
  rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
  ```

- **接続** – 接続は、ODBC ドライバとデータソースによって指定されます。アプリケーションは、その環境に対応する接続を複数確立できます。接続ハンドルを割り付けても、接続は確立されません。最初に接続ハンドルを割り付けてから、接続の確立時に使用します。

  次のコードは、接続ハンドルを割り付ける方法を示します。

  ```
  SQLRETURN rc;
  SQLHDBC  dbc;
  rc = SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
  ```

- **文** – ステートメントハンドルを使って、SQL 文と、結果セットやパラメータなどの関連情報へアクセスできます。接続ごとに複数の文を使用できます。文は、カーソル処理 (データのフェッチ) と単一の文の実行 (INSERT、UPDATE、DELETE など) の両方に使用されます。

  次のコードは、ステートメントハンドルを割り付ける方法を示します。

  ```
  SQLRETURN rc;
  SQLHSTMT stmt;
  rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
  ```

ODBC ハンドルを割り付ける方法

ODBC プログラムに必要なハンドルの型は、次のとおりです。

項目	ハンドルの型
環境	SQLHENV
接続	SQLHDBC
文	SQLHSTMT
記述子	SQLHDESC

ODBC ハンドルを使用するには、次のタスクを実行します。

1. SQLAllocHandle 関数を呼び出します。
2. 後続の関数呼び出しでハンドルを使用します。
3. SQLFreeHandle を使用してオブジェクトを解放します。

SQLAllocHandle は、次のパラメータを取ります。

- 割り付ける項目の型を示す識別子
- 親項目のハンドル
- 割り付けるハンドルのロケーションへのポインタ
  詳細については、http://msdn.microsoft.com/ja-jp/library/ms712455.aspx にある Microsoft の『ODBC API Reference』の「SQLAllocHandle」を参照してください。

SQLFreeHandle は、次のパラメータを取ります。

- 解放する項目の型を示す識別子
- 解放する項目のハンドル
  詳細については、http://msdn.microsoft.com/ja-jp/library/ms710123.aspx にある Microsoft の『ODBC API Reference』の「SQLFreeHandle」を参照してください。

例

次のコードフラグメントは、環境ハンドルを割り付け、解放します。

```
SQLRETURN rc;
SQLHENV env;
rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
if( rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO )
{
    .
    .
    .
}
SQLFreeHandle( SQL_HANDLE_ENV, env );
```

ODBC のサンプル

%IQDIRSAMP16%¥SQLAnywhere¥ODBCConnect¥odbcconnect.cpp にある次の簡単な ODBC プログラムは、SQL Anywhere サンプルデータベースに接続し、直後に切断します。この例では、データベースサーバに接続するための環境の設定に必要な手順のほか、サーバから切断してリソースを解放するのに必要な手順も示しています。

ODBC 接続関数

ODBC には、一連の接続関数が用意されています。どの接続関数を使用するかは、アプリケーションの配備方法と使用方法によって決まります。

- **SQLConnect –** 最も簡単な接続関数です。

  SQLConnect は、データソース名と、オプションでユーザ ID とパスワードをパラメータに取ります。データソース名をアプリケーションにハードコードする場合は、SQLConnect を使用します。

詳細については、http://msdn.microsoft.com/en-us/library/ms711810.aspx にある Microsoft の『ODBC API Reference』の「SQLConnect」を参照してください。

- **SQLDriverConnect –** 接続文字列を使用してデータソースに接続します。

  SQLDriverConnect を使用すると、アプリケーションはデータソースの外部にある SAP Sybase IQ 固有の接続情報を使用できます。また、Sybase IQ ODBC ドライバに対して接続情報を確認するように要求できます。

  データソースを指定しないで接続することもできます。代わりに、Sybase IQ ODBC ドライバ名が指定されます。次の例では、すでに実行されているサーバとデータベースに接続します。

```
SQLSMALLINT cso;
SQLCHAR     scso[2048];

SQLDriverConnect( hdbc, NULL,
   "Driver=Sybase IQ;UID=<user_id>;PWD=<password>", SQL_NTS,
   scso, sizeof(scso)-1,
   &cso, SQL_DRIVER_NOPROMPT );
```

  詳細については、http://msdn.microsoft.com/en-us/library/ms715433.aspx にある Microsoft の『ODBC API Reference』の「SQLDriverConnect」を参照してください。

- **SQLBrowseConnect –** SQLDriverConnect と同様に、接続文字列を使用してデータソースに接続します。

  SQLBrowseConnect を使用すると、アプリケーションは独自のウィンドウを構築して、接続情報を要求するプロンプトを表示したり、特定のドライバ (この場合は Sybase IQ ODBC ドライバ) で使用するデータソースを参照したりできます。

  詳細については、http://msdn.microsoft.com/en-us/library/ms714565.aspx にある Microsoft の『ODBC API Reference』の「SQLBrowseConnect」を参照してください。

ODBC 接続の確立

アプリケーションに ODBC 接続を確立し、データベース操作を行います。

前提条件

この作業を実行するための前提条件はありません。

手順

完全なサンプルは、`%ALLUSERSPROFILE%¥SybaseIQ¥samples¥ODBCConnect¥odbcconnect.cpp` にあります。

1. ODBC 環境を割り付けます。

   次に例を示します。

   ```
   SQLRETURN rc;
   SQLHENV   env;
   rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
   ```

2. ODBC のバージョンを宣言します。

   アプリケーションが ODBC バージョン 3 に準拠するように宣言すると、SQLSTATE 値と他のバージョン依存の機能が適切な動作に設定されます。次に例を示します。

   ```
   rc = SQLSetEnvAttr( env, SQL_ATTR_ODBC_VERSION,
   (void*)SQL_OV_ODBC3, 0 );
   ```

3. ODBC 接続項目を割り付けます。

   次に例を示します。

   ```
   rc = SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
   ```

4. 接続前に必要な接続属性を設定します。

   接続属性には、接続を確立する前または後に必ず設定するものと、確立前に設定しても後に設定してもかまわないものがあります。SQL_AUTOCOMMIT 属性は、接続の確立前にでも後にでも設定できる属性です。

   ```
   rc = SQLSetConnectAttr( dbc, SQL_AUTOCOMMIT,
   (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0 );
   ```

   デフォルトでは、ODBC はオートコミットモードで動作します。このモードは、SQL_AUTOCOMMIT を false に設定してオフにすることができます。

5. 必要な場合は、データソースまたは接続文字列をアセンブルします。

   アプリケーションによっては、データソースや接続文字列をハードコードしたり、柔軟性を高めるために外部に格納したりできます。

6. ODBC 接続関数を呼び出します。

   次に例を示します。

   ```
   if (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO)
   {
      printf( "dbc allocated¥n" );
      rc = SQLConnect( dbc,
         (SQLCHAR *) "Sybase IQ Demo", SQL_NTS,
         (SQLCHAR *) "<user_id>", SQL_NTS,
         (SQLCHAR *) "<password>", SQL_NTS );
      if (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO)
      {
          // Successfully connected.
   ```

ODBC に渡される各文字列には、固有の長さがあります。長さがわからない場合は、終端を NULL 文字 (¥0) でマークした *NULL で終了された文字列*であることを示す SQL_NTS を渡すことができます。

アプリケーションをビルドして実行すると、ODBC 接続が確立されます。

ODBC によって変更されるサーバオプション

SQL Anywhere ODBC ドライバは、SQL Anywhere データベースへの接続時に一部のテンポラリサーバオプションを設定します。次のオプションが、以下で示すように設定されます。

- **date_format –** yyyy-mm-dd
- **date_order –** ymd
- **isolation_level –** SQLSetConnectAttr の SQL_ATTR_TXN_ISOLATION/ SA_SQL_ATTR_TXN_ISOLATION 属性の設定に基づく。次のオプションを使用できます。

```
SQL_TXN_READ_UNCOMMITTED
SQL_TXN_READ_COMMITTED
SQL_TXN_REPEATABLE_READ
SQL_TXN_SERIALIZABLE
SA_SQL_TXN_SNAPSHOT
SA_SQL_TXN_STATEMENT_SNAPSHOT
SA_SQL_TXN_READONLY_STATEMENT_SNAPSHOT
```

- **time_format –** hh:nn:ss
- **timestamp_format –** yyyy-mm-dd hh:nn:ss.ssssss
- **timestamp_with_time_zone_format –** yyyy-mm-dd hh:nn:ss.ssssss +hh:nn

デフォルトのオプション設定に戻すには、SET 文を実行します。次に、timestamp_format オプションをリセットする文の例を示します。

```
set temporary option timestamp_format =
```

SQLSetConnectAttr 拡張接続属性

SQL Anywhere ODBC ドライバは、拡張された一部の接続属性をサポートしています。

- **SA_REGISTER_MESSAGE_CALLBACK –** メッセージは、SQL MESSAGE 文を使用してクライアントアプリケーションからデータベースサーバに送信できます。実行時間が長いデータベースサーバ文によってメッセージも生成できます。

  メッセージハンドラルーチンを作成して、これらのメッセージを捕捉できます。メッセージハンドラのコールバックプロトタイプを次に示します。

```
void SQL_CALLBACK message_handler(
SQLHDBC sqlany_dbc,
unsigned char  msg_type,
long code,
unsigned short  length,
char * message
);
```

msg_type に指定できる次の値は、sqldef.h で定義されています。

- **MESSAGE_TYPE_INFO** – メッセージタイプは INFO でした。
- **MESSAGE_TYPE_WARNING** – メッセージタイプは WARNING でした。
- **MESSAGE_TYPE_ACTION** – メッセージタイプは ACTION でした。
- **MESSAGE_TYPE_STATUS** – メッセージタイプは STATUS でした。
- **MESSAGE_TYPE_PROGRESS** – メッセージタイプは PROGRESS でした。このタイプのメッセージは、BACKUP DATABASE や LOAD TABLE などの実行時間が長いデータベースサーバ文によって生成されます。

メッセージに関連付けられている SQLCODE を *code* に指定することができます。指定がない場合、*code* パラメータの値は 0 です。

メッセージの長さは *length* に記述されています。

メッセージへのポインタは *message* に記述されています。メッセージ文字列は、NULL で終了しません。この問題を処理するようにアプリケーションを設計する必要があります。次はその例です。

```
memcpy( mybuff, msg, len );
mybuff[ len ] = '¥0';
```

メッセージハンドラを ODBC に登録するには、次のようにして SQLSetConnectAttr 関数を呼び出します。

```
rc = SQLSetConnectAttr(
   hdbc,
   SA_REGISTER_MESSAGE_CALLBACK,
   (SQLPOINTER) &message_handler, SQL_IS_POINTER );
```

メッセージハンドラの登録を ODBC から解除するには、次のようにして SQLSetConnectAttr 関数を呼び出します。

```
rc = SQLSetConnectAttr(
   hdbc,
   SA_REGISTER_MESSAGE_CALLBACK,
   NULL, SQL_IS_POINTER );
```

- **SA_GET_MESSAGE_CALLBACK_PARM** – メッセージハンドラのコールバックルーチンに渡される SQLHDBC 接続ハンドルの値を取得するには、SA_GET_MESSAGE_CALLBACK_PARM パラメータを指定して SQLGetConnectAttr 関数を呼び出します。

```
SQLHDBC callback_hdbc = NULL;
rc = SQLGetConnectAttr(
```

```
    hdbc,
    SA_GET_MESSAGE_CALLBACK_PARM,
    (SQLPOINTER) &callback_hdbc, 0, 0 );
```

戻り値は、メッセージハンドラのコールバックルーチンに渡されるパラメータ値と同じです。

- **SA_REGISTER_VALIDATE_FILE_TRANSFER_CALLBACK –** これは、ファイル転送の検証コールバック関数を登録するために使用します。転送を許可する前に、ODBC ドライバは検証コールバックが存在する場合は、それを呼び出します。ストアドプロシージャからなどの間接文の実行中にクライアントのデータ転送が要求された場合、ODBC ドライバはクライアントアプリケーションで検証コールバックが登録されていないかぎり転送を許可しません。どのような状況で検証の呼び出しが行われるかについては、以下でより詳しく説明します。

  コールバックプロトタイプを次に示します。

```
int SQL_CALLBACK file_transfer_callback(
void * sqlca,
char * file_name,
int is_write
);
```

  file_name パラメータは、読み込みまたは書き込み対象のファイルの名前です。*is_write* パラメータは、読み込み (クライアントからサーバへの転送) が要求された場合は 0、書き込みが要求された場合は 0 以外の値になります。ファイル転送が許可されない場合、コールバック関数は 0 を返します。それ以外の場合は 0 以外の値を返します。

  データのセキュリティ上、サーバはファイル転送を要求している文の実行元を追跡します。サーバは、文がクライアントアプリケーションから直接受信されたものかどうかを判断します。クライアントからデータ転送を開始する際に、サーバは文の実行元に関する情報をクライアントソフトウェアに送信します。クライアント側では、クライアントアプリケーションから直接送信された文を実行するためにデータ転送が要求されている場合にかぎり、ODBC ドライバはデータの転送を無条件で許可します。それ以外の場合は、上述の検証コールバックがアプリケーションで登録されていることが必要です。登録されていない場合、転送は拒否されて文が失敗し、エラーが発生します。データベース内に既存しているストアドプロシージャがクライアントの文で呼び出された場合、ストアドプロシージャそのものの実行はクライアントの文で開始されたものと見なされません。ただし、クライアントアプリケーションでテンポラリストアドプロシージャを明示的に作成してストアドプロシージャを実行した場合、そのプロシージャはクライアントによって開始されたものとしてサーバは処理します。同様に、クライアントアプリケーションでバッチ文を実行する場合も、バッチ文はクライアントアプリケーションによって直接実行されるものと見なされます。

- **SA_SQL_ATTR_TXN_ISOLATION –** これは、拡張されたトランザクションの独立性レベルを設定するために使用します。次の例は、Snapshot 独立性レベルを設定します。

```
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLSetConnectAttr( dbc, SA_SQL_ATTR_TXN_ISOLATION,
      SA_SQL_TXN_SNAPSHOT, SQL_IS_UINTEGER );
```

64 ビット ODBC での考慮事項

SQLBindCol、SQLBindParameter、SQLGetData などの ODBC 関数を使用する場合、一部のパラメータは SQLLEN や SQLULEN として関数プロトタイプに型指定されます。参照している Microsoft の『ODBC API Reference』マニュアルによっては、同じパラメータが SQLINTEGER や SQLUINTEGER として記述されている場合があります。

SQLLEN および SQLULEN のデータ項目は、64 ビットの ODBC アプリケーションでは 64 ビット、32 ビットの ODBC アプリケーションでは 32 ビットになります。SQLINTEGER および SQLUINTEGER のデータ項目は、すべてのプラットフォームで 32 ビットです。

この問題を説明するために、次の ODBC 関数プロトタイプを Microsoft の旧版の『ODBC API Reference』から抜粋しました。

```
SQLRETURN SQLGetData(
     SQLHSTMT     StatementHandle,
     SQLUSMALLINT ColumnNumber,
     SQLSMALLINT  TargetType,
     SQLPOINTER   TargetValuePtr,
     SQLINTEGER   BufferLength,
     SQLINTEGER  *StrLen_or_IndPtr);
```

Microsoft Visual Studio バージョン 8 の `sql.h` にある実際の関数プロトタイプと比較してください。

```
SQLRETURN  SQL_API SQLGetData(
    SQLHSTMT      StatementHandle,
    SQLUSMALLINT  ColumnNumber,
    SQLSMALLINT   TargetType,
    SQLPOINTER    TargetValue,
    SQLLEN        BufferLength,
    SQLLEN       *StrLen_or_Ind);
```

BufferLength パラメータと StrLen_or_Ind パラメータが、SQLINTEGER 型ではなく SQLLEN 型と指定されるようになったことがわかります。64 ビットのプラットフォームでは、32 ビット数ではなく 64 ビット数であることが Microsoft のマニュアルからわかります。

異種プラットフォーム間でのコンパイルの問題を回避するために、SQL Anywhere には独自の ODBC ヘッダファイルがあります。Windows プラットフォームの場合は、`ntodbc.h` ヘッダファイルをインクルードしてください。Linux などの UNIX プラットフォームの場合は、`unixodbc.h` ヘッダファイルをインクルードしてください。これらのヘッダファイルを使用することで、対象プラットフォーム用の SQL Anywhere ODBC ドライバとの互換性が確保されます。

次の表に示すのは、一般的な ODBC のタイプの一部です。64 ビットのプラットフォームと 32 ビットのプラットフォームでストレージサイズが同じものもあれば、異なるものもあります。

ODBC API	64 ビットのプラットフォーム	32 ビットのプラットフォーム
SQLINTEGER	32 ビット	32 ビット
SQLUINTEGER	32 ビット	32 ビット
SQLLEN	64 ビット	32 ビット
SQLULEN	64 ビット	32 ビット
SQLSETPOSIROW	64 ビット	16 ビット
SQL_C_BOOKMARK	64 ビット	32 ビット
BOOKMARK	64 ビット	32 ビット

データ変数とパラメータを間違って宣言すると、ソフトウェアが正しく動作しない可能性があります。

次の表は、64 ビットのサポートの導入とともに変更された、ODBC API の関数プロトタイプを示します。影響を受けるパラメータが記載されています。関数プロトタイプで使用される実際のパラメータ名と Microsoft のマニュアルに記載されているパラメータ名が異なる場合は、Microsoft の記載名がカッコ内に示されています。パラメータ名は、Microsoft Visual Studio バージョン 8 のヘッダファイルで使用されるものです。

ODBC API	パラメータ (マニュアル記載のパラメータ名)
SQLBindCol	SQLLEN BufferLength SQLLEN *Strlen_or_Ind
SQLBindParam	SQLULEN LengthPrecision SQLLEN *Strlen_or_Ind

ODBC API	パラメータ (マニュアル記載のパラメータ名)
SQLBindParameter	SQLULEN cbColDef (ColumnSize) SQLLEN cbValueMax (BufferLength) SQLLEN *pcbValue (Strlen_or_IndPtr)
SQLColAttribute	SQLLEN *NumericAttribute
SQLColAttributes	SQLLEN *pfDesc
SQLDescribeCol	SQLULEN *ColumnSize (ColumnSizePtr)
SQLDescribeParam	SQLULEN *pcbParamDef (ParameterSizePtr)
SQLExtendedFetch	SQLLEN irow (FetchOffset) SQLULEN *pcrow (RowCountPtr)
SQLFetchScroll	SQLLEN FetchOffset
SQLGetData	SQLLEN BufferLength SQLLEN *Strlen_or_Ind (Strlen_or_IndPtr)
SQLGetDescRec	SQLLEN *Length (LengthPtr)
SQLParamOptions	SQLULEN crow, SQLULEN *pirow
SQLPutData	SQLLEN Strlen_or_Ind
SQLRowCount	SQLLEN *RowCount (RowCountPtr)
SQLSetConnectOption	SQLULEN Value
SQLSetDescRec	SQLLEN Length SQLLEN *StringLength (StringLengthPtr) SQLLEN *Indicator (IndicatorPtr)
SQLSetParam	SQLULEN LengthPrecision SQLLEN *Strlen_or_Ind (Strlen_or_IndPtr)
SQLSetPos	SQLSETPOSIROW irow (RowNumber)
SQLSetScrollOptions	SQLLEN crowKeyset
SQLSetStmtOption	SQLULEN Value

ポインタを介してODBC API呼び出しに渡され、ODBC API呼び出しから返される値の一部は、64 ビットのアプリケーションに対応するために変更されました。た

とえば、次の SQLSetStmtAttr および SQLSetDescField 関数の値は、SQLINTEGER/SQLUINTEGER ではなくなりました。SQLGetStmtAttr および SQLGetDescField 関数の該当するパラメータに関しても同様です。

ODBC API	Value/ValuePtr 変数の型
SQLSetStmtAttr(SQL_ATTR_FETCH_BOOKMARK_PTR)	SQLLEN * value
SQLSetStmtAttr(SQL_ATTR_KEYSET_SIZE)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_MAX_LENGTH)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_MAX_ROWS)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_PARAM_BIND_OFFSET_PTR)	SQLULEN * value
SQLSetStmtAttr(SQL_ATTR_PARAMS_PROCESSED_PTR)	SQLULEN * value
SQLSetStmtAttr(SQL_ATTR_PARAMSET_SIZE)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_ROW_ARRAY_SIZE)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_ROW_BIND_OFFSET_PTR)	SQLULEN * value
SQLSetStmtAttr(SQL_ATTR_ROW_NUMBER)	SQLULEN value
SQLSetStmtAttr(SQL_ATTR_ROWS_FETCHED_PTR)	SQLULEN * value
SQLSetDescField(SQL_DESC_ARRAY_SIZE)	SQLULEN value
SQLSetDescField(SQL_DESC_BIND_OFFSET_PTR)	SQLLEN * value
SQLSetDescField(SQL_DESC_ROWS_PROCESSED_PTR)	SQLULEN * value
SQLSetDescField(SQL_DESC_DISPLAY_SIZE)	SQLLEN value
SQLSetDescField(SQL_DESC_INDICATOR_PTR)	SQLLEN * value
SQLSetDescField(SQL_DESC_LENGTH)	SQLLEN value
SQLSetDescField(SQL_DESC_OCTET_LENGTH)	SQLLEN value
SQLSetDescField(SQL_DESC_OCTET_LENGTH_PTR)	SQLLEN * value

詳細については、http://support.microsoft.com/kb/298678 にある Microsoft のアーティクル『ODBC 64-Bit API Changes in MDAC 2.7』を参照してください。

データアラインメントの要件

SQLBindCol、SQLBindParameter、または SQLGetData を使用する場合、カラムまたはパラメータには C データ型が指定されます。プラットフォームによっては、

指定された型の値をフェッチまたは格納するために、各カラム用のストレージ (メモリ) を適切にアラインする必要があります。ODBC ドライバは、データアラインメントが適切かどうかを確認します。オブジェクトが適切に揃っていない場合、ODBC ドライバは ["Invalid string or buffer length"] (無効な文字列またはバッファ長) というメッセージ ([SQLSTATE] HY090 または S1090) を出力します。

次の表は、Sun Sparc、Itanium-IA64、ARM ベースのデバイスなどのプロセッサに対するメモリアラインメント要件を示したものです。データ値のメモリアドレスは、示された値の倍数である必要があります。

C データ型	必要なアラインメント
SQL_C_CHAR	なし
SQL_C_BINARY	なし
SQL_C_GUID	なし
SQL_C_BIT	なし
SQL_C_STINYINT	なし
SQL_C_UTINYINT	なし
SQL_C_TINYINT	なし
SQL_C_NUMERIC	なし
SQL_C_DEFAULT	なし
SQL_C_SSHORT	2
SQL_C_USHORT	2
SQL_C_SHORT	2
SQL_C_DATE	2
SQL_C_TIME	2
SQL_C_TIMESTAMP	2
SQL_C_TYPE_DATE	2
SQL_C_TYPE_TIME	2
SQL_C_TYPE_TIMESTAMP	2
SQL_C_WCHAR	2 (すべてのプラットフォームでバッファサイズは 2 の倍数であることが必要)
SQL_C_SLONG	4
SQL_C_ULONG	4

C データ型	必要なアラインメント
SQL_C_LONG	4
SQL_C_FLOAT	4
SQL_C_DOUBLE	8 (ARM の場合は 4)
SQL_C_SBIGINT	8
SQL_C_UBIGINT	8

x86、x64、および PowerPC プラットフォームではメモリアラインメントは必要ありません。x64 プラットフォームには、Advanced Micro Devices (AMD) AMD64 プロセッサや Intel Extended Memory 64 Technology (EM64T) プロセッサなどがあります。

ODBC アプリケーションの結果セット

ODBC アプリケーションは、結果セットの操作と更新にカーソルを使用します。SQL Anywhere は、多種多様なカーソルとカーソル処理をサポートしています。

ODBC トランザクションの独立性レベル

SQLSetConnectAttr を使用して、接続に関するトランザクションの独立性レベルを設定できます。SQL Anywhere に用意されているトランザクションの独立性レベルを決定する特性は、次のとおりです。

- **SQL_TXN_READ_UNCOMMITTED –** 独立性レベルを 0 に設定します。この属性値を設定すると、別のユーザによる変更から読み込まれたデータは分離され、その変更内容は表示されません。READ 文の再実行は別のユーザによって影響されます。繰り返し可能読み出しはサポートされていません。これは独立性レベルのデフォルト値です。
- **SQL_TXN_READ_COMMITTED –** 独立性レベルを 1 に設定します。この属性値を設定すると、別のユーザによる変更から読み込まれたデータは分離されず、その変更内容は表示されます。READ 文の再実行は別のユーザによって影響されます。繰り返し可能読み出しはサポートされていません。
- **SQL_TXN_REPEATABLE_READ –** 独立性レベルを 2 に設定します。この属性値を設定すると、別のユーザによる変更から読み込まれたデータは分離され、その変更内容は表示されません。READ 文の再実行は別のユーザによって影響されます。繰り返し可能読み出しはサポートされています。
- **SQL_TXN_SERIALIZABLE –** 独立性レベルを 3 に設定します。この属性値を設定すると、別のユーザによる変更から読み込まれたデータは分離され、その変更内容は表示されません。READ 文の再実行は別のユーザによって影響されません。繰り返し可能読み出しはサポートされています。

- **SA_SQL_TXN_SNAPSHOT –** 独立性レベルを Snapshot に設定します。この属性値を設定すると、トランザクション全体のデータベースに関する単一ビューが表示されます。
- **SA_SQL_TXN_STATEMENT_SNAPSHOT –** 独立性レベルを Statement-snapshot に設定します。この属性値を設定すると、Snapshot 独立性よりデータの整合性は低くなりますが、トランザクションを長時間実行したためにバージョン情報を格納するテンポラリファイルのサイズが大きくなりすぎる場合には有益です。
- **SA_SQL_TXN_READONLY_STATEMENT_SNAPSHOT –** 独立性レベルを Readonly-statement-snapshot に設定します。この属性値を設定すると、Statement-snapshot 独立性よりデータの整合性は低くなりますが、更新の競合は回避されます。このため、この属性は元々異なる独立性レベルで実行することを想定していたアプリケーションを移植するのに最も適しています。

このオプションを Snapshot、Statement-snapshot、または Readonly-statement-snapshot に設定する場合は、allow_snapshot_isolation データベースオプションを On に設定する必要があります。

詳細については、http://msdn.microsoft.com/ja-jp/library/ms713605.aspx にある Microsoft の『ODBC API Reference』の「SQLSetConnectAttr」を参照してください。

例

次のフラグメントは、独立性レベルを Snapshot に設定します。

```
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLSetConnectAttr( dbc, SQL_ATTR_TXN_ISOLATION,
      SA_SQL_TXN_SNAPSHOT, SQL_IS_UINTEGER );
```

ODBC カーソル特性

文を実行して結果セットを操作する ODBC 関数は、カーソルを使用してタスクを実行します。アプリケーションは SQLExecute または SQLExecDirect 関数を実行するたびに、暗黙的にカーソルを開きます。

結果セットを前方にのみ移動し、更新はしないアプリケーションの場合、カーソルの動作は比較的単純です。ODBC アプリケーションは、デフォルトではこの動作を要求します。ODBC は読み込み専用で前方専用のカーソルを定義します。この場合、SQL Anywhere ではパフォーマンスが向上するように最適化されたカーソルが提供されます。

多くのグラフィカルユーザインタフェースアプリケーションのように、結果セット内で前後にスクロールする必要のあるアプリケーションの場合、カーソルの動作はもっと複雑です。アプリケーションが、他のアプリケーションによって更新されたローに戻るときの動作を考えてみます。ODBC は、アプリケーションに適した動作を組み込めるように、さまざまな**スクロール可能カーソル**を定義してい

ます。SQL Anywhere には、ODBC のスクロール可能カーソルタイプに適合するカーソルのフルセットが用意されています。

必要な ODBC カーソル特性を設定するには、文の属性を定義する SQLSetStmtAttr 関数を呼び出します。SQLSetStmtAttr は、結果セットを作成する文の実行前に呼び出してください。

SQLSetStmtAttr を使用すると、多数のカーソル特性を設定できます。SQL Anywhere に用意されているカーソルタイプを決定する特性は、次のとおりです。

- **SQL_ATTR_CURSOR_SCROLLABLE –** スクロール可能カーソルの場合は SQL_SCROLLABLE、前方専用カーソルの場合は SQL_NONSCROLLABLE に設定します。SQL_NONSCROLLABLE がデフォルトです。
- **SQL_ATTR_CONCURRENCY –** 次のいずれかの値に設定します。
  - **SQL_CONCUR_READ_ONLY –** 更新禁止になります。SQL_CONCUR_READ_ONLY がデフォルトです。
  - **SQL_CONCUR_LOCK –** ローを確実に更新できるロックの最下位レベルを使用します。
  - **SQL_CONCUR_ROWVER –** SQLBase ROWID または Sybase TIMESTAMP などのローバージョンを比較して、最適の同時制御を使用します。
  - **SQL_CONCUR_VALUES –** 値を比較して、最適の同時制御を使用します。

詳細については、http://msdn.microsoft.com/ja-jp/library/ms712631.aspx にある Microsoft の『ODBC API Reference』の「SQLSetStmtAttr」を参照してください。

例

次のフラグメントは、読み込み専用のスクロール可能カーソルを要求します。

```
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
SQLSetStmtAttr( stmt, SQL_ATTR_CURSOR_SCROLLABLE,
      SQL_SCROLLABLE, SQL_IS_UINTEGER );
```

データの取得

データベースからローを取り出すには、SQLExecute または SQLExecDirect を使用して SELECT 文を実行します。これで文のカーソルが開きます。

次に、SQLFetch または SQLFetchScroll を使用し、カーソルを介してローをフェッチします。これらの関数では、結果セットから次のローセットのデータをフェッチし、バインドされているすべてのカラムのデータを返します。SQLFetchScroll を使用すると、ローセットを絶対位置や相対位置で指定したり、ブックマークによって指定できます。ODBC 2.0 仕様の古い SQLExtendedFetch は、SQLFetchScroll に置き換えられました。

アプリケーションは、SQLFreeHandle を使用して文を解放するときにカーソルを閉じます。

カーソルから値をフェッチするため、アプリケーションは SQLBindCol か SQLGetData のいずれかを使用します。SQLBindCol を使用すると、フェッチのたびに値が自動的に取り出されます。SQLGetData を使用する場合は、フェッチ後にカラムごとに呼び出してください。

LONG VARCHAR または LONG BINARY などのカラムの値を分割してフェッチするには、SQLGetData を使用します。または、SQL_ATTR_MAX_LENGTH 文の属性を、カラムの値全体を十分に保持できる大きさの値に設定する方法もあります。SQL_ATTR_MAX_LENGTH のデフォルト値は 256 KB です。

SQL Anywhere ODBC ドライバは、ODBC 仕様で意図されたものとは異なる方法で SQL_ATTR_MAX_LENGTH を実装しています。本来 SQL_ATTR_MAX_LENGTH は、大きなフェッチをトランケートするメカニズムとして使用されることを意図しています。この処理は、データの最初の部分だけを表示するプレビューモードで行われる可能性があります。たとえば、4 MB の blob をサーバからクライアントアプリケーションに転送するのではなく、その先頭 500 バイトだけが転送される可能性があります (SQL_ATTR_MAX_LENGTH が 500 に設定された場合)。SQL Anywhere ODBC ドライバでは、この実装をサポートしていません。

次のコードフラグメントは、クエリに対してカーソルを開き、そのカーソルを介してデータを取り出します。わかりやすくするためにエラーチェックは省いています。このフラグメントは、完全なサンプルから抜粋したものです。サンプルは、%IQDIRSAMP16%¥SQLAnywhere¥ODBCSelect¥odbcselect.cpp にあります。

```
SQLINTEGER cbDeptID = 0, cbDeptName = SQL_NTS, cbManagerID = 0;
SQLCHAR deptName[ DEPT_NAME_LEN + 1 ];
SQLSMALLINT deptID, managerID;
SQLHENV  env;
SQLHDBC  dbc;
SQLHSTMT stmt;
SQLRETURN rc;

SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env );
SQLSetEnvAttr( env,
            SQL_ATTR_ODBC_VERSION,
            (void *)SQL_OV_ODBC3, 0 );
SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
SQLConnect( dbc,
            (SQLCHAR *) "SQL Anywhere 16 Demo", SQL_NTS,
            (SQLCHAR *) "DBA", SQL_NTS,
            (SQLCHAR *) "sql", SQL_NTS );
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
SQLBindCol( stmt, 1,
            SQL_C_SSHORT, &deptID, 0, &cbDeptID );
SQLBindCol( stmt, 2,
            SQL_C_CHAR, deptName,
            sizeof(deptName), &cbDeptName );
SQLBindCol( stmt, 3,
            SQL_C_SSHORT, &managerID, 0, &cbManagerID );
SQLExecDirect( stmt, (SQLCHAR * )
```

```
    "SELECT DepartmentID, DepartmentName, DepartmentHeadID "
    "FROM Departments "
    "ORDER BY DepartmentID", SQL_NTS );
while( ( rc = SQLFetch( stmt ) ) != SQL_NO_DATA )
{
   printf( "%d %20s %d¥n", deptID, deptName, managerID );
}
SQLFreeHandle( SQL_HANDLE_STMT, stmt );
SQLDisconnect( dbc );
SQLFreeHandle( SQL_HANDLE_DBC, dbc );
SQLFreeHandle( SQL_HANDLE_ENV, env );
```

カーソルでフェッチできるローの位置番号は、integer 型のサイズによって管理されます。32 ビット integer に格納できる値より 1 小さい 2147483646 までの番号が付けられたローをフェッチできます。ローの位置番号に、クエリ結果の最後を基準として負の数を使用している場合、integer に格納できる負の最大値より 1 大きい数までの番号のローをフェッチできます。

カーソルを使用したローの更新と削除

Microsoft の『ODBC Programmer's Reference』では、クエリが位置付けオペレーションを使用して更新可能であることを示すために、SELECT...FOR UPDATE を使用するように提案しています。SQL Anywhere では、FOR UPDATE 句を使用する必要はありません。次の条件が満たされている場合は、SELECT 文が自動的に更新可能になります。

- 基本となるクエリが更新をサポートしている。
  つまり、結果のカラムに対するデータ操作文が有効であるかぎり、位置付けデータ操作文をカーソルに対して実行できます。
  ansi_update_constraints データベースオプションは更新可能なクエリの種類を制限します。
- カーソルタイプが更新をサポートしている。
  読み込み専用カーソルを使用している場合、結果セットを更新できません。

ODBC で位置付け更新と位置付け削除を実行するには、2 つの手段があります。

- SQLSetPos 関数を使用する。
  指定されたパラメータ (SQL_POSITION、SQL_REFRESH、SQL_UPDATE、SQL_DELETE) に応じて、SQLSetPos はカーソル位置を設定し、アプリケーションがデータをリフレッシュしたり、結果セットのデータを更新または削除できるようにします。
  これは、SQL Anywhere で使用する方法です。
- SQLExecute を使用して、位置付け UPDATE 文と位置付け DELETE 文を送信する。この方法は、SQL Anywhere では使用しないでください。

ブックマーク

ODBC には**ブックマーク**があります。これはカーソル内のローの識別に使用する値です。SQL Anywhere は、value-sensitive と insensitive カーソルにブックマークをサポートします。たとえば、ODBC カーソルタイプの SQL_CURSOR_STATIC と SQL_CURSOR_KEYSET_DRIVEN ではブックマークをサポートしますが、SQL_CURSOR_DYNAMIC と SQL_CURSOR_FORWARD_ONLY ではブックマークをサポートしていないということです。

ODBC 3.0 より前のバージョンでは、データベースはブックマークをサポートするかどうかを指定するだけであり、カーソルタイプごとにブックマークの情報を提供するインタフェースはありませんでした。このため、サポートされているカーソルブックマークの種類を示す手段が、データベースサーバにはありませんでした。ODBC 2 アプリケーションでは、SQL Anywhere はブックマークをサポートしています。したがって、動的カーソルにブックマークを使用することもできますが、これは実行しないでください。

ストアドプロシージャの考慮事項

この項では、ODBC アプリケーションからストアドプロシージャを作成して呼び出し、その結果を処理する方法について説明します。

プロシージャと結果セット

プロシージャには、結果セットを返すものと返さないものの 2 種類があります。SQLNumResultCols を使用すると、そのどちらであるかを確認できます。プロシージャが結果セットを返さない場合は、結果カラムの数が 0 になります。結果セットがある場合は、他のカーソルの場合と同様に、SQLFetch または SQLExtendedFetch を使用して値をフェッチできます。

プロシージャへのパラメータは、パラメータマーカ (疑問符) を使用して渡してください。INPUT、OUTPUT、または INOUT パラメータのいずれについても、SQLBindParameter を使用して各パラメータマーカ用の記憶領域を割り当てます。

複数の結果セットを処理するために ODBC は、プロシージャが定義した結果セットではなく、現在実行中のカーソルを記述します。したがって、ODBC はストアドプロシージャ定義の RESULT 句で定義されているカラム名を常に記述するわけではありません。この問題を回避するため、プロシージャ結果セットのカーソルでカラムのエイリアスを使用できます。

例 1

この例では、結果セットを返さないプロシージャを作成して呼び出します。このプロシージャは、INOUT パラメータを 1 つ受け取り、その値を増分します。この例では、プロシージャが結果セットを返さないため、変数 num_columns の値は 0 になります。わかりやすくするためにエラーチェックは省いています。

```
HDBC dbc;
SQLHSTMT stmt;
SQLINTEGER I;
SQLSMALLINT num_columns;

SQLAllocStmt( dbc, &stmt );
SQLExecDirect( stmt,
    "CREATE PROCEDURE Increment( INOUT a INT )"
    "BEGIN "
    "    SET a = a + 1 "
    "END", SQL_NTS );

/* Call the procedure to increment 'I' */
I = 1;
SQLBindParameter( stmt, 1, SQL_C_LONG, SQL_INTEGER, 0, 0, &I, NULL );
SQLExecDirect( stmt, "CALL Increment( ? )", SQL_NTS );
SQLNumResultCols( stmt, &num_columns );
```

例 2

この例では、結果セットを返すプロシージャを呼び出しています。ここでは、プロシージャが 2 つのカラムの結果セットを返すため、変数 num_columns の値は 2 になります。わかりやすくするため、エラーチェックは省略しています。

```
SQLRETURN rc;
SQLHDBC dbc;
SQLHSTMT stmt;
SQLSMALLINT num_columns;

SQLCHAR ID[ 10 ];
SQLCHAR Surname[ 20 ];

SQLExecDirect( stmt,
      "CREATE PROCEDURE EmployeeList() "
      "RESULT( ID CHAR(10), Surname CHAR(20) ) "
      "BEGIN "
      "    SELECT EmployeeID, Surname FROM Employees "
      "END", SQL_NTS );

/* Call the procedure - print the results */
SQLExecDirect( stmt, "CALL EmployeeList()", SQL_NTS );
SQLNumResultCols( stmt, &num_columns );
SQLBindCol( stmt, 1, SQL_C_CHAR, &ID, sizeof(ID), NULL );
SQLBindCol( stmt, 2, SQL_C_CHAR, &Surname, sizeof(Surname), NULL );

for( ;; )
{
   rc = SQLFetch( stmt );
```

```
   if( rc == SQL_NO_DATA_FOUND )
   {
      rc = SQLMoreResults( stmt );
      if( rc == SQL_NO_DATA_FOUND ) break;
   }
   else
   {
      do_something( ID, Surname );
   }
}
```

ODBC エスケープ構文

ODBC エスケープ構文は、任意の ODBC アプリケーションで使用できます。エスケープ構文を使用して、使用しているデータベース管理システムとは関係なく、共通の関数セットを呼び出すことができます。エスケープ構文の一般的な形式は次のようになります。

{ `keyword` *parameters* }

次のキーワードセットがあります。

- **{d date-string} –** date-string は、SQL Anywhere が受け取ることのできる任意の日付値です。
- **{t time-string} –** time-string は、SQL Anywhere が受け取ることのできる任意の時刻値です。
- **{ts date-string time-string} –** date-string time-string は、SQL Anywhere が受け取ることのできる任意のタイムスタンプ値です。
- **{guid uuid-string} –** uuid-string は、任意の有効な GUID 文字列です (例：41dfe9ef-db91-11d2-8c43-006008d26a6f)。
- **{oj outer-join-expr} –** outer-join-expr は、SQL Anywhere が受け取ることのできる有効な OUTER JOIN 式です。
- **{? = call func(p1,...)} –** func は、SQL Anywhere が受け取ることのできる任意の有効な関数呼び出しです。
- **{call proc(p1,...)} –** proc は、SQL Anywhere が受け取ることのできる任意の有効なストアドプロシージャ呼び出しです。
- **{fn func(p1,...)} –** func は、以下に示すいずれかの関数ライブラリです。

エスケープ構文を使用して、ODBC ドライバによって実装される関数ライブラリにアクセスできます。このライブラリには、数値、文字列、時刻、日付、システム関数が含まれています。

たとえば、次のコマンドを実行すると、データベース管理システムの種類にかかわらず現在の日付を取得できます。

```
SELECT { FN CURDATE() }
```

次の表は、SQL Anywhere ODBC によってサポートされている関数を示します。

SQL Anywhere ODBC ドライバでサポートされている関数

数値関数	文字列関数	システム関数	日付／時刻関数
ABS	ASCII	DATABASE	CURDATE
ACOS	BIT_LENGTH	IFNULL	CURRENT_DATE
ASIN	CHAR	USER	CURRENT_TIME
ATAN	CHAR_LENGTH	CONVERT	CURRENT_TIMESTAMP
ATAN2	CHARACTER_LENGTH		CURTIME
CEILING	CONCAT		DAYNAME
COS	DIFFERENCE		DAYOFMONTH
COT	INSERT		DAYOFWEEK
DEGREES	LCASE		DAYOFYEAR
EXP	LEFT		EXTRACT
FLOOR	LENGTH		HOUR
LOG	LOCATE		MINUTE
LOG10	LTRIM		MONTH
MOD	OCTET_LENGTH		MONTHNAME
PI	POSITION		NOW
POWER	REPEAT		QUARTER
RADIANS	REPLACE		SECOND
RAND	RIGHT		WEEK
ROUND	RTRIM		YEAR
SIGN	SOUNDEX		
SIN	SPACE		
SQRT	SUBSTRING		
TAN	UCASE		
TRUNCATE			

ODBC エスケープ構文は JDBC エスケープ構文と同じです。JDBC を使用する Interactive SQL では、大カッコ ({) は*必ず*二重にしてください。カッコの間にスペースを入れないでください。"{{" は使用できますが、"{ {" は使用できません。また、文中に改行文字を使用できません。ストアドプロシージャは Interactive SQL で解析されないため、ストアドプロシージャではエスケープ構文を使用できません。

たとえば、SQL エスケープ構文を使用して sa_db_info プロシージャを持つデータベースプロパティを取得するには、InteractiveSQL で次のコマンドを実行します。

```
{{CALL sa_db_info( 0 ) }}
```

ODBC のエラー処理

ODBC のエラーは、各 ODBC 関数呼び出しからの戻り値と SQLError 関数または SQLGetDiagRec 関数を使用してレポートされます。SQLError 関数は、バージョン 3 よりも前の ODBC で使用されていました。バージョン 3 では、SQLError 関数は使用されなくなり、SQLGetDiagRec 関数が代わりに使用されるようになりました。

すべての ODBC 関数は、次のステータスコードのいずれかの SQLRETURN を返します。

ステータスコード	説明
SQL_SUCCESS	エラーはありません。
SQL_SUCCESS_WITH_INFO	関数は完了しましたが、SQLError を呼び出すと警告が示されます。 このステータスは、返される値が長すぎてアプリケーションが用意したバッファに入りきらない場合によく使用されます。
SQL_ERROR	関数はエラーのため完了しませんでした。SQLError を呼び出すと、エラーに関する詳細な情報を取得できます。
SQL_INVALID_HANDLE	パラメータとして渡された環境、接続、またはステートメントハンドルが不正です。 このステータスは、すでに解放済みのハンドルを使用した場合、あるいはハンドルが NULL ポインタである場合によく使用されます。
SQL_NO_DATA_FOUND	情報がありません。 このステータスは、カーソルからフェッチするときに、カーソルにそれ以上ローがないことを示す場合によく使用されます。

ステータスコード	説明
SQL_NEED_DATA	パラメータにデータが必要です。 これは、SQLParamData と SQLPutData の ODBC SDK マニュアルで説明されている高度な機能です。

あらゆる環境、接続、文のハンドルに対して、エラーまたは警告が 1 つ以上発生する可能性があります。SQLError または SQLGetDiagRec を呼び出すたびに、1 つのエラーに関する情報が返され、その情報が削除されます。SQLError または SQLGetDiagRec を呼び出してすべてのエラーを削除しなかった場合は、同じハンドルをパラメータに取る関数が次に呼び出された時点で、残ったエラーが削除されます。

SQLError の各呼び出しで、環境、接続、文に対応する 3 つのハンドルを渡します。最初の呼び出しは、SQL_NULL_HSTMT を使用して接続に関するエラーを取得しています。同様に、SQL_NULL_DBC と SQL_NULL_HSTMT を同時に使用して呼び出すと、環境ハンドルに関するエラーが取得されます。

SQLGetDiagRec を呼び出すたびに、環境、接続、または文のハンドルを渡すことができます。最初の呼び出しでは、型 SQL_HANDLE_DBC のハンドルを渡して、接続に関連するエラーを取得します。2 つ目の呼び出しでは、型 SQL_HANDLE_STMT のハンドルを渡して、直前に実行した文に関連するエラーを取得します。

エラー (SQL_ERROR *以外*) があるうちは SQL_SUCCESS が返され、エラーがなくなると SQL_NO_DATA_FOUND が返されます。

例 1

次のコードフラグメントは SQLError とリターンコードを使用しています。

```
SQLRETURN rc;
SQLHDBC dbc;
SQLHSTMT stmt;
UCHAR errmsg[100];

rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
if( rc == SQL_ERROR )
{
   SQLError( env, dbc, SQL_NULL_HSTMT, NULL, NULL,
         errmsg, sizeof(errmsg), NULL );
   print_error( "Allocation failed", errmsg );
   return;
}

/* Delete items for order 2015 */
rc = SQLExecDirect( stmt,
       "DELETE FROM SalesOrderItems WHERE ID=2015",
       SQL_NTS );
if( rc == SQL_ERROR )
```

```
{
   SQLError( env, dbc, stmt, NULL, NULL,
            errmsg, sizeof(errmsg), NULL );
   print_error( "Failed to delete items", errmsg );
   return;
}
```

例 2

次のコードフラグメントは SQLGetDiagRec とリターンコードを使用しています。

```
SQLRETURN rc;
SQLHDBC dbc;
SQLHSTMT stmt;
SQLSMALLINT errmsglen;
SQLINTEGER errnative;
SQLCHAR errmsg[255];
SQLCHAR errstate[5];

rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
if( rc == SQL_ERROR )
{
   SQLGetDiagRec( SQL_HANDLE_DBC, dbc, 1, errstate,
       &errnative, errmsg, sizeof(errmsg), &errmsglen );
   print_error( "Allocation failed", errstate, errnative, errmsg );
   return;
}

rc = SQLExecDirect( stmt,
       "DELETE FROM SalesOrderItems WHERE ID=2015",
       SQL_NTS );
if( rc == SQL_ERROR )
{
   SQLGetDiagRec( SQL_HANDLE_STMT, stmt, 1, errstate,
       &errnative, errmsg, sizeof(errmsg), &errmsglen );
   print_error( "Failed to delete items", errstate, errnative,
errmsg );
   return;
}
```

データベース内の Java

SAP Sybase IQ では、データベースサーバ環境内から Java クラスを実行するメカニズムが用意されています。データベースサーバで Java メソッドを使用すると、強力な方法でプログラミング論理をデータベースに追加できます。

データベースでの Java サポートの特長を次に示します。

- クライアント、中間層、またはサーバなどアプリケーションの異なるレイヤで Java コンポーネントを再使用したり、最も意味がある場所で使用したりします。SAP Sybase IQ が分散コンピューティング用のプラットフォームになります。
- データベースに論理を構築する場合、Java は SQL ストアドプロシージャ言語よりも高機能な言語です。
- データベースおよびサーバの整合性、セキュリティ、堅牢性を保ちながら、データベースサーバで Java を使用することができます。

SQLJ 標準
データベース内の Java は、SQLJ Part 1 で提唱されている標準 (ANSI/INCITS 331.1-1999) に準拠しています。SQLJ Part 1 は、Java の静的メソッドを SQL ストアドプロシージャおよび関数として呼び出すための仕様です。

データベース内の Java についての FAQ

この項では、データベースにおける Java の主な特徴について説明します。

データベースにおける Java の主な特徴は？

次の各項目については、このあとの項で詳しく説明します。

- **データベースで Java を実行できる** – 外部 Java VM は、データベースサーバに代わって Java コードを実行します。
- **Java からデータにアクセスできる** – SAP Sybase IQ では、Java からデータにアクセスできます。
- **SQL が保持される** – Java を使用しても、既存の SQL 文の動作や他の Java 以外のリレーショナルデータベースの動作は変更されません。

データベースで独自の Java クラスを使用する方法は？

Java 言語は SQL より強力です。Java はオブジェクト指向型言語であるため、その命令 (ソースコード) はクラスの形式を取ります。データベースで Java を実行する

には、データベースの外部で Java 命令を作成し、それらをデータベースの外部でコンパイルし、Java 命令を保持するバイナリファイルであるコンパイル済みクラス (*バイトコード*) にします。

コンパイル済みクラスは、ストアドプロシージャと同じくらい簡単に同じ方法で、クライアントアプリケーションから呼び出すことができます。Java クラスには、サブジェクトに関する情報と計算論理の両方を含めることができます。たとえば、Employees クラスを作成し、Employees テーブルの操作を実行する各種のメソッドを使用して作業を行う Java コードを設計、記述し、コンパイルすることができます。Java のクラスはオブジェクトとしてデータベースにインストールし、SQL のカバー関数やプロシージャを記述して Java クラスのメソッドを呼び出します。

これらのクラスは、インストール後、ストアドプロシージャを使用してデータベースサーバで実行できます。たとえば、次の文は Java プロシージャへのインタフェースを作成するものです。

```
CREATE PROCEDURE MyMethod()
EXTERNAL NAME 'JDBCExample.MyMethod()V'
LANGUAGE JAVA;
```

SAP Sybase IQ は、Java 開発環境ではなく、Java クラスのランタイム環境を支援するものです。Java の記述やコンパイルには、Java Development Kit (JDK) などの Java 開発環境が必要です。また、Java クラスを実行するためには Java Runtime Environment も必要です。

Java Development Kit に含まれる、Java API の一部であるクラスの多くを使用できます。また、Java 開発者が作成し、コンパイルしたクラスも使用できます。

Java はデータベースでどのように実行されるか？

SAP Sybase IQ は Java VM を起動します。Java VM はコンパイル済みの Java 命令を解釈し、データベースサーバに代わってそれらを実行します。データベースサーバは必要に応じて Java VM を自動的に起動します。そのため、ユーザがわざわざ Java VM を起動したり、停止したりする必要はありません。

データベースサーバの SQL 要求プロセッサは、Java VM に呼び出されて、Java 命令を実行できるように拡張されました。このプロセッサは Java VM からの要求も処理でき、Java からのデータアクセスを可能にしました。

Java エラー処理

Java アプリケーションでのエラーによって、そのエラーを表す例外オブジェクトが生成されます (「*例外のスロー*」と呼ばれる)。スローされた例外がキャッチさ

れ、アプリケーションのあるレベルで正しく処理されない限り、その例外は Java プログラムを終了します。

Java API クラスとカスタム作成のクラスは両方とも、例外をスローできます。実際、ユーザは独自の例外クラスを作成でき、それらの例外クラスはカスタム作成されたエラーのクラスをスローします。

例外が発生したメソッド本体に例外ハンドラがない場合は、例外ハンドラの検索が呼び出しスタックを継続します。呼び出しスタックの一番上に達し、例外ハンドラが見つからなかった場合は、アプリケーションを実行する Java インタプリターのデフォルトの例外ハンドラが呼び出され、プログラムが終了します。

SAP Sybase IQ では、SQL 文が Java メソッドを呼び出し、未処理の例外がスローされると、SQL エラーが生成されます。サーバメッセージウィンドウに、Java 例外の完全なテキストと Java スタックトレースが表示されます。

Java クラスをデータベースにインストールする方法

Java クラスは、単一のクラスまたは JAR としてデータベースにインストールできます。

- **単一のクラス –** 単一のクラスを、コンパイル済みクラスファイルからデータベースにインストールできます。通常、クラスファイルには拡張子 `.class` が付いています。
- **JAR ファイル –** 一連のクラスが、圧縮された JAR ファイルと圧縮されていない JAR ファイルのいずれかに保持されている場合は、一度に全部をインストールできます。通常、JAR ファイルには拡張子 `.jar` または `.zip` が付いています。SAP Sybase IQ は、JAR ユーティリティで作成されたすべての圧縮 JAR ファイルと、その他の JAR 圧縮スキームをサポートしています。

クラスファイルの作成

それぞれの手順の詳細は、Java 開発ツールを使用しているかどうかによって異なりますが、独自のクラスを作成する手順は一般的に次のようになっています。

1. クラスを定義します。
   クラスを定義する Java コードを記述します。
2. クラスに名前を付けて保存します。
   クラス宣言 (Java コード) を拡張子 `.java` が付いたファイルに保存します。
   ファイル名とクラス名が同じで、大文字と小文字の使い分けが一致していることを確認します。
   たとえば、クラス Utility は、ファイル `Utility.java` に保存されます。
3. クラスをコンパイルします。

この手順では、Java コードを含むクラス宣言を、バイトコードを含む新しい個別のファイルにします。新しいファイルの名前は、Java コードファイル名と同じですが拡張子 .class が付きます。コンパイルされた Java クラスは、コンパイルを行ったプラットフォームやランタイム環境のオペレーティングシステムに関係なく、Java Runtime Environment で実行することができます。

データベース内の Java クラスの特殊な機能

この項では、データベース内で Java クラスを使用したときの機能について説明します。

main メソッドを呼び出す方法

通常 Java アプリケーションを (データベース外で) 起動するには、main メソッドを持つクラス上で Java VM を起動します。

たとえば %ALLUSERSPROFILE%¥SybaseIQ¥samples¥JavaInvoice¥Invoice.java ファイルの Invoice クラスには main メソッドがあります。次のようなコマンドを使用して、このクラスをコマンドラインから実行すると、main メソッドが実行されます。

```
java Invoice
```

Java アプリケーションでのスレッド

java.lang.Thread パッケージの機能を使用すると、Java アプリケーションでマルチスレッドを使用できます。

Java アプリケーション内のスレッドは、同期、中断、再開、一時停止、または停止することができます。

No Such Method Exception

Java メソッドを呼び出す際に不正な数の引数を指定したり、不正なデータ型を使用したりした場合、Java VM から java.lang.NoSuchMethodException エラーが返されます。引数の数とタイプを確認します。

Java メソッドから結果セットを返す方法

呼び出しを行う環境に結果セットを返す Java メソッドを書き、LANGUAGE JAVA の EXTERNAL NAME であると宣言された SQL ストアドプロシージャにこのメソッドをラップします。

Java メソッドから結果セットを返すには、次のタスクを実行します。

1. パブリッククラスで、Java メソッドが public と static として宣言されていることを確認します。
2. メソッドが返すと思われる各結果セットについて、そのメソッドが java.sql.ResultSet[] 型のパラメータを持っていることを確認します。これらの結果セットパラメータは、必ずパラメータリストの最後になります。
3. このメソッドでは、まず java.sql.ResultSet のインスタンスを作成して、それを ResultSet[] パラメータの 1 つに割り当てます。
4. EXTERNAL NAME LANGUAGE JAVA 型の SQL ストアドプロシージャを作成します。この型のプロシージャは、Java メソッドのラッパーです。結果セットを返す他のプロシージャと同じ方法で、SQL プロシージャの結果セット上でカーソルを使用することができます。

例

次に示す簡単なクラスには 1 つのメソッドがあり、そのメソッドはクエリを実行して、呼び出しを行った環境に結果セットを返します。

```
import java.sql.*;

public class MyResultSet
{
  public static void return_rset( ResultSet[] rset1 )
       throws SQLException
  {
    Connection conn = DriverManager.getConnection(
                      "jdbc:default:connection" );
    Statement stmt = conn.createStatement();
    ResultSet rset =
      stmt.executeQuery (
                      "SELECT Surname " +
                      "FROM Customers" );
    rset1[0] = rset;
  }
}
```

結果セットを公開するには、そのプロシージャから返された結果セットの数と Java メソッドのシグネチャを指定する CREATE PROCEDURE 文を使用します。

結果セットを指定する CREATE PROCEDURE 文は、次のように定義します。

```
CREATE PROCEDURE result_set()
  RESULT (SurName person_name_t)
  DYNAMIC RESULT SETS 1
  EXTERNAL NAME
     'MyResultSet.return_rset([Ljava/sql/ResultSet;)V'
  LANGUAGE JAVA;
```

結果セットを返す SAP Sybase IQ プロシージャでカーソルを開くのと同じように、このプロシージャ上でカーソルを開くことができます。

文字列 `([Ljava/sql/ResultSet;)V` は Java メソッドのシグネチャで、パラメータと戻り値の数や型を簡潔に文字で表現したものです。

Java からストアドプロシージャを経由して返される値

EXTERNAL NAME LANGUAGE JAVA を使用して作成したストアドプロシージャは、Java メソッドのラッパーとして使用できます。この項では、ストアドプロシージャ内で OUT または INOUT パラメータを利用する Java メソッドの記述方法について説明します。

Java は、INOUT または OUT パラメータの明示的なサポートはしていません。ただし、パラメータの配列は使用できます。たとえば、整数の OUT パラメータを使用するには、1 つの整数だけの配列を作成します。

```
public class Invoice
{
  public static boolean testOut( int[] param )
  {
    param[0] = 123;
    return true;
  }
}
```

次のプロシージャでは、testOut メソッドを使用します。

```
CREATE PROCEDURE testOut( OUT p INTEGER )
EXTERNAL NAME 'Invoice.testOut([I)Z'
LANGUAGE JAVA;
```

文字列 `([I)Z` は Java メソッドのシグネチャで、メソッドが単一のパラメータを持ち、このパラメータが整数の配列であり、ブール値を返すことを示しています。OUT または INOUT パラメータとして使用するメソッドパラメータが、OUT または INOUT パラメータの SQL データ型に対応する Java データ型の配列になるように、メソッドを定義します。

これをテストするには、初期化されていない変数を使用してストアドプロシージャを呼び出します。

```
CREATE VARIABLE zap INTEGER;
CALL testOut( zap );
SELECT zap;
```

結果セットは `123` です。

Java のセキュリティ管理

Java には、セキュリティマネージャが用意されています。これを使用すると、ファイルアクセスやネットワークアクセスなど、セキュリティが問題となるアプリケーションの機能に対するユーザのアクセスを制御できます。Java VM でサポートされているセキュリティ管理機能を利用できます。

Java VM を起動し、停止する方法

Java VM は、最初の Java オペレーションが実行されると自動的にロードされます。Java オペレーションを実行する準備として、明示的に Java VM をロードする場合は、次の文を実行します。

```
START JAVA;
```

Java を使用していないときに STOP JAVA 文を実行すると、Java VM をアンロードできます。構文は次のとおりです。

```
STOP JAVA;
```

Java VM でのシャットダウンフック

データベース内の Java サポートを提供するときに使用する SAP Sybase IQ Java VM クラスローダは、アプリケーションがシャットダウンフックをインストールすることを許可しています。これらのシャットダウンフックは、アプリケーションが JVM ランタイムでインストールするシャットダウンフックに非常によく似ています。

データベース内の Java サポートを使用している接続が STOP JAVA 文を実行するか、または切断した場合、その接続用のクラスローダは、アンロードの前に、その接続用にインストールされているすべてのシャットダウンフックを実行します。データベース内にすべての Java クラスをインストールする通常のデータベース内の Java アプリケーションに対しては、シャットダウンフックのインストールが必須ではないことに注目してください。クラスローダシャットダウンフックは、十分な注意を払って使用する必要があり、Java を停止中の特定の接続用に割り付けられたシステム全体のリソースをクリーンアップするためにのみ使用すべきです。また、jdbc:default 接続はクラスローダシャットダウンフックが呼び出される前にすでに閉じられているので、シャットダウンフック内では jdbc:default JDBC 要求は許可されません。

SQL Anywhere Java VM クラスローダのあるシャットダウンフックをインストールするには、アプリケーションが Java コンパイラクラスパス内に `sajvm.jar` を含める必要があり、また、次のようなコードを実行する必要があります。

```
SDHookThread hook = new SDHookThread( ... );
ClassLoader classLoader =
Thread.currentThread().getContextClassLoader();
((ianywhere.sa.jvm.SAClassLoader)classLoader).addShutdownHook( hook
 );
```

SDHookThread クラスは標準 Thread クラスを拡張したものであり、前述のコードは現在の接続用のクラスローダによってロードされたクラスから実行される必要

があります。データベース内にインストールされたクラスと、その後に外部環境呼び出し経由で呼び出されたクラスは、どれも正しい SQL Anywhere Java VM クラスローダによって自動的に実行されます。

SQL Anywhere Java VM クラスローダリストからシャットダウンフックを削除するには、アプリケーションで次に似たコードを実行する必要があります。

```
ClassLoader classLoader =
Thread.currentThread().getContextClassLoader();
((ianywhere.sa.jvm.SAClassLoader)classLoader).removeShutdownHook( h
ook );
```

前述のコードは、現在の接続用のクラスローダによってロードされたクラスから実行される必要があります。

JDBC CLI

JDBC は、Java アプリケーション用のコールレベルインタフェースです。JDBC を使用すると、さまざまなリレーショナルデータベースに同一のインタフェースでアクセスできます。さらに、高いレベルのツールとインタフェースを構築するため基盤にもなります。JDBC は Java の標準部分になっており、JDK に含まれています。

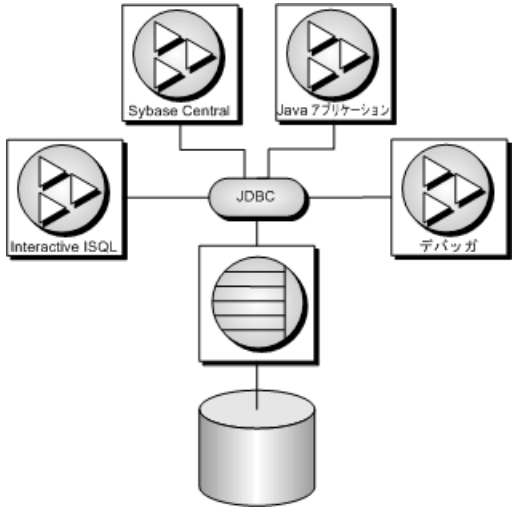
SAP Sybase IQ には、Type 2 ドライバである 4.0 のドライバが含まれています。

また、SAP Sybase IQ では、SAP から利用できる jConnect という pure Java の JDBC ドライバもサポートされています。

JDBC は、クライアント側のアプリケーションプログラミングインタフェースとして使用することもできますし、データベースサーバ内で使用して Java でデータベースのデータにアクセスすることもできます。

JDBC アプリケーション

JDBC API を使用して SAP Sybase IQ に接続する Java アプリケーションを開発できます。Interactive SQL など、SAP Sybase IQ に付属のアプリケーションのいくつかは JDBC を使用しています。



また、Java と JDBC は UltraLite® アプリケーションを開発するための重要なプログラミング言語です。

JDBC はクライアントアプリケーションからとデータベース内からの両方で使用できます。JDBC を使用する Java クラスは、データベースにプログラミング論理を組み込むための、SQL ストアドプロシージャに代わるさらに強力な方法です。

JDBC は Java アプリケーションに対して SQL インタフェースを提供します。Java からリレーショナルデータにアクセスするには、JDBC 呼び出しを使用します。

*クライアントアプリケーション*は、ユーザのコンピュータで動作するアプリケーションを指す場合と、中間層アプリケーションサーバで動作する論理を指す場合があります。

それぞれの例では、SAP Sybase IQ で JDBC を使用する特徴的な機能を示しています。JDBC プログラミングの詳細については、JDBC プログラミングの参考書を参照してください。

SAP Sybase IQ では、次の 2 つの方法で JDBC を使用できます。

- **クライアント側で JDBC を使用する** – Java クライアントアプリケーションは、SAP Sybase IQ に対して JDBC 呼び出しを実行できます。接続は JDBC ドライバを介して行われます。

  SAP Sybase IQ には JDBC 4.0 ドライバ (Type 2 JDBC ドライバ) が含まれており、pure Java アプリケーション用の jConnect ドライバ (Type 4 JDBC ドライバ) もサポートしています。
- **データベース側で JDBC を使用する** – データベースにインストールされている Java クラスは JDBC 呼び出しを行って、データベース内のデータにアクセスしたり、修正したりできます。これには内部 JDBC ドライバを使用します。

JDBC リソース

- **サンプルのソースコード** – この項で示したサンプルのソースコードは、`%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥JDBC` ディレクトリにあります。
- **JDBC 仕様** – JDBC データアクセス API の詳細については、http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html を参照してください。
- **必要なソフトウェア** – jConnect ドライバを使用するには、TCP/IP が必要です。

  jConnect ドライバは、http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect から入手できます。

JDBC ドライバ

SAP Sybase IQ がサポートする JDBC ドライバは次のとおりです。

- **SQL Anywhere 16 JDBC 4.0 ドライバ –** このドライバは、Command Sequence クライアント／サーバプロトコルを使用して SAP Sybase IQ と通信します。ODBC、Embedded SQL、OLE DB アプリケーションと一貫性のある動作をします。SQL Anywhere 16 JDBC 4.0 ドライバは、SAP Sybase IQ データベースに接続する場合の推奨 JDBC ドライバです。JDBC 4.0 ドライバは、JRE 1.6 以降でのみ使用できます。

  JDBC 4.0 ドライバは新しい自動 JDBC ドライバ登録を利用します。したがって、アプリケーションで JDBC 4.0 ドライバを使用すると、Class.forName 呼び出しを実行して JDBC ドライバをロードする必要がなくなります。代わりに、`sajdbc4.jar` ファイルをクラスファイルパスに配置し、jdbc:sqlanywhere で始まる URL を指定して、DriverManager.getConnection() を呼び出すだけですみます。

  JDBC 4.0 ドライバには、OSGi (Open Services Gateway initiative) バンドルとしてのロードを可能にするマニフェスト情報が含まれています。

  JDBC 4.0 ドライバの場合、NCHAR データのメタデータから java.sql.Types.NCHAR、NVARCHAR、または LONGNVARCHAR のカラム型が返されるようになりました。また、アプリケーションで、Get/SetString と Get/SetClob メソッドの代わりに Get/SetNString または Get/SetNClob メソッドを使用して NCHAR データをフェッチできるようになりました。
- **jConnect –** このドライバは、100% pure Java ドライバです。TDS クライアント／サーバプロトコルを使用して SAP Sybase IQ と通信します。

  jConnect ドライバと jConnect のマニュアルは、http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect から入手できます。

使用するドライバを選択するときは、次の要因を考慮します。

- **機能 –** SQL Anywhere 16 JDBC 4.0 ドライバと jConnect は JDBC 4.0 に準拠しています。SQL Anywhere 16 JDBC ドライバでは、SAP Sybase IQ データベースに接続したときにスクロール可能なカーソルを使用できます。jConnect JDBC ドライバでは、SAP Sybase IQ データベースサーバに接続したときは、スクロール可能なカーソルを使用できますが、結果セットはクライアント側でキャッシュされます。jConnect JDBC ドライバでは、SAP Adaptive Server® Enterprise データベースに接続したときに、スクロール可能なカーソルを使用できます。

JDBC 4.0 API のマニュアルは、http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html から入手できます。

- **Pure Java –** jConnect ドライバは pure Java ソリューションです。SQL Anywhere 16 JDBC ドライバは、pure Java ソリューションではなく、SQL Anywhere 16 ODBC ドライバに基づいています。
- **パフォーマンス –** ほとんどの用途で、SQL Anywhere 16 JDBC ドライバのパフォーマンスが jConnect ドライバを上回ります。
- **互換性 –** jConnect ドライバで使用される TDS プロトコルは、Adaptive Server と共通のプロトコルです。ドライバの動作の一部は、このプロトコルで制御されており、Adaptive Server との互換性を持つように設定されています。

SQL Anywhere 16 JDBC ドライバと jConnect のプラットフォームの対応状況については、http://www.sybase.com/detail?id=1061806 を参照してください。

JDBC プログラムの構造

JDBC アプリケーションでは、一般的に次のような一連のイベントが発生します。

- **Connection オブジェクトの作成 –** DriverManager クラスの getConnection クラスメソッドを呼び出すと Connection オブジェクトが作成され、データベースとの接続が確立します。
- **Statement オブジェクトの生成 –** Connection オブジェクトによって Statement オブジェクトが生成されます。
- **SQL 文の引き渡し –** データベース環境で実行する SQL 文が、Statement オブジェクトに渡されます。この SQL 文がクエリの場合、これにより ResultSet オブジェクトが返されます。

  ResultSet オブジェクトには SQL 文から返されたデータが格納されていますが、一度に 1 つのローしか公開されません (カーソルの動きと同じです)。
- **結果セットのローのループ –** ResultSet オブジェクトの next メソッドが次に示す 2 つの動作を実行します。
  - 現在のロー (ResultSet オブジェクトによって公開されている結果セット内のロー) が、1 つ前に送られます。
  - ブール値が返され、前に送るローが存在するかどうかが示されます。
- **それぞれのローに入る値の検索 –** カラムの名前か位置のどちらかを指定すると、ResultSet オブジェクトの各カラムに入る値が検索されます。getData メソッドを使用すると、現在のローにあるカラムから値を取得することができます。

Java オブジェクトは、JDBC オブジェクトを使用してデータベースと対話し、データを取得できます。

クライアント側 JDBC 接続とサーバ側 JDBC 接続の違い

クライアントの JDBC とデータベースサーバ内の JDBC の違いは、データベース環境との接続の確立にあります。

- **クライアント側** – クライアント側の JDBC では、接続を確立するために SQL Anywhere JDBC ドライバまたは jConnect JDBC ドライバが必要です。DriverManager.getConnection に引数を渡すと、接続が確立されます。データベース環境は、クライアントアプリケーションから見て外部アプリケーションとなります。
- **サーバ側** – JDBC がデータベースサーバ内で使用されている場合、接続はすでに確立されています。"jdbc:default:connection" という文字列が DriverManager.getConnection に渡され、JDBC アプリケーションは現在のユーザ接続で動作できるようになります。これは簡単で効率が良く、安全な操作です。それは、接続を確立するためにクライアントアプリケーションがデータベースセキュリティをすでに渡しているためです。ユーザ ID とパスワードがすでに提供されているので、もう一度提供する必要はありません。サーバ側の JDBC ドライバが接続できるのは、現在の接続のデータベースのみです。

URL の構成に 1 つの条件付きの文を使用することによってクライアントとサーバの両方で実行できるように、JDBC クラスを作成します。内部接続には "jdbc:default:connection" が必要ですが、外部接続にはホスト名とポート番号が必要です。

SQL Anywhere JDBC ドライバ

SQL Anywhere JDBC 4.0 ドライバには、pure Java である jConnect JDBC ドライバに比べてパフォーマンスや機能の点で利点があります。ただし、このドライバでは pure Java ソリューションは提供されません。SQL Anywhere JDBC 4.0 ドライバをおすすめします。

SQL Anywhere JDBC 4.0 ドライバをロードする方法

SQL Anywhere JDBC 4.0 ドライバがクラスファイルパスにあることを確認します。

```
set classpath=%IQDIR%¥java¥sajdbc4.jar;%classpath%
```

JDBC 4.0 ドライバは新しい自動 JDBC ドライバ登録を利用します。ドライバがクラスファイルパスにあると、実行が起動された時点で自動的にロードされます。

必要なファイル

SQL Anywhere JDBC 4.0 ドライバの Java コンポーネントは、SAP Sybase IQ インストール環境の `Java` サブディレクトリにインストールされている `sajdbc4.jar` ファイルに含まれています。Windows の場合、ネイティブコンポーネントは SAP Sybase IQ インストール環境の `bin32` または `bin64` サブディレクトリの `dbjdbc16.dll` です。UNIX の場合、ネイティブコンポーネントは `libdbjdbc16.so` です。このコンポーネントは、システムパスにあることが必要です。

SQL Anywhere 16 JDBC ドライバ接続文字列

SQL Anywhere 16 JDBC ドライバを介してデータベースに接続するには、データベースの URL を指定する必要があります。次に例を示します。

```
Connection con = DriverManager.getConnection(
      "jdbc:sqlanywhere:DSN=Sybase IQ Demo" );
```

```
Connection con =
DriverManager.getConnection("jdbc:sqlanywhere:DSN=Sybase IQ Demo" );
```

URL には、`jdbc:sqlanywhere:` の後に接続文字列が含まれています。`sajdbc4.jar` ファイルがクラスファイルパスにあると、JDBC 4.0 ドライバは自動的にロードされて URL を処理します。便宜上、例では ODBC データソース (`DSN`) を指定していますが、データソース接続パラメータとともに、またはデータソース接続パラメータの代わりに、明示的な接続パラメータをセミコロンで区切って使用することもできます。

データソースを使用しない場合は、必要な接続パラメータをすべて接続文字列で指定する必要があります。

```
Connection con = DriverManager.getConnection(

"jdbc:sqlanywhere:UserID=<user_id>;Password=<password>;Start=..." )
;
```

ODBC ドライバと ODBC ドライバマネージャがどちらも使用されないため、`Driver` 接続パラメータは必要ありません。指定しても、無視されます。

jConnect JDBC ドライバ

jConnect ドライバは、別途ダウンロードして入手できます。アプレットから JDBC を使用する場合は、jConnect JDBC ドライバを介して SAP Sybase IQ データベースに接続してください。

jConnect ドライバは、http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect からダウンロードします。jConnect のマニュアルも同じページから入手できます。

jConnect ドライバのファイル

jConnect は `jconn4.jar` という名前の JAR ファイルとして提供されています。このファイルは、jConnect をインストールした場所にあります。

jConnect 用クラスファイルパスの設定

アプリケーションで jConnect を使用するには、コンパイル時と実行時に、jConnect クラスをクラスファイルパスに指定します。これにより、Java コンパイラと Java ランタイムが必要なファイルを見つけられるようになります。

次のコマンドは、既存の CLASSPATH 環境変数に jConnect ドライバを追加します。ここで、*jconnect-path* は jConnect のインストールディレクトリです。

```
set classpath=jconnect-path¥classes¥jconn4.jar;%classpath%
```

jConnect クラスのインポート

jConnect のクラスは、すべて com.sybase.jdbc4.jdbc にあります。これらのクラスを各ソースファイルの先頭でインポートしてください。

```
import com.sybase.jdbc4.jdbc.*
```

パスワードの暗号化

SAP Sybase IQ では、jConnect 接続でのパスワードの暗号化をサポートしています。

jConnect システムオブジェクトのデータベースへのインストール

jConnect を使用してシステムテーブル情報 (データベースメタデータ) にアクセスする場合は、jConnect システムオブジェクトをデータベースに追加してください。

前提条件

ALTER DATABASE システム権限が必要です。また、このデータベースに他の接続がないことが必要です。

データベースファイルをバックアップしてからアップグレードしてください。データベースをアップグレードしようとして失敗した場合、データベースは使用できなくなります。

手順

iqinit ユーティリティを使用すると、jConnect システムオブジェクトはデフォルトで SAP Sybase IQ データベースにインストールされます。jConnect システムオブジェクトは、データベースの作成時、またはその後でも、データベースのアップグレードを実行してデータベースに追加できます。

jConnect ドライバをロードする方法

jConnect ドライバがクラスファイルパスにあることを確認します。ドライバファイル `jconn4.jar` は、jConnect のインストールディレクトリの `classes` サブディレクトリにあります。

```
set classpath=.;c:¥jConnect-7_0¥classes¥jconn4.jar;%classpath%
```

jConnect ドライバは新しい自動 JDBC ドライバ登録を利用します。ドライバがクラスファイルパスにあると、実行が起動された時点で自動的にロードされます。

jConnect ドライバ接続文字列

jConnect を介してデータベースに接続するには、データベースの URL を指定する必要があります。次に例を示します。

```
Connection con = DriverManager.getConnection(
      "jdbc:sybase:Tds:localhost:2638", "<user_id>", "<password>");
```

URL は次のように構成されます。

```
jdbc:sybase:Tds:host:port
```

個々のコンポーネントの説明は次のとおりです。

- **jdbc:sybase:Tds –** TDS アプリケーションプロトコルを使用する、jConnect JDBC ドライバ。
- **host –** サーバが動作しているコンピュータの IP アドレスまたは名前。同じホスト接続を確立している場合は、ログインしているコンピュータシステムを意味する localhost を使用できます。
- **port –** データベースサーバが受信しているポート番号。SAP Sybase IQ に割り当てられているポート番号は 2638 です。ポート番号を変更する理由が特にない場合は、この番号をそのまま使用してください。

接続文字列の長さは、253 文字未満にしてください。

SAP Sybase IQ パーソナルサーバを使用している場合は、サーバの起動時に TCP/IP サポートオプションを必ず含めてください。

jConnect 接続文字列でデータベースを指定する方法

各 SAP Sybase IQ データベースサーバには、1 つまたは複数のデータベースを一度にロードできます。jConnect 経由の接続時に設定する URL でデータベースではなくサーバを指定する場合、そのサーバのデフォルトのデータベースに対して接続が試行されます。

次のいずれかの方法で拡張形式の URL を提供することによって、特定のデータベースを指定できます。

ServiceName パラメータの使用

```
jdbc:sybase:Tds:host:port?ServiceName=database
```

疑問符に続けて一連の割り当てを入力するのは、URL に引数を指定する標準的な方法です。ServiceName の大文字と小文字は区別されません。等号 (=) の前後にはスペースを入れないでください。*database* パラメータはデータベース名で、サーバ名ではありません。データベース名にはパスやファイルサフィックスを含めることはできません。次に例を示します。

```
Connection con = DriverManager.getConnection(
      "jdbc:sybase:Tds:localhost:2638?ServiceName=demo", "DBA",
"sql");
```

RemotePWD パラメータの使用

追加の接続パラメータをサーバに渡すための対処方法があります。

この手法により、RemotePWD フィールドを使用して、データベース名やデータベースファイルなどの追加の接続パラメータを指定できます。put メソッドを使用して、Properties フィールドに RemotePWD を設定します。

次のコードは、このフィールドの使い方を示します。

```
import java.util.Properties;
.
.
.
Properties props = new Properties();
props.put( "User", "DBA" );
props.put( "Password", "sql" );
props.put( "RemotePWD", ",DatabaseFile=mydb.db" );

Connection con = DriverManager.getConnection(
    "jdbc:sybase:Tds:localhost:2638", props );
```

例で示しているように、DatabaseFile 接続パラメータの前にはカンマを入力してください。DatabaseFile パラメータを使用すると、jConnect を使用してサーバ上でデータベースを起動できます。デフォルトでは、データベースは AutoStop=YES で起動されます。utility_db を DatabaseFile (DBF) や DatabaseName (DBN) 接続パラ

メータで指定すると (例：DBN=utility_db)、ユーティリティデータベースは自動的に起動します。

jConnect 接続でのデータベースオプションの設定
アプリケーションが jConnect ドライバを使用してデータベースに接続するとき、sp_tsql_environment ストアドプロシージャが呼び出されます。sp_tsql_environment プロシージャでは、Adaptive Server Enterprise の動作と互換性を保つためのデータベースオプションを設定します。

JDBC クライアントアプリケーションからの接続

SQL Anywhere JDBC ドライバを使用すると、データベースメタデータをいつでも使用できます。

jConnect を使用する JDBC アプリケーションからデータベースシステムテーブル (データベースメタデータ) にアクセスする場合は、jConnect システムオブジェクトのセットをデータベースに追加してください。これらのプロシージャは、すべてのデータベースにデフォルトでインストールされています。iqinit -i オプションを指定すると、このインストールは行われません。

次に示す完全な Java アプリケーションはコマンドラインプログラムであり、稼働中のデータベースに接続して、一連の情報をコマンドラインに出力し、終了します。

すべての JDBC アプリケーションは、データベースのデータを処理する際、最初に接続を確立します。

次の例は、通常のクライアント／サーバ接続である外部接続を示しています。

接続サンプルのコード
この例では、SQL Anywhere JDBC ドライバの JDBC 4.0 バージョンを使用して、デフォルトでデータベースに接続します。異なるドライバを使用するには、コマンドラインでドライバ名 (jdbc4、jConnect) を渡します。JDBC 4.0 ドライバと jConnect の使用例は、コードに含まれています。この例では、サンプルデータベースを使用して、データベースサーバがすでに起動されていることを前提としています。ソースコードは、`%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥JDBC` ディレクトリの `JDBCConnect.java` ファイルにあります。

```
import java.io.*;
import java.sql.*;

public class JDBCConnect
{
    public static void main( String args[] )
    {
```

```
        try
        {
            String arg;
            Connection con;

            // Select the JDBC driver and create a connection.
            // May throw a SQLException.
            // Choices are:
            // 1. jConnect driver
            // 2. SQL Anywhere JDBC 4.0 driver
            arg = "jdbc4";
            if( args.length > 0 ) arg = args[0];
            if( arg.compareToIgnoreCase( "jconnect" ) == 0 )
            {
                con = DriverManager.getConnection(
                    "jdbc:sybase:Tds:localhost:2638", "<user_id>",
"<password>");
            }
            else
            {
                con = DriverManager.getConnection(
                  "jdbc:sqlanywhere:uid=<user_id>;pwd=<password>" );
            }

            System.out.println("Using "+arg+" driver");

            // Create a statement object, the container for the SQL
            // statement. May throw a SQLException.
            Statement stmt = con.createStatement();

            // Create a result set object by executing the query.
            // May throw a SQLException.
            ResultSet rs = stmt.executeQuery(
                "SELECT ID, GivenName, Surname FROM Customers");

            // Process the result set.
            while (rs.next())
            {
                int value = rs.getInt(1);
                String FirstName = rs.getString(2);
                String LastName = rs.getString(3);
               System.out.println(value+" "+FirstName+" "+LastName);
            }
            rs.close();
            stmt.close();
            con.close();
        }
        catch (SQLException sqe)
        {
            System.out.println("Unexpected exception : " +
                               sqe.toString() + ", sqlstate = " +
                               sqe.getSQLState());
            System.exit(1);
        }
        catch (Exception e)
        {
```

```
            e.printStackTrace();
            System.exit(1);
        }

        System.exit(0);
    }
}
```

接続サンプルの動作

この外部接続サンプルは Java コマンドラインプログラムです。

パッケージのインポート
このアプリケーションにはいくつかのパッケージが必要で、そのパッケージは `JDBCConnect.java` の 1 行目でインポートされます。

- java.io パッケージには Java 入出力クラスが含まれています。このクラスはコマンドプロンプトウィンドウに出力するために必要です。
- java.sql パッケージには JDBC クラスが含まれていて、このクラスはすべての JDBC アプリケーションで必要です。

main メソッド
それぞれの Java アプリケーションでは main という名前のメソッドを持つクラスが必要です。このメソッドはプログラムの起動時に呼び出されます。この簡単な例では、JDBCConnect.main がアプリケーションで唯一のパブリックメソッドです。

この JDBCConnect.main メソッドでは、次のタスクを実行します。

1. コマンドラインの引数に基づいて、ロードするドライバを決定します。SQL Anywhere JDBC 4.0 と jConnect 7.0 のドライバがクラスファイルパスにあると、起動時に自動的にロードされます。
2. 選択された JDBC ドライバ URL を使用して、実行しているデフォルトのデータベースに接続します。getConnection メソッドで、指定された URL を使用して接続が確立されます。
3. 文オブジェクトを作成します。このオブジェクトは SQL 文のコンテナです。
4. SQL クエリを実行して結果セットオブジェクトを作成します。
5. 結果セットを反復処理して、カラム情報を表示します。
6. 結果セット、文、接続の各オブジェクトを閉じます。

接続サンプルの実行

JDBC アプリケーションを作成して実行する手順については、例を使用して示します。

前提条件

Java Development Kit (JDK) をインストールしている必要があります。

手順

JDBC を使用した 2 つの異なるタイプの接続を確立できます。1 つはクライアント側接続で、他はサーバ側接続です。次の例ではクライアント側の接続を使用します。

1. コマンドプロンプトで、`%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥JDBC` ディレクトリに移動します。
2. iqdemo.db データベースを含むローカルコンピュータ上のデータベースサーバを起動します。
3. CLASSPATH 環境変数を設定します。この例では、`sajdbc4.jar` にある SQL Anywhere JDBC 4.0 ドライバを使用します。

   ```
   set classpath=.;%IQDIR%¥java¥sajdbc4.jar
   ```

   jConnect ドライバを使用している場合は、次のコマンドを使用します (*path* は jConnect インストールディレクトリ)。

   ```
   set classpath=.;jconnect-path¥classes¥jconn4.jar
   ```
4. 次のコマンドを実行してサンプルをコンパイルします。

   ```
   javac JDBCConnect.java
   ```
5. 次のコマンドを実行してサンプルを実行します。

   ```
   java JDBCConnect
   ```

   `jconnect` などのコマンドライン引数を追加して、異なる JDBC ドライバをロードします。

   ```
   java JDBCConnect jconnect
   ```
6. ID 番号と顧客名のリストがコマンドプロンプトに表示されることを確認します。

   接続が失敗すると、エラーメッセージが表示されます。必要な手順をすべて実行したかどうかを確認します。クラスファイルパスが正しいことを確認します。設定が間違っていると、クラスを検索できません。

ID 番号と顧客名のリストが表示されます。

サーバ側 JDBC クラスから接続を確立する方法

JDBC の SQL 文は、Connection オブジェクトの createStatement メソッドを使用して作成されます。サーバ内で動作するクラスも、Connection オブジェクトを作成するために接続を確立する必要があります。

サーバ側 JDBC クラスから接続を確立する方が、外部接続を確立するよりも簡単です。ユーザはすでにデータベースに接続されているので、クラスでは現在の接続を使用できるからです。

サーバ側接続サンプルのコード

次はサーバ側の接続サンプルのソースコードです。これは、JDBCConnect.java の例に変更を加えたもので、%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥JDBC¥JDBCConnect2.java にあります。

```
import java.io.*;
import java.sql.*;

public class JDBCConnect2
{
  public static void main( String args[] )
  {
    try
    {
      // Open the connection. May throw a SQLException.
      Connection con = DriverManager.getConnection(
          "jdbc:default:connection" );

      // Create a statement object, the container for the SQL
      // statement. May throw a SQLException.
      Statement stmt = con.createStatement();
      // Create a result set object by executing the query.
      // May throw a SQLException.
      ResultSet rs = stmt.executeQuery(
        "SELECT ID, GivenName, Surname FROM Customers");

      // Process the result set.
      while (rs.next())
      {
        int value = rs.getInt(1);
        String FirstName = rs.getString(2);
        String LastName = rs.getString(3);
        System.out.println(value+" "+FirstName+" "+LastName);
      }
      rs.close();
      stmt.close();
      con.close();
    }
```

```
    catch (SQLException sqe)
    {
      System.out.println("Unexpected exception : " +
                sqe.toString() + ", sqlstate = " +
                sqe.getSQLState());
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }
}
```

サーバ側接続サンプルの動作の違い

サーバ側の接続サンプルは、次のことを除いてクライアント側の接続サンプルとほぼ同じです。

1. JDBC ドライバをプリロードする必要はありません。
2. 現在の接続を使用して、稼働中のデフォルトデータベースに接続します。getConnection 呼び出しで指定した URL は次のように変更されています。

   ```
   Connection con = DriverManager.getConnection(
       "jdbc:default:connection" );
   ```
3. `System.exit()` 文は削除されています。

サーバ側接続サンプルの実行

JDBC サーバ側アプリケーションを作成して実行する手順については、例を使用して示します。

前提条件

Java Development Kit (JDK) をインストールしている必要があります。

手順

JDBC を使用した 2 つの異なるタイプの接続を確立できます。1 つはクライアント側接続で、他はサーバ側接続です。次の例ではサーバ側の接続を使用します。

1. コマンドプロンプトで、`%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥JDBC` ディレクトリに移動します。

   ```
   cd %ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥JDBC
   ```
2. サーバ側 JDBC の場合、サーバが異なる現在の作業ディレクトリから起動されないかぎり、CLASSPATH 環境変数を設定する必要はありません。

   ```
   set classpath=.;%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere
   ¥JDBC
   ```

3. iqdemo データベースを含むローカルコンピュータ上のデータベースサーバを起動します。
4. 次のコマンドを入力してサンプルをコンパイルします。
   ```
   javac JDBCConnect2.java
   ```
5. Interactive SQL を使用して、クラスをサンプルデータベースにインストールします。次の文を実行します (クラスファイルへのパスが必要な場合があります)。
   ```
   INSTALL JAVA NEW
   FROM FILE 'JDBCConnect2.class';
   ```
6. クラスの JDBCConnect2.main メソッドのラッパーとして動作する JDBCConnect という名前のストアドプロシージャを定義します。
   ```
   CREATE PROCEDURE JDBCConnect(OUT args LONG VARCHAR)
     EXTERNAL NAME 'JDBCConnect2.main([Ljava/lang/String;)V'
     LANGUAGE JAVA;
   ```
7. 次のように JDBCConnect2.main メソッドを呼び出します。
   ```
   CALL JDBCConnect();
   ```
   セッション内で初めて Java クラスが呼び出されると、Java VM がロードされます。これには数秒間かかる場合があります。
8. ID 番号と顧客名のリストがデータベースサーバメッセージウィンドウに表示されることを確認します。

   接続が失敗すると、エラーメッセージが表示されます。必要な手順をすべて実行したかどうかを確認します。

ID 番号と顧客名のリストがデータベースサーバメッセージウィンドウに表示されます。

JDBC 接続に関する注意

オートコミットの動作、トランザクションの独立性レベル、および接続デフォルトをよく理解してください。

- **オートコミットの動作** – JDBC の仕様により、デフォルトでは各データ操作文が実行されると、COMMIT が実行されます。現在、クライアント側の JDBC はコミットを実行し (オートコミットは true)、サーバ側の JDBC はコミットを実行しない (オートコミットは false) ように動作します。クライアント側とサーバ側のアプリケーションの両方で同じ動作を実行するには、次のような文を使用します。
  ```
  con.setAutoCommit( false );
  ```

この文で、con は現在の接続オブジェクトです。オートコミットを true に設定することもできます。

- **トランザクションの独立性レベルの設定** – トランザクションの独立性レベルを設定するには、次のいずれかの値を使用して、アプリケーションで Connection.setTransactionIsolation メソッドを呼び出す必要があります。

  SQL Anywhere JDBC 4.0 ドライバの場合は、次を使用します。

  TRANSACTION_NONE
  TRANSACTION_READ_COMMITTED
  TRANSACTION_READ_UNCOMMITTED
  TRANSACTION_REPEATABLE_READ
  TRANSACTION_SERIALIZABLE
  sybase.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_SNAPSHOT
  sybase.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_STATEMENT_SNAPSHOT
  sybase.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_STATEMENT_READONLY_SNAPSHOT

  次の例は、JDBC 4.0 ドライバを使用して、トランザクションの独立性レベルを SNAPSHOT に設定します。

```
try
{
    con.setTransactionIsolation(

sybase.jdbc4.sqlanywhere.IConnection.SA_TRANSACTION_SNAPSHOT
    );
}
catch( Exception e )
{
    System.err.println( "Error! Could not set isolation level" );
    System.err.println( e.getMessage() );
    printExceptions( (SQLException)e );
}
```

  getTransactionIsolation と setTransactionIsolation の詳細については、http://docs.oracle.com/javase/6/docs/technotes/guides/jdbc/ にある java.sql.Connection インタフェースのマニュアルを参照してください。

- **接続デフォルト** – サーバ側の JDBC からデフォルト値で新しい接続を作成するのは、`getConnection( "jdbc:default:connection" )` の最初の呼び出しだけです。後続の呼び出しは、接続プロパティを変更せずに、現在の接続のラッパーを返します。最初の接続でオートコミットを false に設定すると、同じ Java コード内の後続の getConnection 呼び出しでは、オートコミットが false に設定された接続を返します。

接続を閉じるときに接続プロパティをデフォルト値に復元し、後続の接続を標準の JDBC 値で取得できるようにしたい場合があります。これを行うには、次のコードを実行します。

```
Connection con =
    DriverManager.getConnection("jdbc:default:connection");

boolean oldAutoCommit = con.getAutoCommit();
try
{
     // main body of code here
}
finally
{
    con.setAutoCommit( oldAutoCommit );
}
```

ここに記載された説明は、オートコミットだけでなく、トランザクションの独立性レベルや読み込み専用モードなどのその他の接続プロパティにも適用されます。

getTransactionIsolation、setTransactionIsolation、isReadOnly の各メソッドの詳細については、http://docs.oracle.com/javase/6/docs/technotes/guides/jdbc/ にある java.sql.Connection インタフェースのマニュアルを参照してください。

JDBC を使用したデータアクセス

データベース内にクラスの一部、またはすべてを格納している Java アプリケーションは、従来の SQL ストアドプロシージャよりもはるかに有利です。ただし、導入段階では、SQL ストアドプロシージャに相当するものを使用して、JDBC の機能を確認した方が便利な場合もあります。次の例では、ローを Departments テーブルに挿入する Java クラスを記述しています。

その他のインタフェースと同様に、JDBC の SQL 文は*静的*または*動的*のどちらでもかまいません。静的 SQL 文は Java アプリケーション内で構成され、データベースに送信されます。データベースサーバは文を解析し、実行プランを選択して SQL 文を実行します。また、実行プランの解析と選択を文の*準備*と呼びます。

同じ文を何度も実行する (たとえば 1 つのテーブルに何度も挿入する) 場合、静的 SQL では著しいオーバヘッドが生じる可能性があります。これは、準備の手順を毎回実行する必要があるためです。

反対に、動的 SQL 文にはプレースホルダがあります。これらのプレースホルダを使用して文を一度準備すれば、それ以降は準備をしなくても何度も文を実行できます。

JDBC サンプルの準備

以下の各項に記載されているコードフラグメントは、完全なクラス %ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥JDBC ¥JDBCExample.java から引用しています。これらの項の準備として、サンプル Java アプリケーションはコンパイルされ、データベースにインストールされます。

前提条件

MANAGE ANY EXTERNAL OBJECT システム権限が必要です。

Java Development Kit (JDK) をインストールしている必要があります。

手順

1. JDBCExample.java ソースコードをコンパイルします。
2. Interactive SQL からデータベースに接続します。
3. Interactive SQL で次の文を実行して、JDBCExample.class ファイルをサンプルデータベースにインストールします。

   ```
   INSTALL JAVA NEW
   FROM FILE 'JDBCExample.class';
   ```

   データベースサーバがクラスファイルと同じディレクトリから起動されず、クラスファイルへのパスがデータベースサーバの CLASSPATH にリスト表示されていない場合、クラスファイルへのパスを INSTALL 文に含める必要があります。

JDBCExample クラスファイルがデータベースにインストールされ、デモの準備が完了しました。

JDBC を使用した挿入、更新、削除

INSERT、UPDATE、DELETE など結果セットを返さない静的 SQL 文の実行には、Statement クラスの executeUpdate メソッドを使用します。CREATE TABLE などの文やその他のデータ定義文も、executeUpdate を使用して実行できます。

Statement クラスの addBatch、clearBatch、executeBatch メソッドも使用できます。Statement クラスの executeBatch メソッドの動作に関する JDBC 仕様は明確でないため、このメソッドを SQL Anywhere JDBC ドライバで使用する場合は、次のことに注意する必要があります。

- SQL 例外または結果セットが発生すると、バッチの処理はすぐに停止します。バッチの処理が停止すると、executeBatch メソッドによって BatchUpdateException がスローされます。BatchUpdateException で

getUpdateCounts メソッドが呼び出されると、ローカウントの整数配列が返されます。この配列では、バッチエラーが発生する前のカウントセットには有効な負でない更新カウントが含まれますが、バッチエラーが発生した時点以降のすべてのカウントには -1 の値が含まれます。BatchUpdateException を SQLException にキャストすると、バッチ処理の停止理由に関する詳細が示されます。

- バッチは、clearBatch メソッドが明示的に呼び出された場合にのみクリアされます。その結果、executeBatch メソッドを繰り返し呼び出すと、バッチが繰り返し再実行されます。また、execute(sql_query) または executeQuery(sql_query) を呼び出すと、指定された SQL クエリは正しく実行されますが、基盤となるバッチはクリアされません。したがって、executeBatch メソッド、execute(sql_query)、executeBatch メソッドの順に呼び出すと、バッチ文のセットが実行され、次に、指定された SQL クエリが実行され、次に、再びバッチ文のセットが実行されます。

次のコードフラグメントは、INSERT 文の実行方法を示しています。ここでは、引数として InsertStatic に渡された Statement オブジェクトを使用しています。

```
public static void InsertStatic( Statement stmt )
{
  try
  {
    int iRows = stmt.executeUpdate(
      "INSERT INTO Departments (DepartmentID, DepartmentName)"
      + " VALUES (201, 'Eastern Sales')" );
    // Print the number of rows inserted
    System.out.println(iRows + " rows inserted");
  }
  catch (SQLException sqe)
  {
    System.out.println("Unexpected exception : " +
               sqe.toString() + ", sqlstate = " +
               sqe.getSQLState());
  }
  catch (Exception e)
  {
    e.printStackTrace();
  }
}
```

説明

- このコードフラグメントは、`%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥JDBC` ディレクトリに含まれている `JDBCExample.java` ファイルの一部です。

- executeUpdate メソッドは整数を返します。この整数は、操作の影響を受けるローの番号を表しています。この場合、INSERT が成功すると、値 1 が返されます。
- サーバ側のクラスとして実行すると、`System.out.println` の出力結果はデータベースサーバメッセージウィンドウに表示されます。

JDBC からの静的 INSERT 文と DELETE 文の使用

サンプル JDBC アプリケーションがデータベースサーバから呼び出され、静的 SQL 文を使用して Departments テーブルでローを挿入、削除します。

前提条件

外部プロシージャを作成するには、CREATE PROCEDURE および CREATE EXTERNAL REFERENCE システム権限が必要です。また、修正しているデータベースオブジェクトでの SELECT、DELETE、INSERT 権限も必要です。

Java Development Kit (JDK) をインストールしている必要があります。

手順

1. Interactive SQL からデータベースに接続します。
2. JDBCExample クラスがインストールされていることを確認します。
3. クラスの JDBCExample.main メソッドのラッパーとして動作する JDBCExample という名前のストアドプロシージャを定義します。
   ```
   CREATE PROCEDURE JDBCExample(IN arg CHAR(50))
     EXTERNAL NAME 'JDBCExample.main([Ljava/lang/String;)V'
     LANGUAGE JAVA;
   ```
4. 次のように JDBCExample.main メソッドを呼び出します。
   ```
   CALL JDBCExample( 'insert' );
   ```
   引数の文字列に `'insert'` を指定すると、InsertStatic メソッドが呼び出されます。
5. ローが Departments テーブルに追加されたことを確認します。
   ```
   SELECT * FROM Departments;
   ```
   サンプルプログラムでは、Departments テーブルの更新された内容をデータベースサーバメッセージウィンドウに表示します。
6. DeleteStatic という名前のサンプルクラスには、追加されたばかりのローを削除する同じようなメソッドがあります。次のように JDBCExample.main メソッドを呼び出します。
   ```
   CALL JDBCExample( 'delete' );
   ```

引数の文字列に 'delete' を指定すると、DeleteStatic メソッドが呼び出されます。

7. ローが Departments テーブルから削除されたことを確認します。

```
SELECT * FROM Departments;
```

サンプルプログラムでは、Departments テーブルの更新された内容をデータベースサーバメッセージウィンドウに表示します。

サーバ側 JDBC アプリケーションで静的 SQL 文を使用して、ローがテーブルで挿入、削除されます。

より効率的なアクセスのために準備文を使用する方法

Statement インタフェースを使用する場合は、データベースに送信するそれぞれの文を解析してアクセスプランを生成し、文を実行します。実行する前の手順を、文の*準備*と呼びます。

PreparedStatement インタフェースを使用すると、パフォーマンス上有利になります。これによりプレースホルダを使用して文を準備し、文の実行時にプレースホルダに値を割り当てることができます。

たくさんのローを挿入するときなど、同じ動作を何度も繰り返す場合は、準備文を使用すると特に便利です。

例

次の例では、PreparedStatement インタフェースの使い方を解説しますが、単一のローを挿入するのは、準備文の正しい使い方ではありません。

JDBCExamples クラスの次の InsertDynamic メソッドによって、準備文を実行します。

```
public static void InsertDynamic( Connection con,
                String ID, String name )
{
  try
  {
    // Build the INSERT statement
    // ? is a placeholder character
    String sqlStr = "INSERT INTO Departments " +
            "( DepartmentID, DepartmentName ) " +
            "VALUES ( ? , ? )";

    // Prepare the statement
    PreparedStatement stmt =
      con.prepareStatement( sqlStr );

    // Set some values
    int idValue = Integer.valueOf( ID );
    stmt.setInt( 1, idValue );
    stmt.setString( 2, name );
```

```
    // Execute the statement
    int iRows = stmt.executeUpdate();

    // Print the number of rows inserted
    System.out.println(iRows + " rows inserted");
  }
  catch (SQLException sqe)
  {
    System.out.println("Unexpected exception : " +
              sqe.toString() + ", sqlstate = " +
              sqe.getSQLState());
  }
  catch (Exception e)
  {
    e.printStackTrace();
  }
}
```

説明

- このコードフラグメントは、%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥JDBC ディレクトリに含まれている JDBCExample.java ファイルの一部です。
- executeUpdate メソッドは整数を返します。この整数は、操作の影響を受けるローの番号を表しています。この場合、INSERT が成功すると、値 1 が返されます。
- サーバ側のクラスとして実行すると、System.out.println の出力結果はデータベースサーバメッセージウィンドウに表示されます。

JDBC からの準備 INSERT 文と DELETE 文の使用

サンプル JDBC アプリケーションがデータベースサーバから呼び出され、準備文を使用して Departments テーブルでローを挿入、削除します。

前提条件

外部プロシージャを作成するには、CREATE PROCEDURE および CREATE EXTERNAL REFERENCE システム権限が必要です。また、修正しているデータベースオブジェクトでの SELECT、DELETE、INSERT 権限も必要です。

Java Development Kit (JDK) をインストールしている必要があります。

手順

1. Interactive SQL からデータベースに接続します。
2. JDBCExample クラスがインストールされていることを確認します。
3. クラスの JDBCExample.Insert メソッドのラッパーとして動作する JDBCInsert という名前のストアドプロシージャを定義します。

```
CREATE PROCEDURE JDBCInsert(IN arg1 INTEGER, IN arg2 CHAR(50))
  EXTERNAL NAME 'JDBCExample.Insert(ILjava/lang/String;)V'
  LANGUAGE JAVA;
```

4. 次のように JDBCExample.Insert メソッドを呼び出します。

   ```
   CALL JDBCInsert( 202, 'Southeastern Sales' );
   ```

   Insert メソッドにより InsertDynamic メソッドが呼び出されます。

5. ローが Departments テーブルに追加されたことを確認します。

   ```
   SELECT * FROM Departments;
   ```

   サンプルプログラムでは、Departments テーブルの更新された内容をデータベースサーバメッセージウィンドウに表示します。

6. DeleteDynamic という名前のサンプルクラスには、追加されたばかりのローを削除する同じようなメソッドがあります。

   クラスの JDBCExample.Delete メソッドのラッパーとして動作する JDBCDelete という名前のストアドプロシージャを定義します。

   ```
   CREATE PROCEDURE JDBCDelete(IN arg1 INTEGER)
     EXTERNAL NAME 'JDBCExample.Delete(I)V'
     LANGUAGE JAVA;
   ```

7. 次のように JDBCExample.Delete メソッドを呼び出します。

   ```
   CALL JDBCDelete( 202 );
   ```

   Delete メソッドにより DeleteDynamic メソッドが呼び出されます。

8. ローが Departments テーブルから削除されたことを確認します。

   ```
   SELECT * FROM Departments;
   ```

   サンプルプログラムでは、Departments テーブルの更新された内容をデータベースサーバメッセージウィンドウに表示します。

サーバ側 JDBC アプリケーションで準備 SQL 文を使用して、ローがテーブルで挿入、削除されます。

JDBC バッチメソッド

PreparedStatement クラスの addBatch メソッドは、バッチ (またはワイド) 挿入を実行するために使用します。次に、このメソッドの使用に関するガイドラインの一部を示します。

1. Connection クラスのいずれかの prepareStatement メソッドを使用して、INSERT 文を準備します。

   ```
   // Build the INSERT statement
   String sqlStr = "INSERT INTO Departments " +
                   "( DepartmentID, DepartmentName ) " +
                   "VALUES ( ? , ? )";
   // Prepare the statement
   ```

```
PreparedStatement stmt =
     con.prepareStatement( sqlStr );
```

2. 次のようにして、準備された INSERT 文のパラメータを設定してバッチ処理します。

```
// loop to batch "n" sets of parameters
for( i=0; i < n; i++ )
{
    // "stmt" is the original prepared insert statement from step
1.
    stmt.setSomeType( 1, param_1 );
    stmt.setSomeType( 2, param_2 );
    .
    .
    .
    // There are "m" parameters in the statement.
    stmt.setSomeType( m , param_m );

    // Add the set of parameters to the batch and
    // move to the next row of parameters.
    stmt.addBatch();
}
```

   例：

```
for( i=0; i < 5; i++ )
{
    stmt.setInt( 1, idValue );
    stmt.setString( 2, name );
    stmt.addBatch();
}
```

3. PreparedStatement クラスの executeBatch メソッドを使用して、バッチを実行する必要があります。

BLOB パラメータはバッチでサポートされていません。

SQL Anywhere JDBC ドライバを使用してバッチ挿入を実行する場合は、小さなカラムサイズを使用することを推奨します。バッチ挿入を使用して、大きなバイナリデータや文字データを long binary カラムや long varchar カラムに挿入すると、パフォーマンスが低下する可能性があるため、推奨しません。パフォーマンスが低下するのは、バッチ挿入されるローをすべて保持するために SQL Anywhere JDBC ドライバが大容量のメモリを割り当てる必要があるためです。これ以外の場合では、バッチ挿入を使用することで個別に挿入するよりも高いパフォーマンスを維持できます。

Java から結果セットを返す方法

この項では、Java メソッドから 1 つ以上の結果セットを取得する方法について説明します。

呼び出し元の環境に 1 つ以上の結果セットを返す Java メソッドを記述し、SQL ストアドプロシージャにこのメソッドをラップします。次のコードフラグメントは、

複数の結果セットをこの Java プロシージャの呼び出し元に返す方法を示しています。ここでは、3 つの executeQuery 文を使用して 3 つの異なる結果セットを取得します。

```
public static void Results( ResultSet[] rset )
       throws SQLException
{
    // Demonstrate returning multiple result sets

    Connection con = DriverManager.getConnection(
                    "jdbc:default:connection" );
    rset[0] = con.createStatement().executeQuery(
        "SELECT * FROM Employees" +
        "   ORDER BY EmployeeID" );
    rset[1] = con.createStatement().executeQuery(
        "SELECT * FROM Departments" +
        "   ORDER BY DepartmentID" );
    rset[2] = con.createStatement().executeQuery(
        "SELECT i.ID,i.LineID,i.ProductID,i.Quantity," +
        "       s.OrderDate,i.ShipDate," +
        "       s.Region,e.GivenName||' '||e.Surname" +
        "   FROM SalesOrderItems AS i" +
        "   JOIN SalesOrders AS s" +
        "   JOIN Employees AS e" +
        "   WHERE s.ID=i.ID" +
        "         AND s.SalesRepresentative=e.EmployeeID" );
    con.close();
}
```

説明

- このサーバ側 JDBC の例は、`%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥JDBC` ディレクトリに含まれている `JDBCExample.java` ファイルの一部です。
- getConnection を使用して、稼働中のデフォルトデータベースへの接続を取得します。
- executeQuery メソッドによって結果セットが返されます。

JDBC から結果セットを返す

サンプル JDBC アプリケーションがデータベースサーバから呼び出され、複数の結果セットを返します。

前提条件

Java Development Kit (JDK) をインストールしている必要があります。

手順

1. Interactive SQL からデータベースに接続します。
2. JDBCExample クラスがインストールされていることを確認します。
3. クラスの JDBCExample.Results メソッドのラッパとして動作する JDBCResults という名前のストアドプロシージャを定義します。

   次に例を示します。

   ```
   CREATE PROCEDURE JDBCResults(OUT args LONG VARCHAR)
     DYNAMIC RESULT SETS 3
     EXTERNAL NAME 'JDBCExample.Results([Ljava/sql/ResultSet;)V'
     LANGUAGE JAVA;
   ```

   例では 3 つの結果セットを返します。
4. 次の Interactive SQL オプションを設定すると、クエリのすべての結果が表示されます。
   a. [ツール] » [オプション] をクリックします。
   b. [Sybase IQ] をクリックします。
   c. [結果] タブをクリックします。
   d. [表示できるローの最大数] の値を `5000` に設定します。
   e. [すべての結果セットを表示] をクリックします。
   f. [OK] をクリックします。
5. JDBCExample.Results メソッドを呼び出します。

   ```
   CALL JDBCResults();
   ```
6. 3 つの結果タブ [結果セット 1]、[結果セット 2]、[結果セット 3] のそれぞれを確認します。

3 つの異なる結果セットは、サーバ側 JDBC アプリケーションから返されます。

JDBC に関する注意事項

Java クラスにアクセスして実行するための権限について説明します。

- **アクセス権限 –** データベースのすべての Java クラスと同様、JDBC 文が含まれているクラスには、Java メソッドのラッパーとして動作するストアドプロシージャを実行する権限が GRANT EXECUTE 文で付与されているどのユーザでもアクセスできます。
- **実行権限 –** Java クラスは、そのクラスを実行する接続の権限によって実行されます。この動作は、所有者の権限によって実行されるストアドプロシージャの動作とは異なります。

JDBC コールバック

SQL Anywhere JDBC ドライバでは、2 つの非同期コールバックをサポートしています。1 つは SQL MESSAGE 文を処理し、もう 1 つはファイル転送要求を検証します。

メッセージは、SQL MESSAGE 文を使用してクライアントアプリケーションからデータベースサーバに送信できます。実行時間が長いデータベースサーバ文によってメッセージも生成できます。

メッセージハンドラルーチンを作成して、これらのメッセージを捕捉できます。次に、メッセージハンドラのコールバックルーチンの例を示します。

```
class T_message_handler implements
sybase.jdbc4.sqlanywhere.ASAMessageHandler
{
    private final int MSG_INFO      = 0x80 | 0;
    private final int MSG_WARNING   = 0x80 | 1;
    private final int MSG_ACTION    = 0x80 | 2;
    private final int MSG_STATUS    = 0x80 | 3;
    T_message_handler()
    {
    }

    public SQLException messageHandler(SQLException sqe)
    {
        String msg_type = "unknown";

        switch( sqe.getErrorCode() ) {
            case MSG_INFO:      msg_type = "INFO   ";   break;
            case MSG_WARNING:   msg_type = "WARNING";   break;
            case MSG_ACTION:    msg_type = "ACTION ";   break;
            case MSG_STATUS:    msg_type = "STATUS ";   break;
        }

        System.out.println( msg_type + ": " + sqe.getMessage() );
        return sqe;
    }
}
```

クライアントファイル転送要求を検証できます。転送を許可する前に、JDBC ドライバは検証コールバックが存在する場合は、それを呼び出します。ストアドプロシージャからなどの間接文の実行中にクライアントのデータ転送が要求された場合、JDBC ドライバはクライアントアプリケーションで検証コールバックが登録されていないかぎり転送を許可しません。どのような状況で検証の呼び出しが行われるかについては、以下でより詳しく説明します。次に、ファイル転送検証のコールバックルーチンの例を示します。

```
class T_filetrans_callback implements
sybase.jdbc4.sqlanywhere.SAValidateFileTransferCallback
```

```
{
    T_filetrans_callback()
    {
    }

    public int callback(String filename, int is_write)
    {
        System.out.println( "File transfer granted for file " +
filename +
                            " with an is_write value of " +
is_write );
        return( 1 ); // 0 to disallow, non-zero to allow
    }
}
```

filename 引数は、読み込みまたは書き込み対象のファイルの名前です。is_write パラメータは、読み込み (クライアントからサーバへの転送) が要求された場合は 0、書き込みが要求された場合は 0 以外の値になります。ファイル転送が許可されない場合、コールバック関数は 0 を返します。それ以外の場合は 0 以外の値を返します。

データのセキュリティ上、サーバはファイル転送を要求している文の実行元を追跡します。サーバは、文がクライアントアプリケーションから直接受信されたものかどうかを判断します。クライアントからデータ転送を開始する際に、サーバは文の実行元に関する情報をクライアントソフトウェアに送信します。クライアント側では、クライアントアプリケーションから直接送信された文を実行するためにデータ転送が要求されている場合にかぎり、JDBC ドライバはデータの転送を無条件で許可します。それ以外の場合は、上述の検証コールバックがアプリケーションで登録されていることが必要です。登録されていない場合、転送は拒否されて文が失敗し、エラーが発生します。データベース内に既存しているストアドプロシージャがクライアントの文で呼び出された場合、ストアドプロシージャそのものの実行はクライアントの文で開始されたものと見なされません。ただし、クライアントアプリケーションでテンポラリストアドプロシージャを明示的に作成してストアドプロシージャを実行した場合、そのプロシージャはクライアントによって開始されたものとしてサーバは処理します。同様に、クライアントアプリケーションでバッチ文を実行する場合も、バッチ文はクライアントアプリケーションによって直接実行されるものと見なされます。

次のサンプル Java アプリケーションは、SQL Anywhere JDBC 4.0 ドライバでサポートされているコールバックの使用を示しています。クラスパスにファイル `%ALLUSERSPROFILE%¥SybaseIQ¥samples¥java¥sajdbc4.jar` を入れる必要があります。

```
import java.io.*;
import java.sql.*;
import java.util.*;

public class callback
{
    public static void main (String args[]) throws IOException
```

```
    {
        Connection          con = null;
        Statement           stmt;

        System.out.println ( "Starting... " );
        con = connect();
        if( con == null )
        {
            return; // exception should already have been reported
        }
        System.out.println ( "Connected... " );
        try
        {
            // create and register message handler callback
            T_message_handler message_worker = new
T_message_handler();

((sybase.jdbc4.sqlanywhere.IConnection)con).setASAMessageHandler( m
essage_worker );

            // create and register validate file transfer callback
            T_filetrans_callback filetran_worker = new
T_filetrans_callback();

((sybase.jdbc4.sqlanywhere.IConnection)con).setSAValidateFileTransf
erCallback( filetran_worker );

            stmt = con.createStatement();

          // execute message statements to force message handler to
be called
            stmt.execute( "MESSAGE 'this is an info   message' TYPE
INFO TO CLIENT" );
            stmt.execute( "MESSAGE 'this is an action message' TYPE
ACTION TO CLIENT" );
            stmt.execute( "MESSAGE 'this is a warning message' TYPE
WARNING TO CLIENT" );
            stmt.execute( "MESSAGE 'this is a status  message' TYPE
STATUS TO CLIENT" );

            System.out.println( "¥n==================¥n" );

            stmt.execute( "set temporary option
allow_read_client_file='on'" );
            try
            {
              stmt.execute( "drop procedure read_client_file_test" );
            }
            catch( SQLException dummy )
            {
                // ignore exception if procedure does not exist
            }
          // create procedure that will force file transfer callback
to be called
          stmt.execute( "create procedure read_client_file_test()" +
```

```
                            "begin" +
                            "    declare v long binary;" +
                        "    set v = read_client_file('sample.txt');" +
                            "end" );

          // call procedure to force validate file transfer callback
to be called
            try
            {
                stmt.execute( "call read_client_file_test()" );
            }
            catch( SQLException filetrans_exception )
            {
               // Note: Since the file transfer callback returns 1,
                 // do not expect a SQL exception to be thrown
                 System.out.println( "SQLException: " +
                                  filetrans_exception.getMessage() );
            }
            stmt.close();
            con.close();
            System.out.println( "Disconnected" );
        }
        catch( SQLException sqe )
        {
            printExceptions(sqe);
        }
    }

    private static Connection connect()
    {
        Connection  connection;

        System.out.println( "Using jdbc4 driver"  );
        try
        {
            connection = DriverManager.getConnection(
                    "jdbc:sqlanywhere:uid=DBA;pwd=sql" );
        }
        catch( Exception e )
        {
            System.err.println( "Error! Could not connect" );
            System.err.println( e.getMessage() );
            printExceptions( (SQLException)e );
            connection = null;
        }
        return connection;
    }

    static private void printExceptions(SQLException sqe)
    {
        while (sqe != null)
        {

            System.out.println("Unexpected exception : " +
                "SqlState: " + sqe.getSQLState()  +
                " " + sqe.toString() +
```

```
                ", ErrorCode: " + sqe.getErrorCode());
            System.out.println( "==================¥n" );
            sqe = sqe.getNextException();
        }
    }
}
```

JDBC エスケープ構文

JDBC エスケープ構文は、Interactive SQL を含む JDBC アプリケーションで使用できます。エスケープ構文を使用して、使用しているデータベース管理システムとは関係なくストアドプロシージャを呼び出すことができます。エスケープ構文の一般的な形式は次のようになります。

{ `keyword` *parameters* }

次のキーワードセットがあります。

- **{d date-string} –** date-string は、SAP Sybase IQ が受け取ることのできる任意の日付値です。
- **{t time-string} –** time-string は、SAP Sybase IQ が受け取ることのできる任意の時刻値です。
- **{ts date-string time-string} –** date-string time-string は、SAP Sybase IQ が受け取ることのできる任意のタイムスタンプ値です。
- **{guid uuid-string} –** uuid-string は、任意の有効な GUID 文字列です (例：41dfe9ef-db91-11d2-8c43-006008d26a6f)。
- **{oj outer-join-expr} –** outer-join-expr は、SAP Sybase IQ が受け取ることのできる有効な OUTER JOIN 式です。
- **{? = call func(p1,...)} –** func は、SAP Sybase IQ が受け取ることのできる任意の有効な関数呼び出しです。
- **{call proc(p1,...)} –** proc は、SAP Sybase IQ が受け取ることのできる任意の有効なストアドプロシージャ呼び出しです。
- **{fn func(p1,...)} –** func は、以下に示すいずれかの関数ライブラリです。

エスケープ構文を使用して、JDBC ドライバによって実装される関数ライブラリにアクセスできます。このライブラリには、数値、文字列、時刻、日付、システム関数が含まれています。

たとえば、次のコマンドを実行すると、データベース管理システムの種類にかかわらず現在の日付を取得できます。

```
SELECT { FN CURDATE() }
```

使用できる関数は、使っている JDBC ドライバによって異なります。次の表は、SQL Anywhere JDBC と jConnect ドライバによってサポートされている関数を示します。

SQL Anywhere JDBC ドライバがサポートする関数

数値関数	文字列関数	システム関数	日付／時刻関数
ABS	ASCII	DATABASE	CURDATE
ACOS	BIT_LENGTH	IFNULL	CURRENT_DATE
ASIN	CHAR	USER	CURRENT_TIME
ATAN	CHAR_LENGTH		CURRENT_TIMESTAMP
ATAN2	CHARACTER_LENGTH		CURTIME
CEILING	CONCAT		DAYNAME
COS	DIFFERENCE		DAYOFMONTH
COT	INSERT		DAYOFWEEK
DEGREES	LCASE		DAYOFYEAR
EXP	LEFT		EXTRACT
FLOOR	LENGTH		HOUR
LOG	LOCATE		MINUTE
LOG10	LTRIM		MONTH
MOD	OCTET_LENGTH		MONTHNAME
PI	POSITION		NOW
POWER	REPEAT		QUARTER
RADIANS	REPLACE		SECOND
RAND	RIGHT		WEEK
ROUND	RTRIM		YEAR
SIGN	SOUNDEX		
SIN	SPACE		
SQRT	SUBSTRING		
TAN	UCASE		
TRUNCATE			

jConnect がサポートする関数

数値関数	文字列関数	システム関数	日付／時刻関数
ABS	ASCII	DATABASE	CURDATE
ACOS	CHAR	IFNULL	CURTIME
ASIN	CONCAT	USER	DAYNAME
ATAN	DIFFERENCE	CONVERT	DAYOFMONTH
ATAN2	LCASE		DAYOFWEEK
CEILING	LENGTH		HOUR
COS	REPEAT		MINUTE
COT	RIGHT		MONTH
DEGREES	SOUNDEX		MONTHNAME
EXP	SPACE		NOW
FLOOR	SUBSTRING		QUARTER
LOG	UCASE		SECOND
LOG10			TIMESTAMPADD
PI			TIMESTAMPDIFF
POWER			YEAR
RADIANS			
RAND			
ROUND			
SIGN			
SIN			
SQRT			
TAN			

エスケープ構文を使用している文は、SAP Sybase IQ、Adaptive Server Enterprise、Oracle、SQL Server、または接続されている他のデータベース管理システムで動作します。

Interactive SQL では、大カッコ ({) は必ず二重にしてください。カッコの間にスペースを入れないでください。"{{" は使用できますが、"{ {" は使用できません。また、文中に改行文字を使用できません。ストアドプロシージャは Interactive SQL で解析されないため、ストアドプロシージャではエスケープ構文を使用できません。

たとえば、SQL エスケープ構文を使用して sa_db_info プロシージャを持つデータベースプロパティを取得するには、Interactive SQL で次のコマンドを実行します。

```
{{CALL sa_db_info( 0 ) }}
```

JDBC 4.0 API のサポート

SQL Anywhere JDBC ドライバでは、JDBC 4.0 仕様のすべての必須クラスとメソッドがサポートされています。java.sql.Blob インタフェースの一部のオプションメソッドはサポートされていません。サポートされていないメソッドは、次のとおりです。

```
long position( Blob pattern, long start );
long position( byte[] pattern, long start );
OutputStream setBinaryStream( long pos )
int setBytes( long pos, byte[] bytes )
int setBytes( long pos, byte[] bytes, int offset, int len );
void truncate( long len );
```

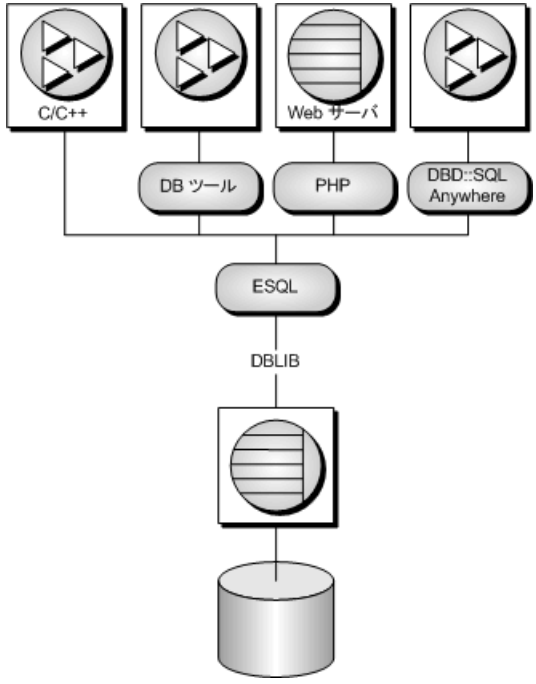
Embedded SQL

C または C++ のソースファイルに組み込まれた SQL 文を、Embedded SQL と呼びます。プリプロセッサがそれらの SQL 文をランタイムライブラリの呼び出しに変換します。Embedded SQL は ISO/ANSI および IBM 規格です。

Embedded SQL は他のデータベースや他の環境に移植可能であり、あらゆる動作環境で同等の機能を実現します。Embedded SQL は、それぞれの製品で使用可能なすべての機能を提供する包括的な低レベルインタフェースです。Embedded SQL を使用するには、C または C++ プログラミング言語に関する知識が必要です。

Embedded SQL アプリケーション

SAP Sybase IQ Embedded SQL インタフェースを使用して SAP Sybase IQ サーバにアクセスする C または C++ アプリケーションを開発できます。たとえば、コマンドラインデータベースツールは、このような方法で開発されたアプリケーションです。



Embedded SQL は、C および C++ プログラミング言語用のデータベースプログラミングインタフェースです。Embedded SQL は、C と C++ のソースコードが混合された (埋め込まれた) SQL 文で構成されます。これらの SQL 文は、SQL プリプロセッ

サによって C または C++ のソースコードに変換されます。その後で、このコードをコンパイルします。

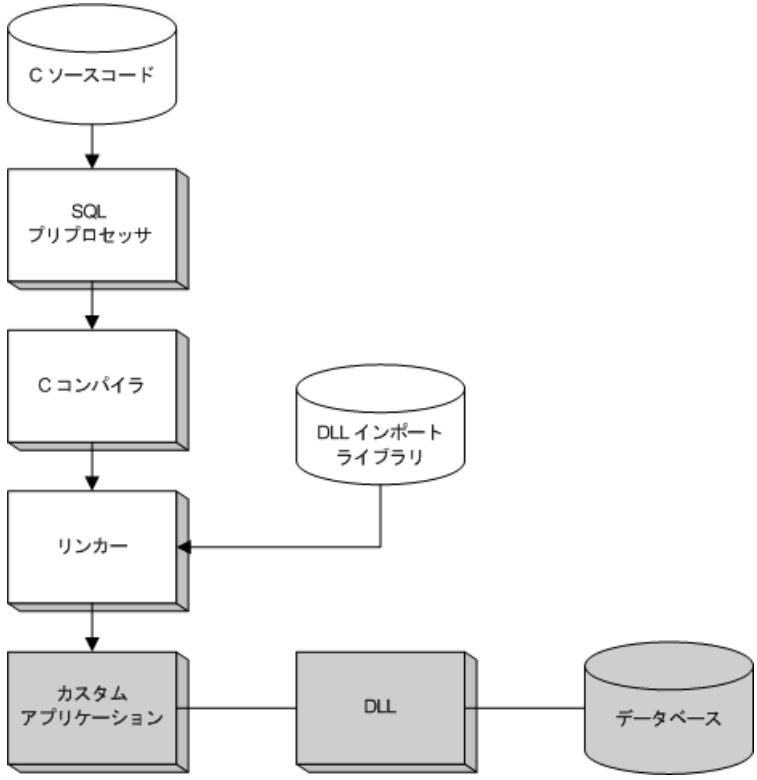
実行時に、Embedded SQL アプリケーションは DBLIB と呼ばれる SAP Sybase IQ の*インタフェースライブラリ*を使用してデータベースサーバと通信します。DBLIB は、ほとんどのプラットフォームで、ダイナミックリンクライブラリ (DLL) または共有オブジェクトです。

- Windows オペレーティングシステムでは、インタフェースライブラリは `dblib16.dll` です。
- UNIX オペレーティングシステムでは、インタフェースライブラリはオペレーティングシステムによって異なり、`libdblib16.so`、`libdblib16.sl`、または `libdblib16.a` です。
- Mac OS X では、インタフェースライブラリは `libdblib16.dylib.1` です。

SAP Sybase IQ には、2 種類の Embedded SQL が用意されています。静的な Embedded SQL は、動的な Embedded SQL に比べて使用方法は単純ですが、柔軟性は乏しくなります。

開発プロセスの概要

Embedded SQL 開発プロセスの概要。



プリプロセッサ処理とコンパイルが成功すると、プログラムは DBLIB 用の*インポートライブラリ*とリンクされ、実行ファイルになります。データベースサーバが実行中のとき、この実行ファイルは DBLIB を使用してデータベースサーバとやりとりをします。プログラムのプリプロセッサ処理はデータベースサーバが実行されていなくても実行できます。

Windows では、Microsoft Visual C++ 用に 32 ビットと 64 ビットのインポートライブラリが用意されています。DLL で関数を呼び出すアプリケーションを開発する方法の 1 つとしてインポートライブラリを使用できます。ただし、ライブラリを動的にロードして、インポートライブラリの使用を避けることをおすすめします。

SQL プリプロセッサ

SQL プリプロセッサの実行プログラム名は iqiqsqlpp です。

プリプロセッサのコマンドラインを次に示します。

```
iqsqlpp [ options ] sql-filename [ output-filename ]
```

プリプロセッサは、C または C++ のソースファイル内の Embedded SQL 文を C コードに変換し、その結果を出力ファイルに出力します。次に、C または C++ コンパイラを使用して出力ファイルを処理します。Embedded SQL を含んだソースプログラムの拡張子は通常 .sqc です。デフォルトの出力ファイル名は、*sql-filename* に拡張子 .c を付けたものになります。*sql-filename* に拡張子 .c が付いている場合、デフォルトの出力ファイル名拡張子は .cc に変更されます。

注意： 新しいメジャーバージョンのデータベースインタフェースライブラリを使用するようにアプリケーションを再構築するときは、同じバージョンの SQL プリプロセッサで Embedded SQL ファイルを前処理する必要があります。

プリプロセッサのオプションを次の表に示します。

オプション	説明
-d	データ領域サイズを減らすコードを生成する。データ構造体を再利用し、実行時に初期化してから使用する。これはコードサイズを増加させる。

オプション	説明
`-e` *level*	指定した標準に含まれない静的 Embedded SQL をエラーとして通知する。*level* 値は、使用する標準を表す。たとえば、`iqsqlpp -e c03 ...` は、コア SQL/2008 規格に含まれない構文を通知する。サポートされる *level* 値は、次のとおり。 • **c08 –** コア SQL/2008 構文でない構文を通知する。 • **p08 –** 上級レベル SQL/2008 構文でない構文を通知する。 • **c03 –** コア SQL/2003 構文でない構文を通知する。 • **p03 –** 上級レベル SQL/2003 構文でない構文を通知する。 • **c99 –** コア SQL/1999 構文でない構文を通知する。 • **p99 –** 上級レベル SQL/1999 構文でない構文を通知する。 • **e92 –** 初級レベル SQL/1992 構文でない構文を通知する。 • **i92 –** 中級レベル SQL/1992 構文でない構文を通知する。 • **f92 –** 上級レベル SQL/1992 構文でない構文を通知する。 • **t –** 標準ではないホスト変数型を通知する。 • **u –** Ultra Light がサポートしていない構文を通知する。 以前のバージョンの SAP Sybase IQ と互換性を保つために、e、i、f を指定することもできる。これらはそれぞれ e92、i92、f92 に対応する。
`-h` *width*	iqsqlpp によって出力される行の最大長を *width* に制限する。行の内容が次の行に続くことを表す文字は円記号 (¥) である。また、*width* に指定できる最小値は 10。
`-k`	コンパイルされるプログラムが SQLCODE のユーザ宣言をインクルードすることをプリプロセッサに通知する。定義は long 型である必要があるが、宣言セクション内で指定する必要はない。

オプション	説明
-m *mode*	Embedded SQL アプリケーションで明示的に指定されていない場合、カーソルの更新可能性モードを指定する。*mode* は次のいずれかを指定する。 • **HISTORICAL** – 以前のバージョンでは、Embedded SQL のカーソルは (クエリや ansi_update_constraints オプション値に応じて) デフォルトで FOR UPDATE または READ ONLY になっていた。明示的なカーソル更新可能性は DECLARE CURSOR で指定された。この動作を維持するためには、このオプションを使用する。 • **READONLY** – Embedded SQL のカーソルはデフォルトで READ ONLY になる。明示的なカーソル更新可能性は PREPARE で指定される。これはデフォルトの動作です。READ ONLY カーソルによってパフォーマンスが向上する場合がある。
-n	C ファイルに行番号情報を生成する。これは、生成された C コード内の適切な場所にある *#line* ディレクティブで構成される。使用しているコンパイラが *#line* ディレクティブをサポートしている場合、このオプションを使うと、コンパイラは SQC ファイル (Embedded SQL が含まれるファイル) の中の行番号を使ってその場所のエラーをレポートする。これは、SQL プリプロセッサによって生成された C ファイルの中の行番号を使って、その場所のエラーをレポートすることとは対照的である。また、ソースレベルデバッガも、*#line* ディレクティブを間接的に使用する。このため、SQC ソースファイルを表示しながらデバッグできる。
-o *operating-system*	ターゲットオペレーティングシステムを指定する。サポートされているオペレーティングシステムは次のとおり。 • **WINDOWS** – Microsoft Windows • **UNIX** – 32 ビットの UNIX アプリケーションを作成している場合はこのオプションを指定する。 • **UNIX64** – 64 ビットの UNIX アプリケーションを作成している場合はこのオプションを指定する。
-q	クワイエットモード (メッセージを表示しない)
-r-	再入力不可コードを生成する。

オプション	説明
`-s` *len*	プリプロセッサが C ファイルに出力する文字列の最大サイズを設定する。この値より長い文字列は、文字のリスト (*'a'*、*'b'*、*'c'* など) を使用して初期化される。ほとんどの C コンパイラには、処理できる文字列リテラルのサイズに制限がある。このオプションを使用して上限を設定する。デフォルト値は 500。
`-u`	Ultra Light 用コードを生成する。
`-w` *level*	指定した標準に含まれない静的 Embedded SQL を警告として通知する。*level* 値は、使用する標準を表す。たとえば `iqsqlpp -w c08 ...` は、コア SQL/2008 構文に含まれない SQL 構文を通知する。サポートされる *level* 値は、次のとおり。 • **c08 –** コア SQL/2008 構文でない構文を通知する。 • **p08 –** 上級レベル SQL/2008 構文でない構文を通知する。 • **c03 –** コア SQL/2003 構文でない構文を通知する。 • **p03 –** 上級レベル SQL/2003 構文でない構文を通知する。 • **c99 –** コア SQL/1999 構文でない構文を通知する。 • **p99 –** 上級レベル SQL/1999 構文でない構文を通知する。 • **e92 –** 初級レベル SQL/1992 構文でない構文を通知する。 • **i92 –** 中級レベル SQL/1992 構文でない構文を通知する。 • **f92 –** 上級レベル SQL/1992 構文でない構文を通知する。 • **t –** 標準ではないホスト変数型を通知する。 • **u –** Ultra Light がサポートしていない構文を通知する。 以前のバージョンの SAP Sybase IQ と互換性を保つために、e、i、f を指定することもできる。これらはそれぞれ e92、i92、f92 に対応する。
`-x`	マルチバイト文字列をエスケープシーケンスに変更して、コンパイラをパススルーできるようにする。
`-z` *cs*	照合順を指定する。推奨する照合順のリストを表示するには、コマンドプロンプトで `iqinit -l` を実行する。 照合順は、プリプロセッサにプログラムのソースコードで使用されている文字を理解させるために使用する。たとえば、識別子に使用できるアルファベット文字の識別などに使用される。-z が指定されていない場合、プリプロセッサは、オペレーティングシステムと SALANG および SACHARSET 環境変数に基づいて、使用する合理的な照合順を決定しようとする。

オプション	説明
sql-filename	処理される Embedded SQL が含まれる C または C++ プログラム。
output-filename	SQL プリプロセッサが作成した C 言語のソースファイル。

対応コンパイラ

C 言語 SQL プリプロセッサは、これまでに次のコンパイラで使用されてきました。

オペレーティングシステム	コンパイラ	バージョン
Windows	Microsoft Visual C++	6.0 以降
Windows Mobile	Microsoft Visual C++	2005
UNIX	GNU またはネイティブコンパイラ	

Embedded SQL ヘッダファイル

ヘッダファイルはすべて、SAP Sybase IQ インストールディレクトリの `SDK¥Include` サブディレクトリにインストールされています。

ファイル名	説明
`sqlca.h`	メインヘッダファイル。すべての Embedded SQL プログラムにインクルードされる。このファイルは SQLCA (SQL Communication Area) の構造体定義と、すべての Embedded SQL データベースインタフェース関数のプロトタイプを含む。
`sqlda.h`	SQLDA (SQL Descriptor Area) の構造体定義。動的 SQL を使用する Embedded SQL プログラムにインクルードされる。
`sqldef.h`	Embedded SQL インタフェースのデータ型定義。このファイルはデータベースサーバを C プログラムから起動するのに必要な構造体定義とリターンコードも含む。
`sqlerr.h`	SQLCA の sqlcode フィールドに返されるエラーコードの定義。
`sqlstate.h`	SQLCA の sqlstate フィールドに返される ANSI/ISO SQL 標準エラーステータスの定義。

ファイル名	説明
pshpk1.h, pshpk4.h, poppk.h	構造体のパックを正しく処理するためのヘッダ。

インポートライブラリ

Windows プラットフォーム上では、インポートライブラリはすべて、SAP Sybase IQ インストールディレクトリの SDK¥Lib サブディレクトリにインストールされています。Windows 用のインポートライブラリは、SDK¥Lib¥x86 および SDK¥Lib¥x64 サブディレクトリに格納されています。エクスポート定義のリストは、SDK¥Lib¥Def¥dblib.def に格納されています。

UNIX プラットフォーム上では、インポートライブラリはすべて、SAP Sybase IQ インストールディレクトリの lib32 および lib64 サブディレクトリにインストールされています。

オペレーティングシステム	コンパイラ	インポートライブラリ
Windows	Microsoft Visual C++	dblibtm.lib
UNIX (非スレッドアプリケーション)	全コンパイラ	libdblib16.so, libdbtasks16.so, libdblib16.sl, libdbtasks16.sl
UNIX (スレッドアプリケーション)	全コンパイラ	libdblib16_r.so, libdbtasks16_r.so,libdblib16_r.sl, libdbtasks16_r.sl

libdbtasks16 ライブラリは、libdblib16 ライブラリに呼び出されます。コンパイラの中には、libdbtasks16 を自動的に検出するものもあります。そうでない場合は、ユーザが明示的に指定する必要があります。

サンプル Embedded SQL プログラム

Embedded SQL プログラムの非常に簡単な例を次に示します。

```
#include <stdio.h>
EXEC SQL INCLUDE SQLCA;
main()
```

```
{
  db_init( &sqlca );
  EXEC SQL WHENEVER SQLERROR GOTO error;
  EXEC SQL CONNECT "DBA" IDENTIFIED BY "sql";
  EXEC SQL UPDATE Employees
    SET Surname =  'Plankton'
    WHERE EmployeeID = 195;
  EXEC SQL COMMIT WORK;
  EXEC SQL DISCONNECT;
  db_fini( &sqlca );
  return( 0 );
error:
  printf( "update unsuccessful -- sqlcode = %ld¥n",
    sqlca.sqlcode );
  db_fini( &sqlca );
  return( -1 );
}
```

この例では、データベースに接続して、従業員番号 195 の姓を更新し、変更内容をコミットして、終了しています。この例では、Embedded SQL コードと C コード間のやりとりはまったくありません。この例では、C コードはフロー制御だけに使用されています。WHENEVER 文はエラーチェックに使用されています。エラーアクション (この例では GOTO) はエラーを起こした SQL 文の後で実行されます。

Embedded SQL プログラムの構造

SQL 文は通常の C または C++ コードの内部に置かれ (埋め込まれ) ます。Embedded SQL 文は、必ず、EXEC SQL で始まり、セミコロン (;) で終わります。Embedded SQL 文の途中に、通常の C 言語のコメントを記述できます。

Embedded SQL を使用する C プログラムでは、ソースファイル内のどの Embedded SQL 文よりも前に、必ず次の文を置きます。

```
EXEC SQL INCLUDE SQLCA;
```

Embedded SQL を使用するすべての C プログラムは、初めに SQLCA を初期化する必要があります。

```
db_init( &sqlca );
```

C プログラムが最初に実行する Embedded SQL 文の 1 つは、CONNECT 文である必要があります。CONNECT 文はデータベースサーバに接続し、ユーザ ID を指定します。このユーザ ID は接続中に実行されるすべての SQL 文の認可に使用されます。

Embedded SQL 文には C コードを生成しないものや、データベースとのやりとりをしないものもあります。このような文は CONNECT 文の前に記述できます。よく使われるのは、INCLUDE 文と、エラー処理を指定する WHENEVER 文です。

Embedded SQL を使用したすべての C プログラムは、初期化された SQLCA をすべてファイナライズする必要があります。

```
db_fini( &sqlca );
```

Windows での DBLIB の動的ロード

インポートライブラリとリンクしなくてもよいように、インストールディレクトリの SDK¥C サブディレクトリにある esqldll.c モジュールを使用して、Embedded SQL アプリケーションから DBLIB を動的にロードします。

前提条件

この作業を実行するための前提条件はありません。

手順

このタスクは、アプリケーションを必要な関数定義が含まれるダイナミックリンクライブラリ (DLL) の静的インポートライブラリにリンクする通常の方法の代替となる方法です。

UNIX プラットフォームで DBLIB を動的にロードする場合にも同様のタスクを使用できます。

1. アプリケーションでは、db_init_dll を呼び出して DBLIB DLL をロードし、db_fini_dll を呼び出して DBLIB DLL を解放します。db_init_dll はすべてのデータベースインタフェース関数の前に呼び出してください。db_fini_dll の後には、インタフェース関数の呼び出しはできません。

   db_init と db_fini ライブラリ関数も呼び出してください。
2. Embedded SQL プログラムでは、EXEC SQL INCLUDE SQLCA 文の前に esqldll.h ヘッダファイルをインクルードするか、sqlca.h をインクルードしてください。esqldll.h ヘッダファイルは sqlca.h をインクルードします。
3. SQL の OS マクロを定義します。sqlos.h にインクルードされるヘッダファイル sqlca.h は、適切なマクロを判断して、定義しようとします。しかし、プラットフォームとコンパイラの組み合わせによっては、定義に失敗することがあります。その場合は、このヘッダファイルの先頭に #define を追加するか、コンパイラオプションを使用してマクロを定義してください。Windows で定義が必要となるマクロを次に示します。

マクロ	プラットフォーム
_SQL_OS_WINDOWS	すべての Windows オペレーティングシステム

4. `esqldll.c` をコンパイルします。
5. インポートライブラリにリンクする代わりに、オブジェクトモジュール `esqldll.obj` を Embedded SQL アプリケーションオブジェクトにリンクします。

Embedded SQL アプリケーションを実行すると、DBLIB インタフェース DLL が動的にロードされます。

サンプル Embedded SQL プログラム

SAP Sybase IQ をインストールすると、Embedded SQL のサンプルプログラムがインストールされます。サンプルプログラムは `%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥C` ディレクトリにあります。

- 静的カーソルを使用した Embedded SQL サンプルである `cur.sqc` は静的 SQL 文の使い方を示します。
- 動的カーソルを使用した Embedded SQL サンプルである `dcur.sqc` は、動的 SQL 文の使い方を示します。

サンプルプログラムで重複するコードの量を減らすために、メインライン部分とデータ出力関数は別ファイルになっています。これは、文字モードシステムでは `mainch.c`、ウィンドウ環境では `mainwin.c` です。

サンプルプログラムには、それぞれ次の 3 つのルーチンがあり、メインライン部分から呼び出されます。

- **WSQLEX_Init –** データベースに接続し、カーソルを開く。
- **WSQLEX_Process_Command –** ユーザのコマンドを処理し、必要に応じてカーソルを操作する。
- **WSQLEX_Finish –** カーソルを閉じ、データベースとの接続を切断する。

メインライン部分の機能を次に示します。

1. WSQLEX_Init ルーチンを呼び出す。
2. ユーザからコマンドを受け取り、ユーザが終了するまで WSQL_Process_Command を呼び出して、ループする。
3. WSQLEX_Finish ルーチンを呼び出す。

データベースへの接続は Embedded SQL の CONNECT 文に適切なユーザ ID とパスワードを指定して実行します。

これらのサンプルに加えて、SAP Sybase IQ には、特定のプラットフォームで使用できる機能を例示するプログラムとソースファイルも用意されています。

静的カーソルのサンプル

これはカーソル使用法の例です。ここで使用されているカーソルはサンプルデータベースの Employees テーブルから情報を取り出します。カーソルは静的に宣言されています。つまり、情報を取り出す実際の SQL 文はソースプログラムにハードコードされています。この例はカーソルの機能を理解するには格好の出発点です。動的カーソルのサンプルでは、この最初のサンプルを使って、これを動的 SQL 文を使用するものに書き換えます。

open_cursor ルーチンは、特定の SQL クエリ用のカーソルを宣言し、同時にカーソルを開きます。

1 ページ分の情報の表示は print ルーチンが行います。このルーチンは、カーソルから 1 つのローをフェッチして表示する動作を *pagesize* 回繰り返します。フェッチルーチンは警告条件 (「ローが見つかりません (SQLCODE 100)」など) を検査し、そうした条件が見つかった場合に適切なメッセージを表示します。また、このプログラムは、カーソルの位置を現在のデータページの先頭に表示されているローの前に変更します。

move、top、bottom ルーチンは適切な形式の FETCH 文を使用して、カーソルを位置付けます。この形式の FETCH 文は実際のデータの取得はしません。単にカーソルを位置付けるだけです。また、汎用の相対位置付けルーチン move はパラメータの符号に応じて移動方向を変えるように実装されています。

ユーザがプログラムを終了すると、カーソルは閉じられ、データベース接続も解放されます。カーソルは ROLLBACK WORK 文によって閉じられ、接続は DISCONNECT によって解放されます。

静的カーソルのサンプルプログラムの実行

静的カーソルのサンプルプログラムを実行します。

前提条件

この作業を実行するための前提条件はありません。

手順

実行ファイルと対応するソースコードは `%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥C` ディレクトリにあります。

1. SAP Sybase IQ サンプルデータベース `iqdemo.db` を起動します。

2. サンプルプログラムの構築用ファイルには、サンプルコードが用意されています。

   Windows で 32 ビットのサンプルを構築するには、`build.bat` を使用します。

   Windows で 64 ビットのサンプルを構築するには、`build64.bat` を使用します。コンパイルとリンクに適した環境を設定する必要があります。x64 プラットフォーム用のサンプルプログラムを構築するコマンド例を次に示します。

   ```
   set mssdk=c:¥MSSDK¥v7.0
   build64
   ```

   Unix でサンプルを構築するには、シェルスクリプトの `build.sh` を使用します。
3. 32 ビットの Windows の例では、ファイル `curwin.exe` を実行します。

   64 ビットの Windows の例では、ファイル `curx64.exe` を実行します。

   UNIX の例では、ファイル `cur` を実行します。
4. 画面に表示される指示に従います。

さまざまなコマンドでデータベースカーソルを操作し、クエリ結果を画面に出力できます。実行するコマンド文字を入力してください。システムによっては、文字入力の後、[Enter] キーを押す必要があります。

動的カーソルのサンプル

このサンプルは、動的 SQL SELECT 文でのカーソルの使用方法を示しています。

動的カーソルのサンプルプログラム (dcur) では、ユーザは N コマンドによって参照したいテーブルを選択できます。プログラムは、そのテーブルの情報を画面に入るかぎり表示します。

起動したら、プロンプトに対して次の接続文字列を入力してください。次はその例です。

```
UID=<userid>;PWD=<your_password>;DBF=iqdemo.db
```

Embedded SQL を使用する C プログラムは、`%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥C` ディレクトリにあります。

dcur プログラムは Embedded SQL インタフェース関数の db_string_connect を使用してデータベースに接続します。この関数はデータベース接続に使用する接続文字列をサポートします。

open_cursor ルーチンは、まず SELECT 文を作成します。

```
SELECT * FROM table-name
```

table-name はルーチンに渡されたパラメータです。この文字列を使用して動的 SQL 文を準備します。

Embedded SQL の DESCRIBE 文は、SELECT 文の結果を SQLDA 構造体に設定するために使用されます。

注意： SQLDA のサイズの初期値は 3 になっています。この値が小さすぎる場合、データベースサーバの返した SELECT リストの実際のサイズを使用して、正しいサイズの SQLDA を割り付けます。

その後、SQLDA 構造体にはクエリの結果を示す文字列を保持するバッファが設定されます。fill_s_sqlda ルーチンは SQLDA のすべてのデータ型を DT_STRING 型に変換し、適切なサイズのバッファを割り付けます。

その後、この文のためのカーソルを宣言して開きます。カーソルを移動して閉じるその他のルーチンは同じです。

fetch ルーチンの場合は多少異なり、ホスト変数のリストの代わりに、SQLDA 構造体に結果を入れます。print ルーチンは大幅に変更され、SQLDA 構造体から結果を取り出して画面の幅一杯まで表示します。print ルーチンは各カラムの見出しを表示するために SQLDA の名前フィールドも使用します。

動的カーソルのサンプルプログラムの実行

動的カーソルのサンプルプログラムを実行します。

前提条件

この作業を実行するための前提条件はありません。

手順

実行ファイルと対応するソースコードは `%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥C` ディレクトリにあります。Windows Mobile の場合は、追加サンプルが `¥SQLAnywhere¥CE¥esql_sample` ディレクトリにあります。

1. SAP Sybase IQ サンプルデータベース `iqdemo.db` を起動します。
2. サンプルプログラムの構築用ファイルには、サンプルコードが用意されています。

   Windows で 32 ビットのサンプルを構築するには、`build.bat` を使用します。

   Windows で 64 ビットのサンプルを構築するには、`build64.bat` を使用します。コンパイルとリンクに適した環境を設定する必要があります。x64 プラットフォーム用のサンプルプログラムを構築するコマンド例を次に示します。

   ```
   set mssdk=c:¥MSSDK¥v7.0
   build64
   ```

Windows Mobile の場合は、Microsoft Visual C++ 用の `esql_sample.sln` プロジェクトファイルを使用してください。このファイルは `SQLAnywhere¥CE ¥esql_sample` にあります。

Unix でサンプルを構築するには、シェルスクリプトの `build.sh` を使用します。

3. 32 ビットの Windows の例では、ファイル `dcurwin.exe` を実行します。

   64 ビットの Windows の例では、ファイル `dcurx64.exe` を実行します。

   Windows Mobile の例では、ファイル `esql_sample.exe` を Windows Mobile デバイスに配備して実行します。

   UNIX の例では、ファイル `dcur` を実行します。

4. 各サンプルプログラムのユーザインタフェースはコンソールタイプであり、プロンプトでコマンドを入力して操作します。次の接続文字列を入力してサンプルデータベースに接続します。

   ```
   DSN=Sybase IQ Demo
   ```

5. 各サンプルプログラムでテーブルを選択するように要求されます。サンプルデータベース内のテーブルを 1 つ選択します。たとえば、`Customers` または `Employees` を入力します。

6. 画面に表示される指示に従います。

さまざまなコマンドでデータベースカーソルを操作し、クエリ結果を画面に出力できます。実行するコマンド文字を入力してください。システムによっては、文字入力の後、[Enter] キーを押す必要があります。

Embedded SQL のデータ型

プログラムとデータベースサーバ間で情報を転送するには、それぞれのデータについてデータ型を設定します。Embedded SQL データ型定数の前には DT_ が付けられ、`sqldef.h` ヘッダファイル内にあります。ホスト変数はサポートされるどのデータ型についても作成できます。これらのデータ型は、データをデータベースと受け渡しするために SQLDA 構造体で使用することもできます。

これらのデータ型の変数を定義するには、`sqlca.h` にリストされている DECL_ マクロを使用します。たとえば、変数が BIGINT 値を保持する場合は DECL_BIGINT と宣言できます。

次のデータ型が、Embedded SQL プログラミングインタフェースでサポートされます。

- **DT_BIT** – 8 ビット符号付き整数
- **DT_SMALLINT** – 16 ビット符号付き整数
- **DT_UNSSMALLINT** – 16 ビット符号なし整数
- **DT_TINYINT** – 8 ビット符号付き整数
- **DT_BIGINT** – 64 ビット符号付き整数
- **DT_UNSBIGINT** – 64 ビット符号なし整数
- **DT_INT** – 32 ビット符号付き整数
- **DT_UNSINT** – 32 ビット符号なし整数
- **DT_FLOAT** – 4 バイト浮動小数点数
- **DT_DOUBLE** – 8 バイト浮動小数点数
- **DT_DECIMAL** – パック 10 進数 (独自フォーマット)

```
typedef struct TYPE_DECIMAL {
  char array[1];
} TYPE_DECIMAL;
```

- **DT_STRING** – CHAR 文字セット内の NULL で終了する文字列。データベースがブランクを埋め込まれた文字列で初期化されると、文字列にブランクが埋め込まれます。
- **DT_NSTRING** – NCHAR 文字セット内の NULL で終了する文字列。データベースがブランクを埋め込まれた文字列で初期化されると、文字列にブランクが埋め込まれます。
- **DT_DATE** – 有効な日付データを含み、NULL で終了する文字列
- **DT_TIME** – 有効な時間データを含み、NULL で終了する文字列
- **DT_TIMESTAMP** – 有効なタイムスタンプを含み、NULL で終了する文字列
- **DT_FIXCHAR** – CHAR 文字セット内のブランクが埋め込まれた固定長文字列。最大長は 32767 で、バイト単位で指定します。データは、NULL で終了しません。
- **DT_NFIXCHAR** – NCHAR 文字セット内のブランクが埋め込まれた固定長文字列。最大長は 32767 で、バイト単位で指定します。データは、NULL で終了しません。
- **DT_VARCHAR** – CHAR 文字セット内の 2 バイトの長さフィールドを持つ可変長文字列。最大長は 32765 バイトです。データを送信する場合は、長さフィールドに値を設定してください。データをフェッチする場合は、データベースサーバが長さフィールドに値を設定します。データは NULL で終了せず、ブランクも埋め込まれません。

```
typedef struct VARCHAR {
 a_sql_ulen len;
 char       array[1];
} VARCHAR;
```

- **DT_NVARCHAR** – NCHAR 文字セット内の 2 バイトの長さフィールドを持つ可変長文字列。最大長は 32765 バイトです。データを送信する場合は、長さ

フィールドに値を設定してください。データをフェッチする場合は、データベースサーバが長さフィールドに値を設定します。データは NULL で終了せず、ブランクも埋め込まれません。

```
typedef struct NVARCHAR {
 a_sql_ulen len;
 char       array[1];
} NVARCHAR;
```

- **DT_LONGVARCHAR** – CHAR 文字セット内の長い可変長文字列

```
typedef struct LONGVARCHAR {
 a_sql_uint32 array_len;  /* number of allocated bytes in array */
 a_sql_uint32 stored_len; /* number of bytes stored in array
                           * (never larger than array_len) */
 a_sql_uint32 untrunc_len;/* number of bytes in untruncated
expression
                           * (may be larger than array_len) */
 char         array[1];   /* the data */
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

32767 バイトを超えるデータには、LONGVARCHAR 構造体を使用できます。このように大きいデータの場合は、全体を一度にフェッチする方法と、GET DATA 文を使用して分割してフェッチする方法があります。また、サーバに対しても、全体を一度に送信する方法と、SET 文を使用してデータベース変数に追加することで分割して送信する方法があります。データは NULL で終了せず、ブランクも埋め込まれません。

- **DT_LONGNVARCHAR** – NCHAR 文字セット内の長い可変長文字列。マクロによって、構造体が次のように定義されます。

```
typedef struct LONGVARCHAR {
 a_sql_uint32 array_len;  /* number of allocated bytes in array */
 a_sql_uint32 stored_len; /* number of bytes stored in array
                           * (never larger than array_len) */
 a_sql_uint32 untrunc_len;/* number of bytes in untruncated
expression
                           * (may be larger than array_len) */
 char         array[1];   /* the data */
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

32767 バイトを超えるデータには、LONGNVARCHAR 構造体を使用できます。このように大きいデータの場合は、全体を一度にフェッチする方法と、GET DATA 文を使用して分割してフェッチする方法があります。また、サーバに対しても、全体を一度に送信する方法と、SET 文を使用してデータベース変数に追加することで分割して送信する方法があります。データは NULL で終了せず、ブランクも埋め込まれません。

- **DT_BINARY** – 2 バイトの長さフィールドを持つ可変長バイナリデータ。最大長は 32765 バイトです。データベースサーバに情報を渡す場合は、長さフィールドを設定してください。データベースサーバから情報をフェッチする場合は、サーバが長さフィールドを設定します。

```
typedef struct BINARY {
 a_sql_ulen len;
 char       array[1];
} BINARY;
```

- **DT_LONGBINARY** – 長いバイナリデータ。マクロによって、構造体が次のように定義されます。

```
typedef struct LONGVARCHAR {
 a_sql_uint32 array_len;  /* number of allocated bytes in array */
 a_sql_uint32 stored_len; /* number of bytes stored in array
                           * (never larger than array_len) */
 a_sql_uint32 untrunc_len;/* number of bytes in untruncated
expression
                           * (may be larger than array_len) */
 char         array[1];   /* the data */
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

  32767 バイトを超えるデータには、LONGBINARY 構造体を使用できます。このように大きいデータの場合は、全体を一度にフェッチする方法と、GET DATA 文を使用して分割してフェッチする方法があります。また、サーバに対しても、全体を一度に送信する方法と、SET 文を使用してデータベース変数に追加することで分割して送信する方法があります。
- **DT_TIMESTAMP_STRUCT** – タイムスタンプの各部分に対応するフィールドを持つ SQLDATETIME 構造体。

```
typedef struct sqldatetime {
 unsigned short year; /* for example 1999 */
 unsigned char  month; /* 0-11 */
 unsigned char  day_of_week; /* 0-6 0=Sunday */
 unsigned short day_of_year; /* 0-365 */
 unsigned char  day; /* 1-31 */
 unsigned char  hour; /* 0-23 */
 unsigned char  minute; /* 0-59 */
 unsigned char  second; /* 0-59 */
 unsigned long  microsecond; /* 0-999999 */
} SQLDATETIME;
```

  SQLDATETIME 構造体は、型が DATE、TIME、TIMESTAMP (または、いずれかの型に変換できるもの) のフィールドを取り出すのに使用できます。アプリケーションは、日付に関して独自のフォーマットで処理をすることがありますが、この構造体を使ってデータをフェッチすると、以後の操作が簡単になります。この構造体の中のデータをフェッチすると、このデータを簡単に操作できます。また、型が DATE、TIME、TIMESTAMP のフィールドは、文字型であれば、どの型でもフェッチと更新が可能です。

  SQLDATETIME 構造体を介してデータベースに日付、時刻、またはタイムスタンプを入力しようとすると、day_of_year と day_of_week メンバーは無視されます。
- **DT_VARIABLE** – NULL で終了する文字列。文字列は SQL 変数名です。その変数の値をデータベースサーバが使用します。このデータ型はデータベースサー

バにデータを与えるときにだけ使用されます。データベースサーバからデータをフェッチするときには使用できません。

これらの構造体は sqlca.h ファイルに定義されています。VARCHAR、NVARCHAR、BINARY、DECIMAL、LONG の各データ型は、データ格納領域が長さ 1 の文字配列のため、ホスト変数の宣言には向いていません。しかし、動的な変数の割り付けや他の変数の型変換を行うのには有効です。

データベースの DATE 型と TIME 型
データベースのさまざまな DATE 型と TIME 型に対応する、Embedded SQL インタフェースのデータ型はありません。これらの型はすべて SQLDATETIME 構造体または文字列を使用してフェッチと更新を行います。

Embedded SQL のホスト変数

ホスト変数とは、SQL プリプロセッサが認識する C 変数です。ホスト変数はデータベースサーバに値を送ったり、データベースサーバから値を受け取ったりするのに使用できます。

ホスト変数はとても使いやすいものですが、制限もあります。SQLDA (SQL Descriptor Area) という構造体を使用するデータベースサーバと情報をやりとりするには、動的 SQL の方が一般的です。SQL プリプロセッサは、ホスト変数が使用されている文ごとに SQLDA を自動的に生成します。

バッチではホスト変数を使用できません。SET 文のサブクエリ内ではホスト変数を使用できません。

ホスト変数の宣言

ホスト変数は、宣言セクションで定義します。ANSI Embedded SQL 標準では、ホスト変数は通常の C の変数宣言を次のように囲んで定義します。

```
EXEC SQL BEGIN DECLARE SECTION;
/* C variable declarations */
EXEC SQL END DECLARE SECTION;
```

こうして定義されたホスト変数は、どの SQL 文でも値定数の代わりに使用できます。データベースサーバが文を実行するときは、ホスト変数の値が使用されます。ホスト変数をテーブル名やカラム名の代わりに使用することはできません。その場合は動的 SQL が必要です。ホスト変数は、SQL 文の中では他の識別子と区別するために、変数名の前にコロン (:) を付けます。

SQL プリプロセッサは、DECLARE SECTION 内でのみ C 言語コードをスキャンします。したがって、DECLARE SECTION 内では TYPEDEF 型および構造体は使用できませんが、変数の初期化は行えます。

例

INSERT 文でホスト変数を使用するコード例です。プログラム側で変数に値を設定してから、データベースに挿入しています。

```
EXEC SQL BEGIN DECLARE SECTION;
long employee_number;
char employee_name[50];
char employee_initials[8];
char employee_phone[15];
EXEC SQL END DECLARE SECTION;
/* program fills in variables with appropriate values
*/
EXEC SQL INSERT INTO Employees
VALUES (:employee_number, :employee_name,
:employee_initials, :employee_phone );
```

C ホスト変数型

ホスト変数として使用できる C のデータ型は非常に限られています。また、ホスト変数の型には、対応する C の型がないものもあります。

`sqlca.h` ヘッダファイルに定義されているマクロを使用すると、NCHAR、VARCHAR、NVARCHAR、LONGVARCHAR、LONGNVARCHAR、BINARY、LONGBINARY、DECIMAL、DT_FIXCHAR、DT_NFIXCHAR、DATETIME (SQLDATETIME)、BIT、BIGINT、または UNSIGNED BIGINT 型のホスト変数を宣言できます。マクロは次のように使用します。

```
EXEC SQL BEGIN DECLARE SECTION;
DECL_NCHAR                  v_nchar[10];
DECL_VARCHAR( 10 )          v_varchar;
DECL_NVARCHAR( 10 )         v_nvarchar;
DECL_LONGVARCHAR( 32768 )   v_longvarchar;
DECL_LONGNVARCHAR( 32768 ) v_longnvarchar;
DECL_BINARY( 4000 )         v_binary;
DECL_LONGBINARY( 128000 )   v_longbinary;
DECL_DECIMAL( 30, 6 )       v_decimal;
DECL_FIXCHAR( 10 )          v_fixchar;
DECL_NFIXCHAR( 10 )         v_nfixchar;
DECL_DATETIME               v_datetime;
DECL_BIT                    v_bit;
DECL_BIGINT                 v_bigint;
DECL_UNSIGNED_BIGINT        v_ubigint;
EXEC SQL END DECLARE SECTION;
```

プリプロセッサは、Embedded SQL 宣言セクション内のこれらのマクロを認識し、変数を適切な型として処理します。10 進数のフォーマットは独自フォーマットであるため、DECIMAL (DT_DECIMAL, DECL_DECIMAL) 型を使用しないことをおすすめします。

次の表は、ホスト変数で使用できる C 変数の型と、対応する Embedded SQL インタフェースのデータ型を示します。

C データ型	Embedded SQL のインタフェースのデータ型	説明
`short si; short int si;`	DT_SMALLINT	16 ビット符号付き整数
`unsigned short int usi;`	DT_UNSSMALLINT	16 ビット符号なし整数
`long l; long int l;`	DT_INT	32 ビット符号付き整数
`unsigned long int ul;`	DT_UNSINT	32 ビット符号なし整数
`DECL_BIGINT ll;`	DT_BIGINT	64 ビット符号付き整数
`DECL_UNSIGNED_BIGINT ull;`	DT_UNSBIGINT	64 ビット符号なし整数
`float f;`	DT_FLOAT	4 バイトの単精度浮動小数点値
`double d;`	DT_DOUBLE	8 バイトの倍精度浮動小数点値
`char a[n]; /*n>=1*/`	DT_STRING	CHAR 文字セット内の NULL で終了する文字列。データベースがブランクを埋め込まれた文字列で初期化されると、文字列にブランクが埋め込まれる。この変数には、n-1 バイトと NULL ターミネータが保持される。
`char *a;`	DT_STRING	CHAR 文字セット内の NULL で終了する文字列。この変数は、最大 32766 バイトと NULL ターミネータを保持できる領域を指す。

C データ型	Embedded SQL のインタフェースのデータ型	説明
`DECL_NCHAR a[n]; /*n>=1*/`	DT_NSTRING	NCHAR 文字セット内の NULL で終了する文字列。データベースがブランクを埋め込まれた文字列で初期化されると、文字列にブランクが埋め込まれる。この変数には、n-1 バイトと NULL ターミネータが保持される。
`DECL_NCHAR *a;`	DT_NSTRING	NCHAR 文字セット内の NULL で終了する文字列。この変数は、最大 32766 バイトと NULL ターミネータを保持できる領域を指す。
`DECL_VARCHAR(n) a;`	DT_VARCHAR	CHAR 文字セット内の 2 バイトの長さフィールドを持つ可変長文字列。文字列は NULL で終了せず、ブランクも埋め込まれない。n の最大値は 32765 (バイト単位)。
`DECL_NVARCHAR(n) a;`	DT_NVARCHAR	NCHAR 文字セット内の 2 バイトの長さフィールドを持つ可変長文字列。文字列は NULL で終了せず、ブランクも埋め込まれない。n の最大値は 32765 (バイト単位)。
`DECL_LONGVARCHAR(n) a;`	DT_LONGVARCHAR	CHAR 文字セット内の 4 バイトの長さフィールドを 3 つ持つ長い可変長文字列。文字列は NULL で終了せず、ブランクも埋め込まれない。
`DECL_LONGNVARCHAR(n) a;`	DT_LONGNVARCHAR	NCHAR 文字セット内の 4 バイトの長さフィールドを 3 つ持つ長い可変長文字列。文字列は NULL で終了せず、ブランクも埋め込まれない。
`DECL_BINARY(n) a;`	DT_BINARY	2 バイトの長さフィールドを持つ可変長バイナリデータ。n の最大値は 32765 (バイト単位)。

C データ型	Embedded SQL のインタフェースのデータ型	説明
`DECL_LONGBINARY(n) a;`	DT_LONGBINARY	4 バイトの長さフィールドを 3 つ持つ長い可変長バイナリデータ。
`char a; /*n=1*/ DECL_FIXCHAR(n) a;`	DT_FIXCHAR	CHAR 文字セット内の固定長文字列。ブランクが埋め込まれるが、NULL で終了しない。n の最大値は 32767 (バイト単位)。
`DECL_NCHAR a; /*n=1*/ DECL_NFIXCHAR(n) a;`	DT_NFIXCHAR	NCHAR 文字セット内の固定長文字列。ブランクが埋め込まれるが、NULL で終了しない。n の最大値は 32767 (バイト単位)。
`DECL_DATETIME a;`	DT_TIMESTAMP_STRUCT	SQLDATETIME 構造体

文字セット

DT_FIXCHAR、DT_STRING、DT_VARCHAR、DT_LONGVARCHAR の場合、文字データはアプリケーションの CHAR 文字セット内にあります。この文字セットは、通常、アプリケーションのロケールの文字セットです。アプリケーションでは、CHARSET 接続パラメータを使用するか、db_change_char_charset 関数を呼び出すことで CHAR 文字セットを変更できます。

DT_NFIXCHAR、DT_NSTRING、DT_NVARCHAR、DT_LONGNVARCHAR の場合、文字データはアプリケーションの NCHAR 文字セット内にあります。デフォルトでは、アプリケーションの NCHAR 文字セットは CHAR 文字セットと同じです。アプリケーションでは、db_change_nchar_charset 関数を呼び出すことで NCHAR 文字セットを変更できます。

データの長さ

使用している CHAR や NCHAR 文字セットに関係なく、すべてのデータ長はバイトで指定します。

サーバとアプリケーションの間で文字セットを変換する場合は、変換されたデータを処理するためのバッファが十分に確保されていることを確認し、データがトランケートされた場合に追加の GET DATA 文を発行するのはアプリケーション側の責任です。

文字ポインタ
pointer to char (*char * a*) として宣言されたホスト変数は、データベースインタフェースでは 32767 バイトの長さであると見なされます。pointer to char 型のホスト変数を使用してデータベースから情報を取り出す場合は、ポインタの指すバッファを、データベースから返ってくる可能性のある値を格納するのに十分な大きさにしてください。

これはかなりの危険性があります。プログラムが作成された後でデータベースのカラムの定義が変更され、カラムのサイズが大きくなっている可能性があるからです。そうなると、ランダムメモリが破壊される可能性があります。関数のパラメータに pointer to char を渡す場合でも、配列を宣言して使用する方が安全です。この方法により、Embedded SQL 文で配列のサイズを知ることができます。

ホスト変数のスコープ
標準のホスト変数の宣言セクションは、C 変数を宣言できる通常の場所であれば、どこにでも記述できます。C の関数のパラメータの宣言セクションにも記述できます。C 変数は通常のスコープを持っています (定義されたブロック内で使用可能)。ただし、SQL プリプロセッサは C コードをスキャンしないため、C ブロックを重視しません。

SQL プリプロセッサに関しては、ホスト変数はソースファイルにおいてグローバルです。同じ名前のホスト変数は使用できません。

ホスト変数の使用法

ホスト変数は次の場合に使用できます。

- SELECT、INSERT、UPDATE、DELETE 文で数値定数または文字列定数を書ける場所。
- SELECT、FETCH 文の INTO 句。
- ホスト変数は、文名、カーソル名、Embedded SQL 固有の文のオプション名としても使用できます。
- CONNECT、DISCONNECT、SET CONNECT 文では、ホスト変数はサーバ名、データベース名、接続名、ユーザ ID、パスワード、接続文字列として使用できます。
- SET OPTION と GET OPTION では、オプション値の代わりにホスト変数を使用できます。

ホスト変数は次の場合には使用できません。

- ホスト変数は、どの文でもテーブル名、カラム名としては使用できません。
- バッチではホスト変数を使用できません。
- SET 文のサブクエリ内ではホスト変数を使用できません。

SQLCODE および SQLSTATE ホスト変数

ISO/ANSI 標準を使用することで、Embedded SQL ソースファイルの Embedded SQL 宣言セクション内で次の特別なホスト変数を宣言できます。

```
long SQLCODE;
char SQLSTATE[6];
```

使用する場合、これらの変数が設定されるのは、データベース要求を生成する任意の Embedded SQL 文 (DECLARE SECTION、INCLUDE、WHENEVER SQLCODE などを除く EXEC SQL 文) の後になります。したがって、SQLCODE および SQLSTATE ホスト変数は、データベース要求を生成するすべての Embedded SQL 文のスコープで参照可能である必要があります。

次に示すのは、有効な Embedded SQL です。

```
EXEC SQL INCLUDE SQLCA;
// declare SQLCODE with global scope
EXEC SQL BEGIN DECLARE SECTION;
long SQLCODE;
EXEC SQL END DECLARE SECTION;
sub1() {
 EXEC SQL BEGIN DECLARE SECTION;
 char SQLSTATE[6];
 EXEC SQL END DECLARE SECTION;
 exec SQL CREATE TABLE ...
}
sub2() {
 EXEC SQL BEGIN DECLARE SECTION;
 char SQLSTATE[6];
 EXEC SQL END DECLARE SECTION;
 exec SQL DROP TABLE ...
}
```

次の例は、SQLSTATE が関数 sub2 のスコープ内で定義されていないため、有効な Embedded SQL ではありません。

```
EXEC SQL INCLUDE SQLCA;
sub1() {
 EXEC SQL BEGIN DECLARE SECTION;
 char SQLSTATE[6];
 EXEC SQL END DECLARE SECTION;
 exec SQL CREATE TABLE...
}
sub2() {
 exec SQL DROP TABLE...
}
```

インジケータ変数

インジケータ変数とは、データのやりとりをするときに補足的な情報を保持する C 変数のことです。インジケータ変数の役割は、場合によってまったく異なります。

- **NULL 値** – アプリケーションが NULL 値を扱えるようにする。
- **文字列のトランケーション** – フェッチした値がホスト変数に収まるようにトランケートされた場合に、アプリケーションが対応できるようにする。
- **変換エラー** – エラー情報を保持する。

インジケータ変数は a_sql_len 型のホスト変数で、SQL 文では通常のホスト変数の直後に書きます。たとえば、次の INSERT 文では、:ind_phone がインジケータ変数です。

```
EXEC SQL INSERT INTO Employees
 VALUES (:employee_number, :employee_name,
 :employee_initials, :employee_phone:ind_phone );
```

フェッチ時または実行時にデータベースサーバからローを受信しなかった場合 (エラーが発生したか、結果セットの末尾に到達した場合)、インジケータの値は変更されません。

注意： 32 ビット長および 64 ビット長とインジケータを今後使用できるように、Embedded SQL での short int の使用は推奨されなくなりました。a_sql_len を代わりに使用します。

インジケータ変数：SQL NULL 値

SQL での NULL を同じ名前の C 言語の定数と混同しないでください。SQL 言語では、NULL は属性が不明であるか情報が適切でないかのいずれかを表します。C 言語の定数は、ポイント先がメモリのロケーションではないポインタ値を表します。

SAP Sybase IQ のマニュアルで使用されている NULL の場合は、上記のような SQL データベースを指します。C 言語の定数を指す場合は、null ポインタ (小文字) のように表記されます。

NULL は、カラムに定義されるどのデータ型の値とも同じではありません。したがって、NULL 値をデータベースに渡したり、結果に NULL を受け取ったりするためには、通常のホスト変数の他に何か特別なものが必要です。このために使用されるのが、インジケータ変数です。

NULL を挿入する場合のインジケータ変数

INSERT 文は、次のようにインジケータ変数を含むことができます。

```
EXEC SQL BEGIN DECLARE SECTION;
short int employee_number;
```

```
char employee_name[50];
char employee_initials[6];
char employee_phone[15];
a_sql_len ind_phone;
EXEC SQL END DECLARE SECTION;
/*
This program fills in the employee number,
name, initials, and phone number.
*/
if( /* Phone number is unknown */ ) {
 ind_phone = -1;
} else {
 ind_phone = 0;
}
EXEC SQL INSERT INTO Employees
 VALUES (:employee_number, :employee_name,
 :employee_initials, :employee_phone:ind_phone );
```

インジケータ変数の値が -1 の場合は、NULL が書き込まれます。値が 0 の場合は、employee_phone の実際の値が書き込まれます。

NULL をフェッチする場合のインジケータ変数
インジケータ変数は、データをデータベースから受け取るときにも使用されます。この場合は、NULL 値がフェッチされた (インジケータが負) ことを示すために使用されます。NULL 値がデータベースからフェッチされたときにインジケータ変数が渡されない場合は、エラーが発生します (SQLE_NO_INDICATOR)。

インジケータ変数： トランケートされた値
インジケータ変数は、ホスト変数に収まるようにトランケートされたフェッチされた変数があるかどうかを示します。これによって、アプリケーションがトランケーションに適切に対応できるようになります。

フェッチの際に値がトランケートされると、インジケータ変数は正の値になり、トランケーション前のデータベース値の実際の長さを示します。データベース値の実際の長さが 32767 バイトを超える場合は、インジケータ変数は 32767 になります。

インジケータ変数： 変換エラー
デフォルトでは、conversion_error データベースオプションは On に設定され、データ型変換が失敗するとエラーになってローは返されません。

この場合、インジケータ変数を使用して、どのカラムでデータ型変換が失敗したかを示すことができます。データベースオプション conversion_error を Off にすると、データ型変換が失敗した場合はエラーではなく CANNOT_CONVERT 警告を発します。変換エラーが発生したカラムにインジケータ変数がある場合、その変数の値は -2 になります。

conversion_error オプションを Off にすると、データをデータベースに挿入するときに変換が失敗した場合は NULL 値が挿入されます。

インジケータ変数値のまとめ

次の表は、インジケータ変数の使用法をまとめたものです。

インジケータの値	データベースに渡す値	データベースから受け取る値
> 0	ホスト変数値	取り出された値はトランケートされている。インジケータ変数は実際の長さを示す。
0	ホスト変数値	フェッチが成功、または conversion_error が On に設定されている。
-1	NULL 値	NULL 結果。
-2	NULL 値	変換エラー (conversion_error が Off に設定されている場合のみ)。SQLCODE は CANNOT_CONVERT 警告を示す。
< -2	NULL 値	NULL 結果。

SQLCA (SQL Communication Area)

SQLCA (SQL Communication Area) とは、データベースへの要求のたびに、アプリケーションとデータベースサーバの間で統計情報とエラーをやりとりするのに使用されるメモリ領域です。SQLCA は、アプリケーションとデータベース間の通信リンクのハンドルとして使用されます。データベースサーバとやりとりする必要のあるデータベースライブラリ関数には SQLCA が必ず渡されます。また、Embedded SQL 文でも必ず暗黙的に渡されます。

インタフェースライブラリ内には、グローバル SQLCA 変数が 1 つ定義されています。プリプロセッサはこのグローバル SQLCA 変数の外部参照と、そのポインタの外部参照を生成します。外部参照の名前は sqlca、型は SQLCA です。ポインタの名前は sqlcaptr です。実際のグローバル変数は、インポートライブラリ内で宣言されています。

SQLCA は、インストールディレクトリの `sqlca.h` サブディレクトリにある `SDK¥Include` ヘッダファイルで定義されています。

SQLCA にはエラーコードが入る

SQLCA を参照すると、特定のエラーコードの検査ができます。データベースへの要求でエラーがあると、sqlcode フィールドと sqlstate フィールドにエラーコードが

入ります。sqlcode や sqlstate などの SQLCA のフィールドを参照するために、C マクロが定義されています。

SQLCA のフィールド

SQLCA のフィールドの意味を次に示します。

- **sqlcaid –** SQLCA 構造体の ID として文字列 SQLCA が格納される 8 バイトの文字フィールド。このフィールドはデバッグ時にメモリの中身を見るときに役立ちます。
- **sqlcabc –** SQLCA 構造体の長さ (136 バイト) が入る 32 ビットの整数。
- **sqlcode –** データベースが検出した要求エラーのエラーコードが入る 32 ビットの整数。エラーコードの定義はヘッダファイル `sqlerr.h` にあります。エラーコードは、0 (ゼロ) は成功、正は警告、負はエラーを示します。
- **sqlerrml –** sqlerrmc フィールドの情報の長さ。
- **sqlerrmc –** エラーメッセージに挿入される文字列。挿入されない場合もあります。エラーメッセージに 1 つまたは複数のプレースホルダ文字列 (*%1*、*%2*、...) があると、このフィールドの文字列と置換されます。

  たとえば、「`Table Not Found`」(テーブルが見つかりません) というエラーが発生した場合、sqlerrmc にはテーブル名が入り、これがエラーメッセージの適切な場所に挿入されます。
- **sqlerrp –** 予約済み。
- **sqlerrd –** 32 ビット整数の汎用配列。
- **sqlwarn –** 予約済み。
- **sqlstate –** SQLSTATE ステータス値。ANSI SQL 標準では、SQLCODE 値の他に、SQL 文からのこの型の戻り値が定義されます。SQLSTATE 値は NULL で終了する長さ 5 の文字列で、前半 2 バイトがクラス、後半 3 バイトがサブクラスを表します。各バイトは 0 ～ 9 の数字、または、A ～ Z の英大文字です。

  0 ～ 4 または A ～ H の文字で始まるクラス、サブクラスはすべて SQL 標準で定義されています。それ以外のクラスとサブクラスは実装依存です。SQLSTATE 値 '00000' はエラーや警告がなかったことを意味します。

sqlerror 配列

sqlerror フィールドの配列要素を次に示します。

- **sqlerrd[1] (SQLIOCOUNT) –** 文を完了するために必要とされた入出力操作の実際の回数。

  データベースサーバによって、文の実行ごとにこの値が 0 にリセットされることはありません。一連の文を実行する前にこの変数が 0 にリセットされるよう

にプログラムすることもできます。最後の文が実行された後、この値は一連の文の入出力操作の合計回数になります。

- **sqlerrd[2] (SQLCOUNT) –** このフィールドの値の意味は実行中の文によって変わります。
  - **INSERT、UPDATE、PUT、DELETE 文 –** 文によって影響を受けたローの数。
  - **OPEN 文と RESUME 文 –** カーソルを開いたとき、または再開したとき、このフィールドには、カーソル内の実際のロー数 (0 以上の値)、または、その推定値 (負の数で、その絶対値が推定値) が入ります。データベースサーバによって計算されたローの数は、ローの実際の数です。ローを数える必要はありません。row_counts オプションを使って、常にローの実際の数を返すようにデータベースを設定することもできます。
  - **FETCH カーソル文 –** SQLCOUNT フィールドは、警告 SQLE_NOTFOUND が返った場合に設定されます。このフィールドには、FETCH RELATIVE または FETCH ABSOLUTE 文によって、カーソル位置の可能な範囲を超えたローの数が入ります (カーソルは、ローの上にも、最初のローより前または最後のローより後にも置くことができます)。ワイドフェッチの場合、SQLCOUNT は実際にフェッチされたローの数であり、要求されたローの数と同じかそれより少なくなります。ワイドフェッチ中に SQLE_NOTFOUND が設定されるのは、ローがまったく返されなかった場合のみです。

    ローが見つからなくても位置が有効な場合は、値は 0 です。たとえば、カーソル位置が最後のローのときに FETCH RELATIVE 1 を実行した場合です。カーソルの最後を超えてフェッチしようとした場合、値は正の数です。カーソルの先頭を超えてフェッチしようとした場合、値は負の数です。
  - **GET DATA 文 –** SQLCOUNT フィールドには値の実際の長さが入っています。
  - **DESCRIBE 文 –** WITH VARIABLE RESULT 句を使用して複数の結果セットを返す可能性のあるプロシージャを記述した場合、SQLCOUNT は次のいずれかの値に設定されます。
    - **0 –** 結果セットは変更される場合があります。各 OPEN 文の後でプロシージャコールを記述し直してください。
    - **1 –** 結果セットは固定です。再度記述する必要はありません。

    SQLE_SYNTAX_ERROR 構文エラーの場合、このフィールドには文内のおおよそのエラー検出位置が入ります。
- **sqlerrd[3] (SQLIOESTIMATE) –** 文の完了に必要な入出力操作の推定回数。このフィールドは OPEN 文または EXPLAIN 文によって値が設定されます。

マルチスレッドまたは再入可能コードでの SQLCA 管理

Embedded SQL 文はマルチスレッドまたは再入可能コードでも使用できます。ただし、単一接続の場合は、アクティブな要求は 1 接続あたり 1 つに制限されます。

マルチスレッドアプリケーションにおいて、セマフォを使ったアクセス制御をしない場合は、1 つのデータベース接続を各スレッドで共用しないでください。

データベースを使用する各スレッドが別々の接続を使用する場合は制限がまったくありません。ランタイムライブラリは SQLCA を使用してスレッドのコンテキストを区別します。したがって、データベースを同時に使用するスレッドには、それぞれ専用の SQLCA が必要です。ただし例外として、スレッドでは db_cancel_request 関数を使用して、そのスレッドの SQLCA を使用している別のスレッドで実行されている文をキャンセルできます。

次は再入可能マルチスレッド Embedded SQL コードの例です。

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <ctype.h>
#include <stdlib.h>
#include <process.h>
#include <windows.h>
EXEC SQL INCLUDE SQLCA;
EXEC SQL INCLUDE SQLDA;

#define TRUE 1
#define FALSE 0

// multithreading support

typedef struct a_thread_data {
    SQLCA sqlca;
    int num_iters;
    int thread;
    int done;
} a_thread_data;

// each thread's ESQL test

EXEC SQL SET SQLCA "&thread_data->sqlca";

static void PrintSQLError( a_thread_data * thread_data )
/*****************************************************/
{
  char                buffer[200];

  printf( "%d: SQL error %d -- %s ... aborting¥n",
      thread_data->thread,
      SQLCODE,
      sqlerror_message( &thread_data->sqlca,
               buffer, sizeof( buffer ) ) );
    exit( 1 );
}

EXEC SQL WHENEVER SQLERROR { PrintSQLError( thread_data ); };

static void do_one_iter( void * data )
```

```
{
  a_thread_data *  thread_data = (a_thread_data *)data;
  int    i;
  EXEC SQL BEGIN DECLARE SECTION;
  char    user[ 20 ];
  EXEC SQL END DECLARE SECTION;

  if( db_init( &thread_data->sqlca ) != 0 ) {
    for( i = 0; i < thread_data->num_iters; i++ ) {
     EXEC SQL CONNECT "dba" IDENTIFIED BY "sql";
     EXEC SQL SELECT USER INTO :user;
     EXEC SQL DISCONNECT;
    }
    printf( "Thread %d did %d iters successfully¥n",
        thread_data->thread, thread_data->num_iters );
    db_fini( &thread_data->sqlca );
  }
  thread_data->done = TRUE;
}

int main()
{
  int num_threads = 4;
  int thread;
  int num_iters = 300;
  int num_done = 0;
  a_thread_data *thread_data;
  thread_data = (a_thread_data *)malloc( sizeof(a_thread_data) *
num_threads );
  for( thread = 0; thread < num_threads; thread++ ) {
    thread_data[ thread ].num_iters = num_iters;
    thread_data[ thread ].thread = thread;
    thread_data[ thread ].done = FALSE;
    if( _beginthread( do_one_iter,
      8096,
      (void *)&thread_data[thread] ) <= 0 ) {
      printf( "FAILED creating thread.¥n" );
      return( 1 );
    }
  }
    while( num_done != num_threads ) {
    Sleep( 1000 );
    num_done = 0;
    for( thread = 0; thread < num_threads; thread++ ) {
      if( thread_data[ thread ].done == TRUE ) {
        num_done++;
      }
    }
  }
  return( 0 );
}
```

複数の SQLCA

再入力不可コードを生成する SQL プリプロセッサオプション (-r-) を使用しないでください。再入可能コードは、静的に初期化されたグローバル変数を使用できないため、少しだけサイズが大きく、遅いコードになります。ただし、その影響は最小限です。

プログラムで使用する各 SQLCA は db_init を呼び出して初期化し、最後に db_fini を呼び出してクリーンアップします。

Embedded SQL 文の SET SQLCA を使用して、SQL プリプロセッサにデータベース要求で別の SQLCA を使用することを伝えます。通常は、`EXEC SQL SET SQLCA 'task_data->sqlca';` のような文をプログラムの先頭かヘッダファイルに置いて、SQLCA 参照がそのタスク固有のデータを指すようにします。この文はコードを生成しないため、パフォーマンスに影響はありません。この文はプリプロセッサ内部の状態を変更して、指定の文字列で SQLCA を参照するようにします。

各スレッドには専用の SQLCA が必要です。この条件は、Embedded SQL を使用していて、アプリケーションの複数のスレッドから呼び出される、共有ライブラリ (DLL など) 内のコードにも適用されます。

複数 SQLCA のサポートは、サポートされるどの Embedded SQL 環境でも使用できますが、再入可能コードでは必須です。

複数のデータベースに接続するために複数の SQLCA を使用したり、単一のデータベースに対して複数の接続を持ったりする必要はありません。

各 SQLCA は、無名の接続を 1 つ持つことができます。各 SQLCA はアクティブな接続、つまり現在の接続を持ちます。

特定のデータベース接続に対するすべての操作では、その接続が確立されたときに使用されたのと同じ SQLCA を使用します。

注意： 異なる接続に対する操作では通常のレコードロックメカニズムが使用され、互いにブロックしてデッドロックを発生させる可能性があります。

静的 SQL と動的 SQL

SQL 文を C プログラムに埋め込むには次の 2 つの方法があります。

静的文
動的文

静的 SQL 文

標準的 SQL のデータ操作文とデータ定義文はすべて、文の前に EXEC SQL を付け、後ろにセミコロン (;) を付けることで、C プログラムに埋め込むことができます。このような文を*静的*文と呼びます。

静的文にはホスト変数への参照を含めることができます。ホスト変数は文字列定数または数値定数の代わりにしか使えません。カラム名やテーブル名としては使用できません。このような操作には動的文が必要です。

動的 SQL 文

C 言語では、文字列は文字の配列に格納されます。動的文は C 言語の文字列で構成されます。この文は PREPARE 文と EXECUTE 文を使用して実行できます。この SQL 文は静的文と同じようにしてホスト変数を参照することはできません。C 言語の変数は、C プログラムの実行中に変数名でアクセスできないためです。

SQL 文と C 言語の変数との間で情報をやりとりするために、SQLDA (*SQL Descriptor Area*) 構造体が使用されます。EXECUTE 文の USING 句を使ってホスト変数のリストを指定すると、SQL プリプロセッサが自動的にこの構造体を用意します。ホスト変数のリストは、準備文の適切な位置にあるプレースホルダに順番に対応しています。

*プレースホルダ*は文の中に置いて、どこでホスト変数にアクセスするかを指定します。プレースホルダは、疑問符 (?) か静的文と同じホスト変数参照です (ホスト変数名の前にはコロンを付けます)。ホスト変数参照の場合も、実際の文テキスト内のホスト変数名は SQLDA を参照することを示すプレースホルダの役割しかありません。

データベースに情報を渡すのに使用するホスト変数を*バインド変数*と呼びます。

例

```
EXEC SQL BEGIN DECLARE SECTION;
char      comm[200];
char      street[30];
char      city[20];
a_sql_len cityind;
long      empnum;
EXEC SQL END DECLARE SECTION;

...
sprintf( comm,
    "UPDATE %s SET Street = :?, City = :?"
    "WHERE EmployeeID = :?",
    tablename );
EXEC SQL PREPARE S1 FROM :comm FOR UPDATE;
EXEC SQL EXECUTE S1 USING :street, :city:cityind, :empnum;
```

この方法では、文中にいくつのホスト変数があるかを知っている必要があります。通常はそのようなことはありません。そこで、自分で SQLDA 構造体を設定し、この SQLDA を EXECUTE 文の USING 句で指定します。

DESCRIBE BIND VARIABLES 文は、準備文内にあるバインド変数のホスト変数名を返します。これにより、C プログラムでホスト変数を管理するのが容易になります。一般的な方法を次に示します。

```
EXEC SQL BEGIN DECLARE SECTION;
char comm[200];
EXEC SQL END DECLARE SECTION;
...
sprintf( comm,
    "UPDATE %s SET Street = :street, City = :city"
   " WHERE EmployeeID = :empnum",
   tablename );
EXEC SQL PREPARE S1 FROM :comm FOR UPDATE;
/* Assume that there are no more than 10 host variables.
 * See next example if you cannot put a limit on it. */
sqlda = alloc_sqlda( 10 );
EXEC SQL DESCRIBE BIND VARIABLES FOR S1 INTO sqlda;
/* sqlda->sqld will tell you how many
   host variables there were. */
/* Fill in SQLDA_VARIABLE fields with
   values based on name fields in sqlda. */
...
EXEC SQL EXECUTE S1 USING DESCRIPTOR sqlda;
free_sqlda( sqlda );
```

SQLDA の内容

SQLDA は変数記述子の配列です。各記述子は、対応する C プログラム変数の属性、または、データベースがデータを出し入れするロケーションを記述します。

データ型
*型*が文字列型の場合は長さ
メモリアドレス
インジケータ変数

インジケータ変数と NULL

インジケータ変数は、データベースに NULL 値を渡したり、データベースから NULL 値を取り出したりするのに使用されます。インジケータ変数は、データベース操作中にトランケーション条件が発生したことをデータベースサーバが示すのにも使用されます。インジケータ変数はデータベースの値を受け取るのに十分な領域がない場合、正の値に設定されます。

動的 SELECT 文

シングルローだけを返す SELECT 文は、動的に準備し、その後に EXECUTE 文に INTO 句を指定してローを 1 つだけ取り出すようにできます。ただし、複数のローを返す SELECT 文では動的カーソルを使用します。

動的カーソルでは、結果はホスト変数のリスト、または FETCH 文 (FETCH INTO と FETCH USING DESCRIPTOR) で指定する SQLDA に入ります。通常、SELECT リスト項目の数は不明であるため、多くの場合、SQLDA を使用します。DESCRIBE SELECT LIST 文で SQLDA に SELECT リスト項目の型を設定します。その後、fill_sqlda 関数または fill_s_sqlda 関数を使用して、値用の領域を割り付けます。情報は FETCH USING DESCRIPTOR 文で取り出します。

次は典型的な例です。

```
EXEC SQL BEGIN DECLARE SECTION;
char comm[200];
EXEC SQL END DECLARE SECTION;
int actual_size;
SQLDA * sqlda;
...
sprintf( comm, "SELECT * FROM %s", table_name );
EXEC SQL PREPARE S1 FROM :comm;
/* Initial guess of 10 columns in result.
   If it is wrong, it is corrected right
   after the first DESCRIBE by reallocating
   sqlda and doing DESCRIBE again. */
sqlda = alloc_sqlda( 10 );
EXEC SQL DESCRIBE SELECT LIST FOR S1
  INTO sqlda;
if( sqlda->sqld > sqlda->sqln )
{
  actual_size = sqlda->sqld;
  free_sqlda( sqlda );
  sqlda = alloc_sqlda( actual_size );
  EXEC SQL DESCRIBE SELECT LIST FOR S1
    INTO sqlda;
}
fill_sqlda( sqlda );
EXEC SQL DECLARE C1 CURSOR FOR S1;
EXEC SQL OPEN C1;
EXEC SQL WHENEVER NOTFOUND {break};
for( ;; )
{
  EXEC SQL FETCH C1 USING DESCRIPTOR sqlda;
  /* do something with data */
}
EXEC SQL CLOSE C1;
EXEC SQL DROP STATEMENT S1;
```

注意： リソースを無駄に消費しないように、文は使用後に削除してください。

SQLDA (SQL Descriptor Area)

SQLDA (SQL Descriptor Area) は動的 SQL 文で使用されるインタフェース構造体です。この構造体を使用して、ホスト変数と SELECT 文の結果に関する情報を、データベースとの間で受け渡しします。SQLDA はヘッダファイル `sqlda.h` に定義されています。

データベースのインタフェース共有ライブラリまたは DLL には SQLDA の管理に使用できる関数が用意されています。

ホスト変数を静的 SQL 文で使用するときは、プリプロセッサがホスト変数用の SQLDA を構成します。実際にデータベースサーバとの間でやりとりされるのは、この SQLDA です。

SQLDA ヘッダファイル

`sqlda.h` の内容を、次に示します。

```
#ifndef _SQLDA_H_INCLUDED
#define _SQLDA_H_INCLUDED
#define II_SQLDA
#include "sqlca.h"
#if defined( _SQL_PACK_STRUCTURES )
  #if defined( _MSC_VER ) && _MSC_VER > 800
    #pragma warning(push)
    #pragma warning(disable:4103)
  #endif
  #include "pshpk1.h"
#endif
#define SQL_MAX_NAME_LEN    30
#define _sqldafar
typedef short int a_sql_type;

struct sqlname {
  short int   length; /* length of char data */
  char        data[ SQL_MAX_NAME_LEN ]; /* data */
};

struct sqlvar {          /* array of variable descriptors */
  short int   sqltype;   /* type of host variable */
  a_sql_len   sqllen;    /* length of host variable */
  void        *sqldata;  /* address of variable */
  a_sql_len   *sqlind;   /* indicator variable pointer */
  struct sqlname sqlname;
};

#if defined( _SQL_PACK_STRUCTURES )
  #include "poppk.h"
  /* The SQLDA should be 4-byte aligned */
  #include "pshpk4.h"
```

```
#endif

struct sqlda {
  unsigned char   sqldaid[8]; /* eye catcher "SQLDA" */
  a_sql_int32     sqldabc; /* length of sqlda structure */
  short int       sqln;    /* descriptor size in number of entries */
  short int       sqld;    /* number of variables found by DESCRIBE */
  struct sqlvar   sqlvar[1]; /* array of variable descriptors */
};

#define SCALE(sqllen)       ((sqllen)/256)
#define PRECISION(sqllen)   ((sqllen)&0xff)
#define SET_PRECISION_SCALE(sqllen,precision,scale)   ¥
                sqllen = (scale)*256 + (precision)
#define DECIMALSTORAGE(sqllen)  (PRECISION(sqllen)/2 + 1)
typedef struct sqlda    SQLDA;
typedef struct sqlvar   SQLVAR, SQLDA_VARIABLE;
typedef struct sqlname  SQLNAME, SQLDA_NAME;
#ifndef SQLDASIZE
#define SQLDASIZE(n)  ( sizeof( struct sqlda ) +  ¥
              (n-1) * sizeof( struct sqlvar) )
#endif
#if defined( _SQL_PACK_STRUCTURES )
  #include "poppk.h"
  #if defined( _MSC_VER ) && _MSC_VER > 800
    #pragma warning(pop)
  #endif
#endif
#endif
```

SQLDA のフィールド

SQLDA のフィールドの意味を次に示します。

フィールド	説明
sqldaid	SQLDA 構造体の ID として文字列 SQLDA が格納される 8 バイトの文字フィールド。このフィールドはデバッグ時にメモリの中身を見るときに役立ちます。
sqldabc	SQLDA 構造体の長さが入る 32 ビットの整数。
sqln	sqlvar 配列に割り付けられた変数記述子の数。
sqld	有効な変数記述子の数 (ホスト変数の記述情報を含む)。このフィールドは DESCRIBE 文で設定される。また、このフィールドは、データベースサーバにデータを渡すときに設定できる。
sqlvar	struct sqlvar 型の記述子の配列。各要素がホスト変数を記述する。

SQLDA のホスト変数の記述

SQLDA の sqlvar 構造体がそれぞれ 1 つのホスト変数を記述しています。sqlvar 構造体のフィールドの意味を次に示します。

- **sqltype –** この記述子で記述している変数の型。

  低位ビットは NULL 値が可能かどうかを示します。有効な型と定数の定義は `sqldef.h` ヘッダファイルにあります。

  このフィールドは DESCRIBE 文で設定されます。データベースサーバにデータを渡したり、データベースサーバからデータを取り出したりするときに、このフィールドはどの型にでも設定できます。必要な型変換は自動的に行われます。
- **sqllen –** 変数の長さ。sqllen 値は a_sql_len 型です。長さが実際に何を意味するかは、型情報と SQLDA の使用法によって決まります。

  データ型 LONG VARCHAR、LONG NVARCHAR、LONG BINARY の場合は、sqllen フィールドの代わりに、データ型の構造体 DT_LONGVARCHAR、DT_LONGNVARCHAR、または DT_LONGBINARY の array_len フィールドが使用されます。
- **sqldata –** この変数が占有するメモリへのポインタ。このメモリ領域は sqltype と sqllen フィールドに合致させてください。

  UPDATE 文、INSERT 文では、sqldata ポインタが NULL ポインタの場合、この変数は操作に関係しません。FETCH では、sqldata ポインタが NULL ポインタの場合、データは返されません。つまり、sqldata ポインタが返すカラムは、*バインドされていないカラム*です。

  DESCRIBE 文が LONG NAMES を使用している場合、このフィールドは結果セットカラムのロングネームを保持します。さらに、DESCRIBE 文が DESCRIBE USER TYPES 文の場合は、このフィールドはカラムではなくユーザ定義のデータ型のロングネームを保持します。型がベースタイプの場合、フィールドは空です。
- **sqlind –** インジケータ値へのポインタ。インジケータ値は a_sql_len 型です。負のインジケータ値は NULL 値を意味します。正のインジケータ値は、この変数が FETCH 文でトランケートされたことを示し、インジケータ値にはトランケートされる前のデータの長さが入ります。conversion_error データベースオプションを Off に設定した場合、-2 の値は変換エラーを示します。

  sqlind ポインタが NULL ポインタの場合、このホスト変数に対応するインジケータ変数はありません。

  sqlind フィールドは、DESCRIBE 文でパラメータタイプを示すのにも使用されます。ユーザ定義のデータ型の場合、このフィールドは

DT_HAS_USERTYPE_INFO に設定されます。この場合、DESCRIBE USER TYPES を実行すると、ユーザ定義のデータ型についての情報を取得できます。

- **sqlname –** 次のような VARCHAR に似た構造体。

```
struct sqlname {
   short int  length;
   char  data[ SQL_MAX_NAME_LEN ];
};
```

DESCRIBE 文によって設定され、それ以外では使用されません。このフィールドは DESCRIBE 文のフォーマットによって意味が異なります。

- **SELECT LIST –** 名前データバッファには SELECT リストの対応する項目のカラム見出しが入ります。
- **BIND VARIABLES –** 名前データバッファにはバインド変数として使用されたホスト変数名が入ります。無名のパラメータマーカが使用されている場合は、"?" が入ります。

DESCRIBE SELECT LIST 文では、指定のインジケータ変数にはすべて、SELECT リスト項目が更新可能かどうかを示すフラグが設定されます。このフラグの詳細は、`sqldef.h` ヘッダファイルにあります。

DESCRIBE 文が DESCRIBE USER TYPES 文の場合、このフィールドはカラムではなくユーザ定義のデータ型のロングネームを保持します。型がベースタイプの場合、フィールドは空です。

SQLDA の sqllen フィールドの値

DESCRIBE 実行後、値を送信するとき、およびデータを取得するときの SQLDA の sqllen フィールドの値です。

DESCRIBE 実行後の SQLDA の sqllen フィールドの値

DESCRIBE 文は、データベースから取り出したデータを格納するために必要なホスト変数、またはデータベースにデータを渡すために必要なホスト変数に関する情報を取得します。

次の表は、データベースのさまざまな型に対して DESCRIBE 文 (SELECT LIST 文と BIND VARIABLE 文の両方) が返す sqllen と sqltype 構造体のメンバーの値を示します。ユーザ定義のデータベースデータ型の場合、ベースタイプが記述されます。

プログラムでは DESCRIBE の返す型と長さを使用できます。別の型も使用できます。データベースサーバはどの型でも型変換を行います。sqldata フィールドの指すメモリは sqltype と sqllen フィールドに合致させてください。Embedded SQL の型は、sqltype でビット処理 AND と DT_TYPES を指定して (sqltype & DT_TYPES) 取得します。

データベースのフィールドの型	返される Embedded SQL の型	describe で返される長さ (バイト単位)
BIGINT	DT_BIGINT	8
BINARY(n)	DT_BINARY	n
BIT	DT_BIT	1
CHAR(n)	DT_FIXCHAR[1]	データベースの文字セットからクライアントの CHAR 文字セットへの変換時の最大データ拡張に n を掛けた値。この長さが 32767 バイトを超える場合、返される Embedded SQL の型は、32767 バイト長の DT_LONGVARCHAR になる。
CHAR(n CHAR)	DT_FIXCHAR[1]	クライアントの CHAR 文字セット内の文字の最大長に n を掛けた値。この長さが 32767 バイトを超える場合、返される Embedded SQL の型は、32767 バイト長の DT_LONGVARCHAR になる。
DATE	DT_DATE	フォーマットされた文字列の最大長
DECIMAL(p,s)	DT_DECIMAL	SQLDA の長さフィールドの低位バイトが p に、高位バイトが s に設定される。sqlda.h の PRECISION および SCALE マクロを参照。
DOUBLE	DT_DOUBLE	8
FLOAT	DT_FLOAT	4
INT	DT_INT	4
LONG BINARY	DT_LONGBINARY	32767
LONG NVARCHAR	DT_LONGVARCHAR / DT_LONGNVARCHAR[2]	32767
LONG VARCHAR	DT_LONGVARCHAR	32767

データベースのフィールドの型	返される Embedded SQL の型	describe で返される長さ (バイト単位)
NCHAR(n)	DT_FIXCHAR / DT_NFIXCHAR[2]	クライアントの NCHAR 文字セット内の文字の最大長に n を掛けた値。この長さが 32767 バイトを超える場合、返される Embedded SQL の型は、32767 バイト長の DT_LONGNVARCHAR になる。
NVARCHAR(n)	DT_VARCHAR / DT_NVARCHAR[2]	クライアントの NCHAR 文字セット内の文字の最大長に n を掛けた値。この長さが 32767 バイトを超える場合、返される Embedded SQL の型は、32767 バイト長の DT_LONGNVARCHAR になる。
REAL	DT_FLOAT	4
SMALLINT	DT_SMALLINT	2
TIME	DT_TIME	フォーマットされた文字列の最大長
TIMESTAMP	DT_TIMESTAMP	フォーマットされた文字列の最大長
TINYINT	DT_TINYINT	1
UNSIGNED BIGINT	DT_UNSBIGINT	8
UNSIGNED INT	DT_UNSINT	4
UNSIGNED SMALLINT	DT_UNSSMALLINT	2
VARCHAR(n)	DT_VARCHAR[1]	データベースの文字セットからクライアントの CHAR 文字セットへの変換時の最大データ拡張に n を掛けた値。この長さが 32767 バイトを超える場合、返される Embedded SQL の型は、32767 バイト長の DT_LONGVARCHAR になる。
VARCHAR(n CHAR)	DT_VARCHAR[1]	クライアントの CHAR 文字セット内の文字の最大長に n を掛けた値。この長さが 32767 バイトを超える場合、返される Embedded SQL の型は、32767 バイト長の DT_LONGVARCHAR になる。

[1] クライアントの CHAR 文字セットの最大バイト長が 32767 バイトを超える場合、CHAR および VARCHAR について返される型は DT_LONGVARCHAR になります。

[2] クライアントの NCHAR 文字セットの最大バイト長が 32767 バイトを超える場合、NCHAR および NVARCHAR について返される型は DT_LONGNVARCHAR になります。NCHAR、NVARCHAR、LONG NVARCHAR はそれぞれデフォルトで DT_FIXCHAR、DT_VARCHAR、DT_LONGVARCHAR と記述されます。db_change_nchar_charset 関数が呼び出された場合、これらの型はそれぞれ DT_NFIXCHAR、DT_NVARCHAR、DT_LONGNVARCHAR と記述されます。

値を送信するときの SQLDA の sqllen フィールドの値

次の表は、SQLDA においてデータベースサーバにデータを渡すとき、値の長さをどう指定するかを示します。

この場合は、表で示したデータ型だけを使用できます。DT_DATE、DT_TIME、DT_TIMESTAMP 型は、データベースに情報を渡すときは、DT_STRING 型と同じものとして扱われます。値は、NULL で終了する適切な日付または時刻形式の文字列にしてください。

Embedded SQL のデータ型	長さを設定するプログラム動作
DT_BIGINT	動作不要
DT_BINARY(n)	BINARY 構造体の長さフィールドから取る
DT_BIT	動作不要
DT_DATE	末尾の NULL 文字によって長さが決まる
DT_DOUBLE	動作不要
DT_FIXCHAR(n)	SQLDA の長さフィールドが文字列の長さを決定する
DT_FLOAT	動作不要
DT_INT	動作不要
DT_LONGBINARY	長さフィールドが無視される
DT_LONGNVARCHAR	長さフィールドが無視される
DT_LONGVARCHAR	長さフィールドが無視される
DT_NFIXCHAR(n)	SQLDA の長さフィールドが文字列の長さを決定する

Embedded SQL のデータ型	長さを設定するプログラム動作
DT_NSTRING	末尾の ¥0 によって長さが決まる。ansi_blanks オプションが On に設定されていて、データベースでブランクが埋め込まれる場合、SQLDA の長さフィールドは、値が含まれているバッファの長さ (少なくても値の長さと末尾の NULL 文字の分を足した長さ) に設定される。
DT_NVARCHAR	NVARCHAR 構造体の長さフィールドから取る
DT_SMALLINT	動作不要
DT_STRING	末尾の ¥0 によって長さが決まる。ansi_blanks オプションが On に設定されていて、データベースでブランクが埋め込まれる場合、SQLDA の長さフィールドは、値が含まれているバッファの長さ (少なくても値の長さと末尾の NULL 文字の分を足した長さ) に設定される。
DT_TIME	末尾の NULL 文字によって長さが決まる
DT_TIMESTAMP	末尾の NULL 文字によって長さが決まる
DT_TIMESTAMP_STRUCT	動作不要
DT_UNSBIGINT	動作不要
DT_UNSINT	動作不要
DT_UNSSMALLINT	動作不要
DT_VARCHAR(n)	VARCHAR 構造体の長さフィールドから取る
DT_VARIABLE	末尾の ¥0 によって長さが決まる

データを取得するときの SQLDA の sqllen フィールドの値

次の表は、SQLDA を使用してデータベースからデータを取り出すときの、長さフィールドの値を示します。データを取り出すときには、sqllen フィールドは変更されません。

この場合に使用できるのは、表で示したインタフェースデータ型だけです。DT_DATE、DT_TIME、DT_TIMESTAMP 型はデータベースから情報を取り出すときは DT_STRING と同じものとして扱われます。値は現在の日付形式に従って文字列としてフォーマットされます。

Embedded SQL のデータ型	データを受け取るときにプログラムが長さフィールドに設定する値	値をフェッチした後、データベースが長さ情報を返す方法
DT_BIGINT	動作不要	動作不要
DT_BINARY(n)	BINARY 構造体の最大長 (n+2)。n の最大値は 32765。	BINARY 構造体の len フィールドに実際の長さをバイト単位で設定
DT_BIT	動作不要	動作不要
DT_DATE	バッファの長さ	文字列の末尾の NULL 文字
DT_DOUBLE	動作不要	動作不要
DT_FIXCHAR(n)	バッファの長さ (バイト)。n の最大値は 32767。	バッファの長さまでブランクを埋め込む
DT_FLOAT	動作不要	動作不要
DT_INT	動作不要	動作不要
DT_LONGBINARY	長さフィールドが無視される	長さフィールドが無視される
DT_LONGNVARCHAR	長さフィールドが無視される	長さフィールドが無視される
DT_LONGVARCHAR	長さフィールドが無視される	長さフィールドが無視される
DT_NFIXCHAR(n)	バッファの長さ (バイト)。n の最大値は 32767。	バッファの長さまでブランクを埋め込む
DT_NSTRING	バッファの長さ	文字列の末尾の NULL 文字
DT_NVARCHAR(n)	NVARCHAR 構造体の最大長 (n+2)。n の最大値は 32765。	NVARCHAR 構造体の len フィールドに実際の長さをバイト単位で設定
DT_SMALLINT	動作不要	動作不要
DT_STRING	バッファの長さ	文字列の末尾の NULL 文字
DT_TIME	バッファの長さ	文字列の末尾の NULL 文字
DT_TIMESTAMP	バッファの長さ	文字列の末尾の NULL 文字

Embedded SQL のデータ型	データを受け取るときにプログラムが長さフィールドに設定する値	値をフェッチした後、データベースが長さ情報を返す方法
DT_TIMESTAMP_STRUCT	動作不要	動作不要
DT_UNSBIGINT	動作不要	動作不要
DT_UNSINT	動作不要	動作不要
DT_UNSSMALLINT	動作不要	動作不要
DT_VARCHAR(n)	VARCHAR 構造体の最大長 (n+2)。n の最大値は 32765。	VARCHAR 構造体の len フィールドに実際の長さをバイト単位で設定

Embedded SQL を使用してデータをフェッチする方法

Embedded SQL でデータをフェッチするには SELECT 文を使用します。これには 2 つの場合があります。

- **SELECT 文がローを返さないか、1 つだけ返す場合 –** INTO 句を使用して、戻り値をホスト変数に直接割り当てます。
- **SELECT 文が複数のローを返す可能性がある場合 –** カーソルを使用して結果セットのローを管理します。

ローを返さないか、1 つだけ返す SELECT 文

シングルロークエリがデータベースから取り出すローの数は多くても 1 つだけです。シングルロークエリの SELECT 文では、INTO 句が SELECT リストの後、FROM 句の前にきます。INTO 句には、SELECT リストの各項目の値を受け取るホスト変数のリストを指定します。SELECT リスト項目と同数のホスト変数を指定してください。ホスト変数と一緒に、結果が NULL であることを示すインジケータ変数も指定できます。

SELECT 文が実行されると、データベースサーバは結果を取り出して、ホスト変数に格納します。クエリの結果、複数のローが取り出されると、データベースサーバはエラーを返します。

クエリの結果、選択されたローが存在しない場合、ローが見つからないことを示すエラー (SQLCODE 100) が返されます。エラーと警告は、SQLCA 構造体で返されます。

例

次のコードは Employees テーブルから正しくローをフェッチできた場合は 1 を、ローが存在しない場合は 0 を、エラーが発生した場合は -1 を返します。

```
EXEC SQL BEGIN DECLARE SECTION;
long      id;
char      name[41];
char      sex;
char      birthdate[15];
a_sql_len ind_birthdate;
EXEC SQL END DECLARE SECTION;
...
int find_employee( long employee_id )
{
  id = employee_id;
  EXEC SQL SELECT GivenName ||
    ' ' || Surname, Sex, BirthDate
    INTO :name, :sex,
      :birthdate:ind_birthdate
    FROM Employees
    WHERE EmployeeID = :id;
  if( SQLCODE == SQLE_NOTFOUND )
  {
    return( 0 ); /* employee not found */
  }
  else if( SQLCODE < 0 )
  {
    return( -1 ); /* error */
  }
  else
  {
    return( 1 ); /* found */
  }
}
```

Embedded SQL でのカーソル

カーソルは、結果セットに複数のローがあるクエリからローを取り出すために使用されます。カーソルは、SQL クエリのためのハンドルつまり識別子であり、結果セット内の位置を示します。

Embedded SQL でのカーソル管理では、次の手順を実行します。

1. DECLARE CURSOR 文を使って、特定の SELECT 文のためのカーソルを宣言します。
2. OPEN 文を使って、カーソルを開きます。
3. FETCH 文を使って、一度に 1 つのローをカーソルから取り出します。
4. 警告「`Row Not Found`」(ローが見つかりません) が返されるまで、ローをフェッチします。
   エラーと警告は、SQLCA 構造体で返されます。

5. CLOSE 文を使ってカーソルを閉じます。

デフォルトによって、カーソルはトランザクション終了時 (COMMIT または ROLLBACK 時) に自動的に閉じられます。WITH HOLD 句を指定して開いたカーソルは、明示的に閉じるまで以降のトランザクション中も開いたままになります。

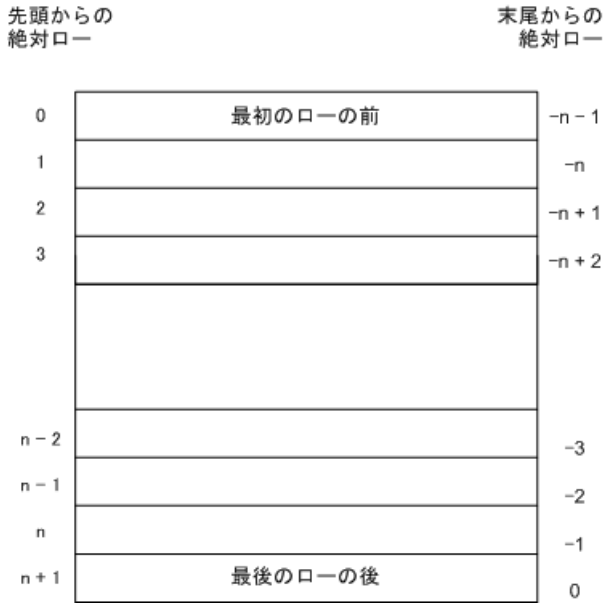
次は、簡単なカーソル使用の例です。

```
void print_employees( void )
{
  EXEC SQL BEGIN DECLARE SECTION;
  char      name[50];
  char      sex;
  char      birthdate[15];
  a_sql_len ind_birthdate;
  EXEC SQL END DECLARE SECTION;
  EXEC SQL DECLARE C1 CURSOR FOR
    SELECT GivenName || ' ' || Surname, Sex, BirthDate FROM Employees;
  EXEC SQL OPEN C1;
  for( ;; )
  {
    EXEC SQL FETCH C1 INTO :name, :sex, :birthdate:ind_birthdate;
    if( SQLCODE == SQLE_NOTFOUND )
    {
      break;
    }
    else if( SQLCODE < 0 )
    {
      break;
    }

    if( ind_birthdate < 0 )
    {
      strcpy( birthdate, "UNKNOWN" );
    }
    printf( "Name: %s Sex: %c Birthdate: %s¥n", name, sex,
birthdate );
  }
  EXEC SQL CLOSE C1;
}
```

カーソル位置
カーソルは、次のいずれかの位置にあります。

- ローの上
- 最初のローの前
- 最後のローの後



カーソルを開くと最初のローの前に置かれます。カーソル位置は FETCH 文を使用して移動できます。カーソルはクエリ結果の先頭または末尾を基点にした絶対位置に位置付けできます。カーソルの現在位置を基準にした相対位置にも移動できます。

カーソルの現在位置のローを更新または削除するために、特別な*位置付け*型の UPDATE 文と DELETE 文があります。先頭のローの前か、末尾のローの後にカーソルがある場合、カーソルに対応するローがないことを示すエラーが返されます。

PUT 文で、カーソルにローを挿入できます。

カーソル位置に関する問題

DYNAMIC SCROLL カーソルに挿入や更新をいくつか行うと、カーソルの位置の問題が生じます。SELECT 文に ORDER BY 句を指定しないかぎり、データベースサーバはカーソル内の予測可能な位置にはローを挿入しません。場合によっては、カーソルを閉じてもう一度開かないと、挿入したローが表示されないことがあります。

SAP Sybase IQ では、カーソルを開くためにテンポラリテーブルを作成する必要がある場合にこうしたことが起こります。

UPDATE 文は、カーソル内でローを移動させることがあります。これは、既存のインデックスを使用する ORDER BY 句がカーソルに指定されている場合に発生します (テンポラリテーブルは作成されません)。

ワイドフェッチ (配列フェッチ)

FETCH 文は一度に複数のローをフェッチするように変更できます。こうするとパフォーマンスが向上することがあります。これを*ワイドフェッチ*または*配列フェッチ*といいます。

SAP Sybase IQ は、ワイドプットとワイド挿入もサポートします。

Embedded SQL でワイドフェッチを使用するには、コードに次のような FETCH 文を含めます。

```
EXEC SQL FETCH ... ARRAY nnn
```

ARRAY *nnn* は FETCH 文の最後の項目です。フェッチ回数を示す *nnn* にはホスト変数も使用できます。SQLDA 内の変数の数はローあたりのカラム数と *nnn* との積にしてください。最初のローは SQLDA の変数 0 から (ロー当たりのカラム数) - 1 に入り、以後のローも同様です。

各カラムは、SQLDA の各ローと同じ型にしてください。型が同じでない場合、SQLDA_INCONSISTENT エラーが返されます。

サーバはフェッチしたレコード数を SQLCOUNT に返します。この値は、エラーまたは警告がないかぎり、常に正の数です。ワイドフェッチでは、エラーではなくて SQLCOUNT が 1 の場合、有効なローが 1 つフェッチされたことを示します。

例

次は、ワイドフェッチの使用例です。このコードは `%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥esqlwidefetch¥widefetch.sqc` にもあります。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqldef.h"
EXEC SQL INCLUDE SQLCA;

EXEC SQL WHENEVER SQLERROR { PrintSQLError();
        goto err; };

static void PrintSQLError()
{
  char buffer[200];

  printf( "SQL error %d -- %s¥n",
     SQLCODE,
     sqlerror_message( &sqlca,
              buffer,
              sizeof( buffer ) ) );
}
static SQLDA * PrepareSQLDA(
```

```
  a_sql_statement_number stat0,
  unsigned  width,
  unsigned  *cols_per_row )

/* Allocate a SQLDA to be used for fetching from
   the statement identified by "stat0". "width"
   rows are retrieved on each FETCH request.
   The number of columns per row is assigned to
   "cols_per_row". */
{
  int           num_cols;
  unsigned      row, col, offset;
  SQLDA *       sqlda;
  EXEC SQL BEGIN DECLARE SECTION;
  a_sql_statement_number  stat;
  EXEC SQL END DECLARE SECTION;
  stat = stat0;
  sqlda = alloc_sqlda( 100 );
  if( sqlda == NULL ) return( NULL );
  EXEC SQL DESCRIBE :stat INTO sqlda;
  *cols_per_row = num_cols = sqlda->sqld;
  if( num_cols * width > sqlda->sqln )
  {
    free_sqlda( sqlda );
    sqlda = alloc_sqlda( num_cols * width );
    if( sqlda == NULL ) return( NULL );
    EXEC SQL DESCRIBE :stat INTO sqlda;
  }
  // copy first row in SQLDA setup by describe
  // to following (wide) rows
  sqlda->sqld = num_cols * width;
  offset = num_cols;
  for( row = 1; row < width; row++ )
  {
    for( col = 0;
      col < num_cols;
      col++, offset++ )
    {
      sqlda->sqlvar[offset].sqltype =
          sqlda->sqlvar[col].sqltype;
      sqlda->sqlvar[offset].sqllen =
          sqlda->sqlvar[col].sqllen;
      // optional: copy described column name
      memcpy( &sqlda->sqlvar[offset].sqlname,
              &sqlda->sqlvar[col].sqlname,
              sizeof( sqlda->sqlvar[0].sqlname ) );
    }
  }
  fill_s_sqlda( sqlda, 40 );
  return( sqlda );
err:
  return( NULL );
}
static void PrintFetchedRows(
  SQLDA * sqlda,
  unsigned cols_per_row )
```

```
{
  /* Print rows already wide fetched in the SQLDA */
  long      rows_fetched;
  int       row, col, offset;

  if( SQLCOUNT == 0 )
  {
    rows_fetched = 1;
  }
  else
  {
    rows_fetched = SQLCOUNT;
  }
  printf( "Fetched %d Rows:¥n", rows_fetched );
  for( row = 0; row < rows_fetched; row++ )
  {
    for( col = 0; col < cols_per_row; col++ )
    {
      offset = row * cols_per_row + col;
      printf( " ¥"%s¥"",
        (char *)sqlda->sqlvar[offset].sqldata );
     }
     printf( "¥n" );
  }
}
static int DoQuery(
  char * query_str0,
  unsigned fetch_width0 )
{
  /* Wide Fetch "query_str0" select statement
   * using a width of "fetch_width0" rows" */
  SQLDA *                 sqlda;
  unsigned                cols_per_row;
  EXEC SQL BEGIN DECLARE SECTION;
  a_sql_statement_number  stat;
  char *                  query_str;
  unsigned                fetch_width;
  EXEC SQL END DECLARE SECTION;

  query_str = query_str0;
  fetch_width = fetch_width0;

  EXEC SQL PREPARE :stat FROM :query_str;
  EXEC SQL DECLARE QCURSOR CURSOR FOR :stat
       FOR READ ONLY;
  EXEC SQL OPEN QCURSOR;
  sqlda = PrepareSQLDA( stat,
    fetch_width,
    &cols_per_row );
  if( sqlda == NULL )
  {
    printf( "Error allocating SQLDA¥n" );
    return( SQLE_NO_MEMORY );
  }
  for( ;; )
```

```
  {
    EXEC SQL FETCH QCURSOR INTO DESCRIPTOR sqlda
      ARRAY :fetch_width;
    if( SQLCODE != SQLE_NOERROR ) break;
    PrintFetchedRows( sqlda, cols_per_row );
  }
  EXEC SQL CLOSE QCURSOR;
  EXEC SQL DROP STATEMENT :stat;
  free_filled_sqlda( sqlda );
err:
  return( SQLCODE );
}
void main( int argc, char *argv[] )
{
  /* Optional first argument is a select statement,
   * optional second argument is the fetch width */
  char *query_str =
    "SELECT GivenName, Surname FROM Employees";
  unsigned fetch_width = 10;

  if( argc > 1 )
  {
    query_str = argv[1];
    if( argc > 2 )
    {
      fetch_width = atoi( argv[2] );
      if( fetch_width < 2 )
      {
        fetch_width = 2;
      }
    }
  }
  db_init( &sqlca );
  EXEC SQL CONNECT "DBA" IDENTIFIED BY "sql";

  DoQuery( query_str, fetch_width );

  EXEC SQL DISCONNECT;
err:
  db_fini( &sqlca );
}
```

ワイドフェッチの使用上の注意

- PrepareSQLDA 関数では、alloc_sqlda 関数を使用して SQLDA 用のメモリを割り付けています。この関数では、alloc_sqlda_noind 関数とは違って、インジケータ変数用の領域が確保できます。
- フェッチされたローの数が要求より少ないが 0 ではない場合 (たとえばカーソルの終端に達したとき)、SQLDA のフェッチされなかったローに対応する項目は、インジケータ変数に値を設定して、NULL として返されます。インジケータ変数が指定されていない場合は、エラーが発生します

(SQLE_NO_INDICATOR: NULL の結果に対してインジケータ変数がありません)。

- フェッチしようとしたローが更新され、警告 (SQLE_ROW_UPDATED_WARNING) が出された場合、フェッチは警告を引き起こしたロー上で停止します。そのときまでに処理されたすべてのロー (警告を起こしたローも含む) の値が返されます。SQLCOUNT には、フェッチしたローの数 (警告を引き起こしたローも含む) が入ります。残りの SQLDA の項目はすべて NULL になります。
- フェッチしようとしたローが削除またはロックされ、エラー (SQLE_NO_CURRENT_ROW または SQLE_LOCKED) が発生した場合、SQLCOUNT にはエラー発生までに読み込まれたローの数が入ります。この値にはエラーを起こしたローは含みません。SQLDA にはローの値は入りません。エラーのときは、SQLDA に値が返らないためです。SQLCOUNT の値は、ローを読み込む必要がある場合、カーソルの再位置付けに使用できます。

Embedded SQL を使用して long 値を送信し、取り出す方法

Embedded SQL アプリケーションで LONG VARCHAR、LONG NVARCHAR、LONG BINARY の値を送信し、取り出す方法は、他のデータ型とは異なります。標準的な SQLDA フィールドのデータ長は 32767 バイトに制限されています。これは、長さの情報を保持するフィールド (sqllen、*sqlind) が 16 ビット値であるためです。これらの値を 32 ビット値に変更すると、既存のアプリケーションが中断します。

LONG VARCHAR、LONG NVARCHAR、LONG BINARY の値の記述方法は、他のデータ型の場合と同じです。

静的 SQL 構造体

データ型 LONG BINARY、LONG VARCHAR、LONG NVARCHAR の割り付けられた長さ、格納された長さ、トランケートされていない長さを保持するには、別々のフィールドが使用されます。静的 SQL データ型は、`sqlca.h` に次のように定義されています。

```
#define DECL_LONGVARCHAR( size )         ¥
  struct { a_sql_uint32    array_len;    ¥
           a_sql_uint32    stored_len;   ¥
           a_sql_uint32    untrunc_len;  ¥
           char            array[size+1];¥
         }
#define DECL_LONGNVARCHAR( size )        ¥
  struct { a_sql_uint32    array_len;    ¥
           a_sql_uint32    stored_len;   ¥
           a_sql_uint32    untrunc_len;  ¥
```

```
          char            array[size+1];¥
        }
#define DECL_LONGBINARY( size )          ¥
  struct { a_sql_uint32    array_len;    ¥
           a_sql_uint32    stored_len;   ¥
           a_sql_uint32    untrunc_len;  ¥
           char            array[size];  ¥
        }
```

動的 SQL 構造体
動的 SQL の場合は、sqltype フィールドを必要に応じて DT_LONGVARCHAR、DT_LONGNVARCHAR、または DT_LONGBINARY に設定します。対応する LONGVARCHAR、LONGNVARCHAR、LONGBINARY の構造体は、次のとおりです。

```
typedef struct LONGVARCHAR {
    a_sql_uint32    array_len;
    a_sql_uint32    stored_len;
    a_sql_uint32    untrunc_len;
    char            array[1];
} LONGVARCHAR, LONGNVARCHAR, LONGBINARY;
```

構造体メンバーの定義
静的 SQL 構造体と動的 SQL 構造体のいずれの場合も、構造体メンバーは次のように定義します。

- **array_len –** (送信と取得。) 構造体の配列部分に割り付けられたバイト数。
- **stored_len –** (送信と取得。) 配列に格納されるバイト数。常に array_len および untrunc_len 以下になります。
- **untrunc_len –** (取得のみ。) 値がトランケートされなかった場合に配列に格納されるバイト数。常に stored_len 以上になります。トランケーションが発生すると、値は array_len より大きくなります。

静的 SQL を使用した LONG データの取り出し

静的 SQL を使用して LONG VARCHAR、LONG NVARCHAR、LONG BINARY の値を受信します。

前提条件

この作業を実行するための前提条件はありません。

手順

1. 必要に応じて、DECL_LONGVARCHAR、DECL_LONGNVARCHAR、または DECL_LONGBINARY 型のホスト変数を宣言します。array_len メンバーの値は自動的に設定されます。
2. FETCH、GET DATA、または EXECUTE INTO を使用してデータを取り出します。SAP Sybase IQ によって次の情報が設定されます。
   - **インジケータ変数 –** 値が NULL の場合は負、トランケーションなしの場合は 0 で、トランケートされていない最大 32767 バイトの正の長さです。
   - **stored_len –** 配列に格納されるバイト数。常に array_len および untrunc_len 以下になります。
   - **untrunc_len –** 値がトランケートされなかった場合に配列に格納されるバイト数。常に stored_len 以上になります。トランケーションが発生すると、値は array_len より大きくなります。

LONG データは静的 SQL を使用して取得されます。

動的 SQL を使用した LONG データの取り出し

動的 SQL を使用して LONG VARCHAR、LONG NVARCHAR、LONG BINARY の値を受信します。

前提条件

この作業を実行するための前提条件はありません。

手順

1. sqltype フィールドを必要に応じて DT_LONGVARCHAR、DT_LONGNVARCHAR、または DT_LONGBINARY に設定します。
2. sqldata フィールドを、LONGVARCHAR、LONGNVARCHAR、または LONGBINARY 構造体を指すように設定します。

   LONGVARCHARSIZE(*n*)、LONGNVARCHARSIZE(*n*)、または LONGBINARYSIZE(*n*) マクロを使用して、array フィールドに *n* バイトのデータを保持するために割り付ける合計バイト数を決定できます。
3. ホスト変数構造体の array_len フィールドを、array フィールドに割り付けるバイト数に設定します。
4. FETCH、GET DATA、または EXECUTE INTO を使用してデータを取り出します。SAP Sybase IQ によって次の情報が設定されます。

- *** sqlind –** この sqlda フィールドは、値が NULL の場合は負、トランケーションなしの場合は 0 で、トランケートされていない最大 32767 バイトの正の長さです。
- **stored_len –** 配列に格納されるバイト数。常に array_len および untrunc_len 以下になります。
- **untrunc_len –** 値がトランケートされなかった場合に配列に格納されるバイト数。常に stored_len 以上になります。トランケーションが発生すると、値は array_len より大きくなります。

LONG データは動的 SQL を使用して取得されます。

静的 SQL を使用した LONG データの送信

Embedded SQL アプリケーションから静的 SQL を使用して、LONG 値をデータベースに送信します。

前提条件

この作業を実行するための前提条件はありません。

手順

1. 必要に応じて、DECL_LONGVARCHAR、DECL_LONGNVARCHAR、または DECL_LONGBINARY 型のホスト変数を宣言します。
2. NULL を送信する場合は、インジケータ変数を負の値に設定します。
3. ホスト変数構造体の stored_len フィールドを、array フィールド内のデータのバイト数に設定します。
4. カーソルを開くか、文を実行して、データを送信します。

Embedded SQL アプリケーションでの、データベースへの LONG 値の送信準備ができています。

動的 SQL を使用した LONG データの送信

Embedded SQL アプリケーションから動的 SQL を使用して、LONG 値をデータベースに送信します。

前提条件

この作業を実行するための前提条件はありません。

手順

1. sqltype フィールドを必要に応じて DT_LONGVARCHAR、DT_LONGNVARCHAR、または DT_LONGBINARY に設定します。
2. NULL を送信する場合は、`* sqlind` を負の値に設定します。
3. NULL 値を送信しない場合は、sqldata フィールドを LONGVARCHAR、LONGNVARCHAR、LONGBINARY ホスト変数構造体を指すように設定します。

   LONGVARCHARSIZE(*n*)、LONGNVARCHARSIZE(*n*)、または LONGBINARYSIZE(*n*) マクロを使用して、array フィールドに *n* バイトのデータを保持するために割り付ける合計バイト数を決定できます。
4. ホスト変数構造体の array_len フィールドを、array フィールドに割り付けるバイト数に設定します。
5. ホスト変数構造体の stored_len フィールドを、array フィールド内のデータのバイト数に設定します。このバイト数は array_len 以下にしてください。
6. カーソルを開くか、文を実行して、データを送信します。

Embedded SQL アプリケーションでの、データベースへの LONG 値の送信準備ができています。

Embedded SQL での簡単なストアドプロシージャ

Embedded SQL でストアドプロシージャを作成して呼び出すことができます。

CREATE PROCEDURE は、CREATE TABLE など、他のデータ定義文と同じように埋め込むことができます。また、ストアドプロシージャを実行する CALL 文を埋め込むこともできます。次のコードフラグメントは、Embedded SQL でストアドプロシージャを作成して実行する方法を示しています。

```
EXEC SQL CREATE PROCEDURE pettycash(
 IN Amount DECIMAL(10,2) )
BEGIN
 UPDATE account
 SET balance = balance - Amount
 WHERE name = 'bank';

 UPDATE account
 SET balance = balance + Amount
 WHERE name = 'pettycash expense';
END;
EXEC SQL CALL pettycash( 10.72 );
```

ホスト変数の値をストアドプロシージャに渡したい場合、または出力変数を取り出したい場合は、CALL 文を準備して実行します。次のコードフラグメントは、

ホスト変数の使用方法を示しています。EXECUTE 文では、USING 句と INTO 句の両方を使用しています。

```
EXEC SQL BEGIN DECLARE SECTION;
double  hv_expense;
double  hv_balance;
EXEC SQL END DECLARE SECTION;

// Code here
EXEC SQL CREATE PROCEDURE pettycash(
   IN expense  DECIMAL(10,2),
   OUT endbalance DECIMAL(10,2) )
BEGIN
  UPDATE account
  SET balance = balance - expense
  WHERE name = 'bank';
  UPDATE account
  SET balance = balance + expense
  WHERE name = 'pettycash expense';

  SET endbalance = ( SELECT balance FROM account
         WHERE name = 'bank' );
END;

EXEC SQL PREPARE S1 FROM 'CALL pettycash( ?, ? )';
EXEC SQL EXECUTE S1 USING :hv_expense INTO :hv_balance;
```

結果セットを持つストアドプロシージャ

データベースプロシージャでは SELECT 文も使用できます。プロシージャの宣言に RESULT 句を使用して、結果セットのカラムの数、名前、型を指定します。結果セットのカラムは出力パラメータとは異なります。結果セットを持つプロシージャでは、SELECT 文の代わりに CALL 文を使用してカーソル宣言を行うことができます。

```
EXEC SQL BEGIN DECLARE SECTION;
 char hv_name[100];
EXEC SQL END DECLARE SECTION;

EXEC SQL CREATE PROCEDURE female_employees()
  RESULT( name char(50) )
BEGIN
  SELECT GivenName || Surname FROM Employees
  WHERE Sex = 'f';
END;

EXEC SQL PREPARE S1 FROM 'CALL female_employees()';

EXEC SQL DECLARE C1 CURSOR FOR S1;
EXEC SQL OPEN C1;
for(;;)
{
  EXEC SQL FETCH C1 INTO :hv_name;
  if( SQLCODE != SQLE_NOERROR ) break;
```

```
  printf( "%s¥n", hv_name );
}
EXEC SQL CLOSE C1;
```

この例では、プロシージャは EXECUTE 文ではなく OPEN 文を使用して呼び出されています。OPEN 文の場合は、SELECT 文が見つかるまでプロシージャが実行されます。このとき、C1 はデータベースプロシージャ内の SELECT 文のためのカーソルです。操作を終了するまで FETCH 文のすべての形式 (後方スクロールと前方スクロール) を使用できます。CLOSE 文によってプロシージャの実行が終了します。

この例では、たとえプロシージャ内の SELECT 文の後に他の文があっても、その文は実行されません。SELECT の後の文を実行するには、RESUME cursor-name 文を使用してください。RESUME 文は警告 (SQLE_PROCEDURE_COMPLETE)、または別のカーソルが残っていることを意味する SQLE_NOERROR を返します。次は select が 2 つあるプロシージャの例です。

```
EXEC SQL CREATE PROCEDURE people()
  RESULT( name char(50) )
BEGIN
 SELECT GivenName || Surname
 FROM Employees;

 SELECT GivenName || Surname
 FROM Customers;
END;

EXEC SQL PREPARE S1 FROM 'CALL people()';
EXEC SQL DECLARE C1 CURSOR FOR S1;
EXEC SQL OPEN C1;
while( SQLCODE == SQLE_NOERROR )
{
  for(;;)
  {
    EXEC SQL FETCH C1 INTO :hv_name;
    if( SQLCODE != SQLE_NOERROR ) break;
    printf( "%s¥n", hv_name );
  }
  EXEC SQL RESUME C1;
}
EXEC SQL CLOSE C1;
```

CALL 文の動的カーソル

ここまでの例は静的カーソルを使用していました。CALL 文では完全に動的なカーソルも使用できます。

DESCRIBE 文はプロシージャコールでも完全に機能します。DESCRIBE OUTPUT で、結果セットの各カラムを記述した SQLDA を生成します。

プロシージャに結果セットがない場合、SQLDA にはプロシージャの INOUT パラメータまたは OUT パラメータの記述が入ります。DESCRIBE INPUT 文はプロシージャの IN または INOUT の各パラメータを記述した SQLDA を生成します。

DESCRIBE ALL
DESCRIBE ALL は IN、INOUT、OUT、RESULT セットの全パラメータを記述します。DESCRIBE ALL は SQLDA のインジケータ変数に追加情報を設定します。

CALL 文を記述すると、インジケータ変数の DT_PROCEDURE_IN と DT_PROCEDURE_OUT ビットが設定されます。DT_PROCEDURE_IN は IN または INOUT パラメータを示し、DT_PROCEDURE_OUT は INOUT または OUT パラメータを示します。プロシージャの RESULT カラムはどちらのビットもクリアされています。

DESCRIBE OUTPUT の後、これらのビットは結果セットを持っている文 (OPEN、FETCH、RESUME、CLOSE を使用する必要がある) と持っていない文 (EXECUTE を使用する必要がある) を区別するのに使用できます。

複数の結果セット
複数の結果セットを返すプロシージャにおいて、結果セットの形が変わる場合は、各 RESUME 文の後で再記述してください。

カーソルの現在位置を記述するには、文ではなくカーソルを記述する必要があります。

Embedded SQL を使用した要求管理

通常の Embedded SQL アプリケーションは、各データベース要求が完了するまで待ってから次のステップを実行する必要があります。そのため、アプリケーションで複数の実行スレッドを使用することで、他のタスクと同時に実行できます。

1つの実行スレッドを使用する必要がある場合は、db_register_a_callback function 関数を DB_CALLBACK_WAIT オプションとともに使用してコールバック関数を登録することにより、ある程度のマルチタスクを実現できます。コールバック関数は、データベースサーバまたはクライアントライブラリがデータベース要求を処理している間、インタフェースライブラリによって繰り返し呼び出されます。

コールバック関数では、別のデータベース要求を開始することはできませんが、db_cancel_request 関数を使用して現在の要求をキャンセルすることはできます。メッセージハンドラ内で db_is_working 関数を使用して、処理中のデータベース要求があるかどうかを判断できます。

Embedded SQL を使用したデータベースのバックアップ

データベースのバックアップには **BACKUP DATABASE** 文の使用をお奨めします。

別の方法として、db_backup 関数を使用して Embedded SQL アプリケーションでオンラインバックアップを実行できます。SAP Sybase IQ ユーティリティも、この関数を使用しています。

データベースツールの DBBackup 関数を使用して、SAP Sybase IQ Backup ユーティリティと直接やりとりすることもできます。

他のどのバックアップ方法でも希望どおりのバックアップができない場合にのみ、db_backup 関数を使用するプログラムを記述してください。

ライブラリ関数のリファレンス

SQL プリプロセッサはインタフェースライブラリまたは DLL 内の関数呼び出しを生成します。SQL プリプロセッサが生成する呼び出しの他に、データベース操作を容易にする一連のライブラリ関数も用意されています。このような関数のプロトタイプは EXEC SQL INCLUDE SQLCA 文で含めます。

この項では、これらの関数のリファレンスについて説明します。

DLL のエントリポイント
DLL のエントリポイントはすべて同じです。ただし、プロトタイプには、次のように各 DLL に適した変更子が付きます。

エントリポイントを移植可能な方法で宣言するには、`sqlca.h` で定義されている `_esqlentry_` を使用します。これは、__stdcall の値に解析されます。

alloc_sqlda 関数

SQLDA に *numvar* 変数の記述子を割り付けます。

構文

```
struct sqlda * alloc_sqlda( unsigned numvar );
```

パラメータ

- **numvar –** 割り付ける変数記述子の数。

戻り値
成功した場合は SQLDA へのポインタを返し、十分なメモリがない場合は null ポインタを返します。

備考
SQLDA に *numvar* 変数の記述子を割り付けます。SQLDA の sqln フィールドを *numvar* に初期化します。インジケータ変数用の領域が割り付けられ、この領域を指すようにインジケータポインタが設定されて、インジケータ値が 0 に初期化されます。メモリを割り付けできない場合は、NULL ポインタが返されます。alloc_sqlda_noind 関数の代わりに、この関数を使用することをおすすめします。

alloc_sqlda_noind 関数

SQLDA に *numvar* 変数の記述子を割り付けます。

構文

```
struct sqlda * alloc_sqlda_noind( unsigned numvar );
```

パラメータ

- **numvar –** 割り付ける変数記述子の数。

戻り値
成功した場合は SQLDA へのポインタを返し、十分なメモリがない場合は null ポインタを返します。

備考
SQLDA に *numvar* 変数の記述子を割り付けます。SQLDA の sqln フィールドを *numvar* に初期化します。インジケータ変数用の領域は割り付けられず、インジケータポインタは NULL ポインタとして設定されます。メモリを割り付けできない場合は、NULL ポインタが返されます。

db_backup 関数

この関数を使用してアプリケーションにバックアップ機能を追加することもできますが、このタスクには **BACKUP DATABASE** 文を使用することをお奨めします。

構文

```
void db_backup(
SQLCA * sqlca,
int op,
int file_num,
unsigned long page_num,
struct sqlda * sqlda);
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。
- **op –** 実行する動作または操作。
- **file_num –** データベースのファイル番号。
- **page_num –** データベースのページ番号。範囲内の値は、0 からページの最大数から 1 を引いた値までになります。
- **sqlda –** SQLDA 構造体へのポインタ。

権限
BACKUP DATABASE システム権限を持つユーザとして接続しているか、SYS_RUN_REPLICATION_ROLE システムロールを持っている必要があります。

備考
この関数を使用してアプリケーションにバックアップ機能を追加することもできますが、このタスクには **BACKUP DATABASE** 文を使用することをお奨めします。

実行されるアクションは、*op* パラメータの値によって決まります。

- **DB_BACKUP_START –** これを呼び出してからバックアップを開始します。1 つのデータベースサーバに対して同時に実行できるバックアップはデータベースごとに 1 つだけです。バックアップが完了するまで (*op* 値に DB_BACKUP_END を指定して db_backup が呼び出されるまで)、データベースチェックポイントは無効にされます。バックアップが開始できない場合は、SQLCODE が SQLE_BACKUP_NOT_STARTED になります。それ以外の場合は、*sqlca* の SQLCOUNT フィールドには各データベースページのサイズが設定されます。バックアップは一度に 1 ページずつ処理されます。

  file_num、*page_num*、および *sqlda* パラメータは無視されます。
- **DB_BACKUP_OPEN_FILE –** *file_num* で指定されたデータベースファイルを開きます。これによって、指定されたファイルの各ページを DB_BACKUP_READ_PAGE を使用してバックアップできます。有効なファイル番号は、ルートデータベースファイルの場合は 0 から DB_BACKUP_MAX_FILE の値までで、トランザクションログファイルの場合は 0 から DB_BACKUP_TRANS_LOG_FILE の値までです。指定されたファイルが存在しない場合は、SQLCODE は SQLE_NOTFOUND になります。その他の場合は、SQLCOUNT はファイルのページ数を含み、SQLIOESTIMATE にはデータベースファイルが作成された時間を示す 32 ビットの値 (POSIX time_t) が含まれます。オペレーティングシステムファイル名は SQLCA の *sqlerrmc* フィールドにあります。

page_num および *sqlda* パラメータは無視されます。

- **DB_BACKUP_READ_PAGE –** *file_num* で指定されたデータベースファイルから 1 ページを読み込みます。*page_num* の値は、0 から、DB_BACKUP_OPEN_FILE オペレーションを使用した db_backup に対する呼び出しの成功によって SQLCOUNT に返されるページ数未満の値までです。その他の場合は、SQLCODE は SQLE_NOTFOUND になります。*sqlda* 記述子は、バッファを指す DT_BINARY または DT_LONG_BINARY 型の変数で設定してください。このバッファは、DB_BACKUP_START オペレーションを使用した db_backup の呼び出しで SQLCOUNT フィールドに返されるサイズのバイナリデータを保持するのに十分な大きさにしてください。

  DT_BINARY データは、2 バイトの長さフィールドの後に実際のバイナリデータを含んでいるので、バッファはページサイズより 2 バイトだけ大きくなければなりません。

  注意： この呼び出しによって、指定されたデータベースのページがバッファにコピーされます。ただし、バックアップメディアにバッファを保存するのはアプリケーションの役割です。

- **DB_BACKUP_READ_RENAME_LOG –** このアクションは、トランザクションログの最後のページが返された後にデータベースサーバがトランザクションログの名前を変更して新しいログを開始する点を除けば、DB_BACKUP_READ_PAGE と同じです。

  データベースサーバが現時点でログの名前を変更できない場合 (バージョン 7.0.x 以前のデータベースで、完了していないトランザクションがある場合など) は、SQLE_BACKUP_CANNOT_RENAME_LOG_YET エラーが設定されます。この場合は、返されたページを使用しないで、要求を再発行して SQLE_NOERROR を受け取ってからページを書き込んでください。SQLE_NOTFOUND 条件を受け取るまでページを読むことを続けてください。

  SQLE_BACKUP_CANNOT_RENAME_LOG_YET エラーは、何回も、複数のページについて返されることがあります。リトライループでは、要求が多すぎてサーバが遅くなることがないように遅延を入れてください。

  SQLE_NOTFOUND 条件を受け取った場合は、トランザクションログはバックアップに成功してファイルの名前は変更されています。古い方のトランザクションログファイルの名前は、SQLCA の *sqlerrmc* フィールドに返されます。

  db_backup を呼び出した後に、sqlda->sqlvar[0].sqlind の値を調べてください。この値が 0 より大きい場合は、最後のログページは書き込まれていて、ログファイルの名前は変更されています。新しい名前はまだ sqlca.sqlerrmc にありますが、SQLCODE 値は SQLE_NOERROR になります。

この後、ファイルを閉じてバックアップを終了するとき以外は、db_backup を再度呼び出さないでください。再度呼び出すと、バックアップされているログファイルの 2 番目のコピーが得られ、SQLE_NOTFOUND を受け取ります。

- **DB_BACKUP_CLOSE_FILE –** 1 つのファイルの処理が完了したときに呼び出して、*file_num* で指定されたデータベースファイルを閉じます。

  page_num および *sqlda* パラメータは無視されます。
- **DB_BACKUP_END –** バックアップの最後に呼び出します。このバックアップが終了するまで、他のバックアップは開始できません。チェックポイントが再度有効にされます。

  file_num、*page_num*、および *sqlda* パラメータは無視されます。
- **DB_BACKUP_PARALLEL_START –** 並列バックアップを開始します。DB_BACKUP_START と同様、1 つのデータベースサーバに対して同時に実行できるバックアップはデータベースごとに 1 つだけです。バックアップが完了するまで (*op* 値に DB_BACKUP_END を指定して db_backup が呼び出されるまで)、データベースチェックポイントは無効にされます。バックアップが開始できない場合は、SQLE_BACKUP_NOT_STARTED を受け取ります。それ以外の場合は、sqlca の SQLCOUNT フィールドには各データベースページのサイズが設定されます。

  file_num パラメータは、トランザクションログの名前を変更し、トランザクションログの最後のページが返された後で新しいログを開始するようデータベースに指示します。値が 0 以外の場合、トランザクションログの名前が変更されるか、トランザクションログが再起動されます。それ以外の場合は、名前の変更も再起動も行われません。このパラメータにより、並列バックアップオペレーションの間は実行できない DB_BACKUP_READ_RENAME_LOG オペレーションが必要なくなります。

  page_num パラメータは、データベースのページ数で表したクライアントバッファの最大サイズをデータベースサーバに通知します。サーバ側では、並列バックアップの読み込みで連続したページブロックを読み込もうとします。この値によって、サーバは割り付けるブロックのサイズを知ることができます。*nnn* の値を渡すと、サーバはクライアントが最大 *nnnn* ページのデータベースページをサーバから一度に受け入れる準備があることを認識します。サーバは、*nnn* ページのブロックに十分なメモリを割り付けられない場合、*nnn* より小さいサイズのページブロックを返す可能性があります。クライアント側で DB_BACKUP_PARALLEL_START を呼び出すまでデータベースページのサイズがわからない場合は、DB_BACKUP_INFO オペレーションでこの値をサーバに渡すことができます。この値は、バックアップページを取得する初回の呼び出し (DB_BACKUP_PARALLEL_READ) を実行する前に指定する必要があります。

注意： db_backup を使用して並列バックアップを開始すると、ライタースレッドは作成されません。db_backup の呼び出し元でデータを受け取り、ライターとして動作するようにしてください。

- **DB_BACKUP_INFO –** このパラメータは、並列バックアップに関する追加情報をデータベースに提供します。*file_num* パラメータは提供される情報の種類を示し、*page_num* パラメータには値が指定されます。DB_BACKUP_INFO で次の追加情報を指定できます。
  - **DB_BACKUP_INFO_PAGES_IN_BLOCK –** *page_num* 引数には、1 つのブロックで送信される最大ページ数が含まれます。
  - **DB_BACKUP_INFO_CHKPT_LOG –** これは、クライアント側では **BACKUP DATABASE** 文の WITH CHECKPOINT LOG オプションと同等です。DB_BACKUP_CHKPT_COPY の *page_num* 値は COPY を示しますが、DB_BACKUP_CHKPT_NOCOPY の値は NO COPY を示します。値が指定されないと、デフォルトで COPY に設定されます。
- **DB_BACKUP_PARALLEL_READ –** このオペレーションでは、データベースサーバから 1 ブロック分のページを読み込みます。DB_BACKUP_OPEN_FILE オペレーションを使用してバックアップするファイルをすべて開いてから DB_BACKUP_PARALLEL_READ オペレーションを呼び出します。DB_BACKUP_PARALLEL_READ では、*file_num* および *page_num* 引数は無視されます。

  sqlda 記述子は、バッファを指す DT_LONGBINARY 型の変数で設定してください。DB_BACKUP_START_PARALLEL または DB_BACKUP_INFO オペレーションで指定した、*nnn* ページのサイズのバイナリデータを格納するのに十分なバッファを確保してください。

  サーバは特定のデータベースファイルについてデータベースページの連続したブロックを返します。ブロックの最初のページのページ番号は、SQLCOUNT フィールドに返されます。ページが含まれているファイルのファイル番号は SQLIOESTIMATE フィールドに返され、この値は DB_BACKUP_OPEN_FILE 呼び出しで使用されるファイル番号の 1 つに一致します。返されるデータのサイズは DT_LONGBINARY 変数の *stored_len* フィールドから取得でき、常にデータベースページのサイズの倍数になります。この呼び出しによって返されるデータには指定されたファイルの連続したページのブロックが含まれていますが、別のデータブロックが順番に返されることや、データベースファイルのすべてのページが別のデータベースファイルのページの前に返されることを想定するのは危険です。呼び出し元では、他の別個のファイルの一部分や、別の呼び出しによって開かれたデータベースファイルの一部分を順番に関係なく受信できるよう準備しておく必要があります。

  アプリケーションでは、読み込むデータのサイズが 0 になるか、sqlda->sqlvar[0].sqlind の値が 0 より大きくなるまで、このオペレーションを繰り返し

呼び出してください。トランザクションログの名前を変更するか再起動してバックアップを開始すると、SQLERROR は SQLE_BACKUP_CANNOT_RENAME_LOG_YET に設定される場合があります。この場合は、返されたページを使用しないで、要求を再発行して SQLE_NOERROR を受け取ってからデータを書き込んでください。SQLE_BACKUP_CANNOT_RENAME_LOG_YET エラーは、何回も、複数のページについて返されることがあります。リトライループでは、要求が多すぎてデータベースサーバが遅くなることがないように遅延を入れてください。最初の 2 つの条件のいずれかを満たすまで、引き続きページを読み込みます。

dbbackup ユーティリティは、次のようなアルゴリズムを使用します。これは、C のコードではなく、エラーチェックは含んでいません。

```
sqlda->sqld = 1;
sqlda->sqlvar[0].sqltype = DT_LONGBINARY

/* Allocate LONGBINARY value for page buffer. It MUST have */
/* enough room to hold the requested number (128) of database pages
*/
sqlda->sqlvar[0].sqldata = allocated buffer

/* Open the server files needing backup */
for file_num = 0 to DB_BACKUP_MAX_FILE
  db_backup( ... DB_BACKUP_OPEN_FILE, file_num ... )
  if SQLCODE == SQLE_NO_ERROR
    /* The file exists */
    num_pages = SQLCOUNT
    file_time = SQLE_IO_ESTIMATE
    open backup file with name from sqlca.sqlerrmc
end for

/* read pages from the server, write them locally */
while TRUE
  /* file_no and page_no are ignored */
  db_backup( &sqlca, DB_BACKUP_PARALLEL_READ, 0, 0, &sqlda );

  if SQLCODE != SQLE_NO_ERROR
     break;

  if buffer->stored_len == 0 || sqlda->sqlvar[0].sqlind > 0
     break;

  /* SQLCOUNT contains the starting page number of the block    */
  /* SQLIOESTIMATE contains the file number the pages belong to */
  write block of pages to appropriate backup file
end while

/* close the server backup files */
for file_num = 0 to DB_BACKUP_MAX_FILE
    /* close backup file */
    db_backup( ... DB_BACKUP_CLOSE_FILE, file_num ... )
end for
```

```
/* shut down the backup */
db_backup( ... DB_BACKUP_END ... )

/* cleanup */
free page buffer
```

db_cancel_request 関数

現在アクティブなデータベースサーバ要求をキャンセルします。この関数は、キャンセル要求を送信する前に、データベースサーバ要求がアクティブかどうかを調べます。

構文

```
int db_cancel_request( SQLCA * sqlca );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。

戻り値
キャンセル要求が送信された場合は 1、要求が送信されなかった場合は 0 を返します。

備考
戻り値が 0 でないことが、要求がキャンセルされたことを意味するわけではありません。キャンセル要求とデータベースまたはサーバからの応答が行き違いになるようなタイミング上の危険性はほとんどありません。このような場合は、関数が TRUE を返しても、キャンセルは効力を持ちません。

db_cancel_request 関数は非同期で呼び出すことができます。別の要求が使用している可能性のある SQLCA を使用して非同期で呼び出すことができるのは、データベースインタフェースライブラリではこの関数と db_is_working だけです。

カーソル操作実行要求をキャンセルした場合は、カーソルの位置は確定されません。キャンセルした後は、カーソルを絶対位置に位置付けるか、閉じます。

db_change_char_charset 関数

この接続用にアプリケーションの CHAR 文字セットを変更します。

構文

```
unsigned int db_change_char_charset(
SQLCA * sqlca,
char * charset );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。
- **charset –** 文字セットを表す文字列。

戻り値
変更が成功した場合は 1 を返し、それ以外の場合は 0 を返します。

備考
DT_FIXCHAR、DT_VARCHAR、DT_LONGVARCHAR、および DT_STRING 型を使用して送信およびフェッチされたデータは CHAR 文字セットにあります。

db_change_nchar_charset 関数

この接続用にアプリケーションの NCHAR 文字セットを変更します。

構文

```
unsigned int db_change_nchar_charset(
SQLCA * sqlca,
char * charset );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。
- **charset –** 文字セットを表す文字列。

戻り値
変更が成功した場合は 1 を返し、それ以外の場合は 0 を返します。

備考
DT_NFIXCHAR、DT_NVARCHAR、DT_LONGNVARCHAR、DT_NSTRING ホスト変数型を使用して送信およびフェッチされたデータの文字セットは NCHAR です。

db_change_nchar_charset 関数が呼び出されないと、すべてのデータは CHAR 文字セットを使用して送信およびフェッチされます。通常、Unicode データを送信およびフェッチするアプリケーションでは、NCHAR 文字セットを UTF-8 に設定します。

この関数が呼び出される場合、文字セットのパラメータは一般に "UTF-8" です。NCHAR 文字セットは UTF-16 に設定できません。

Embedded SQL の場合、NCHAR、NVARCHAR、LONG NVARCHAR はそれぞれデフォルトで DT_FIXCHAR、DT_VARCHAR、DT_LONGVARCHAR と記述されま

す。db_change_nchar_charset 関数が呼び出された場合、これらの型はそれぞれ DT_NFIXCHAR、DT_NVARCHAR、DT_LONGNVARCHAR と記述されます。

db_find_engine 関数

ローカルデータベースサーバのステータス情報を返します。

構文

```
unsigned short db_find_engine(
SQLCA * sqlca,
char * name );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。
- **name –** NULL またはサーバの名前を含む文字列。

戻り値

サーバのステータスを unsigned short 値として返します。共有メモリにサーバが見つからない場合は 0 を返します。

備考

name という名前のローカルデータベースサーバのステータス情報を示す unsigned short 値を返します。指定された名前のサーバが共有メモリを介して見つからない場合、戻り値は 0 です。0 以外の値は、ローカルサーバが現在稼働中であることを示します。

name に NULL ポインタが指定されている場合は、デフォルトデータベースサーバについて情報が返されます。

戻り値の各ビットには特定の情報が保持されています。さまざまな情報に対するビット表現の定数は、`sqldef.h` ヘッダファイルに定義されています。次にその意味を説明します。

- **DB_ENGINE –** このフラグは常に設定されています。
- **DB_CLIENT –** このフラグは常に設定されています。
- **DB_CAN_MULTI_DB_NAME –** このフラグは使用されなくなりました。
- **DB_DATABASE_SPECIFIED –** このフラグは常に設定されています。
- **DB_ACTIVE_CONNECTION –** このフラグは常に設定されています。
- **DB_CONNECTION_DIRTY –** このフラグは使用されなくなりました。
- **DB_CAN_MULTI_CONNECT –** このフラグは使用されなくなりました。
- **DB_NO_DATABASES –** このフラグは、サーバがデータベースを起動していない場合に設定されます。

db_fini 関数

この関数は、データベースインタフェースまたは DLL で使用されたリソースを解放します。

構文

```
int db_fini( SQLCA * sqlca );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。

戻り値
成功した場合は 0 以外の値を返します。それ以外の場合は 0 を返します。

備考
db_fini が呼び出された後に、他のライブラリ呼び出しをしたり、Embedded SQL 文を実行したりしないでください。処理中にエラーが発生すると、SQLCA 内でエラーコードが設定され、関数は 0 を返します。エラーがなければ、0 以外の値が返されます。

使用する SQLCA ごとに 1 回ずつ db_fini を呼び出します。

Windows ダイナミックリンクライブラリ内の DllMain 関数から直接的または間接的に db_fini 関数を呼び出さないでください。DllMain のエントリポイント関数は、簡単な初期化および終了処理のみを実行するためのものです。db_fini を呼び出した場合、デッドロックや循環依存が発生する可能性があります。

db_get_property 関数

接続するデータベースインタフェースまたはサーバに関する情報を取得します。

構文

```
unsigned int db_get_property(
SQLCA * sqlca,
a_db_property property,
char * value_buffer,
int value_buffer_size );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。

- **a_db_property –** 要求されるプロパティ。DB_PROP_CLIENT_CHARSET、DB_PROP_SERVER_ADDRESS、または DB_PROP_DBLIB_VERSION のいずれかです。
- **value_buffer –** NULL で終了する文字列としてプロパティ値が入ります。
- **value_buffer_size –** 末尾の NULL 文字用の領域を含む、文字列 value_buffer の最大長。

戻り値
正常終了すると 1 を返し、それ以外は 0 を返します。

備考
次のプロパティがサポートされます。

- **DB_PROP_CLIENT_CHARSET –** このプロパティ値はクライアントの文字セットを取得します ("windows-1252" など)。
- **DB_PROP_SERVER_ADDRESS –** このプロパティ値は、現在の接続のサーバネットワークアドレスを印刷可能な文字列として取得します。共有メモリプロトコルは、アドレスに対して必ず空の文字列を返します。TCP/IP プロトコルは、空でない文字列アドレスを返します。
- **DB_PROP_DBLIB_VERSION –** このプロパティ値は、データベースインタフェースライブラリのバージョンを取得します ("16.0.0.1297" など)。

db_init 関数

この関数は、データベースインタフェースライブラリを初期化します。

構文

```
int db_init( SQLCA * sqlca );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。

戻り値
成功した場合は 0 以外の値を返します。それ以外の場合は 0 を返します。

備考
この関数は、他のライブラリが呼び出される前、および Embedded SQL 文が実行される前に呼び出します。インタフェースライブラリがプログラムのために必要とするリソースは、この呼び出しで割り付けられて初期化されます。

プログラムの最後でリソースを解放するには db_fini を使用します。処理中にエラーが発生した場合は、SQLCA に渡されて 0 が返されます。エラーがなかった場合は、0 以外の値が返され、Embedded SQL 文と関数の使用を開始できます。

通常は、この関数を一度だけ呼び出して、ヘッダファイル sqlca.h に定義されているグローバル変数 sqlca のアドレスを渡してください。DLL または Embedded SQL を使用する複数のスレッドがあるアプリケーションを作成する場合は、使用する SQLCA ごとに 1 回ずつ db_init を呼び出します。

db_is_working 関数

アプリケーションで *sqlca* を使用するデータベース要求が処理中である場合は 1 を返します。*sqlca* を使用する要求が処理中でない場合は 0 を返します。

構文

```
unsigned short db_is_working( SQLCA * sqlca );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。

戻り値
アプリケーションで *sqlca* を使用するデータベース要求が処理中である場合は 1、*sqlca* を使用する要求が処理中でない場合は 0。

備考
この関数は、非同期で呼び出すことができます。別の要求が使用している可能性のある SQLCA を使用して非同期で呼び出すことができるのは、データベースインタフェースライブラリではこの関数と db_cancel_request だけです。

db_locate_servers 関数

TCP/IP で受信しているローカルネットワーク上のすべての SAP Sybase IQ データベースサーバをリストして、dblocate ユーティリティによって表示される情報にプログラムからアクセスできるようにします。

構文

```
unsigned int db_locate_servers(
SQLCA * sqlca,
SQL_CALLBACK_PARM callback_address,
void * callback_user_data );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。
- **callback_address –** コールバック関数のアドレス。
- **callback_user_data –** データを格納するユーザ定義領域のアドレス。

戻り値
正常終了すると 1 を返し、それ以外は 0 を返します。

備考
コールバック関数には、次のプロトタイプが必要です。

```
int (*)( SQLCA * sqlca,
a_server_address * server_addr,
void * callback_user_data );
```

コールバック関数は、検出されたサーバごとに呼び出されます。コールバック関数が 0 を返すと、db_locate_servers はサーバ間の反復を中止します。

コールバック関数に渡される sqlca と callback_user_data は、db_locate_servers に渡されるものと同じです。2 番目のパラメータは a_server_address 構造体へのポインタです。a_server_address は次の内容を `sqlca.h` で定義します。

```
typedef struct a_server_address {
    a_sql_uint32 port_type;
    a_sql_uint32 port_num;
    char         *name;
    char         *address;
} a_server_address;
```

- **port_type –** この時点では常に PORT_TYPE_TCP です (`sqlca.h` 内では 6 に定義されています)。
- **port_num –** このサーバが受信している TCP ポート番号です。
- **name –** サーバ名があるバッファを指します。
- **address –** サーバの IP アドレスがあるバッファを指します。

db_locate_servers_ex 関数

TCP/IP で受信しているローカルネットワーク上のすべての SAP Sybase IQ データベースサーバをリストして、dblocate ユーティリティによって表示される情報にプログラムからアクセスできるようにします。また、コールバック関数に渡されるアドレスの選択に使用するマスクパラメータを提供します。

構文

```
unsigned int db_locate_servers_ex(
SQLCA * sqlca,
SQL_CALLBACK_PARM callback_address,
void * callback_user_data,
unsigned int bitmask);
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。

- **callback_address –** コールバック関数のアドレス。
- **callback_user_data –** データを格納するユーザ定義領域のアドレス。
- **bitmask –** DB_LOOKUP_FLAG_NUMERIC、
  DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT、
  DB_LOOKUP_FLAG_DATABASES のいずれかで構成されるマスク。

戻り値
正常終了すると 1 を返し、それ以外は 0 を返します。

備考
コールバック関数には、次のプロトタイプが必要です。

```
int (*)( SQLCA * sqlca,
a_server_address * server_addr,
void * callback_user_data );
```

コールバック関数は、検出されたサーバごとに呼び出されます。コールバック関数が 0 を返すと、db_locate_servers_ex はサーバ間の反復を中止します。

コールバック関数に渡される sqlca と callback_user_data は、db_locate_servers に渡されるものと同じです。2 番目のパラメータは a_server_address 構造体へのポインタです。a_server_address は次の内容を `sqlca.h` で定義します。

```
typedef struct a_server_address {
    a_sql_uint32   port_type;
    a_sql_uint32   port_num;
    char           *name;
    char           *address;
    char           *dbname;
} a_server_address;
```

- **port_type –** この時点では常に PORT_TYPE_TCP です (`sqlca.h` 内では 6 に定義されています)。
- **port_num –** このサーバが受信している TCP ポート番号です。
- **name –** サーバ名があるバッファを指します。
- **address –** サーバの IP アドレスがあるバッファを指します。
- **dbname –** データベース名があるバッファを指します。

3 つのビットマスクフラグがサポートされています。

DB_LOOKUP_FLAG_NUMERIC
DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT
DB_LOOKUP_FLAG_DATABASES

これらのフラグは `sqlca.h` で定義されており、OR を使用して併用できます。

DB_LOOKUP_FLAG_NUMERIC は、コールバック関数に渡されたアドレスがホスト名ではなく IP アドレスであることを確認します。

DB_LOOKUP_FLAG_ADDRESS_INCLUDES_PORT では、コールバック関数に渡された a_server_address 構造体内の TCP/IP ポート番号がアドレスに含まれていることを示します。

DB_LOOKUP_FLAG_DATABASES は、検出されたデータベースごと、または検出されたデータベースサーバごと (データベース情報の送信をサポートしていないバージョン 9.0.2 以前のデータベースサーバの場合) にコールバック関数が 1 回呼び出されることを示します。

db_register_a_callback 関数

この関数は、コールバック関数を登録します。

構文

```
void db_register_a_callback(
SQLCA * sqlca,
a_db_callback_index index,
( SQL_CALLBACK_PARM ) callback );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。
- **index –** 後述するように、コールバックのタイプを特定するインデックス値。
- **callback –** ユーザ定義コールバック関数のアドレス。

備考

DB_CALLBACK_WAIT コールバックを登録しない場合は、デフォルトでは何もアクションを実行しません。アプリケーションはブロックして、データベースの応答を待ちます。MESSAGE TO CLIENT 文のコールバックを登録してください。

コールバックを削除するには、*callback* 関数として NULL ポインタを渡します。

index パラメータに指定できる値を次に示します。

- **DB_CALLBACK_DEBUG_MESSAGE –** 指定の関数がデバッグメッセージごとに 1 回呼び出され、デバッグメッセージのテキストを含む NULL で終了する文字列が渡されます。デバッグメッセージは、LogFile のファイルに記録されるメッセージです。デバッグメッセージをこのコールバックに渡すには、LogFile 接続パラメータを使用する必要があります。通常、この文字列の末尾の NULL 文字の直前に改行文字 (¥n) が付いています。コールバック関数のプロトタイプを次に示します。

  ```
  void SQL_CALLBACK debug_message_callback(
  SQLCA * sqlca,
  char * message_string );
  ```

- **DB_CALLBACK_START –** プロトタイプを次に示します。

```
void SQL_CALLBACK start_callback( SQLCA * sqlca );
```

この関数は、データベース要求がサーバに送信される直前に呼び出されます。DB_CALLBACK_START は、Windows でのみ使用されます。

- **DB_CALLBACK_FINISH –** プロトタイプを次に示します。

```
void SQL_CALLBACK finish_callback( SQLCA * sqlca );
```

この関数は、データベース要求に対する応答を DBLIB インタフェース DLL が受け取った後に呼び出されます。DB_CALLBACK_FINISH は、Windows オペレーティングシステムでのみ使用されます。

- **DB_CALLBACK_CONN_DROPPED –** プロトタイプを次に示します。

```
void SQL_CALLBACK conn_dropped_callback (
SQLCA * sqlca,
char * conn_name );
```

この関数は、DROP CONNECTION 文を通じた活性タイムアウトのため、またはデータベースサーバがシャットダウンされているために、データベースサーバが接続を切断しようとするときに呼び出されます。複数の接続を区別できるように、接続名 *conn_name* が渡されます。接続が無名の場合は、値が NULL になります。

- **DB_CALLBACK_WAIT –** プロトタイプを次に示します。

```
void SQL_CALLBACK wait_callback( SQLCA * sqlca );
```

この関数は、データベースサーバまたはクライアントライブラリがデータベース要求を処理している間、インタフェースライブラリによって繰り返し呼び出されます。

このコールバックは次のように登録します。

```
db_register_a_callback( &sqlca,
   DB_CALLBACK_WAIT,
   (SQL_CALLBACK_PARM)&db_wait_request );
```

- **DB_CALLBACK_MESSAGE –** この関数は、要求の処理中にサーバから受け取ったメッセージをアプリケーションが処理できるようにするために使用します。メッセージは、SQL MESSAGE 文を使用してクライアントアプリケーションからデータベースサーバに送信できます。実行時間が長いデータベースサーバ文によってメッセージも生成できます。

コールバックプロトタイプを次に示します。

```
void SQL_CALLBACK message_callback(
SQLCA * sqlca,
unsigned char msg_type,
an_sql_code code,
unsigned short length,
char *  msg
);
```

msg_type パラメータは、メッセージの重大度を示します。異なるメッセージタイプを異なる方法で処理できます。*msg_type* に指定できる次の値は、`sqldef.h` で定義されています。

- **MESSAGE_TYPE_INFO –** メッセージタイプは INFO でした。
- **MESSAGE_TYPE_WARNING –** メッセージタイプは WARNING でした。
- **MESSAGE_TYPE_ACTION –** メッセージタイプは ACTION でした。
- **MESSAGE_TYPE_STATUS –** メッセージタイプは STATUS でした。
- **MESSAGE_TYPE_PROGRESS –** メッセージタイプは PROGRESS でした。このタイプのメッセージは、BACKUP DATABASE や LOAD TABLE などの実行時間が長いデータベースサーバ文によって生成されます。

code フィールドにはメッセージに関連付けられた SQLCODE を指定できます。それ以外の場合、値は 0 です。*length* フィールドはメッセージの長さを示します。メッセージは NULL で終了しません。SAP Sybase IQ DBLIB および ODBC クライアントは、DB_CALLBACK_MESSAGE パラメータを使用して進行メッセージを取得します。

たとえば、Interactive SQL のコールバックは STATUS と INFO メッセージを [メッセージ] タブに表示しますが、ACTION と WARNING メッセージはウィンドウに表示されます。アプリケーションがコールバックを登録しない場合は、デフォルトのコールバックが使用されます。これは、すべてのメッセージをサーバログファイルに書き込みます (デバッグがオンでログファイルが指定されている場合)。さらに、メッセージタイプ MESSAGE_TYPE_WARNING と MESSAGE_TYPE_ACTION は、オペレーティングシステムに依存した方法で表示されます。

アプリケーションによってメッセージコールバックが登録されていない場合、クライアントに送信されたメッセージは LogFile 接続パラメータが指定された際にログファイルに保存されます。また、クライアントに送信された ACTION または STATUS メッセージは、Windows オペレーティングシステムではウィンドウに表示され、UNIX オペレーティングシステムでは stderr に記録されます。

- **DB_CALLBACK_VALIDATE_FILE_TRANSFER –** これは、ファイル転送の検証コールバック関数を登録するために使用します。転送を許可する前に、クライアントライブラリは検証コールバックが存在している場合は、それを呼び出します。ストアドプロシージャからなどの間接文の実行中にクライアントのデータ転送が要求された場合、クライアントライブラリはクライアントアプリケーションで検証コールバックが登録されていないかぎり転送を許可しません。どのような状況で検証の呼び出しが行われるかについては、以下でより詳しく説明します。

  コールバックプロトタイプを次に示します。

```
int SQL_CALLBACK file_transfer_callback(
SQLCA * sqlca,
```

```
char * file_name,
int is_write
);
```

file_name パラメータは、読み込みまたは書き込み対象のファイルの名前です。*is_write* パラメータは、読み込み (クライアントからサーバへの転送) が要求された場合は 0、書き込みが要求された場合は 0 以外の値になります。ファイル転送が許可されない場合、コールバック関数は 0 を返します。それ以外の場合は 0 以外の値を返します。

データのセキュリティ上、サーバはファイル転送を要求している文の実行元を追跡します。サーバは、文がクライアントアプリケーションから直接受信されたものかどうかを判断します。クライアントからデータ転送を開始する際に、サーバは文の実行元に関する情報をクライアントソフトウェアに送信します。クライアント側では、クライアントアプリケーションから直接送信された文を実行するためにデータ転送が要求されている場合に限り、Embedded SQL クライアントライブラリはデータの転送を無条件で許可します。それ以外の場合は、上述の検証コールバックがアプリケーションで登録されていることが必要です。登録されていない場合、転送は拒否されて文が失敗し、エラーが発生します。データベース内に既存しているストアドプロシージャがクライアントの文で呼び出された場合、ストアドプロシージャそのものの実行はクライアントの文で開始されたものと見なされません。ただし、クライアントアプリケーションでテンポラリストアドプロシージャを明示的に作成してストアドプロシージャを実行した場合、そのプロシージャはクライアントによって開始されたものとしてサーバは処理します。同様に、クライアントアプリケーションでバッチ文を実行する場合も、バッチ文はクライアントアプリケーションによって直接実行されるものと見なされます。

db_start_database 関数

可能であれば、既存のサーバでデータベースを起動します。そうでない場合は、新しいサーバを起動します。

構文

```
unsigned int db_start_database( SQLCA * sqlca, char * parms );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。
- **parms –** NULL で終了する文字列で、KEYWORD=*value* 形式のパラメータ設定をセミコロンで区切ったリストが含まれています。次に例を示します。

  ```
  "UID=DBA;PWD=sql;DBF=c:¥¥db¥¥mydatabase.db"
  ```

戻り値

成功した場合は 0 以外を返します。それ以外の場合は 0 を返します。

備考
可能であれば、データベースは既存のサーバで起動します。そうでない場合は、新しいサーバを起動します。

データベースがすでに実行している場合、または正しく起動した場合、戻り値は true (0 以外) であり、SQLCODE には 0 が設定されます。エラー情報は SQLCA に返されます。

ユーザ ID とパスワードがパラメータに指定されても、それらは無視されます。

データベースの開始と停止に必要な権限は、サーバコマンドラインで -gd オプションを使用して設定します。

db_start_engine 関数

データベースサーバが実行されていない場合、データベースサーバを起動します。

構文

```
unsigned int db_start_engine( SQLCA * sqlca, char * parms );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。
- **parms –** NULL で終了する文字列で、KEYWORD=*value* 形式のパラメータ設定をセミコロンで区切ったリストが含まれています。次に例を示します。

  ```
  "UID=DBA;PWD=sql;DBF=c:¥¥db¥¥mydatabase.db"
  ```

戻り値
成功した場合は 0 以外を返します。それ以外の場合は 0 を返します。

備考
データベースサーバがすでに実行している場合、または正しく起動した場合、戻り値は TRUE (0 以外) であり、SQLCODE には 0 が設定されます。エラー情報は SQLCA に返されます。

次の db_start_engine 呼び出しは、データベースサーバを起動し、指定されたデータベースをロードして、サーバに demo という名前を付けます。

```
db_start_engine( &sqlca, "DBF=demo.db;START=iqsrv16" );
```

ForceStart (FORCE) 接続パラメータを使用して YES に設定しないかぎり、db_start_engine 関数は、サーバを起動する前にサーバに接続を試みます。これは、すでに稼働しているサーバに起動を試みないようにするためです。

ForceStart 接続を YES に設定した場合は、サーバを起動する前にサーバに接続を試みることはありません。これにより、次の 1 組のコマンドが期待どおりに動作します。

1. server_1 と名付けたデータベースサーバを起動します。
   ```
   iqsrv16 -n server_1 demo.db
   ```
2. 新しいサーバを強制的に起動しそれに接続します。
   ```
   db_start_engine( &sqlda,
     "START=iqsrv16 -n server_2 mydb.db;ForceStart=YES" )
   ```

ForceStart (FORCE) を使用せず、ServerName (Server) パラメータも使用しない場合、2 番目のコマンドでは server_1 に接続しようとします。db_start_engine 関数を実行しても、StartLine (START) パラメータの -n オプションでサーバ名を取得することはできません。

db_stop_database 関数

ServerName (Server) で識別されるサーバ上の DatabaseName (DBN) で識別されるデータベースを停止します。ServerName を指定しない場合は、デフォルトサーバが使用されます。

構文

```
unsigned int db_stop_database( SQLCA * sqlca, char * parms );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。
- **parms –** NULL で終了する文字列で、KEYWORD=*value* 形式のパラメータ設定をセミコロンで区切ったリストが含まれています。次に例を示します。
  ```
  "UID=DBA;PWD=sql;DBF=c:¥¥db¥¥mydatabase.db"
  ```

戻り値
成功した場合は 0 以外を返します。それ以外の場合は 0 を返します。

備考
デフォルトでは、この関数は既存の接続があるデータベースは停止させません。Unconditional (UNC) を `yes` に設定した場合は、既存の接続に関係なくデータベースは停止します。

戻り値 TRUE は、エラーがなかったことを示します。

データベースの開始と停止に必要な権限は、サーバコマンドラインで -gd オプションを使用して設定します。

db_stop_engine 関数

データベースサーバの実行を終了します。

構文

```
unsigned int db_stop_engine( SQLCA * sqlca, char * parms );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。
- **parms –** NULL で終了する文字列で、KEYWORD=*value* 形式のパラメータ設定をセミコロンで区切ったリストが含まれています。次に例を示します。

  ```
  "UID=DBA;PWD=sql;DBF=c:¥¥db¥¥mydatabase.db"
  ```

戻り値
成功した場合は 0 以外を返します。それ以外の場合は 0 を返します。

備考
この関数によって実行されるステップは次のとおりです。

- ServerName (Server) パラメータと一致する名前のローカルデータベースサーバを探します。ServerName の指定がない場合は、デフォルトのローカルデータベースサーバを探します。
- 一致するサーバが見つからない場合は、この関数は正常に値を返します。
- チェックポイントをとってすべてのデータベースを停止するように指示する要求をサーバに送信します。
- データベースサーバをアンロードします。

デフォルトでは、この関数は既存の接続があるデータベースサーバは停止させません。Unconditional=yes パラメータを指定した場合は、既存の接続に関係なくデータベースサーバは停止します。

C のプログラムでは、dbstop を生成する代わりにこの関数を使用できます。戻り値 TRUE は、エラーがなかったことを示します。

db_stop_engine の使用には、-gk サーバオプションで設定される権限が適用されます。

db_string_connect 関数

Embedded SQL CONNECT 文に対する拡張機能を提供します。

構文

```
unsigned int db_string_connect( SQLCA * sqlca, char * parms );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。
- **parms –** NULL で終了する文字列で、KEYWORD=*value* 形式のパラメータ設定をセミコロンで区切ったリストが含まれています。次に例を示します。

```
"UID=DBA;PWD=sql;DBF=c:¥¥db¥¥mydatabase.db"
```

戻り値
成功した場合は 0 以外を返します。それ以外の場合は 0 を返します。

備考
戻り値は、接続の確立に成功した場合は TRUE (0 以外)、失敗した場合は FALSE (0) です。サーバの起動、データベースの開始、または接続に対するエラー情報は SQLCA に返されます。

db_string_disconnect 関数

この関数は、ConnectionName パラメータで識別される接続を解除します。他のパラメータはすべて無視されます。

構文

```
unsigned int db_string_disconnect(
    SQLCA * sqlca,
    char * parms );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。
- **parms –** NULL で終了する文字列で、KEYWORD=*value* 形式のパラメータ設定をセミコロンで区切ったリストが含まれています。次に例を示します。

```
"UID=DBA;PWD=sql;DBF=c:¥¥db¥¥mydatabase.db"
```

戻り値
成功した場合は 0 以外を返します。それ以外の場合は 0 を返します。

備考
文字列に ConnectionName パラメータを指定しない場合は、無名の接続が解除されます。これは、Embedded SQL DISCONNECT 文と同等の機能です。接続に成功した場合、戻り値は TRUE です。エラー情報は SQLCA に返されます。

この関数は、AutoStop=yes 接続パラメータを使用して起動されたデータベースへの接続が他にない場合は、そのデータベースを停止します。また、サーバが

AutoStop=yes パラメータを使用してすでに起動されており、実行中のデータベースが他にない場合も、サーバは停止します。

db_string_ping_server 関数

この関数は、サーバの検索が可能かどうかを判断するために使用され、オプションで、データベースへの正常な接続が実行できるかどうかを判断するために使用されます。

構文

```
unsigned int db_string_ping_server(
SQLCA * sqlca,
char * connect_string,
unsigned int connect_to_db );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。
- **connect_string –** *connect_string* は、通常の接続文字列です。サーバとデータベースの情報を含んでいる場合もあれば、含んでいない場合もあります。
- **connect_to_db –** *connect_to_db* が 0 以外 (TRUE) なら、この関数はサーバ上のデータベースに接続を試みます。TRUE を返すのは、接続文字列で指定したサーバ上の指定したデータベースに接続できた場合のみです。

  connect_to_db が 0 なら、この関数はサーバの検索を試みるだけです。TRUE を返すのは、接続文字列でサーバを検索できた場合のみです。データベースには接続を試みません。

戻り値

サーバまたはデータベースを正常に検索できた場合は TRUE (0 以外)、検索できなかった場合は FALSE (0)。サーバまたはデータベースの検索に対するエラー情報は SQLCA に返されます。

db_time_change 関数

この関数を使用すると、クライアントは、クライアント側での時刻の変更をサーバに通知できます。

構文

```
unsigned int db_time_change(
SQLCA * sqlca);
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

備考
この関数は、タイムゾーン調整を再計算して、その値をサーバに送信します。Windows プラットフォームでは、アプリケーションが WM_TIMECHANGE メッセージを受信したときにこの関数を呼び出すようにすることをおすすめします。これにより、時刻の変更、タイムゾーンの変更、または夏時間に伴う変更があっても、UTC タイムスタンプの整合性が保たれます。

fill_s_sqlda 関数

sqlda 内のすべてのデータ型を DT_STRING 型に変更することを除いて、fill_sqlda と同じです。

構文

```
struct sqlda * fill_s_sqlda(
struct sqlda * sqlda,
unsigned int maxlen );
```

パラメータ

- **sqlda –** SQLDA 構造体へのポインタ。
- **maxlen –** 文字列を割り付ける最大バイト数。

戻り値
成功した場合は *sqlda* を返します。十分なメモリがない場合は NULL を返します。

備考
SQLDA によって最初に指定されたデータ型の文字列表現を保持するために十分な領域が割り付けられます。最大 *maxlen* バイトまでです。SQLDA 内の長さフィールド (sqllen) は適切に修正されます。

SQLDA は、free_filled_sqlda 関数を使用して解放する必要があります。

fill_sqlda 関数

sqlda の各記述子に記述されている各変数に領域を割り付け、このメモリのアドレスを対応する記述子の sqldata フィールドに割り当てます。

構文

```
struct sqlda * fill_sqlda( struct sqlda * sqlda );
```

パラメータ

- **sqlda –** SQLDA 構造体へのポインタ。

戻り値
成功した場合は *sqlda* を返します。十分なメモリがない場合は NULL を返します。

備考
記述子に示されるデータベースのタイプと長さに対して十分な領域が割り付けられます。

SQLDA は、free_filled_sqlda 関数を使用して解放する必要があります。

fill_sqlda_ex 関数

sqlda の各記述子に記述されている各変数に領域を割り付け、このメモリのアドレスを対応する記述子の sqldata フィールドに割り当てます。

構文

```
struct sqlda * fill_sqlda_ex( struct sqlda * sqlda , unsigned int flags);
```

パラメータ

- **sqlda –** SQLDA 構造体へのポインタ。
- **flags –** 0 または FILL_SQLDA_FLAG_RETURN_DT_LONG

戻り値
成功した場合は *sqlda* を返します。十分なメモリがない場合は NULL を返します。

備考
記述子に示されるデータベースのタイプと長さに対して十分な領域が割り付けられます。

SQLDA は、free_filled_sqlda 関数を使用して解放する必要があります。

1つのフラグビット FILL_SQLDA_FLAG_RETURN_DT_LONG がサポートされています。このフラグは sqlca.h で定義されています。

FILL_SQLDA_FLAG_RETURN_DT_LONG は、DT_LONGVARCHAR、DT_LONGNVARCHAR、DT_LONGBINARY 型を入力された記述子に保持します。このフラグビットを指定しない場合、fill_sqlda_ex は DT_LONGVARCHAR、DT_LONGNVARCHAR、DT_LONGBINARY 型をそれぞれ DT_VARCHAR、DT_NVARCHAR、DT_BINARY に変換します。DT_LONG*xyz* 型を使用すると、

DT_VARCHAR、DT_NVARCHAR、DT_BINARY における制限である 32765 バイトではなく、32767 バイトをフェッチできます。

fill_sqlda(sqlda) は、fill_sqlda_ex(sqlda, 0) と同じです。

free_filled_sqlda 関数

各 sqldata ポインタに割り付けられていたメモリと、SQLDA 自体に割り付けられていた領域を解放します。NULL ポインタであるものは解放されません。

構文

```
void free_filled_sqlda( struct sqlda * sqlda );
```

パラメータ

- **sqlda –** SQLDA 構造体へのポインタ。

備考

これが呼び出されるのは、SQLDA の sqldata フィールドの割り付けに fill_sqlda、fill_sqlda_ex、または fill_s_sqlda が使用された場合のみです。

この関数を呼び出すと、free_sqlda が自動的に呼び出されて、alloc_sqlda が割り付けたすべての記述子が解放されます。

free_sqlda 関数

この *sqlda* に割り付けられている領域を解放し、fill_sqlda などで割り付けられたインジケータ変数領域を解放します。

構文

```
void free_sqlda( struct sqlda * sqlda );
```

パラメータ

- **sqlda –** SQLDA 構造体へのポインタ。

備考

各 sqldata ポインタによって参照されているメモリは解放しません。

free_sqlda_noind 関数

この *sqlda* に割り付けられている領域を解放します。各 sqldata ポインタによって参照されているメモリは解放しません。インジケータ変数ポインタは無視されます。

構文

```
void free_sqlda_noind( struct sqlda * sqlda );
```

パラメータ

- **sqlda –** SQLDA 構造体へのポインタ。

sql_needs_quotes 関数

この関数は、引用符が必要かどうかを調べるための要求を生成してデータベースサーバに送信します。関連する情報は、sqlcode フィールドに格納されます。

構文

```
unsigned int sql_needs_quotes( SQLCA *sqlca, char * str );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。
- **str –** SQL 識別子の候補である文字列。

戻り値
文字列を SQL 識別子として使用するときに二重引用符で囲む必要があるかどうかを示す TRUE または FALSE を返します。

備考
戻り値とコードの組み合わせには、次の 3 つの場合があります。

- **return = FALSE、sqlcode = 0 –** この文字列に引用符は必要ありません。
- **return = TRUE –** sqlcode は常に SQLE_WARNING となり、文字列には引用符が必要です。
- **return = FALSE –** sqlcode が 0 でも SQLE_WARNING でもない場合は、このテストでは確定できません。

sqlda_storage 関数

varno 変数の値を格納するために必要な記憶領域の量を表す符号なし 32 ビット整数値。

構文

```
a_sql_uint32 sqlda_storage( struct sqlda * sqlda, int varno );
```

パラメータ

- **sqlda –** SQLDA 構造体へのポインタ。
- **varno –** sqlvar ホスト変数のインデックス。

戻り値
変数の値を格納するために必要な記憶領域の量を表す符号なし 32 ビット整数値。

sqlda_string_length 関数

C の文字列 (DT_STRING データ型) の長さを表す符号なし 32 ビット整数値を返します。これは、変数 sqlda->sqlvar[varno] を保持するためにどのデータ型の場合にも必要です。

構文

```
a_sql_uint32 sqlda_string_length( struct sqlda * sqlda, int
varno );
```

パラメータ

- **sqlda –** SQLDA 構造体へのポインタ。
- **varno –** sqlvar ホスト変数のインデックス。

戻り値
C の文字列 (DT_STRING データ型) の長さを表す符号なし 32 ビット整数値。これは、変数 sqlda->sqlvar[varno] を保持するためにどのデータ型の場合にも必要です。

sqlerror_message 関数

エラーメッセージを含んでいる文字列へのポインタを返します。エラーメッセージには、SQLCA 内のエラーコードに対するテキストが含まれます。エラーがなかった場合は、NULL ポインタが返されます。エラーメッセージは、指定されたバッファに入れられ、必要に応じて長さ *max* にトランケートされます。

構文

```
char * sqlerror_message( SQLCA * sqlca, char * buffer, int max );
```

パラメータ

- **sqlca –** SQLCA 構造体へのポインタ。
- **buffer –** メッセージを入れるバッファ (最大 *max* 文字)。
- **max –** バッファの最大長。

戻り値
エラーメッセージを含んでいる文字列へのポインタを返します。エラーが示されなかった場合は NULL を返します。

Embedded SQL 文のまとめ

必ず、Embedded SQL 文の前には EXEC SQL、後ろにはセミコロン (;) を付けてください。

Embedded SQL 文は大きく 2 つに分類できます。標準の SQL 文は、単純に EXEC SQL とセミコロン (;) で囲んで、C プログラム内に置いて使用します。CONNECT、DELETE、SELECT、SET、UPDATE には、Embedded SQL でのみ使用できる追加形式があります。この追加の形式は、Embedded SQL 固有の文になります。

いくつかの SQL 文は Embedded SQL 固有であり、C プログラム内でのみ使用できます。

標準的なデータ操作文とデータ定義文は、Embedded SQL アプリケーションから使用できます。また、次の文は Embedded SQL プログラミング専用です。

- **ALLOCATE DESCRIPTOR 文 [ESQL] –** 記述子にメモリを割り付ける。
- **CLOSE 文 [ESQL] [SP] –** カーソルを閉じる。
- **CONNECT 文 [ESQL] [Interactive SQL] –** データベースに接続する。
- **DEALLOCATE DESCRIPTOR 文 [ESQL] –** 記述子のメモリを再使用する。
- **宣言セクション [ESQL] –** データベースとのやりとりに使用するホスト変数を宣言する。
- **DECLARE CURSOR 文 [ESQL] [SP] –** カーソルを宣言する。
- **DELETE 文 (位置付け) [ESQL] [SP] –** カーソルの現在位置のローを削除する。
- **DESCRIBE 文 [ESQL] –** 特定の SQL 文用のホスト変数を記述する。
- **DISCONNECT 文 [ESQL] [Interactive SQL] –** データベースサーバとの接続を切断する。
- **DROP STATEMENT 文 [ESQL] –** 準備文が使用したリソースを解放する。
- **EXECUTE 文 [ESQL] –** 特定の SQL 文を実行する。
- **EXPLAIN 文 [ESQL] –** 特定のカーソルの最適化方式を説明する。
- **FETCH 文 [ESQL] [SP] –** カーソルからローをフェッチする。
- **GET DATA 文 [ESQL] –** カーソルから長い値をフェッチする。
- **GET DESCRIPTOR 文 [ESQL] –** SQLDA 内の変数に関する情報を取り出す。
- **GET OPTION 文 [ESQL] –** 特定のデータベースオプションの設定を取得する。
- **INCLUDE 文 [ESQL] –** SQL 前処理用のファイルをインクルードする。
- **OPEN 文 [ESQL] [SP] –** カーソルを開く。
- **PREPARE 文 [ESQL] –** 特定の SQL 文を準備する。
- **PUT 文 [ESQL] –** カーソルにローを挿入する。

- **SET CONNECTION 文 [Interactive SQL] [ESQL] –** アクティブな接続を変更する。
- **SET DESCRIPTOR 文 [ESQL] –** SQLDA 内で変数を記述し、SQLDA にデータを置く。
- **SET SQLCA 文 [ESQL] –** デフォルトのグローバル SQLCA 以外の SQLCA を使用する。
- **UPDATE (位置付け) 文 [ESQL] [SP] –** カーソルの現在位置のローを更新する。
- **WHENEVER 文 [ESQL] –** SQL 文でエラーが発生した場合の動作を指定する。

C/C++ 用の SAP Sybase IQ データベース API

SAP Sybase IQ C アプリケーションプログラミングインタフェース (API) は、C/C++ 言語用のデータアクセス API です。C API 仕様は、実際に使用されているデータベースとは関係なく一貫したデータベースインタフェースを提供する一連の関数、変数、規則を定義します。SAP Sybase IQ C API を使用すると、C/C++ アプリケーションから SAP Sybase IQ 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。

Perl DBI サポート

DBD::SQLAnywhere は DBI 用の SAP Sybase IQ データベースドライバで、Perl 言語用のデータアクセス API です。DBI API 仕様は、実際に使用されているデータベースとは関係なく一貫したデータベースインタフェースを提供する一連の関数、変数、規則を定義します。DBI と DBD::SQLAnywhere を使用すると、Perl スクリプトから SAP Sybase IQ 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。

DBD::SQLAnywhere

DBD::SQLAnywhere は、Tim Bunce によって作成された Database Independent Interface for Perl (DBI) モジュールのドライバです。DBI モジュールと DBD::SQLAnywhere をインストールすると、Perl から SAP Sybase IQ データベースの情報にアクセスして変更できるようになります。

DBD::SQLAnywhere ドライバは、ithread が採用された Perl を使用するときにスレッドに対応します。

稼働条件
DBD::SQLAnywhere インタフェースには、次のコンポーネントが必要です。

- Perl 5.6.0 以降。Windows では、ActivePerl 5.6.0 ビルド 616 以降が必要です。
- DBI 1.34 以降。
- C コンパイラ。Windows では、Microsoft Visual C++ コンパイラのみがサポートされています。

Windows での DBD::SQLAnywhere のインストール

サポートされている Windows プラットフォームに DBD::SQLAnywhere インタフェースをインストールし、Perl を使用して SAP Sybase IQ データベースにアクセスします。

前提条件

- iqdemo データベースを作成します。
- ActivePerl 5.6.0 以降をインストールします。ActivePerl インストーラを使用して、Perl をインストールし、コンピュータを設定できます。Perl を再コンパイルする必要はありません。

- Microsoft Visual Studio をインストールして環境を設定します。
  インストール時に環境を設定しなかった場合は、作業を行う前に PATH、LIB、INCLUDE 環境変数を正しく設定します。Microsoft は、このためのバッチファイルを用意しています。32 ビットのビルドの場合、Visual Studio 2005 または 2008 のインストール環境の `vc¥bin` サブディレクトリに `vcvars32.bat` というバッチファイルが格納されています。64 ビットのビルドの場合は、`vcvarsamd64.bat` など、このバッチファイルの 64 ビットバージョンを探します。作業を続ける前に、新しいシステムコマンドプロンプトを開き、このバッチファイルを実行してください。
  64 ビットの Visual C++ ビルド環境の設定の詳細については、http://msdn.microsoft.com/en-us/library/x4d2c09s.aspx を参照してください。

手順

1. コマンドプロンプトで、ActivePerl のインストールディレクトリの `bin` サブディレクトリに移動します。

   別のシェルからは次の手順を使用できないことがあるため、システムコマンドプロンプトを使用することを強くおすすめします。
2. Perl Module Manager を使用して、次のコマンドを入力します。

   ```
   ppm query dbi
   ```

   ppm を実行できない場合は、Perl が正しくインストールされていることを確認してください。

   このコマンドを実行すると、次のような 2 行のテキストが生成されます。この場合、この情報は、ActivePerl バージョン 5.8.1 ビルド 807 が動作しており、DBI バージョン 1.38 がインストールされていることを示します。

   ```
   Querying target 1 (ActivePerl 5.8.1.807)
     1. DBI [1.38] Database independent interface for Perl
   ```

   それ以降のバージョンの Perl では、次のようなテーブルが表示される場合があります。この場合は、DBI バージョン 1.58 がインストールされていることを示します。

name	version	abstract	area
DBI	1.58	Database independent interface for Perl	perl

   DBI がインストールされていない場合は、インストールしてください。インストールするには、ppm プロンプトで次のコマンドを入力します。

   ```
   ppm install dbi
   ```

3. コマンドプロンプトで、SAP Sybase IQ インストール環境の SDK¥Perl サブディレクトリに移動します。
4. 次のコマンドを入力し、DBD::SQLAnywhere を構築してテストします。

```
perl Makefile.PL
```

```
nmake
```

   何らかの理由によって最初から作業をやり直す必要がある場合は、コマンド nmake clean を実行し、部分的に構築されたターゲットを削除できます。

5. DBD::SQLAnywhere をテストするには、サンプルデータベースファイルを SDK¥Perl ディレクトリにコピーして、テストを実行します。

```
copy "%ALLUSERSPROFILE%¥SybaseIQ¥demo¥iqdemo.db" .
```

```
iqsrv16 demo
```

```
nmake test
```

   テストが行われない場合は、SAP Sybase IQ インストール環境の bin32 または bin64 サブディレクトリがパスに含まれていることを確認してください。

6. インストールを完了するには、同じプロンプトで次のコマンドを実行します。

```
nmake install
```

DBI Perl モジュールと DBD::SQLAnywhere インタフェースが使用できるようになりました。

UNIX での DBD::SQLAnywhere のインストール

サポートされている UNIX プラットフォームに DBD::SQLAnywhere インタフェースをインストールし、Perl を使用して SAP Sybase IQ データベースにアクセスします。

前提条件

ActivePerl 5.6.0 ビルド 616 以降と C コンパイラがインストールされている必要があります。

手順

1. DBI モジュールソースを http://www.cpan.org からダウンロードします。
2. このファイルの内容を新しいディレクトリに抽出します。
3. コマンドプロンプトで、新しいディレクトリに変更し、次のコマンドを実行して DBI モジュールを構築します。

```
perl Makefile.PL
```

```
make
```

何らかの理由によって最初から作業をやり直す必要がある場合は、コマンド `make clean` を使用し、部分的に構築されたターゲットを削除できます。

4. 次のコマンドを使用して、DBI モジュールをテストします。

```
make test
```

5. インストールを完了するには、同じプロンプトで次のコマンドを実行します。

```
make install
```

6. SAP Sybase IQ の環境が設定されていることを確認します。

使用しているシェルに応じて適切なコマンドを入力して、SAP Sybase IQ のインストールディレクトリから SAP Sybase IQ の設定スクリプトのコマンドを実行します。

シェル	使用するコマンド
sh、ksh、または bash	`.bin/sa_config.sh`
csh または tcsh	source `bin/sa_config.csh`

7. シェルプロンプトで、SAP Sybase IQ インストール環境の `sdk/perl` サブディレクトリに移動します。
8. コマンドプロンプトで、次のコマンドを実行して DBD::SQLAnywhere を構築します。

```
perl Makefile.PL
```

```
make
```

何らかの理由によって最初から作業をやり直す必要がある場合は、コマンド `make clean` を使用し、部分的に構築されたターゲットを削除できます。

9. DBD::SQLAnywhere をテストするには、サンプルデータベースファイルを `sdk/perl` ディレクトリにコピーして、テストを実行します。

```
cp samples-dir/demo.db .
```

```
iqsrv16 demo
```

```
make test
```

テストが行われない場合は、SAP Sybase IQ インストール環境の `bin32` または `bin64` サブディレクトリがパスに含まれていることを確認してください。

10. インストールを完了するには、同じプロンプトで次のコマンドを実行します。

```
make install
```

DBI Perl モジュールと DBD::SQLAnywhere インタフェースが使用できるようになりました。

次のステップ

オプションとして、ここで DBI ソースツリーを削除できます。このツリーは必要なくなりました。

DBD::SQLAnywhere を使用する Perl スクリプト

この項では、DBD::SQLAnywhere インタフェースを使用する Perl スクリプトを作成する方法の概要を説明します。

DBD::SQLAnywhere は、DBI モジュールのドライバです。DBI モジュールの完全なドキュメントについては、オンラインで http://dbi.perl.org を参照してください。

DBI モジュール

Perl スクリプトから DBD::SQLAnywhere インタフェースを使用するには、DBI モジュールを使用することを最初に Perl に通知する必要があります。これを行うには、ファイルの先頭に次の行を挿入します。

```
use DBI;
```

また、Perl を厳密モードで実行することを強くおすすめします。たとえば、明示的な変数定義を必須とするこの文によって、印刷上のエラーなどの一般的なエラーが原因の不可解なエラーが発生する可能性を大幅に減らせる見込みがあります。

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
```

DBI モジュールは、必要に応じて DBD ドライバ (DBD::SQLAnywhere など) を自動的にロードします。

Perl DBI を使用してデータベース接続を開いたり閉じたりする方法

通常、データベースに対して 1 つの接続を開いてから、一連の SQL 文を実行して必要なすべての操作を実行します。接続を開くには、connect メソッドを使用します。この戻り値は、接続時に後続の操作を行うために使用するデータベース接続のハンドルです。

connect メソッドのパラメータは、次のとおりです。

1. "DBI:SQLAnywhere:" とセミコロンで分けられた追加接続パラメータ。
2. ユーザ名。この文字列がブランクでないかぎり、接続文字列に ";UID=*value*" が追加されます。

3. パスワード値。この文字列がブランクでないかぎり、接続文字列に ";PWD=*value*" が追加されます。
4. デフォルト値のハッシュへのポインタ。AutoCommit、RaiseError、PrintError などの設定は、この方法で設定できます。

次のコードサンプルは、SAP Sybase IQ サンプルデータベースへの接続を開いて閉じます。スクリプトを実行するには、データベースサーバとサンプルデータベースを起動します。

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd      = "sql";
my %defaults = (
     AutoCommit => 1, # Autocommit enabled.
     PrintError => 0  # Errors not automatically printed.
   );
my $dbh = DBI->connect($data_src, $uid, $pwd, ¥%defaults)
  or die "Cannot connect to $data_src: $DBI::errstr¥n";
$dbh->disconnect;
exit(0);
__END__
```

オプションとして、ユーザ名またはパスワード値を個々のパラメータとして指定する代わりに、これらをデータソース文字列に追加できます。このオプションを実施する場合は、該当する引数にブランク文字列を指定します。たとえば、上記の場合、接続を開く文を次の文に置き換えることにより、スクリプトを変更できます。

```
$data_src .= ";UID=$uid";
$data_src .= ";PWD=$pwd";
my $dbh = DBI->connect($data_src, '', '', ¥%defaults)
  or die "Cannot connect to $data_src: $DBI::errstr¥n";
```

Perl DBI を使用して結果セットを取得する方法

開かれた接続へのハンドルを取得したら、データベースに格納されているデータにアクセスして修正できます。最も単純な操作は、おそらくいくつかのローを取得して出力することです。

ローのセットを返す SQL 文は、先に準備してから実行する必要があります。prepare メソッドは、文のハンドルを返します。このハンドルを使用して文を実行し、結果セットに関するメタ情報と、結果セットのローを取得できます。

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
```

```
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd      = "sql";
my $sel_stmt = "SELECT ID, GivenName, Surname
                FROM Customers
                ORDER BY GivenName, Surname";
my %defaults = (
     AutoCommit => 0, # Require explicit commit or rollback.
     PrintError => 0
   );
my $dbh = DBI->connect($data_src, $uid, $pwd, ¥%defaults)
  or die "Cannot connect to $data_src: $DBI::errstr¥n";
&db_query($sel_stmt, $dbh);
$dbh->rollback;
$dbh->disconnect;
exit(0);

sub db_query {
      my($sel, $dbh) = @_;
      my($row, $sth) = undef;
      $sth = $dbh->prepare($sel);
      $sth->execute;
      print "Fields:     $sth->{NUM_OF_FIELDS}¥n";
      print "Params:     $sth->{NUM_OF_PARAMS}¥n¥n";
      print join("¥t¥t", @{$sth->{NAME}}), "¥n¥n";
      while($row = $sth->fetchrow_arrayref) {
         print join("¥t¥t", @$row), "¥n";
      }
      $sth = undef;
}
__END__
```

準備文は、Perl 文のハンドルが破棄されないかぎりデータベースサーバから削除されません。文のハンドルを破棄するには、変数を再使用するか、変数を undef に設定します。finish メソッドを呼び出してもハンドルは削除されません。実際には、結果セットの読み込みを終了しないと決定した場合を除いて、finish メソッドは呼び出さないようにしてください。

ハンドルのリークを検出するために、SAP Sybase IQ データベースサーバでは、カーソルと準備文の数はデフォルトで接続ごとに最大 50 に制限されています。これらの制限を超えると、リソースガバナーによってエラーが自動的に生成されます。このエラーが発生したら、破棄されていない文のハンドルを確認してください。文のハンドルが破棄されていない場合は、prepare_cached を慎重に使用してください。

必要な場合、max_cursor_count と max_statement_count オプションを設定してこれらの制限を変更できます。

Perl DBI を使用して複数の結果セットを処理する方法

クエリからの複数の結果セットを処理するメソッドでは、結果セット間を移動する別のループの中でフェッチループを囲い込みます。

実行する前に、複数の結果セットを返す SQL 文を準備しておく必要があります。prepare メソッドは、文のハンドルを返します。このハンドルを使用して文を実行し、結果セットに関するメタ情報と、それぞれの結果セットのローを取得します。

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd      = "sql";
my $sel_stmt = "SELECT ID, GivenName, Surname
                   FROM Customers
                   ORDER BY GivenName, Surname;
                SELECT *
                   FROM Departments
                   ORDER BY DepartmentID";
my %defaults = (
     AutoCommit => 0, # Require explicit commit or rollback.
     PrintError => 0
   );
my $dbh = DBI->connect($data_src, $uid, $pwd, ¥%defaults)
  or die "Cannot connect to $data_src: $DBI::errstr¥n";
&db_query($sel_stmt, $dbh);
$dbh->rollback;
$dbh->disconnect;
exit(0);

sub db_query {
      my($sel, $dbh) = @_;
      my($row, $sth) = undef;
      $sth = $dbh->prepare($sel);
      $sth->execute;
      do {
          print "Fields:     $sth->{NUM_OF_FIELDS}¥n";
          print "Params:     $sth->{NUM_OF_PARAMS}¥n¥n";
          print join("¥t¥t", @{$sth->{NAME}}), "¥n¥n";
          while($row = $sth->fetchrow_arrayref) {
             print join("¥t¥t", @$row), "¥n";
          }
          print "---end of results---¥n¥n";
      } while (defined $sth->more_results);
      $sth = undef;
}
__END__
```

Perl DBI を使用してローを挿入する方法

ローを挿入するには、開かれた接続へのハンドルが必要です。最も簡単な方法は、パラメータ化された INSERT 文を使用する方法です。この場合、疑問符が値のプレースホルダとして使用されます。この文は最初に準備されてから、新しいローごとに 1 回実行されます。新しいローの値は、execute メソッドのパラメータとして指定されます。

次のサンプルプログラムは、2 人の新しい顧客を挿入します。ローの値はリテラル文字列として表示されますが、これらの値はファイルから読み込むことができます。

```
#!/usr/local/bin/perl -w
#
use DBI;
use strict;
my $database = "demo";
my $data_src = "DBI:SQLAnywhere:SERVER=$database;DBN=$database";
my $uid      = "DBA";
my $pwd      = "sql";
my $ins_stmt = "INSERT INTO Customers (ID, GivenName, Surname,
                        Street, City, State, Country, PostalCode,
                        Phone, CompanyName)
                VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
my %defaults = (
     AutoCommit => 0, # Require explicit commit or rollback.
     PrintError => 0
   );
my $dbh = DBI->connect($data_src, $uid, $pwd, ¥%defaults)
     or die "Can't connect to $data_src: $DBI::errstr¥n";
&db_insert($ins_stmt, $dbh);
$dbh->commit;
$dbh->disconnect;
exit(0);

sub db_insert {
      my($ins, $dbh) = @_;
      my($sth) = undef;
      my @rows = (
        "801,Alex,Alt,5 Blue Ave,New York,NY,USA,
10012,5185553434,BXM",
        "802,Zach,Zed,82 Fair St,New York,NY,USA,
10033,5185552234,Zap"
       );
      $sth = $dbh->prepare($ins);
      my $row = undef;
      foreach $row ( @rows ) {
        my @values = split(/,/, $row);
        $sth->execute(@values);
      }
}
__END__
```

Python サポート

SAP Sybase IQ Python データベースインタフェース sqlanydb は、Python 言語のデータアクセス API です。この項では、SAP Sybase IQ を Python と一緒に使用する方法について説明します。

sqlanydb

SQL Anywhere Python データベースインタフェース (sqlanydb) は、Python 言語のデータアクセス API です。Python データベース API 仕様は、実際に使用されているデータベースとは関係なく一貫したデータベースインタフェースを提供する一連のメソッドを定義します。sqlanydb モジュールを使用すると、Python スクリプトから SAP Sybase IQ 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。

sqlanydb モジュールは、Marc-André Lemburg が作成した Python データベース API 仕様 v2.0 を拡張して実装しています。sqlanydb モジュールをインストールすると、Python から SAP Sybase IQ データベースの情報にアクセスして変更できるようになります。

Python データベース API 仕様 v2.0 の詳細については、http://www.python.org/dev/peps/pep-0249/ を参照してください。

Python でスレッドを使用している場合、sqlanydb モジュールはスレッド対応になります。

稼働条件
sqlanydb モジュールには次のコンポーネントが必要です。

- Python。サポートされるバージョンの一覧は、http://www.sybase.com/detail?id=1068981 を参照してください。
- ctypes モジュールは必須です。ctypes モジュールがインストールされているかどうかをテストするには、コマンドプロンプトウィンドウを開いて Python を実行します。
  Python プロンプトで次の文を入力します。

  ```
  import ctypes
  ```

  エラーメッセージが表示された場合は、ctypes がインストールされていません。次はその例です。

  ```
  >>> import ctypes
  Traceback (most recent call last):
    File "<stdin>", line 1, in ?
  ImportError: No module named ctypes
  ```

Python のインストール環境に ctypes が含まれていない場合は、インストールしてください。インストールについては、http://sourceforge.net/project/showfiles.php?group_id=71702 で SourceForge.net ファイルのセクションを参照してください。
ctypes は Peak EasyInstall でもインストールされます。Peak EasyInstal をダウンロードするには、http://peak.telecommunity.com/DevCenter/EasyInstall にアクセスしてください。

Windows での Python サポートのインストール

Python サポートは、SAP Sybase IQ インストール環境の `SDK¥Python` サブディレクトリから該当する設定 Python スクリプトを実行し、Windows に設定できます。

前提条件

Python および ctypes モジュールがインストールされていることを確認します。サポートされる Python バージョンの一覧は、http://www.sybase.com/detail?id=1068981 を参照してください。

手順

1. システムのコマンドプロンプトで、SAP Sybase IQ インストール環境の `SDK¥Python` サブディレクトリに移動します。
2. 次のコマンドを実行して sqlanydb をインストールします。
   ```
   python setup.py install
   ```
3. sqlanydb をテストするには、サンプルデータベースのコピーを現在のディレクトリに作成し、テストを実行します。
   ```
   newdemo
   cd "%ALLSERPROFILE%"¥SybaseIQ¥demo
   start_iq @iqdemo.cfg iqdemo.db
   python Scripts¥test.py
   ```
   テストスクリプトによりデータベースサーバに接続され、SQL クエリが実行されます。正常に動作すると、`sqlanydb successfully installed.` というメッセージが表示されます。
   テストが行われない場合は、SAP Sybase IQ インストール環境の `bin32` または `bin64` サブディレクトリがパスに含まれていることを確認してください。

これで、sqlanydb モジュールが使用可能になりました。

Unix での Python サポートのインストール

Python サポートは、SAP Sybase IQ インストール環境の sdk/python サブディレクトリから該当する設定 Python スクリプトを実行し、Unix に設定できます。

前提条件

Python および ctypes モジュールがインストールされていることを確認します。サポートされる Python バージョンの一覧は、http://www.sybase.com/detail?id=1068981 を参照してください。

手順

1. SAP Sybase IQ の環境が設定されていることを確認します。

   使用しているシェルに応じて適切なコマンドを入力して、SAP Sybase IQ のインストールディレクトリから SAP Sybase IQ の設定スクリプトのコマンドを実行します (64 ビットのソフトウェアがインストールされている場合は、bin32 の代わりに bin64 を使用できます)。

シェル	使用するコマンド
sh、ksh、または bash	`.bin32/sa_config.sh`
csh または tcsh	source `bin32/sa_config.csh`

2. シェルプロンプトで、SAP Sybase IQ インストール環境の sdk/python サブディレクトリに移動します。
3. 次のコマンドを入力して sqlanydb をインストールします。

   ```
   python setup.py install
   ```

4. sqlanydb をテストするには、サンプルデータベースのコピーを現在のディレクトリに作成し、テストを実行します。

   ```
   newdemo
   cd "%ALLSERPROFILE%"¥SybaseIQ¥demo
   start_iq @iqdemo.cfg iqdemo.db
   python scripts/test.py
   ```

   テストスクリプトによりデータベースサーバに接続され、SQL クエリが実行されます。正常に動作すると、`sqlanydb successfully installed.` というメッセージが表示されます。

テストが行われない場合は、SAP Sybase IQ インストール環境の `bin32` または `bin64` サブディレクトリがパスに含まれていることを確認してください。

これで、sqlanydb モジュールが使用可能になりました。

sqlanydb を使用する Python スクリプト

この項では、sqlanydb インタフェースを使用する Python スクリプトを作成する方法の概要を説明します。

API の完全なドキュメントについては、オンラインで http://www.python.org/dev/peps/pep-0249/ を参照してください。

sqlanydb モジュール

sqlanydb モジュールを Python スクリプトから使用するには、ファイルの先頭に次の行を含めて、sqlanydb モジュールをロードします。

```
import sqlanydb
```

Python を使用してデータベース接続を開いたり閉じたりする方法

通常、データベースに対して 1 つの接続を開いてから、一連の SQL 文を実行して必要なすべての操作を実行します。接続を開くには、connect メソッドを使用します。この戻り値は、接続時に後続の操作を行うために使用するデータベース接続のハンドルです。

connect メソッドのパラメータは、一連の「キーワード=値」ペアをカンマで区切って指定します。

```
sqlanydb.connect( keyword=value, ...)
```

一般的な接続パラメータを次に示します。

- **DataSourceName="*dsn*" –** この接続パラメータの省略形は `DSN="dsn"` です。たとえば、DataSourceName="Sybase IQ demo" と指定します。
- **UserID="*user-id*" –** この接続パラメータの省略形は `UID="user-id"` です。
- **Password="*passwd*" –** この接続パラメータの省略形は `PWD="passwd"` です。
- **DatabaseFile="*db-file*" –** この接続パラメータの省略形は `DBF="db-file"` です。たとえば、DatabaseFile="iqdemo.db" と指定します。

次のコードサンプルは、SAP Sybase IQ サンプルデータベースへの接続を開いて閉じます。スクリプトを実行するには、データベースサーバとサンプルデータベースを起動します。

```
import sqlanydb
```

```
# Create a connection object
con = sqlanydb.connect( userid="<user_id>",
                        password="<password>" )
# Close the connection
con.close()
```

データベースサーバが手動で開始されないようにするには、サーバを起動するように設定されたデータソースを使用します。この例を次に示します。

```
import sqlanydb

# Create a connection object
con = sqlanydb.connect( DSN="Sybase IQ Demo" )

# Close the connection
con.close()
```

Python を使用して結果セットを取得する方法

開かれた接続へのハンドルを取得したら、データベースに格納されているデータにアクセスして修正できます。最も単純な操作は、おそらくいくつかのローを取得して出力することです。

cursor メソッドは、開いている接続にカーソルを作成するために使用します。execute メソッドは、結果セットを作成するために使用します。fetchall メソッドは、この結果セットからローを取得するために使用します。

```
import sqlanydb

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="<user_id>",
                        password="<password>" )
cursor = con.cursor()

# Execute a SQL string
sql = "SELECT * FROM Employees"
cursor.execute(sql)

# Get a cursor description which contains column names
desc = cursor.description
print len(desc)

# Fetch all results from the cursor into a sequence,
# display the values as column name=value pairs,
# and then close the connection
rowset = cursor.fetchall()
for row in rowset:
    for col in range(len(desc)):
        print "%s=%s" % (desc[col][0], row[col] )
    print
cursor.close()
con.close()
```

Python を使用してローを挿入する方法

ローをテーブルに挿入する最も簡単な方法は、パラメータ化されていない INSERT 文を使用することです。この方法では、値は SQL 文の一部として指定されます。新しい文が新しいローごとに構築されて実行されます。前の例でみたように、SQL 文を実行するにはカーソルが必要です。

次のサンプルプログラムは、2 人の新規顧客をサンプルデータベースに挿入します。切断される前に、データベースに対してトランザクションをコミットします。

```
import sqlanydb

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="<user_id>", pwd="<password>" )
cursor = con.cursor()
cursor.execute("DELETE FROM Customers WHERE ID > 800")

rows = ((801,'Alex','Alt','5 Blue Ave','New York','NY',
        'USA','10012','5185553434','BXM'),
        (802,'Zach','Zed','82 Fair St','New York','NY',
        'USA','10033','5185552234','Zap'))

# Set up a SQL INSERT
parms = ("'%s'," * len(rows[0]))[:-1]
sql = "INSERT INTO Customers VALUES (%s)" % (parms)
print sql % rows[0]
cursor.execute(sql % rows[0])
print sql % rows[1]
cursor.execute(sql % rows[1])
cursor.close()
con.commit()
con.close()
```

パラメータ化された INSERT 文を使用してローをテーブルに挿入する方法もあります。この場合、疑問符が値のプレースホルダとして使用されます。executemany メソッドは、ローのセットのメンバーごとに INSERT 文を実行するために使用します。新しいローの値は、1 つの引数として executemany メソッドに渡されます。

```
import sqlanydb

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="<user_id>", pwd="<password>" )
cursor = con.cursor()
cursor.execute("DELETE FROM Customers WHERE ID > 800")

rows = ((801,'Alex','Alt','5 Blue Ave','New York','NY',
        'USA','10012','5185553434','BXM'),
        (802,'Zach','Zed','82 Fair St','New York','NY',
        'USA','10033','5185552234','Zap'))

# Set up a parameterized SQL INSERT
parms = ("?," * len(rows[0]))[:-1]
sql = "INSERT INTO Customers VALUES (%s)" % (parms)
```

```
print sql
cursor.executemany(sql, rows)
cursor.close()
con.commit()
con.close()
```

どちらのサンプルプログラムも、ローのデータをテーブルに挿入する方法として適切であるようですが、いくつかの理由で後者の方が優れています。入力を要求されてデータ値が取得された場合、最初のサンプルプログラムでは SQL 文を含む不良データが挿入される可能性があります。最初のサンプルプログラムでは、execute メソッドはテーブルに挿入されるローごとに呼び出されます。2 番目のサンプルプログラムでは、すべてのローをテーブルに挿入するときに一度だけ executemany メソッドが呼び出されます。

データベースタイプの変換

データベースサーバから結果がフェッチされたときに、データベースタイプを Python オブジェクトにマッピングする方法を制御するには、変換コールバックを登録します。

コールバックの登録には、モジュールレベルの register_converter メソッドを使用できます。このメソッドは、データベースタイプを最初のパラメータ、変換関数を 2 番目のパラメータとして呼び出されます。たとえば、データ型 DT_DECIMAL として記述されるカラムのデータに対して Decimal オブジェクトを作成するよう sqlanydb に要求する場合は、次の例を使用します。

```
import sqlanydb
import decimal

def convert_to_decimal(num):
    return decimal.Decimal(num)

sqlanydb.register_converter(sqlanydb.DT_DECIMAL,
convert_to_decimal)
```

コンバータは、次のデータベースタイプに登録できます。

```
DT_DATE
DT_TIME
DT_TIMESTAMP
DT_VARCHAR
DT_FIXCHAR
DT_LONGVARCHAR
DT_DOUBLE
DT_FLOAT
DT_DECIMAL
DT_INT
DT_SMALLINT
DT_BINARY
DT_LONGBINARY
DT_TINYINT
DT_BIGINT
```

```
DT_UNSINT
DT_UNSSMALLINT
DT_UNSBIGINT
DT_BIT
```

次の例は、小数の結果を、小数点以下の桁数をトランケートして整数に変換する方法を示します。アプリケーションの実行時には、給与額が整数値で表示されます。

```
import sqlanydb

def convert_to_int(num):
    return int(float(num))

sqlanydb.register_converter(sqlanydb.DT_DECIMAL, convert_to_int)

# Create a connection object, then use it to create a cursor
con = sqlanydb.connect( userid="<user_id>",
                        password="<password>" )
cursor = con.cursor()

# Execute a SQL string
sql = "SELECT * FROM Employees WHERE EmployeeID=105"
cursor.execute(sql)

# Get a cursor description which contains column names
desc = cursor.description
print len(desc)

# Fetch all results from the cursor into a sequence,
# display the values as column name=value pairs,
# and then close the connection
rowset = cursor.fetchall()
for row in rowset:
    for col in range(len(desc)):
        print "%s=%s" % (desc[col][0], row[col] )
    print
cursor.close()
con.close()
```

PHP サポート

PHP には一般的なデータベースから情報を取得する機能があります。SAP Sybase IQ には PHP から SAP Sybase IQ データベースにアクセスするためのモジュールが用意されています。PHP 言語を使用すると、SAP Sybase IQ データベースから情報を取得し、独自の Web サイトで動的な Web コンテンツを提供できます。

SAP Sybase IQ PHP 拡張

PHP (`PHP: Hypertext Preprocessor`) は、オープンソーススクリプト言語です。PHP は汎用スクリプト言語として使用できますが、HTML 文書に組み込むことができるスクリプトを作成するときに役に立つ言語として設計されました。クライアントによって頻繁に実行される JavaScript で作成されたスクリプトとは異なり、PHP スクリプトは Web サーバによって処理され、クライアントには処理結果の HTML 出力が送信されます。PHP の構文は、Java や Perl などのその他の一般的な言語の構文から派生されたものです。

動的な Web ページを開発するときに役に立つ言語として使用できるように、PHP には SAP Sybase IQ を含む多くの一般的なデータベースから情報を取得する機能が含まれています。SAP Sybase IQ には、PHP から SAP Sybase IQ データベースにアクセスするための拡張が用意されています。SAP Sybase IQ PHP 拡張と PHP 言語を使用することによって、スタンドアロンのスクリプトを作成し、SAP Sybase IQ データベースの情報に依存する動的な Web ページを作成できます。

SAP Sybase IQ PHP 拡張を使用すると、PHP からデータベースにネイティブな方法でアクセスできます。この PHP 拡張は、単純であり、他の PHP データアクセス方法では発生する可能性のあるシステムリソースのリークを防ぐことができるため、優先して使用するようにしてください。

ビルド済みバージョンの PHP 拡張は Windows、Linux、および Solaris 用に用意されており、SAP Sybase IQ インストール環境のバイナリサブディレクトリにインストールされます。SAP Sybase IQ PHP 拡張のソースコードは、SAP Sybase IQ インストール環境の `sdk¥php` サブディレクトリにインストールされています。

詳細および最新の SAP Sybase IQ PHP ドライバについては、http://www.sybase.com/detail?id=1019698 を参照してください。

PHP 拡張のテスト

簡単なチェックを行って、SAP Sybase IQ PHP 拡張が正常に動作することを確認します。

前提条件

必要なすべての PHP コンポーネントがシステムにインストールされている必要があります。

手順

1. SAP Sybase IQ インストール環境の `bin32` サブディレクトリがパスに含まれていることを確認してください。SAP Sybase IQ PHP 拡張では、`bin32` ディレクトリがパスに含まれている必要があります。
2. コマンドプロンプトで、次のコマンドを実行して SAP Sybase IQ サンプルデータベースを起動します。

   ```
   cd "%ALLUSERSPROFILE%"¥SybaseIQ¥demo
   start_iq @iqdemo.cfg iqdemo.db
   ```

   このコマンドは、サンプルデータベースを使用してデータベースサーバを起動します。
3. コマンドプロンプトで、SAP Sybase IQ インストール環境の `SDK¥PHP ¥Examples` サブディレクトリに移動します。実行プログラムディレクトリの `php` がパスに含まれていることを確認します。次のコマンドを入力します。

   ```
   php test.php
   ```

   次のようなメッセージが表示されます。PHP コマンドが認識されない場合は、PHP がパスにあるかを確認します。

   ```
   Installation successful
   Using php-5.2.11_sqlanywhere.dll
   Connected successfully
   ```

   SAP Sybase IQ PHP 拡張がロードされない場合は、コマンド "php -i" を使用して PHP セットアップに関する情報を取得できます。このコマンドの出力で `extension_dir` と `sqlanywhere` を検索してください。
4. ここまで終了したら、データベースサーバメッセージウィンドウで[シャットダウン]をクリックして、SAP Sybase IQ データベースサーバを停止します。

テストが成功し、SAP Sybase IQ PHP 拡張が正常に動作していることを示します。

PHP テストページの作成と実行

PHP が適切に設定されているかどうかをテストするいくつかの Web ページを作成し、実行します。

前提条件

PHP をインストールします。PHP のインストールについては、http://us2.php.net/install を参照してください。

手順

このプロシージャはすべての設定に適用されます。

1. ルートの Web コンテンツディレクトリに `info.php` という名前のファイルを作成します。

   使用するディレクトリがわからない場合は、Web サーバの設定ファイルを確認してください。Apache のインストール環境では、多くの場合、そのコンテンツディレクトリの名前は `htdocs` です。
2. ファイルに次のコードを挿入します。

   ```
   <?php phpinfo(); ?>
   ```

   phpinfo は、システム設定情報のページを生成する PHP 関数です。これによって、PHP と Web サーバのインストールがともに適切に機能していることが確認されます。
3. ファイル `connect.php` を `sdk¥php¥examples` ディレクトリからルートの Web コンテンツディレクトリにコピーします。これによって、PHP と SQL Anywhere のインストールがともに適切に機能していることが確認されます。
4. `sa_test.php` という名前のルートの Web コンテンツディレクトリにファイルを作成し、このファイルに次のコードを挿入します。

   ```
   <?php
     $conn = sasql_connect( "UID=DBA;PWD=sql" );
     $result = sasql_query( $conn, "SELECT * FROM Employees" );
     sasql_result_all( $result );
     sasql_free_result( $result );
     sasql_disconnect( $conn );
   ?>
   ```

   sa_test ページには、Employees テーブルの内容が表示されます。
5. 必要な場合は、Web サーバを起動します。

   たとえば、Apache Web サーバを起動するには、Apache のインストール環境の `bin` サブディレクトリから次のコマンドを実行します。

```
apachectl start
```

6. Linux では、提供されているスクリプトのいずれかを使用して SAP Sybase IQ の環境変数を設定します。

   使用しているシェルに応じて適切なコマンドを入力して、SAP Sybase IQ のインストールディレクトリから SAP Sybase IQ の設定スクリプトのコマンドを実行します。

シェル	使用するコマンド
sh、ksh、または bash	`. /bin32/sa_config.sh`
csh または tcsh	`source /bin32/sa_config.csh`

7. コマンドプロンプトで、iqdemo.db サンプルデータベースを起動します。
8. PHP と Web サーバが SAP Sybase IQ で正常に機能していることをテストするには、サーバと同じコンピュータで実行されているブラウザからテストページにアクセスします。

テストページ	使用する URL
`info.php`	`http://localhost/info.php`
`connect.php`	`http://localhost/connect.php`
`sa_test.php`	`http://localhost/sa_test.php`

info ページには、`phpinfo()` 呼び出しからの出力が表示されます。

connect ページには、`Connected successfully` というメッセージが表示されます。

sa_test ページには、Employees テーブルの内容が表示されます。

PHP スクリプトの開発

この項では、SAP Sybase IQ PHP 拡張を使用して SAP Sybase IQ データベースにアクセスする PHP スクリプトの作成方法について説明します。

ここで使用される例のソースコードは他の例と同様に、SAP Sybase IQ インストール環境の `SDK¥PHP¥Examples` サブディレクトリにあります。

PHP を使用してデータベースに接続する方法

データベースに接続するには、標準の SAP Sybase IQ 接続文字列を sasql_connect 関数のパラメータとしてデータベースサーバに渡します。<?php と ?> のタグは、この間のコードを PHP が実行し、そのコードを PHP 出力に置き換えることを Web サーバに指定します。

この例のソースコードは、SAP Sybase IQ インストール環境の connect.php という名前のファイルにあります。

```
<?php
  # Connect using the default user ID and password
  $conn = sasql_connect( "UID=DBA;PWD=sql" );
  if( ! $conn ) {
      echo "Connection failed¥n";
  } else {
      echo "Connected successfully¥n";
      sasql_close( $conn );
  }?>
```

このスクリプトは、ローカルサーバでデータベースに接続しようとします。このコードを正常に実行するには、SAP Sybase IQ サンプルデータベースまたは同じクレデンシャルを持つデータベースがローカルサーバで実行されている必要があります。

PHP を使用してデータベースからデータを取得する方法

Web ページでの PHP スクリプトの用途の 1 つとして、データベースに含まれる情報の取り出しと表示があります。次に示す例で、役に立つテクニックをいくつか紹介します。

単純な select クエリ

次の PHP コードでは、Web ページに SELECT 文の結果セットを含める便利な方法を示します。この例は、SAP Sybase IQ サンプルデータベースに接続し、顧客のリストを返すように設計されています。

PHP スクリプトを実行するように Web サーバを設定している場合、このコードを Web ページに埋め込むことができます。

このサンプルのソースコードは、SAP Sybase IQ インストール環境の query.php という名前のファイルにあります。

```
<?php
  # Connect using the default user ID and password
  $conn = sasql_connect( "UID=DBA;PWD=sql" );
  if( ! $conn ) {
      echo "sasql_connect failed¥n";
  } else {
      echo "Connected successfully¥n";
      # Execute a SELECT statement
      $result = sasql_query( $conn, "SELECT * FROM Customers" );
      if( ! $result ) {
          echo "sasql_query failed!";
      } else {
          echo "query completed successfully¥n";
          # Generate HTML from the result set
          sasql_result_all( $result );
          sasql_free_result( $result );
```

```
      }
      sasql_close( $conn );
  }
?>
```

sasql_result_all 関数が、結果セットのすべてのローをフェッチし、それを表示するための HTML 出力テーブルを生成します。sasql_free_result 関数が、結果セットを格納するために使用されたリソースを解放します。

カラム名によるフェッチ
状況によって、結果セットからすべてのデータを表示する必要がない場合、または別の方法でデータを表示したい場合があります。次の例は、結果セットの出力フォーマットを自由に制御する方法を示します。PHP によって、必要とする情報を選択した希望の方法で表示できます。

このサンプルのソースコードは、SAP Sybase IQ インストール環境の `fetch.php` という名前のファイルにあります。

```
<?php
  # Connect using the default user ID and password
  $conn = sasql_connect( "UID=DBA;PWD=sql" );
  if( ! $conn ) {
      die ("Connection failed");
  } else {
      # Connected successfully.
  }
  # Execute a SELECT statement
  $result = sasql_query( $conn, "SELECT * FROM Customers" );
  if( ! $result ) {
      echo "sasql_query failed!";
      return 0;
  } else {
      echo "query completed successfully¥n";
  }
  # Retrieve meta information about the results
  $num_cols = sasql_num_fields( $result );
  $num_rows = sasql_num_rows( $result );
  echo "Num of rows = $num_rows¥n";
  echo "Num of cols = $num_cols¥n";
  while( ($field = sasql_fetch_field( $result )) ) {
      echo "Field # : $field->id ¥n";
      echo "¥tname   : $field->name ¥n";
      echo "¥tlength : $field->length ¥n";
      echo "¥ttype   : $field->type ¥n";
  }
  # Fetch all the rows
  $curr_row = 0;
  while( ($row = sasql_fetch_row( $result )) ) {
      $curr_row++;
      $curr_col = 0;
      while( $curr_col < $num_cols ) {
          echo "$row[$curr_col]¥t|";
```

```
            $curr_col++;
        }
        echo "¥n";
    }
    # Clean up.
    sasql_free_result( $result );
    sasql_disconnect( $conn );
?>
```

sasql_fetch_array 関数が、テーブルの単一ローを返します。データは、カラム名とカラムインデックスを基準に取り出すことができます。

sasql_fetch_assoc が、テーブルの単一ローを連想配列として返します。カラム名をインデックスとして使用して、データを取得できます。次はその例です。

```
<?php
  # Connect using the default user ID and password
  $conn = sasql_connect("UID=DBA;PWD=sql");

  /* check connection */
  if( sasql_errorcode() ) {
      printf("Connect failed: %s¥n", sasql_error());
      exit();
  }

  $query = "SELECT Surname, Phone FROM Employees ORDER by
EmployeeID";

  if( $result = sasql_query($conn, $query) ) {

      /* fetch associative array */
      while( $row = sasql_fetch_assoc($result) ) {
          printf ("%s (%s)¥n", $row["Surname"], $row["Phone"]);
      }

      /* free result set */
      sasql_free_result($result);
  }

  /* close connection */
  sasql_close($conn);
?>
```

この他に PHP インタフェースには 2 つの類似したメソッドがあります。sasql_fetch_row はカラムインデックスのみで検索し、sasql_fetch_object はカラム名のみで検索してローを返します。

sasql_fetch_object 関数の例については、fetch_object.php のサンプルスクリプトを参照してください。

ネストされた結果セット
SELECT 文がデータベースに送信されると、結果セットが返されます。sasql_fetch_row 関数と sasql_fetch_array 関数を使用すると、結果セットの個々のローからデータが取り出され、さらに問い合わせることができるカラムの配列としてそれぞれのローが返されます。

このサンプルのソースコードは、SAP Sybase IQ インストール環境の `nested.php` という名前のファイルにあります。

```
<?php
  $conn = sasql_connect( "UID=DBA;PWD=sql" );
  if( $conn ) {
      // get the GROUPO user id
      $result = sasql_query( $conn,
            "SELECT user_id FROM SYS.SYSUSER " .
            "WHERE user_name='GROUPO'" );
      if( $result ) {
          $row = sasql_fetch_array( $result );
          $user = $row[0];
      } else {
          $user = 0;
      }
      // get the tables created by user GROUPO
      $result = sasql_query( $conn,
            "SELECT table_id, table_name FROM SYS.SYSTABLE " .
            "WHERE creator = $user" );
      if( $result ) {
          $num_rows = sasql_num_rows( $result );
          echo "Returned rows : $num_rows¥n";
          while( $row = sasql_fetch_array( $result ) ) {
              echo "Table: $row[1]¥n";
            $query = "SELECT table_id, column_name FROM SYS.SYSCOLUMN
" .
                        "WHERE table_id = '$row[table_id]'";
              $result2 = sasql_query( $conn, $query );
              if( $result2 ) {
                  echo "Columns:";
                while( $detailed = sasql_fetch_array( $result2 ) ) {
                      echo " $detailed[column_name]";
                  }
                  sasql_free_result( $result2 );
              }
              echo "¥n¥n";
          }
          sasql_free_result( $result );
      }
      sasql_disconnect( $conn );
  }
?>
```

上の例では、SQL 文で SYSTAB から各テーブルのテーブル ID とテーブル名を選択します。sasql_query 関数が、ローの配列を返します。スクリプトは、

sasql_fetch_array 関数を使用してそれらのローを反復し、ローを配列から取り出します。内部反復がその各ローのカラムで行われ、それらの値が出力されます。

Web フォーム

PHP では、Web フォームからユーザ入力を受け取って SQL クエリとしてデータベースサーバに渡し、返される結果を表示できます。次の例は、ユーザが SQL 文を使用してサンプルデータベースを問い合わせ、結果を HTML テーブルに表示する簡単な Web フォームを示しています。

このサンプルのソースコードは、SAP Sybase IQ インストール環境の webisql.php という名前のファイルにあります。

```
<?php
  echo "<HTML>¥n";
  $qname = $_POST["qname"];
  $qname = str_replace( "¥¥", "", $qname );
  echo "<form method=post action=webisql.php>¥n";
  echo "<br>Query: <input type=text Size=80 name=qname value=¥"$qname
¥">¥n";
  echo "<input type=submit>¥n";
  echo "</form>¥n";
  echo "<HR><br>¥n";
  if( ! $qname ) {
      echo "No Current Query¥n";
      return;
  }
  # Connect to the database
  $con_str =
"UID=<user_id>;PWD=<password>;SERVER=iqdemo;LINKS=tcpip";
  $conn = sasql_connect( $con_str );
  if( ! $conn ) {
      echo "sasql_connect failed¥n";
      echo "</html>¥n";
      return 0;
  }
  $qname = str_replace( "¥¥", "", $qname );
  $result = sasql_query( $conn, $qname );
  if( ! $result ) {
        echo "sasql_query failed!";
  } else {
        // echo "query completed successfully¥n";
        sasql_result_all( $result, "border=1" );
      sasql_free_result( $result );
  }
  sasql_disconnect( $conn );
  echo "</html>¥n";
?>
```

この設計は、ユーザによって入力される値に基づいてカスタマイズされた SQL クエリを作成することによって、複雑な Web フォームを処理できるように拡張できます。

PHP アプリケーション内の BLOB

SAP Sybase IQ データベースでは、あらゆる種類のデータをバイナリラージオブジェクト (BLOB) として格納できます。Web ブラウザで読み取れるデータであれば、PHP スクリプトによって簡単にデータベースからそのデータを取り出して動的に生成したページに表示できます。

BLOB フィールドは、多くの場合、GIF や JPG 形式のイメージなどのテキスト以外のデータを格納するために使用します。サードパーティソフトウェアやデータ型変換を必要とせずに、さまざまな種類のデータを Web ブラウザに渡すことができます。次の例は、イメージをデータベースに追加し、それを再び取り出して Web ブラウザに表示する処理を示します。

このサンプルは、SAP Sybase IQ インストール環境の `image_insert.php` ファイルと `image_retrieve.php` ファイルにあるサンプルコードに似ています。これらのサンプルでは、イメージを格納する BLOB カラムの使用についても示しています。

```
<?php
  $conn = sasql_connect( "UID=DBA;PWD=sql" )
        or die("Cannot connect to database");
  $create_table = "CREATE TABLE images (ID INTEGER PRIMARY KEY, img
IMAGE)";
  sasql_query( $conn, $create_table);
  $insert = "INSERT INTO images VALUES (99,
xp_read_file('ianywhere_logo.gif'))";
  sasql_query( $conn, $insert );
  $query = "SELECT img FROM images WHERE ID = 99";
  $result = sasql_query($conn, $query);
  $data = sasql_fetch_row($result);
  $img = $data[0];
  header("Content-type: image/gif");
  echo $img;
  sasql_disconnect($conn);
?>
```

バイナリデータをデータベースから直接 Web ブラウザに送信するには、スクリプトでヘッダ関数を使用してデータの MIME タイプを設定する必要があります。この例の場合、ブラウザは GIF イメージを受け取るように指定されているので、イメージが正しく表示されます。

Unix で SAP Sybase IQ PHP 拡張を構築する方法

UNIX で SAP Sybase IQ PHP 拡張を使用して PHP を SAP Sybase IQ に接続するには、SAP Sybase IQ PHP 拡張のファイルを PHP のソースツリーに追加し、PHP を再コンパイルする必要があります。

Unix での SAP Sybase IQ PHP 拡張ファイルの PHP ソースツリーへの追加

このトピックでは、SAP Sybase IQ PHP 拡張のファイルを PHP のソースツリーに追加するために必要な手順について説明します。

前提条件

UNIX で SAP Sybase IQ PHP 拡張を使用する場合、次のソフトウェアをシステムにインストールしておく必要があります。

- SAP Sybase IQ インストール環境。Apache Web サーバと同じコンピュータでも異なるコンピュータでも実行できます。
- SQL Anywhere PHP 拡張のソースコードは、http://download.sybase.com/ianywhere/php/2.0.3/src/sasql_php.zip からダウンロードできます。
  また、`sqlpp` と `libdblib16.so` (Unix) がインストールされている必要があります (SAP Sybase IQ `lib32` ディレクトリを確認してください)。
- PHP ソースコード。http://www.php.net からダウンロードできます。
  サポートされるバージョンの一覧は、http://www.sybase.com/detail?id=1068981 を参照してください。
- Apache Web サーバのソースコード。http://httpd.apache.org からダウンロードできます。
  ビルド済みバージョンの Apache を使用する場合は、`apache` と `apache-devel` がインストールされていることを確認してください。
- Unified ODBC PHP 拡張を使用する場合は、`libdbodbc16.so` (Unix) がインストールされている必要があります (SAP Sybase IQ `lib32` ディレクトリを確認してください)。

UNIX インストールディスクから次のバイナリをインストールする必要があります (すでにインストールされていない場合)。これらのバイナリは RPM パッケージとして提供されています。

make
automake
autoconf
makeinfo
bison

gcc
cpp
glibc-devel
kernel-headers
flex

インストール作業の特定の手順を実行するためには、PHP をインストールした人と同じアクセス権が必要になります。UNIX ベースのシステムには通常 `sudo` コマンドがあり、十分なパーミッションを持たないユーザでも、権限を持ったユーザと同様に特定のコマンドを実行できます。

手順

インストール作業の特定の手順を実行するためには、PHP をインストールした人と同じアクセス権が必要になります。UNIX ベースのシステムには通常 `sudo` コマンドがあり、十分なパーミッションを持たないユーザでも、権限を持ったユーザと同様に特定のコマンドを実行できます。

1. SAP Sybase IQ PHP 拡張ソースコードを http://www.sybase.com/detail?id=1019698 からダウンロードします。「Building the Driver from Source」というタイトルの項を探してください。
2. SAP Sybase IQ PHP 拡張が保存されているディレクトリから、ファイルを PHP ソースツリーの `ext` サブディレクトリに抽出します。

   ```
   $ tar -xzf sasql_php.zip -C PHP-source-directory/ext/
   ```

   次の例は、PHP バージョン 5.2.11 用のコマンドです。以下の php-5.2.11 を、使用している PHP のバージョンに変更する必要があります。

   ```
   $ tar -xzf sqlanywhere_php-1.0.8.tar.gz -C ~/php-5.2.11/ext
   ```

3. PHP に拡張を認識させます。

   ```
   $ cd PHP-source-directory/ext/sqlanywhere
   $ touch *
   $ cd ~/PHP-source-directory
   $ ./buildconf
   ```

   次の例は、PHP バージョン 5.2.11 用のコマンドです。以下の php-5.2.11 を、使用している PHP のバージョンに変更する必要があります。

   ```
   $ cd ~/php-5.2.11/ext/sqlanywhere
   $ touch *
   $ cd ~/php-5.2.11
   $ ./buildconf
   ```

4. PHP が拡張を認識していることを確認します。

   ```
   $ ./configure -help | egrep sqlanywhere
   ```

PHP による SAP Sybase IQ 拡張の認識が成功した場合は、次のテキストが表示されます。

```
--with-sqlanywhere=[DIR]
```

失敗した場合は、このコマンドの出力を記録し、SQL Anywhere フォーラム (http://sqlanywhere-forum.sybase.com/) に投稿してサポートを受けてください。

Apache と PHP をコンパイルする方法

PHP は、Web サーバ (Apache など) の共有モジュールとして、または CGI 実行プログラムとしてコンパイルできます。PHP でサポートされていない Web サーバを使用している場合、または PHP スクリプトを Web ページではなくコマンドシェルで実行したい場合は、PHP を CGI 実行プログラムとしてコンパイルします。それ以外の場合、PHP をインストールして Apache で動作するようにするには、Apache モジュールとしてコンパイルします。

Apache モジュールとしての PHP のコンパイル

PHP をインストールして Apache で動作するようにするには、Apache モジュールとしてコンパイルします。

前提条件

次の手順を使用して、Apache が共有モジュールを認識するように設定します。

- Apache を設定して共有モジュールを認識させます。
  Apache ファイルが抽出されたディレクトリから、次のようなコマンドを実行します。

  ```
  $ cd Apache-source-directory
  $ ./configure --enabled-shared=max --enable-module=most --
  prefix=/Apache-installation-directory
  ```

  次の例は、Apache バージョン 2.2.9 用のコマンドです。`apache_2.2.9` を、使用している Apache のバージョンに変更する必要があります。

  ```
  $ cd ~/apache_2.2.9
  $ ./configure --enabled-shared=max --enable-module=most --
  prefix=/usr/local/web/apache
  ```
- 関連するコンポーネントを再コンパイルしてインストールします。

  ```
  $ make
  $ make install
  ```

  これで、Apache モジュールとして動作するように PHP をコンパイルできます。

手順

1. SAP Sybase IQ の環境が設定されていることを確認します。

使用しているシェルに応じて、SAP Sybase IQ がインストールされているディレクトリから適切なコマンドを入力します。

シェル	使用するコマンド
sh、ksh、bash	`../IQ_16.sh`
csh、tcsh	`./IQ_16.csh`

2. PHP を Apache モジュールとして設定して、SAP Sybase IQ PHP 拡張を含めます。

   次のコマンドを実行します。

   ```
   $ cd PHP-source-directory
   $ ./configure --with-sqlanywhere --with- apxs=/Apache-
   installation-directory/bin/apxs
   ```

   次の例は、PHP バージョン 5.2.11 用のコマンドです。php-5.2.11 を、使用している PHP のバージョンに変更する必要があります。

   ```
   $ cd ~/php-5.2.11
   $ ./configure --with-sqlanywhere --with- apxs=/usr/local/web/
   apache/bin/apxs
   ```

   `configure` スクリプトによって、インストールされている SAP Sybase IQ のバージョンとロケーションの特定が試行されます。
3. 関連するコンポーネントを再コンパイルします。

   ```
   $ make
   ```
4. ライブラリが正常にリンクされていることを確認します。
   - Linux ユーザの場合 (次の例は、PHP バージョン 5 を使用していると仮定)

     ```
     ldd ./.libs/libphp5.so
     ```
5. PHP バイナリを Apache の `lib` ディレクトリにインストールします。

   ```
   $ make install
   ```
6. 検証を実行します。検証は PHP により自動的に行われます。ユーザによる確認が必要なのは、`httpd.conf` 設定ファイルが検証され、Apache が `.php` ファイルを PHP スクリプトとして認識するかどうかの確認だけです。

   `httpd.conf` は、`Apache` ディレクトリの `conf` サブディレクトリに格納されています。

   ```
   $ cd Apache-installation-directory/conf
   ```

   次に例を示します。

   ```
   $ cd /usr/local/web/apache/conf
   ```

   `httpd.conf` のバックアップコピーを作成してからファイルを編集します (`pico` を好みのテキストエディタと置き換えることができます)。

```
$ cp httpd.conf httpd.conf.backup
$ pico httpd.conf
```

次の行を `httpd.conf` に追加するか、コメント解除します (ファイル内の同じ場所にはありません)。

```
LoadModule php5_module   libexec/libphp5.so
AddModule mod_php5.c
AddType application/x-httpd-php .php
AddType application/x-httpd-php-source .phps
```

最初の 2 行により、PHP コードの解釈に使用されるファイルに Apache がポイントされます。残りの 2 行により、拡張子が `.php` または `.phps` のファイルのファイルタイプが宣言されます。これにより、Apache はファイルを適切に認識および処理できるようになります。

PHP は共有モジュールとして正常にコンパイルされます。

CGI 実行プログラムとしての PHP のコンパイル

PHP でサポートされていない Web サーバを使用している場合、または PHP スクリプトを Web ページではなくコマンドシェルで実行したい場合は、PHP を CGI 実行プログラムとしてコンパイルします。

前提条件

この作業を実行するための前提条件はありません。

手順

1. SAP Sybase IQ の環境が設定されていることを確認します。

   使用しているシェルに応じて、SAP Sybase IQ がインストールされているディレクトリから適切なコマンドを入力します。

シェル	使用するコマンド
sh、ksh、bash	`../IQ_16.sh`
csh、tcsh	`./IQ_16.csh`

2. PHP を CGI 実行プログラムとして設定して、SAP Sybase IQ PHP 拡張を含めます。

   PHP ファイルが抽出されたディレクトリから、次のコマンドを実行します。

```
$ cd PHP-source-directory
$ ./configure --with-sqlanywhere
```

   次の例は、PHP バージョン 5.2.11 用のコマンドです。php-5.2.11 を、使用している PHP のバージョンに変更する必要があります。

```
$ cd ~/php-5.2.11
$ ./configure --with-sqlanywhere
```

設定スクリプトによって、インストールされている SAP Sybase IQ のバージョンとロケーションの特定が試行されます。

3. 実行プログラムをコンパイルします。
```
$ make
```
4. コンポーネントをインストールします。
```
$ make install
```

PHP は CGI 実行プログラムとして正常にコンパイルされます。

SAP Sybase IQ PHP API リファレンス

PHP API では、次の関数をサポートしています。

sasql_affected_rows

最後の SQL 文の影響を受けるローの数を返します。通常、この関数は INSERT 文、UPDATE 文、または DELETE 文に使用します。SELECT 文には sasql_num_rows 関数を使用します。

プロトタイプ

```
int sasql_affected_rows( sasql_conn $conn )
```

パラメータ

$conn – 接続関数から返される接続リソース。

戻り値
影響を受けたローの数。

sasql_commit

SQL Anywhere サーバのトランザクションを終了し、トランザクション中に加えられたすべての変更を永続的なものにします。auto_commit オプションが Off である場合にのみ有効です。

プロトタイプ

```
bool sasql_commit( sasql_conn $conn )
```

パラメータ

$conn – 接続関数から返される接続リソース。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_close

開いていたデータベース接続を閉じます。

プロトタイプ

```
bool sasql_close( sasql_conn $conn )
```

パラメータ

$conn – 接続関数から返される接続リソース。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_connect

SAP Sybase IQ データベースへの接続を確立します。

プロトタイプ

```
sasql_conn sasql_connect( string $con_str )
```

パラメータ

$con_str – SAP Sybase IQ によって認識される接続文字列。

戻り値
成功の場合は正の SAP Sybase IQ 接続リソース、失敗の場合はエラーまたは 0。

sasql_data_seek

sasql_query を使用して開かれた *$result* のロー *row_num* にカーソルを配置します。

プロトタイプ

```
bool sasql_data_seek( sasql_result $result, int row_num )
```

パラメータ

$result – sasql_query 関数によって返される結果リソース。

row_num – 結果リソース内のカーソルの新しい位置を表す整数。たとえば、0 に指定するとカーソルは結果セットの最初のローに移動し、5 に指定すると 6 番目のローに移動します。負の数は結果セットの最後の位置に相対的なローを表します。

たとえば、-1 に指定するとカーソルは結果セットの最後のローに移動し、-2 に指定すると最後から 2 番目のローに移動します。

戻り値
成功の場合は TRUE、エラーの場合は FALSE。

sasql_disconnect

sasql_connect または sasql_pconnect によって開かれている接続を閉じます。

プロトタイプ
```
bool sasql_disconnect( sasql_conn $conn )
```

パラメータ

$conn – 接続関数から返される接続リソース。

戻り値
成功の場合は TRUE、エラーの場合は FALSE。

sasql_error

最後に実行された SAP Sybase IQ PHP 関数のエラーテキストを返します。エラーメッセージは接続ごとに格納されます。*$conn* を指定しないと、sasql_error は接続が使用できなかったときの最後のエラーメッセージを返します。たとえば、sasql_connect を呼び出して接続が失敗した場合、*$conn* のパラメータを設定せずに sasql_error を呼び出すと、エラーメッセージを取得します。対応するエラーコード値を取得する場合は、sasql_errorcode 関数を使用します。

プロトタイプ
```
string sasql_error( [ sasql_conn $conn ] )
```

パラメータ

$conn – 接続関数から返される接続リソース。

戻り値
エラーが記述された文字列。

sasql_errorcode

最後に実行された SAP Sybase IQ PHP 関数のエラーコードを返します。エラーコードは接続ごとに格納されます。*$conn* を指定しないと、sasql_errorcode は接続が使用できなかったときの最後のエラーコードを返します。たとえば、sasql_connect を呼び出して接続が失敗した場合、*$conn* のパラメータを設定せずに

sasql_errorcode を呼び出すと、エラーコードを取得します。対応するエラーメッセージを取得する場合は、sasql_error 関数を使用します。

プロトタイプ

```
int sasql_errorcode( [ sasql_conn $conn ] )
```

パラメータ

$conn – 接続関数から返される接続リソース。

戻り値
エラーコードを表す整数。エラーコードが 0 の場合は、処理が正常に終了したことを示します。エラーコードが正数の場合は、警告を伴う正常終了を示します。エラーコードが負数の場合は、処理が失敗したことを示します。

sasql_escape_string

指定された文字列内の特殊文字をすべてエスケープします。エスケープされる特殊文字は、¥r、¥n、'、"、;、¥、および NULL 文字です。この関数は、sasql_real_escape_string のエイリアスです。

プロトタイプ

```
string sasql_escape_string( sasql_conn $conn, string $str )
```

パラメータ

$conn – 接続関数から返される接続リソース。

$string – エスケープする文字列。

戻り値
エスケープされた文字列。

sasql_fetch_array

結果セットから 1 つのローをフェッチします。このローは、カラム名またはカラムインデックスによってインデックス付けが可能な配列として返されます。

プロトタイプ

```
array sasql_fetch_array( sasql_result $result [, int
$result_type ])
```

パラメータ

$result – sasql_query 関数によって返される結果リソース。

$result_type – このオプションのパラメータは、現在のローデータから生成される配列の種類を示す定数です。このパラメータに指定できる値は、定数 SASQL_ASSOC、SASQL_NUM、または SASQL_BOTH です。デフォルトで SASQL_BOTH になります。

SASQL_ASSOC 定数を使用すると、この関数は sasql_fetch_assoc 関数と同じ動作をし、SASQL_NUM 定数を使用すると sasql_fetch_row 関数と同じ動作をします。最後のオプション SASQL_BOTH を使用すると、両方の属性を持つ単一配列が作成されます。

戻り値
結果セットのローを表す配列。ローがない場合は FALSE。

sasql_fetch_assoc

連想配列として結果セットからローを 1 つフェッチします。

プロトタイプ

```
array sasql_fetch_assoc( sasql_result $result )
```

パラメータ

$result – sasql_query 関数によって返される結果リソース。

戻り値
結果セットからフェッチされたローを表す文字列の連想配列。配列内の各キーは結果セットの 1 つのカラムの名前を表します。結果セットにそれ以上ローがない場合は、FALSE を返します。

sasql_fetch_field

特定のカラムに関する情報を含むオブジェクトを返します。

プロトタイプ

```
object sasql_fetch_field( sasql_result $result [, int
$field_offset ] )
```

パラメータ

$result – sasql_query 関数によって返される結果リソース。

$field_offset – 取り出したい情報のカラム (フィールド) を表す整数。カラムは 0 から始まります。最初のカラムを取得するには、値 0 を指定します。このパラメータを省略すると、次のフィールドオブジェクトが返されます。

戻り値
次のプロパティを持つオブジェクトが返されます。

- **id –** フィールド番号を示します。
- **name –** フィールド名を示します。
- **numeric –** フィールドが数値であるかどうかを示します。
- **length –** フィールドのネイティブの記憶サイズを返します。
- **type –** フィールドのタイプを返します。
- **native_type –** フィールドのネイティブタイプを返します。DT_FIXCHAR、DT_DECIMAL、DT_DATE などの値です。
- **precision –** フィールドの数値精度を返します。このプロパティが設定されるのは、native_type が DT_DECIMAL のフィールドのみです。
- **scale –** フィールドの数値の位取りを返します。このプロパティが設定されるのは、native_type が DT_DECIMAL のフィールドのみです。

sasql_fetch_object

オブジェクトとして結果セットからローを 1 つフェッチします。

プロトタイプ

```
object sasql_fetch_object( sasql_result $result  )
```

パラメータ

$result – sasql_query 関数によって返される結果リソース。

戻り値
結果セットからフェッチされたローを表すブジェクト。各プロパティ名は結果セットの 1 つのカラム名と一致します。結果セットにそれ以上ローがない場合は、FALSE を返します。

sasql_fetch_row

結果セットから 1 つのローをフェッチします。このローは、カラムインデックスのみによってインデックス付けが可能な配列として返されます。

プロトタイプ

```
array sasql_fetch_row( sasql_result $result )
```

パラメータ

$result – sasql_query 関数によって返される結果リソース。

戻り値
結果セットのローを表す配列。ローがない場合は FALSE。

sasql_field_count

最後の結果に含まれているカラム (フィールド) の数を返します。

プロトタイプ

```
int sasql_field_count( sasql_conn $conn )
```

パラメータ

$conn – 接続関数から返される接続リソース。

戻り値
カラムの正数。*$conn* が有効でない場合は FALSE。

sasql_field_seek

フィールドカーソルを特定のオフセットに設定します。次の sasql_fetch_field 呼び出し時に、そのオフセットに関連付けられたカラムのフィールド定義を取得します。

プロトタイプ

```
bool sasql_field_seek( sasql_result $result, int $field_offset )
```

パラメータ

$result – sasql_query 関数によって返される結果リソース。

$field_offset – 取り出したい情報のカラム (フィールド) を表す整数。カラムは 0 から始まります。最初のカラムを取得するには、値 0 を指定します。このパラメータを省略すると、次のフィールドオブジェクトが返されます。

戻り値
成功の場合は TRUE、エラーの場合は FALSE。

sasql_free_result

sasql_query から返される結果リソースに関連付けられているデータベースリソースを解放します。

プロトタイプ

```
bool sasql_free_result( sasql_result $result )
```

パラメータ

$result – sasql_query 関数によって返される結果リソース。

戻り値
成功の場合は TRUE、エラーの場合は FALSE。

sasql_get_client_info

クライアントのバージョン情報を返します。

プロトタイプ

```
string sasql_get_client_info( )
```

パラメータ
なし

戻り値
SQL Anywhere クライアントソフトウェアのバージョンを表す文字列。文字列は X.Y.Z.W の形式で返されます。X はメジャーバージョン番号、Y はマイナーバージョン番号、Z はパッチ番号、W はビルド番号を表します。たとえば、10.0.1.3616 のようになります。

sasql_insert_id

IDENTITY カラムまたは DEFAULT AUTOINCREMENT カラムに最後に挿入された値を返します。最後に挿入したテーブルに IDENTITY や DEFAULT AUTOINCREMENT カラムが含まれていないと、0 を返します。sasql_insert_id 関数は、MySQL データベースとの互換性のために用意されています。

プロトタイプ

```
int sasql_insert_id( sasql_conn $conn )
```

パラメータ

$conn – 接続関数から返される接続リソース。

戻り値
前回の INSERT 文で生成された AUTOINCREMENT カラムの ID。最後の挿入が AUTOINCREMENT カラムに影響しなかった場合は 0。*$conn* が有効でない場合は FALSE。

sasql_message

メッセージをサーバメッセージウィンドウに書き込みます。

プロトタイプ

```
bool sasql_message( sasql_conn $conn, string $message )
```

パラメータ

$conn – 接続関数から返される接続リソース。

$message – サーバメッセージウィンドウに書き込まれるメッセージ。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_multi_query

指定された接続リソースを使用して、*$sql_str* によって指定された 1 つまたは複数の SQL クエリを準備して実行します。各クエリはセミコロンによって区切られます。最初のクエリ結果は、sasql_use_result または sasql_store_result を使用して取得または格納できます。sasql_field_count を使用すると、クエリから結果セットが返されるかどうかを確認できます。それ以降のクエリ結果は、sasql_next_result および sasql_use_result/sasql_store_result を使用して処理できます。

プロトタイプ

```
bool sasql_multi_query( sasql_conn $conn, string $sql_str )
```

パラメータ

$conn – 接続関数から返される接続リソース。

$sql_str – セミコロンで区切られた 1 つ以上の SQL 文。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_next_result

$conn で実行された最後のクエリの次の結果セットを準備します。

プロトタイプ

```
bool sasql_next_result( sasql_conn $conn )
```

パラメータ

$conn – 接続関数から返される接続リソース。

戻り値
取り出す結果セットが他にない場合は FALSE。取り出す結果セットが別にある場合は TRUE。sasql_use_result または sasql_store_result を呼び出して、次の結果セットを取り出します。

sasql_num_fields

$result 内のローに含まれるフィールドの数を返します。

プロトタイプ

```
int sasql_num_fields( sasql_result $result )
```

パラメータ

$result – sasql_query 関数によって返される結果リソース。

戻り値
指定された結果セット内のフィールド数を返します。

sasql_num_rows

$result に含まれるローの数を返します。

プロトタイプ

```
int sasql_num_rows( sasql_result $result )
```

パラメータ

$result – sasql_query 関数によって返される結果リソース。

戻り値
ローの数が厳密である場合は正の数、概数である場合は負の数。ローの厳密な数を取得するには、データベースオプション row_counts をデータベースで永続的に設定するか、接続で一時的に設定します。

sasql_pconnect

SAP Sybase IQ データベースへの永続的な接続を確立します。sasql_connect の代わりに sasql_pconnect を使用すると、Apache が子プロセスを生成する方法に応じて、パフォーマンスが向上することがあります。場合によって、永続的な接続では、接続プーリングと同様にパフォーマンスが向上します。データベースサーバに接

続数の制限がある場合 (たとえば、パーソナルデータベースサーバで同時接続の数を 10 に制限)、永続的な接続を使用するには注意が必要です。永続的な接続はそれぞれの子プロセスにアタッチされるので、使用可能な接続数を超えた子プロセスが Apache にあると、接続エラーが発生します。

プロトタイプ

```
sasql_conn sasql_pconnect( string $con_str )
```

パラメータ

$con_str – SAP Sybase IQ によって認識される接続文字列。

戻り値
成功の場合は正の SAP Sybase IQ 永続接続リソース、失敗の場合はエラーまたは 0。

sasql_prepare

指定された SQL 文字列を準備します。

プロトタイプ

```
sasql_stmt sasql_prepare( sasql_conn $conn, string $sql_str )
```

パラメータ

$conn – 接続関数から返される接続リソース。

$sql_str – 準備される SQL 文。適切な位置に疑問符を埋め込んで、文字列にパラメータマーカーを含めることができます。

戻り値
文のオブジェクト。失敗した場合は FALSE。

sasql_query

sasql_connect または sasql_pconnect を使用してすでに開かれている、*$conn* によって識別される接続で、SQL クエリ *$sql_str* を準備して実行します。sasql_query 関数は、2 つの関数 (sasql_real_query と、sasql_store_result または sasql_use_result のいずれか 1 つ) を呼び出すことと同等です。

プロトタイプ

```
mixed sasql_query( sasql_conn $conn, string $sql_str [, int
$result_mode ] )
```

パラメータ

$conn – 接続関数から返される接続リソース。

$sql_str – SQL Anywhere によってサポートされている SQL 文。

$result_mode – SASQL_USE_RESULT または SASQL_STORE_RESULT (デフォルト) のどちらか。

戻り値
失敗した場合は FALSE。INSERT、UPDATE、DELETE、CREATE で成功した場合は TRUE。SELECT で成功した場合は sasql_result。

sasql_real_escape_string

指定された文字列内の特殊文字をすべてエスケープします。エスケープされる特殊文字は、¥r、¥n、'、"、;、¥、および NULL 文字です。

プロトタイプ

```
string sasql_real_escape_string( sasql_conn $conn, string $str )
```

パラメータ

$conn – 接続関数から返される接続リソース。

$string – エスケープする文字列。

戻り値
エスケープされた文字列。エラーの場合は FALSE。

sasql_real_query

指定された接続リソースを使用して、データベースに対してクエリを実行します。クエリ結果は、sasql_store_result または sasql_use_result を使用して取得または格納できます。sasql_field_count 関数を使用すると、クエリから結果セットが返されるかどうかを確認できます。sasql_query 関数は、この関数と sasql_store_result または sasql_use_result のいずれか 1 つを呼び出すことと同等です。

プロトタイプ

```
bool sasql_real_query( sasql_conn $conn, string $sql_str )
```

パラメータ

$conn – 接続関数から返される接続リソース。

$sql_str – SQL Anywhere によってサポートされている SQL 文。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_result_all

$result のすべての結果をフェッチし、オプションのフォーマット文字列に従って HTML 出力テーブルを生成します。

プロトタイプ

```
bool sasql_result_all( resource $result
[, $html_table_format_string
[, $html_table_header_format_string
[, $html_table_row_format_string
[, $html_table_cell_format_string
] ] ] ] )
```

パラメータ

$result – sasql_query 関数によって返される結果リソース。

$html_table_format_string – HTML テーブルに適用されるフォーマット文字列。たとえば、"Border=1; Cellpadding=5" のようになります。特別な値 none を指定すると、HTML テーブルは作成されません。これは、カラム名やスクリプトをカスタマイズする場合に便利です。このパラメータの明示的値を指定しなくても済むように、パラメータ値には NULL を使用します。

$html_table_header_format_string – HTML テーブルのカラム見出しに適用されるフォーマット文字列。たとえば、"bgcolor=#FF9533" のようになります。特別な値 none を指定すると、HTML テーブルは作成されません。これは、カラム名やスクリプトをカスタマイズする場合に便利です。このパラメータの明示的値を指定しなくても済むように、パラメータ値には NULL を使用します。

$html_table_row_format_string – HTML テーブルのローに適用されるフォーマット文字列。たとえば、"onclick='alert('this')'" のようになります。交互に変わるフォーマットを使用する場合は、特別なトークン >< を使用します。トークンの左側は、奇数ローで使用するフォーマットを示し、トークンの右側は偶数ローで使用するフォーマットを示します。このトークンをフォーマット文字列に含めなかった場合は、すべてのローが同じフォーマットになります。このパラメータに明示的な値を指定したくない場合は、パラメータ値として NULL を使用します。

$html_table_cell_format_string – HTML テーブルローのセルに適用されるフォーマット文字列。たとえば、"onclick='alert('this')'" のようになります。

このパラメータに明示的な値を指定したくない場合は、パラメータ値として NULL を使用します。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_rollback

データベースのトランザクションを終了し、トランザクション中に加えられたすべての変更を破棄します。この関数は、auto_commit オプションが Off である場合にのみ有効です。

プロトタイプ

```
bool sasql_rollback( sasql_conn $conn )
```

パラメータ

$conn – 接続関数から返される接続リソース。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_set_option

指定した接続で、指定したオプションの値を設定します。

プロトタイプ

```
bool sasql_set_option( sasql_conn $conn, string $option, mixed
$value )
```

説明
次のオプションの値を設定できます。

名前	説明	デフォルト
auto_commit	このオプションを on に設定すると、データベースサーバは各文の実行後にコミットする。	on
row_counts	このオプションを FALSE に設定すると、sasql_num_rows 関数は影響を受ける推定ロー数を返す。正確なロー数を取得するには、このオプションを TRUE に設定する。	FALSE

名前	説明	デフォルト
verbose_errors	このオプションを TRUE に設定すると、PHP ドライバは冗長エラーを返す。このオプションを FALSE に設定した場合、詳細なエラー情報を取得するには、sasql_error または sasql_errorcode 関数を呼び出す。	TRUE

php.ini ファイルに次の行を追加することによって、オプションのデフォルト値を変更できます。次の例では、auto_commit オプションのデフォルト値が設定されます。

```
sqlanywhere.auto_commit=0
```

パラメータ

$conn – 接続関数から返される接続リソース。

$option – 設定するオプションの名前。

$value – 新しいオプションの値。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_stmt_affected_rows

文の実行の影響を受けるローの数を返します。

プロトタイプ

```
int sasql_stmt_affected_rows( sasql_stmt $stmt )
```

パラメータ

$stmt – sasql_stmt_execute によって実行された文リソース。

戻り値
影響を受けたローの数。失敗した場合は FALSE。

sasql_stmt_bind_param

PHP 変数を文のパラメータにバインドします。

プロトタイプ

```
bool sasql_stmt_bind_param( sasql_stmt $stmt, string $types,
mixed &$var_1 [, mixed &$var_2 .. ] )
```

パラメータ

$stmt – sasql_prepare 関数によって返された準備文リソース。

**$types –** 対応するバインドの種類を指定する 1 つ以上の文字を含む文字列。次のいずれかを指定してください。文字列には `s`、整数には `i`、二重には `d`、blob には `b` です。$types 文字列の長さは、$types パラメータに続くパラメータ ($var_1、$var_2、...) の数と一致する必要があります。文字数も、準備文内のパラメータマーカー (疑問符) の数と一致する必要があります。

$var_n – 変数の参照。

戻り値
変数のバインドに成功した場合は TRUE、それ以外の場合は FALSE。

sasql_stmt_bind_param_ex

PHP 変数を文のパラメータにバインドします。

プロトタイプ

```
bool sasql_stmt_bind_param_ex( sasql_stmt $stmt, int
$param_number, mixed &$var, string $type [, bool $is_null [, int
$direction ] ] )
```

パラメータ

$stmt – sasql_prepare 関数によって返された準備文リソース。

**$param_number –** パラメータ番号。これは 0 ～ (sasql_stmt_param_count($stmt) - 1) の間の数です。

$var – PHP 変数。PHP 変数への参照のみが許可されます。

$type – 変数の型。次のいずれかを指定してください。文字列には `s`、整数には `i`、二重には `d`、blob には `b` です。

$is_null – 変数値が NULL であるか否かを示します。

$direction – SASQL_D_INPUT、SASQL_D_OUTPUT、または SASQL_INPUT_OUTPUT を指定できます。

戻り値
変数のバインドに成功した場合は TRUE、それ以外の場合は FALSE。

sasql_stmt_bind_result

実行された文の結果カラムに 1 つ以上の PHP 変数をバインドして、結果セットを返します。

プロトタイプ

```
bool sasql_stmt_bind_result( sasql_stmt $stmt, mixed &$var1 [,
mixed &$var2 .. ] )
```

パラメータ

$stmt – sasql_stmt_execute によって実行された文リソース。

$var1 – sasql_stmt_fetch によって返される結果セットのカラムにバインドされる PHP 変数への参照。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_stmt_close

指定された文リソースを閉じて、関連付けられているリソースを解放します。この関数は、sasql_stmt_result_metadata によって返された結果オブジェクトも解放します。

プロトタイプ

```
bool sasql_stmt_close( sasql_stmt $stmt )
```

パラメータ

$stmt – sasql_prepare 関数によって返された準備文リソース。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_stmt_data_seek

この関数は、結果セット内で指定されたオフセットを検索します。

プロトタイプ

```
bool sasql_stmt_data_seek( sasql_stmt $stmt, int $offset )
```

パラメータ

$stmt – 文リソース。

**$offset –** 結果セット内のオフセット。これは、0 ～ (sasql_stmt_num_rows($stmt) - 1) の間の数です。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_stmt_errno

指定された文リソースを使用して最後に実行された文関数のエラーコードを返します。

プロトタイプ

```
int sasql_stmt_errno( sasql_stmt $stmt )
```

パラメータ

$stmt – sasql_prepare 関数によって返された準備文リソース。

戻り値
整数のエラーコード。

sasql_stmt_error

指定された文リソースを使用して最後に実行された文関数のエラーテキストを返します。

プロトタイプ

```
string sasql_stmt_error( sasql_stmt $stmt )
```

パラメータ

$stmt – sasql_prepare 関数によって返された準備文リソース。

戻り値
エラーが記述された文字列。

sasql_stmt_execute

準備文を実行します。sasql_stmt_result_metadata を使用して、文が結果セットを返すかどうかを確認できます。

プロトタイプ

```
bool sasql_stmt_execute( sasql_stmt $stmt )
```

パラメータ

$stmt – sasql_prepare 関数によって返された準備文リソース。変数をバインドしてから実行してください。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_stmt_fetch

この関数は、文の結果からローを 1 つフェッチし、sasql_stmt_bind_result を使用してバインドされた変数にカラムを配置します。

プロトタイプ

```
bool sasql_stmt_fetch( sasql_stmt $stmt )
```

パラメータ

$stmt – 文リソース。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_stmt_field_count

この関数は、文の結果セット内のカラム数を返します。

プロトタイプ

```
int sasql_stmt_field_count( sasql_stmt $stmt )
```

パラメータ

$stmt – 文リソース。

戻り値
文の結果内のカラム数。文から結果が返されない場合は 0 を返します。

sasql_stmt_free_result

この関数は、キャッシュされた文の結果セットを解放します。

プロトタイプ

```
bool sasql_stmt_free_result( sasql_stmt $stmt )
```

パラメータ

$stmt – sasql_stmt_execute を使用して実行された文リソース。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_stmt_insert_id

IDENTITY カラムまたは DEFAULT AUTOINCREMENT カラムに最後に挿入された値を返します。最後に挿入したテーブルに IDENTITY または DEFAULT AUTOINCREMENT カラムが含まれていないと、0 を返します。

プロトタイプ

```
int sasql_stmt_insert_id( sasql_stmt $stmt )
```

パラメータ

$stmt – sasql_stmt_execute によって実行された文リソース。

戻り値
前回の INSERT 文によって生成された、IDENTITY カラムまたは DEFAULT AUTOINCREMENT カラムの ID。最後の挿入が IDENTITY カラムまたは DEFAULT AUTOINCREMENT カラムに影響しなかった場合は 0。$stmt が有効でない場合は、FALSE (0) を返します。

sasql_stmt_next_result

この関数は、文の次の結果に進みます。別の結果セットがある場合は、現在キャッシュされている結果が破棄されて、それに関連付けられている結果セットオブジェクト (sasql_stmt_result_metadata によって返されたもの) が削除されます。

プロトタイプ

```
bool sasql_stmt_next_result( sasql_stmt $stmt )
```

パラメータ

$stmt – 文リソース。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_stmt_num_rows

結果セット内のロー数を返します。結果セット内の実際のロー数は、sasql_stmt_store_result 関数が呼び出されて結果セット全体がバッファに格納された後でのみ特定できます。sasql_stmt_store_result 関数が呼び出されなかった場合は 0 が返されます。

プロトタイプ

```
int sasql_stmt_num_rows( sasql_stmt $stmt )
```

パラメータ

$stmt – sasql_stmt_execute によって実行され、sasql_stmt_store_result が呼び出された文リソース。

戻り値
結果で使用できるロー数。失敗した場合は 0。

sasql_stmt_param_count

指定された準備文リソース内のパラメータ数を返します。

プロトタイプ

```
int sasql_stmt_param_count( sasql_stmt $stmt )
```

パラメータ

$stmt – sasql_prepare 関数によって返される文リソース。

戻り値
パラメータ数。エラーの場合は FALSE。

sasql_stmt_reset

この関数は、*$stmt* オブジェクトを記述直後の状態にリセットします。バインドされた変数はすべてバインドを解除され、sasql_stmt_send_long_data を使用して送信されたデータはすべて削除されます。

プロトタイプ

```
bool sasql_stmt_reset( sasql_stmt $stmt )
```

パラメータ

$stmt – 文リソース。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_stmt_result_metadata

指定された文の結果セットオブジェクトを返します。

プロトタイプ

```
sasql_result sasql_stmt_result_metadata( sasql_stmt $stmt )
```

パラメータ

$stmt – 準備され、実行された文リソース。

戻り値
sasql_result オブジェクト。文から結果が返されない場合は FALSE。

sasql_stmt_send_long_data

ユーザがパラメータデータをチャンク単位で送信できるようにします。ユーザは、最初に sasql_stmt_bind_param または sasql_stmt_bind_param_ex を呼び出してから、データを送信します。バインドパラメータのデータ型は string または blob にする必要があります。この関数を繰り返して呼び出すと、結果は前に送信された内容に追加されます。

プロトタイプ

```
bool sasql_stmt_send_long_data( sasql_stmt $stmt, int
$param_number, string $data )
```

パラメータ

$stmt – sasql_prepare を使用して準備された文リソース。

**$param_number –** パラメータ番号。これは 0 ～ (sasql_stmt_param_count($stmt) - 1) の間の数です。

$data – 送信されるデータ。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_stmt_store_result

この関数を使用すると、クライアントで文の結果セット全体をキャッシュできるようになります。キャッシュされた結果は、関数 sasql_stmt_free_result を使用して解放できます。

プロトタイプ

```
bool sasql_stmt_store_result( sasql_stmt $stmt )
```

パラメータ

$stmt – sasql_stmt_execute を使用して実行された文リソース。

戻り値
成功した場合は TRUE、失敗した場合は FALSE。

sasql_store_result

データベース接続 *$conn* での最後のクエリの結果セットを、sasql_data_seek 関数で使用できるように転送します。

プロトタイプ

```
sasql_result sasql_store_result( sasql_conn $conn )
```

パラメータ

$conn – 接続関数から返される接続リソース。

戻り値
クエリが結果オブジェクトを返さない場合は FALSE。それ以外の場合は、結果のすべてのローを含む結果セットオブジェクト。結果はクライアントでキャッシュされます。

sasql_sqlstate

最新の SQLSTATE 文字列を返します。SQLSTATE は、最後に実行された SQL 文が成功、エラー、または警告条件になったかどうかを示します。SQLSTATE コードは 5 文字で構成され、「00000」はエラーがないことを示します。この値は ISO/ANSI SQL 標準で定められています。

プロトタイプ

```
string sasql_sqlstate( sasql_conn $conn )
```

パラメータ

$conn – 接続関数から返される接続リソース。

戻り値
現在の SQLSTATE コードを含む 5 文字の文字列を返します。「00000」という結果はエラーがないことを示します。

sasql_use_result

接続で最後に実行されたクエリの結果セットの取得を開始します。

プロトタイプ

```
sasql_result sasql_use_result( sasql_conn $conn )
```

パラメータ

$conn – 接続関数から返される接続リソース。

戻り値
クエリが結果オブジェクトを返さない場合は FALSE。それ以外の場合は、結果セットオブジェクト。結果はクライアントでキャッシュされません。

Ruby サポート

SAP Sybase IQ では、3 種類の Ruby API (Application Programming Interface) がサポートされています。1 つ目は SAP Sybase IQ Ruby API です。この API は、SAP Sybase IQ C API によって公開されているインタフェースを Ruby でラップします。2 つ目は ActiveRecord のサポートです。ActiveRecord は、Web 開発フレームワーク Ruby on Rails の一部として普及しているオブジェクト関係マッパです。3 つ目は Ruby DBI のサポートです。SAP Sybase IQ は、DBI とともに使用できる Ruby データベースドライバ (DBD) を提供しています。

Ruby API サポート

SAP Sybase IQ Ruby プロジェクトには 3 つの別個のパッケージがあります。すべてのパッケージをインストールする最も簡単な方法は、`RubyGems` を使用することです。

SAP Sybase IQ Ruby プロジェクトのホームは http://sqlanywhere.rubyforge.org/ です。

ネイティブ Ruby ドライバ

sqlanywhere – このパッケージは Ruby コードから SAP Sybase IQ データベースへのインタフェースを可能にする低レベルのドライバです。このパッケージは、SAP Sybase IQ C API によって公開されているインタフェースを Ruby でラップします。このパッケージは C 言語で記述され、Windows と Linux 用に、ソースまたは事前にコンパイルされた gem として提供されています。`RubyGems` がインストールされている場合は、次のコマンドを実行してこのパッケージを入手できます。

```
gem install sqlanywhere
```

SQL Anywhere のこれ以外の Ruby パッケージを使用するには、このパッケージが必要です。

Rails

Rails は、Ruby 言語で記述された Web 開発フレームワークです。その強みは、Web アプリケーションの開発にあります。Rails で開発を行う前に Ruby プログラミング言語に慣れておくことを強くおすすめします。http://www.rubyonrails.org/ を参照してください。

ActiveRecord アダプタ

activerecord-sqlanywhere-adapter – このパッケージは、ActiveRecord と SAP Sybase IQ の対話を可能にするアダプタです。ActiveRecord は、Web 開発フレームワーク Ruby on Rails の一部として普及しているオブジェクト関係マッパです。このパッケージは Pure Ruby で記述され、ソースまたは gem フォーマットで提供されています。このアダプタでは `sqlanywhere` gem が使用され、この gem に依存します。`RubyGems` がインストールされている場合は、次のコマンドを実行してこのパッケージとその依存ファイルをインストールできます。

```
gem install activerecord-sqlanywhere-adapter
```

SQL Anywhere の Ruby/DBI ドライバ

dbi – このパッケージは Ruby 用の DBI ドライバです。RubyGems がインストールされている場合は、次のコマンドを実行してこのパッケージとその依存ファイルをインストールできます。

```
gem install dbi
```

dbd-sqlanywhere – このパッケージは、Ruby/DBI と SAP Sybase IQ の対話を可能にするドライバです。Ruby/DBI は一般的な Perl DBI モジュールをモデルとする汎用のデータベースインタフェースです。このパッケージは Pure Ruby で記述され、ソースまたは gem フォーマットで提供されています。このドライバでは sqlanywhere gem が使用され、この sqlanywhere gem に依存します。RubyGems がインストールされている場合は、次のコマンドを実行してこのパッケージとその依存ファイルをインストールできます。

```
gem install dbd-sqlanywhere
```

これらのパッケージに関するフィードバックがある場合は、sqlanywhere-users@rubyforge.com のメーリングリストを使用してください。

SAP Sybase IQ での Rails サポートの設定

SAP Sybase IQ に Ruby on Rails サポートを設定できます。

前提条件

この作業を実行するための前提条件はありません。

手順

1. RubyGems をインストールします。これにより、Ruby パッケージのインストールが簡単になります。Ruby on Rails のダウンロードページに、インストールす

る適切なバージョンが記載されています。http://www.rubyonrails.org/ を参照してください。

2. システムに Ruby のインタプリタをインストールします。Ruby on Rails のダウンロードページに、インストールする推奨バージョンが記載されています。http://www.rubyonrails.org/ を参照してください。
3. 次のコマンドを実行し、Ruby Rails とその依存性をインストールします。
```
gem install rails
```
4. Ruby Development Kit (`DevKit`) をインストールします。`DevKit` を http://rubyinstaller.org/downloads/ からダウンロードして、http://github.com/oneclick/rubyinstaller/wiki/Development-Kit の手順に従ってください。
5. 次のコマンドを実行し、SQL Anywhere ActiveRecord サポート (activerecord-sqlanywhere-adapter) をインストールします。
```
gem install activerecord-sqlanywhere-adapter
```
6. Rails によってサポートされたデータベース管理システムのセットに SAP Sybase IQ を追加します。本書作成時点での最新リリースバージョンは、Rails 3.1.3 です。
   a. Rails の `configs¥databases` ディレクトリに `sqlanywhere.yml` ファイルを作成し、データベースを設定します。Ruby を `¥Ruby` ディレクトリにインストールし、Rails のバージョン 3.1.3 をインストールした場合、このファイルへのパスは `¥Ruby¥lib¥ruby¥gems¥1.9.1¥gems¥railties-3.1.3¥lib¥rails¥generators¥rails¥app¥templates¥config¥databases` になります。このファイルの内容は次のようになります。
```
#
# SQL Anywhere database configuration
#
# This configuration file defines the patten used for
# database filenames. If your application is called "blog",
# then the database names will be blog_development,
# blog_test, blog_production. The specified username and
# password should permit DBA access to the database.
#

development:
  adapter: sqlanywhere
  server: <%= app_name %>
  database: <%= app_name %>_development
  username: DBA
  password: sql

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run
"rake".
# Do not set this db to the same as development or production.
test:
```

```
  adapter: sqlanywhere
  server: <%= app_name %>
  database: <%= app_name %>_test
  username: DBA
  password: sql

production:
  adapter: sqlanywhere
  server: <%= app_name %>
  database: <%= app_name %>_production
  username: DBA
  password: sql
```

sqlanywhere.yml ファイルは、Rails プロジェクトで database.yml ファイルを作成するためのテンプレートになります。次のデータベースオプションを指定できます。

- **adapter –** (必須、デフォルトなし)。SQL Anywhere ActiveRecord アダプタを使用する場合、このオプションを sqlanywhere に設定する必要があります。
- **database –** (必須、デフォルトなし)。このオプションは、接続文字列の "DatabaseName" に対応しています。
- **server –** (オプション、デフォルトは database オプション)。このオプションは、接続文字列の "ServerName" に対応しています。
- **username –** (オプション、デフォルトは 'DBA')。このオプションは、接続文字列の "UserID" に対応しています。
- **password –** (オプション、デフォルトは 'sql')。このオプションは、接続文字列の "Password" に対応しています。
- **encoding –** (オプション、デフォルトは OS の文字セット)。このオプションは、接続文字列の "CharSet" に対応しています。
- **commlinks –** (省略可)。このオプションは、接続文字列の "CommLinks" に対応しています。
- **connection_name –** (省略可)。このオプションは、接続文字列の "ConnectionName" に対応しています。

b. Rails の app_base.rb ファイルを更新します。以前の手順と同じ条件の場合、このファイルはパス ¥Ruby¥lib¥ruby¥gems¥1.9.1¥gems¥railties-3.1.3¥lib¥rails¥generators¥app_base.rb にあります。app_base.rb ファイルを編集し、次の行を検索します。

```
DATABASES = %w( mysql oracle postgresql sqlite3 frontbase
ibm_db sqlserver )
```

このリストに次のように sqlanywhere を追加します。

```
DATABASES = %w( sqlanywhere mysql oracle postgresql sqlite3
frontbase ibm_db sqlserver )
```

7. 次の SAP Sybase IQ 固有の注意事項に従って、Ruby on Rails Web サイト (http://guides.rails.info/getting_started.html) にあるチュートリアルに従います。
   - このチュートリアルには、`blog` プロジェクトを初期化するコマンドが記載されています。SAP Sybase IQ で使用するように `blog` プロジェクトを初期化するコマンドは次のようになります。
     ```
     rails new blog -d sqlanywhere
     ```
   - `blog` アプリケーションを作成したら、そのフォルダに移動して、そのアプリケーションで直接作業を続けます。
     ```
     cd blog
     ```
   - `gemfile` ファイルを編集して、SQL Anywhere ActiveRecord アダプタ用の `gem` ディレクティブを含めます。下に示す行の後に新しいディレクティブを追加します。
     ```
     gem 'sqlanywhere'
     gem 'activerecord-sqlanywhere-adapter'
     ```
   - `config¥database.yml` ファイルは、開発、テスト、運用データベースを参照します。チュートリアルで示されているように `rake` コマンドでデータベースを作成する代わりに、プロジェクトの `db` ディレクトリに移動して、次のように 3 つのデータベースを作成します。
     ```
     cd db
     iqinit -dba DBA,sql blog_development
     iqinit -dba DBA,sql blog_test
     iqinit -dba DBA,sql blog_production
     cd ..
     ```

SAP Sybase IQ で Rails サポートが設定されました。

次のステップ

次のように、データベースサーバと 3 つのデータベースを起動します。

```
iqsrv16 -n blog blog_development.db blog_production.db blog_test.db
```

コマンドラインのデータベースサーバ名 (`blog`) は、`database.yml` ファイルの `server:` タグで指定した名前と同じにする必要があります。`sqlanywhere.yml` テンプレートファイルは、データベースサーバ名が、生成されるすべての `database.yml` ファイルのプロジェクト名と必ず同じになるように設定されています。

Ruby-DBI ドライバ

この項では、DBI ドライバを使用する Ruby アプリケーションを作成する方法の概要を説明します。

DBI モジュールのロード

Ruby アプリケーションから DBI:SQLAnywhere インタフェースを使用するには、Ruby DBI モジュールを使用することを最初に Ruby に通知する必要があります。これを行うには、Ruby のソースファイルの先頭近くに次の行を挿入します。

```
require 'dbi'
```

DBI モジュールは、必要に応じて SQLAnywhere データベースドライバ (DBD) インタフェースを自動的にロードします。

接続を開いて閉じる

通常、データベースに対して 1 つの接続を開いてから、一連の SQL 文を実行して必要なすべての操作を実行します。接続を開くには、connect 関数を使用します。この戻り値は、接続時に後続の操作を行うために使用するデータベース接続のハンドルです。

connect 関数の呼び出しは、一般的に次の形式で行います。

```
dbh = DBI.connect('DBI:SQLAnywhere:server-name', user-id, password,
options)
```

- ***server-name*** – 接続先のデータベースサーバ名です。"option1=value1;option2=value2;..." というフォーマットで接続文字列を指定することもできます。
- ***user-id*** – 有効なユーザ ID です。この文字列が空白でない場合、接続文字列に ";UID=value" が付加されます。
- ***password*** – ユーザ ID に対応するパスワードです。この文字列が空白でない場合、接続文字列に ";PWD=value" が付加されます。
- ***options*** – DatabaseName、DatabaseFile、ConnectionName などの追加接続パラメータのハッシュです。"option1=value1;option2=value2;..." というフォーマットで接続文字列に付加されます。

connect 関数を使用してみるには、iqdemo データベースを作成し、データベースサーバと iqdemo データベースを起動してからサンプルの Ruby スクリプトを実行します。

```
$IQDIR16/demo
 mkiqdemo.sh
start_iq @iqdemo.cfg iqdemo.db
```

次のコードサンプルは、iqdemo データベースへの接続を開いて閉じます。次の例では文字列 "myserver" がサーバ名です。

```
require 'dbi'
DBI.connect('DBI:SQLAnywhere:myserver', '<user_id>', '<password>')
do |dbh|
    if dbh.ping
       print "Successfully Connected¥n"
       dbh.disconnect()
    end
end
```

サーバ名の代わりに接続文字列を指定することもできます。たとえば、上記の場合、connect 関数の最初のパラメータを次のように置き換えることにより、スクリプトを変更できます。

```
require 'dbi'
DBI.connect('DBI:SQLAnywhere:SERVER=myserver;DBN=iqdemo',
'<user_id>', '<password>') do |dbh|
    if dbh.ping
       print "Successfully Connected¥n"
       dbh.disconnect()
    end
end
```

接続文字列にはユーザ ID とパスワードを指定できません。これらの引数を省略すると、Ruby DBI によってデフォルトのユーザ名とパスワードが自動的に入力されるため、UID または PWD 接続パラメータを接続文字列に含めないでください。含めた場合は、例外がスローされます。

次の例は、追加接続パラメータをキーと値のペアのハッシュとして connect 関数に渡す方法を示しています。

```
require 'dbi'
DBI.connect('DBI:SQLAnywhere:myserver', '<user_id>', '<password>',
       { :ConnectionName => "RubyDemo",
         :DatabaseFile => "iqdemo.db",
         :DatabaseName => "iqdemo" }
        ) do |dbh|
    if dbh.ping
       print "Successfully Connected¥n"
       dbh.disconnect()
    end
end
```

データの選択

開かれた接続へのハンドルを取得したら、データベースに格納されているデータにアクセスして修正できます。最も単純な操作は、おそらくいくつかのローを取得して出力することです。

SQL 文は最初に実行する必要があります。文から結果セットが返された場合、ステートメントハンドルを使用して、結果セットに関するメタ情報と、結果セットのローを取得できます。次の例では、メタデータからカラム名を取得し、フェッチされた各ローのカラム名と値を表示しています。

```
require 'dbi'

def db_query( dbh, sql )
    sth = dbh.execute(sql)
    print "# of Fields:  #{sth.column_names.size}¥n"
    sth.fetch do |row|
        print "¥n"
        sth.column_info.each_with_index do |info, i|
            unless info["type_name"] == "LONG VARBINARY"
                print "#{info["name"]}=#{row[i]}¥n"
            end
        end
    end
    sth.finish
end

begin
    dbh = DBI.connect('DBI:SQLAnywhere:demo', '<user_id>',
'<password>')
    db_query(dbh, "SELECT * FROM Products")
rescue DBI::DatabaseError => e
    puts "An error occurred"
    puts "Error code: #{e.err}"
    puts "Error message: #{e.errstr}"
    puts "Error SQLSTATE: #{e.state}"
ensure
    dbh.disconnect if dbh
end
```

表示される出力の最初の数行を次に示します。

```
# of Fields:  8

ID=300
Name=Tee Shirt
Description=Tank Top
Size=Small
Color=White
Quantity=28
UnitPrice=9.00

ID=301
Name=Tee Shirt
Description=V-neck
Size=Medium
Color=Orange
Quantity=54
UnitPrice=14.00
```

終了したら、finish を呼び出してステートメントハンドルを解放することが重要です。finish を呼び出さなかった場合、次のようなエラーが表示される場合があります。

```
Resource governor for 'prepared statements' exceeded
```

ハンドルのリークを検出するために、SAP Sybase IQ データベースサーバでは、カーソルと準備文の数はデフォルトで接続ごとに最大 50 に制限されています。これらの制限を越えると、リソースガバナによってエラーが自動的に生成されます。このエラーが発生したら、破棄されていない文のハンドルを確認してください。文のハンドルが破棄されていない場合は、prepare_cached を慎重に使用してください。

必要な場合、max_cursor_count と max_statement_count オプションを設定してこれらの制限を変更できます。

ローの挿入

ローを挿入するには、開かれた接続へのハンドルが必要です。ローを挿入する最も簡単な方法は、パラメータ化された INSERT 文を使用する方法です。この場合、疑問符が値のプレースホルダとして使用されます。この文は最初に準備されてから、新しいローごとに 1 回実行されます。新しいローの値は、execute メソッドのパラメータとして指定されます。

```
require 'dbi'

def db_query( dbh, sql )
    sth = dbh.execute(sql)
    print "# of Fields:  #{sth.column_names.size}¥n"
    sth.fetch do |row|
        print "¥n"
        sth.column_info.each_with_index do |info, i|
            unless info["type_name"] == "LONG VARBINARY"
                print "#{info["name"]}=#{row[i]}¥n"
            end
        end
    end
    sth.finish
end

def db_insert( dbh, rows )
    sql = "INSERT INTO Customers (ID, GivenName, Surname,
                Street, City, State, Country, PostalCode,
                Phone, CompanyName)
           VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
    sth = dbh.prepare(sql);
    rows.each do |row|
        sth.execute(row[0],row[1],row[2],row[3],row[4],
                    row[5],row[6],row[7],row[8],row[9])
    end
end

begin
    dbh = DBI.connect('DBI:SQLAnywhere:demo', '<user_id>',
'<password>')
    rows = [
        [801,'Alex','Alt','5 Blue Ave','New York','NY','USA',
            '10012','5185553434','BXM'],
```

```
        [802,'Zach','Zed','82 Fair St','New York','NY','USA',
            '10033','5185552234','Zap']
       ]
    db_insert(dbh, rows)
    dbh.commit
    db_query(dbh, "SELECT * FROM Customers WHERE ID > 800")
rescue DBI::DatabaseError => e
    puts "An error occurred"
    puts "Error code: #{e.err}"
    puts "Error message: #{e.errstr}"
    puts "Error SQLSTATE: #{e.state}"
ensure
    dbh.disconnect if dbh
end
```

SAP Sybase IQ Ruby API リファレンス

SAP Sybase IQ には、SAP Sybase IQ C API への低レベルのインタフェースがあります。以降の各項で説明する API によって、SQL アプリケーションの迅速な開発が可能になります。Ruby によるアプリケーション開発の長所を示すために、次の Ruby のサンプルプログラムについて考えてみます。このプログラムは、SAP Sybase IQ Ruby 拡張をロードし、サンプルデータベースに接続し、Products テーブルのカラム値を表示し、接続を切断し、終了します。

```
begin
    require 'rubygems'
    gem 'sqlanywhere'
    unless defined? SQLAnywhere
        require 'sqlanywhere'
    end
end
api = SQLAnywhere::SQLAnywhereInterface.new()
SQLAnywhere::API.sqlany_initialize_interface( api )
api.sqlany_init()
conn = api.sqlany_new_connection()
api.sqlany_connect( conn, "DSN=Sybase IQ Demo" )
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Products" )
num_rows = api.sqlany_num_rows( stmt )
num_rows.times {
    api.sqlany_fetch_next( stmt )
    num_cols = api.sqlany_num_cols( stmt )
    for col in 1..num_cols do
        info = api.sqlany_get_column_info( stmt, col - 1 )
        unless info[3]==1 # Don't do binary
            rc, value = api.sqlany_get_column( stmt, col - 1 )
            print "#{info[2]}=#{value}¥n"
        end
    end
    print "¥n"
}
api.sqlany_free_stmt( stmt )
```

```
api.sqlany_disconnect(conn)
api.sqlany_free_connection(conn)
api.sqlany_fini()
SQLAnywhere::API.sqlany_finalize_interface( api )
```

この Ruby プログラムの結果セットから出力される最初の 2 つのローは、次のとおりです。

```
ID=300
Name=Tee Shirt
Description=Tank Top
Size=Small
Color=White
Quantity=28
UnitPrice=9.00

ID=301
Name=Tee Shirt
Description=V-neck
Size=Medium
Color=Orange
Quantity=54
UnitPrice=14.00
```

以降の各項では、サポートされている各関数について説明します。

sqlany_affected_rows

準備文の実行の影響を受けるローの数を返します。

構文

```
sqlany_affected_rows ( $stmt )
```

パラメータ

- **$stmt –** 準備および実行が成功したものの、結果セットが返されなかった文。たとえば、INSERT 文、UPDATE 文、または DELETE 文が実行された場合です。

戻り値

影響を受けたローの数のスカラ値を返します。失敗した場合は -1 を返します。

例

```
affected = api.sqlany_affected( stmt )
```

sqlany_bind_param 関数

ユーザが指定するバッファを準備文のパラメータとしてバインドします。

構文

```
sqlany_bind_param ( $stmt, $index, $param )
```

パラメータ

- **$stmt –** sqlany_prepare の実行によって返された文オブジェクト。
- **$index –** パラメータのインデックス。数値は、0 ～ sqlany_num_params() - 1 の間である必要があります。
- **$param –** sqlany_describe_bind_param から取得された、設定済みのバインドオブジェクト。

戻り値
成功した場合はスカラ値 1、失敗した場合は 0 を返します。

例

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
     SET Contacts.ID = Contacts.ID + 1000
     WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
print "Param name = ", param.get_name(), "¥n"
print "Param dir = ", param.get_direction(), "¥n"
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
```

sqlany_clear_error 関数

最後に格納されたエラーコードをクリアします。

構文

```
sqlany_clear_error ( $conn )
```

パラメータ

- **$conn –** sqlany_new_connection から返された接続オブジェクト。

戻り値
NULL を返します。

例

```
api.sqlany_clear_error( conn )
```

sqlany_client_version 関数

現在のクライアントバージョンを返します。

構文

```
sqlany_client_version ( )
```

戻り値
クライアントのバージョン文字列のスカラ値を返します。

例

```
buffer = api.sqlany_client_version()
```

sqlany_commit 関数

現在のトランザクションをコミットします。

構文

```
sqlany_commit ( $conn )
```

パラメータ

- **$conn –** コミット操作が実行される接続オブジェクト。

戻り値
成功した場合はスカラ値 1、失敗した場合は 0 を返します。

例

```
rc = api.sqlany_commit( conn )
```

sqlany_connect 関数

指定された接続オブジェクトと接続文字列を使用して、SQL Anywhere データベースサーバへの接続を作成します。

構文

```
sqlany_connect ( $conn, $str )
```

パラメータ

- **$conn –** sqlany_new_connection によって作成された接続オブジェクト。
- **$str –** SQL Anywhere 接続文字列。

戻り値
接続が確立された場合はスカラ値 1、接続が失敗した場合は 0 を返します。
sqlany_error を使用してエラーコードとエラーメッセージを取得します。

例

```
# Create a connection
conn = api.sqlany_new_connection()

# Establish a connection
```

```
status = api.sqlany_connect( conn, "UID=DBA;PWD=sql" )
print "Connection status = #{status}¥n"
```

sqlany_describe_bind_param 関数

準備文のバインドパラメータを記述します。

構文

```
sqlany_describe_bind_param ( $stmt, $index )
```

パラメータ

- **$stmt –** sqlany_prepare を使用して準備された文。
- **$index –** パラメータのインデックス。数値は、0 ～ sqlany_num_params() - 1 の間である必要があります。

戻り値

2 要素の配列を返します。最初の要素は、成功した場合は 1、失敗した場合は 0 です。2 番目の要素は記述されたパラメータです。

備考

この関数により、呼び出し元は準備文のパラメータに関する情報を判断できます。提供される情報の量は、準備文のタイプ (ストアドプロシージャまたは DML) によって決まります。パラメータの方向 (入力、出力、または入出力) に関する情報は常に提供されます。

例

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
     SET Contacts.ID = Contacts.ID + 1000
     WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
print "Param name = ", param.get_name(), "¥n"
print "Param dir = ", param.get_direction(), "¥n"
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
```

sqlany_disconnect 関数

SQL Anywhere 接続を切断します。コミットされていないトランザクションはすべてロールバックされます。

構文

```
sqlany_disconnect ( $conn )
```

パラメータ

- **$conn –** sqlany_connect を使用して確立された接続の接続オブジェクト。

戻り値
成功した場合はスカラ値 1、失敗した場合は 0 を返します。

例

```
# Disconnect from the database
status = api.sqlany_disconnect( conn )
print "Disconnect status = #{status}¥n"
```

sqlany_error 関数

接続オブジェクトに最後に格納されたエラーコードとエラーメッセージを返します。

構文

```
sqlany_error ( $conn )
```

パラメータ

- **$conn –** sqlany_new_connection から返された接続オブジェクト。

戻り値
2 要素の配列を返します。最初の要素は SQL エラーコード、2 番目の要素はエラーメッセージ文字列です。

エラーコードでは、正の値が警告、負の値がエラー、0 が成功を示します。

例

```
code, msg = api.sqlany_error( conn )
print "Code=#{code} Message=#{msg}¥n"
```

sqlany_execute 関数

準備文を実行します。

構文

```
sqlany_execute ( $stmt )
```

パラメータ

- **$stmt –** sqlany_prepare を使用して準備された文。

戻り値
成功した場合はスカラ値 1、失敗した場合は 0 を返します。

備考
sqlany_num_cols を使用して、文が結果セットを返したかどうか確認できます。

例

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
     SET Contacts.ID = Contacts.ID + 1000
     WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
rc = api.sqlany_execute( stmt )
```

sqlany_execute_direct 関数

文字列引数によって指定された SQL 文を実行します。

構文

```
sqlany_execute_direct ( $conn, $sql )
```

パラメータ

- **$conn –** sqlany_connect を使用して確立された接続の接続オブジェクト。
- **$sql –** SQL 文字列。SQL 文字列には、? のようなパラメータを含めることはできません。

戻り値
文オブジェクトを返します。失敗した場合は NULL を返します。

備考
文の準備と実行を 1 つの手順で実行する場合にこの関数を使用します。パラメータを持つ SQL 文を実行する際には使用しないでください。

例

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
rc = api.sqlany_fetch_next( stmt )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID  = api.sqlany_get_column( stmt, 1 )
rc, surname = api.sqlany_get_column( stmt, 2 )
rc, givenName = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
     surname, ",", givenName, ",", departmentID, "¥n"
```

sqlany_execute_immediate 関数

指定された SQL 文を、結果セットを返さずにただちに実行します。結果セットを返さない文の場合に使用すると便利です。

構文

```
sqlany_execute_immediate ( $conn, $sql )
```

パラメータ

- **$conn –** sqlany_connect を使用して確立された接続の接続オブジェクト。
- **$sql –** SQL 文字列。SQL 文字列には、? のようなパラメータを含めることはできません。

戻り値

成功した場合はスカラ値 1、失敗した場合は 0 を返します。

例

```
rc = api.sqlany_execute_immediate(conn, "UPDATE Contacts
     SET Contacts.ID = Contacts.ID + 1000
     WHERE Contacts.ID >= 50" )
```

sqlany_fetch_absolute 関数

結果セット内の現在のローを、指定されたロー番号に移し、そのローのデータをフェッチします。

構文

```
sqlany_fetch_absolute ( $stmt, $row_num )
```

パラメータ

- **$stmt –** sqlany_execute または sqlany_execute_direct によって実行された文オブジェクト。
- **$row_num –** フェッチされるローの番号。最初のローは 1、最後のローは -1。

戻り値

成功した場合はスカラ値 1、失敗した場合は 0 を返します。

例

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Fetch the second row
rc = api.sqlany_fetch_absolute( stmt, 2 )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID  = api.sqlany_get_column( stmt, 1 )
rc, surname = api.sqlany_get_column( stmt, 2 )
```

```
rc, givenName = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
     surname, ",", givenName, ",", departmentID, "¥n"
```

sqlany_fetch_next 関数

結果セットから次のローを返します。この関数は、ローポインタを進めてから新しいローのデータをフェッチします。

構文

```
sqlany_fetch_next ( $stmt )
```

パラメータ

- **$stmt –** sqlany_execute または sqlany_execute_direct によって実行された文オブジェクト。

戻り値
成功した場合はスカラ値 1、失敗した場合は 0 を返します。

例

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Fetch the second row
rc = api.sqlany_fetch_next( stmt )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID  = api.sqlany_get_column( stmt, 1 )
rc, surname = api.sqlany_get_column( stmt, 2 )
rc, givenName = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
     surname, ",", givenName, ",", departmentID, "¥n"
```

sqlany_fini 関数

API によって割り付けられたリソースを解放します。

構文

```
sqlany_fini ( )
```

戻り値
NULL を返します。

例

```
# Disconnect from the database
api.sqlany_disconnect( conn )

# Free the connection resources
api.sqlany_free_connection( conn )
```

```
# Free resources the api object uses
api.sqlany_fini()

# Close the interface
SQLAnywhere::API.sqlany_finalize_interface( api )
```

sqlany_free_connection 関数

接続オブジェクトに関連付けられているリソースを解放します。

構文

```
sqlany_free_connection ( $conn )
```

パラメータ

- **$conn –** sqlany_new_connection によって作成された接続オブジェクト。

戻り値

NULL を返します。

例

```
# Disconnect from the database
api.sqlany_disconnect( conn )

# Free the connection resources
api.sqlany_free_connection( conn )

# Free resources the api object uses
api.sqlany_fini()

# Close the interface
SQLAnywhere::API.sqlany_finalize_interface( api )
```

sqlany_free_stmt 関数

文オブジェクトに関連付けられているリソースを解放します。

構文

```
sqlany_free_stmt ( $stmt )
```

パラメータ

- **$stmt –** sqlany_prepare または sqlany_execute_direct の実行によって返された文オブジェクト。

戻り値

NULL を返します。

例

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
     SET Contacts.ID = Contacts.ID + 1000
     WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
rc = api.sqlany_execute( stmt )
rc = api.sqlany_free_stmt( stmt )
```

sqlany_get_bind_param_info 関数

sqlany_bind_param を使用してバインドされたパラメータに関する情報を取得します。

構文

```
sqlany_get_bind_param_info ( $stmt, $index )
```

パラメータ

- **$stmt –** sqlany_prepare を使用して準備された文。
- **$index –** パラメータのインデックス。数値は、0 ～ sqlany_num_params() - 1 の間である必要があります。

戻り値

2 要素の配列を返します。最初の要素は、成功した場合は 1、失敗した場合は 0 です。2 番目の要素は記述されたパラメータです。

例

```
# Get information on first parameter (0)
rc, param_info = api.sqlany_get_bind_param_info( stmt, 0 )
print "Param_info direction = ", param_info.get_direction(), "¥n"
print "Param_info output = ", param_info.get_output(), "¥n"
```

sqlany_get_column 関数

指定されたカラムについてフェッチされた値を返します。

構文

```
sqlany_get_column ( $stmt, $col_index )
```

パラメータ

- **$stmt –** sqlany_execute または sqlany_execute_direct によって実行された文オブジェクト。

- **$col_index –** 取り出すカラムの数。カラムの数は、0 ～ sqlany_num_cols() - 1 の間です。

戻り値
2 要素の配列を返します。最初の要素は、成功した場合は 1、失敗した場合は 0 です。2 番目の要素はカラム値です。

例

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Fetch the second row
rc = api.sqlany_fetch_next( stmt )
rc, employeeID = api.sqlany_get_column( stmt, 0 )
rc, managerID  = api.sqlany_get_column( stmt, 1 )
rc, surname = api.sqlany_get_column( stmt, 2 )
rc, givenName = api.sqlany_get_column( stmt, 3 )
rc, departmentID = api.sqlany_get_column( stmt, 4 )
print employeeID, ",", managerID, ",",
     surname, ",", givenName, ",", departmentID, "¥n"
```

sqlany_get_column_info 関数

指定された結果セットのカラムに対するカラム情報を取得します。

構文

```
sqlany_get_column_info ( $stmt, $col_index )
```

パラメータ

- **$stmt –** sqlany_execute または sqlany_execute_direct によって実行された文オブジェクト。
- **$col_index –** カラムの数は、0 ～ sqlany_num_cols() - 1 の間です。

戻り値
結果セット内のカラムに関する情報を格納した 9 要素の配列を返します。最初の要素は、成功した場合は 1、失敗した場合は 0 です。配列の各要素を次の表に示します。

要素番号	データ型	説明
0	整数	成功した場合は 1、失敗した場合は 0。
1	整数	カラムのインデックス (0 ～ sqlany_num_cols() - 1)。
2	文字列	カラム名。
3	整数	カラム型。

要素番号	データ型	説明
4	整数	カラムのネイティブ型。
5	整数	カラム精度 (数値型の場合)。
6	整数	カラムの位取り (数値型の場合)。
7	整数	カラムサイズ。
8	整数	カラムが NULL 入力可かどうか (1 = NULL 入力可、0 = NULL 入力不可)。

例

```
# Get column info for first column (0)
rc, col_num, col_name, col_type, col_native_type, col_precision,
col_scale,
    col_size, col_nullable = api.sqlany_get_column_info( stmt, 0 )
```

sqlany_get_next_result 関数

複数の結果セットクエリのうちの次の結果セットに進みます。

構文

```
sqlany_get_next_result ( $stmt )
```

パラメータ

- **$stmt –** sqlany_execute または sqlany_execute_direct によって実行された文オブジェクト。

戻り値

成功した場合はスカラ値 1、失敗した場合は 0 を返します。

例

```
stmt = api.sqlany_prepare(conn, "call two_results()" )
rc = api.sqlany_execute( stmt )
# Fetch from first result set
rc = api.sqlany_fetch_absolute( stmt, 3 )
# Go to next result set
rc = api.sqlany_get_next_result( stmt )
# Fetch from second result set
rc = api.sqlany_fetch_absolute( stmt, 2 )
```

sqlany_init 関数

インタフェースを初期化します。

構文

```
sqlany_init ( )
```

戻り値

2 要素の配列を返します。最初の要素は、成功した場合は 1、失敗した場合は 0 です。2 番目の要素は Ruby インタフェースバージョンです。

例

```
# Load the SQLAnywhere gem
begin
    require 'rubygems'
    gem 'sqlanywhere'
    unless defined? SQLAnywhere
        require 'sqlanywhere'
    end
end
# Create an interface
api = SQLAnywhere::SQLAnywhereInterface.new()
# Initialize the interface (loads the DLL/SO)
SQLAnywhere::API.sqlany_initialize_interface( api )
# Initialize our api object
api.sqlany_init()
```

sqlany_new_connection 関数

接続オブジェクトを作成します。

構文

```
sqlany_new_connection ()
```

戻り値

接続オブジェクトのスカラ値を返します。

備考

データベース接続が確立される前に接続オブジェクトが作成されている必要があります。接続オブジェクトからエラーが取得される場合があります。各接続で一度に処理できる要求は 1 つだけです。

例

```
# Create a connection
conn = api.sqlany_new_connection()

# Establish a connection
```

```
status = api.sqlany_connect( conn, "UID=DBA;PWD=sql" )
print "Status=#{status}¥n"
```

sqlany_num_cols 関数

結果セット内のカラム数を返します。

構文

```
sqlany_num_cols ( $stmt )
```

パラメータ

- **$stmt –** sqlany_execute または sqlany_execute_direct によって実行された文オブジェクト。

戻り値

結果セット内のカラム数のスカラ値を返します。失敗した場合は -1 を返します。

例

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Get number of result set columns
num_cols = api.sqlany_num_cols( stmt )
```

sqlany_num_params 関数

準備文で必要とされるパラメータ数を返します。

構文

```
sqlany_num_params ( $stmt )
```

パラメータ

- **$stmt –** sqlany_prepare の実行によって返された文オブジェクト。

戻り値

準備文内のパラメータ数のスカラ値を返します。失敗した場合は -1 を返します。

例

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
      SET Contacts.ID = Contacts.ID + 1000
      WHERE Contacts.ID >= ?" )
num_params = api.sqlany_num_params( stmt )
```

sqlany_num_rows 関数

結果セット内のロー数を返します。

構文

```
sqlany_num_rows ( $stmt )
```

パラメータ

- **$stmt –** sqlany_execute または sqlany_execute_direct によって実行された文オブジェクト。

戻り値

結果セット内のロー数のスカラ値を返します。ロー数が推定値の場合は負の値を返します。また、その推定値が、返された整数の絶対値となります。ロー数が正確な値の場合は正の値を返します。

備考

デフォルトでは、この関数は推定値のみを返します。正確なロー数が返されるようにするには、接続の ROW_COUNTS オプションを設定します。

複数の結果セットを返す文の場合、最初の結果セット内のロー数だけが返されます。sqlany_get_next_result を使用して次の結果セットに進んでも、sqlany_num_rows によって返されるのは最初の結果セット内のロー数のみです。

例

```
stmt = api.sqlany_execute_direct( conn, "SELECT * FROM Employees" )
# Get number of rows in result set
num_rows = api.sqlany_num_rows( stmt )
```

sqlany_prepare 関数

指定された SQL 文字列を準備します。

構文

```
sqlany_prepare ( $conn, $sql )
```

パラメータ

- **$conn –** sqlany_connect を使用して確立された接続の接続オブジェクト。
- **$sql –** 準備される SQL 文。

戻り値

文オブジェクトのスカラ値を返します。失敗した場合は NULL を返します。

備考
文オブジェクトに関連付けられた文は sqlany_execute によって実行されます。sqlany_free_stmt を使用して、文オブジェクトに関連付けられたリソースを解放できます。

例

```
stmt = api.sqlany_prepare(conn, "UPDATE Contacts
     SET Contacts.ID = Contacts.ID + 1000
     WHERE Contacts.ID >= ?" )
rc, param = api.sqlany_describe_bind_param( stmt, 0 )
param.set_value(50)
rc = api.sqlany_bind_param( stmt, 0, param )
rc = api.sqlany_execute( stmt )
```

sqlany_rollback 関数

現在のトランザクションをロールバックします。

構文

```
sqlany_rollback ( $conn )
```

パラメータ

- **$conn –** ロールバック操作が実行される接続オブジェクト。

戻り値
成功した場合はスカラ値 1、失敗した場合は 0 を返します。

例

```
rc = api.sqlany_rollback( conn )
```

sqlany_sqlstate 関数

現在の SQLSTATE を取得します。

構文

```
sqlany_sqlstate ( $conn )
```

パラメータ

- **$conn –** sqlany_new_connection から返された接続オブジェクト。

戻り値
現在の SQLSTATE を表す 5 文字のスカラ値を返します。

例

```
sql_state = api.sqlany_sqlstate( conn )
```

カラムの型

次の Ruby クラスで、一部の SQL Anywhere Ruby 関数によって返されるカラムの型が定義されています。

```
class Types
    A_INVALID_TYPE = 0
    A_BINARY       = 1
    A_STRING       = 2
    A_DOUBLE       = 3
    A_VAL64        = 4
    A_UVAL64       = 5
    A_VAL32        = 6
    A_UVAL32       = 7
    A_VAL16        = 8
    A_UVAL16       = 9
    A_VAL8         = 10
    A_UVAL8        = 11
end
```

ネイティブのカラム型

次の表に、一部の SQL Anywhere 関数によって返されるネイティブのカラム型を示します。

ネイティブ型の値	ネイティブ型
384	DT_DATE
388	DT_TIME
390	DT_TIMESTAMP_STRUCT
392	DT_TIMESTAMP
448	DT_VARCHAR
452	DT_FIXCHAR
456	DT_LONGVARCHAR
460	DT_STRING
480	DT_DOUBLE
482	DT_FLOAT
484	DT_DECIMAL
496	DT_INT
500	DT_SMALLINT

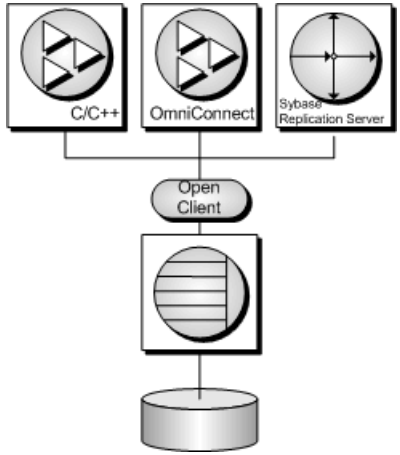
ネイティブ型の値	ネイティブ型
524	DT_BINARY
528	DT_LONGBINARY
600	DT_VARIABLE
604	DT_TINYINT
608	DT_BIGINT
612	DT_UNSINT
616	DT_UNSSMALLINT
620	DT_UNSBIGINT
624	DT_BIT
628	DT_NSTRING
632	DT_NFIXCHAR
636	DT_NVARCHAR
640	DT_LONGNVARCHAR

Sybase Open Client のサポート

Sybase Open Client は、カスタマアプリケーション、サードパーティ製品、その他の Sybase 製品に、SAP Sybase IQ およびその他の Open Server と通信するために必要なインタフェースを提供します。

どのようなときに Open Client を使用するか
Adaptive Server Enterprise との互換性が必要なとき、または Open Client インタフェースをサポートする他の Sybase 製品を使用しているときに、Open Client インタフェースの使用が考えられます。

Open Client アプリケーション
C または C++ で開発したアプリケーションを Open Client API を使用して SAP Sybase IQ に接続できます。その他の Sybase アプリケーションの中にも OmniConnect のように Open Client を使用しているものがあります。Open Client API は Sybase Adaptive Server Enterprise でもサポートされています。



Open Client アーキテクチャ

この項では、SAP Sybase IQ 用の Sybase Open Client プログラミングインタフェースについて説明します。Sybase Open Client アプリケーション開発の基本のマニュアルは、SAP から入手できる Sybase Open Client マニュアルです。この項は、SAP Sybase IQ 特有の機能について説明していますが、Sybase Open Client アプリケーションプログラミングの包括的なガイドではありません。

Sybase Open Client には、2つのコンポーネントがあります。プログラミングインタフェースとネットワークサービスです。

DB-Library と Client Library
Sybase Open Client には、クライアントアプリケーションを記述するための主要なプログラミングインタフェースが 2 つ用意されています。DB-Library と Client-Library です。

Open Client DB-Library は、以前の Open Client アプリケーションをサポートする、Client-Library とはまったく別のプログラミングインタフェースです。DB-Library については、Sybase Open Client 製品に付属する『Open Client DB-Library/C リファレンスマニュアル』を参照してください。

Client-Library プログラムも CS-Library に依存しています。CS-Library は、Client-Library アプリケーションと Server-Library アプリケーションの両方が使用するルーチンを提供します。Client-Library アプリケーションは、Bulk-Library のルーチンを使用して高速データ転送を行うこともできます。

CS-Library と Bulk-Library はどちらも Sybase Open Client に含まれていますが、別々に使用できます。

ネットワークサービス
Open Client ネットワークサービスは、TCP/IP や DECnet などの特定のネットワークプロトコルをサポートする Sybase Net-Library を含みます。Net-Library インタフェースはアプリケーション開発者からは見えません。ただしプラットフォームによっては、アプリケーションがシステムネットワーク構成ごとに別の Net-Library ドライバを必要とする場合もあります。Net-Library ドライバの指定は、ホストプラットフォームに応じて、システムの Sybase 設定で行うか、またはプログラムをコンパイルしてリンクするときに行います。

ドライバ設定の詳細については、『Open Client/Server 設定ガイド』を参照してください。

Client-Library プログラムの作成方法については、『Open Client/Server プログラマーズガイド補足』を参照してください。

Open Client アプリケーション作成に必要なもの

Open Client アプリケーションを実行するためには、アプリケーションを実行しているコンピュータに Sybase Open Client コンポーネントをインストールして構成する必要があります。これらのコンポーネントは、他の Sybase 製品の一部として入手するか、ライセンス契約の条項に従って、SAP Sybase IQ とともに、これらのライブラリをオプションでインストールできます。

データベースサーバを実行しているコンピュータでは、Open Client アプリケーションは Open Client コンポーネントを一切必要としません。

Open Client アプリケーションを作成するには、SAP から入手可能な Open Client の開発バージョンが必要です。

デフォルトでは、SAP Sybase IQ データベースは大文字と小文字を区別しないように、Adaptive Server データベースは区別するように作成されます。

Open Client データ型マッピング

Sybase Open Client は独自の内部データ型を持っており、そのデータ型は SAP Sybase IQ で使用されるものと細部が多少異なります。このため、SAP Sybase IQ は Open Client アプリケーションで使用されるデータ型と SAP Sybase IQ で使用されるデータ型を内部的にマッピングします。

Open Client アプリケーションを作成するには、Open Client の開発バージョンが必要です。Open Client アプリケーションを使用するには、そのアプリケーションが動作するコンピュータに、Open Client ランタイムをインストールし構成しておく必要があります。

SAP Sybase IQ サーバは Open Client アプリケーションをサポートするために、外部通信のランタイムを一切必要としません。

Open Client の各データ型は、同等の SAP Sybase IQ のデータ型にマッピングされます。Open Client のデータ型は、すべてサポートされます。

Open Client とは異なる名前を持つ SAP Sybase IQ のデータ型
次の表は、SAP Sybase IQ でサポートされるデータ型と Open Client のデータ型のマッピングリストです。これらは同じデータ型名ではないデータ型です。

SAP Sybase IQ データ型	Open Client データ型
unsigned short	int
unsigned int	bigint
unsigned bigint	numeric(20,0)
string	varchar
timestamp	datetime

Open Client データ型マッピングの範囲制限

データ型によっては、SAP Sybase IQ と Open Client で範囲が異なります。このような場合には、データを検索または挿入するときにオーバーフローエラーが発生することがあります。

次の表にまとめた Open Client アプリケーションのデータ型は、SAP Sybase IQ データ型にマッピングできますが、 使用可能な値の範囲に制限があります。

通常、Open Client データ型からマッピングされる SAP Sybase IQ データ型のほうが使用可能な値の範囲が大きくなっています。このため、Open Client アプリケーションでは大きすぎてフェッチできない値を SAP Sybase IQ に渡してデータベースに格納することも可能です。

データ型	Open Client の最小値	Open Client の最大値	SAP Sybase IQ の最小値	SAP Sybase IQ の最大値
MONEY	-922 377 203 685 477.5808	922 377 203 685 477.5807	-999 999 999 999 999.9999	999 999 999 999 999.9999
SMALLMONEY	-214 748.3648	214 748.3647	-999 999.9999	-999 999.9999
DATETIME [1]	January 1, 1753	December 31, 9999	January 1, 0001	December 31, 9999
SMALLDATETIME	January 1, 1900	June 6, 2079	January 1, 0001	December 31, 9999

[1] OpenClient 15.5 より前のバージョン用。これ以降のバージョンでは、0001-01-01 ～ 9999-12-31 の範囲内のすべての日付がサポートされています。

例

たとえば、Open Client の MONEY および SMALLMONEY データ型は、基本となる SAP Sybase IQ 実装の全数値範囲を超えることはありません。したがって、Open Client のデータ型 MONEY の境界を超える値を SAP Sybase IQ のカラムに設定できます。クライアントが SAP Sybase IQ 経由でこのような違反値をフェッチすると、エラーになります。

タイムスタンプ
クライアントが Open Client 15.1 以前を使用している場合、SAP Sybase IQ に挿入した TIMESTAMP 値または SAP Sybase IQ から取り出した TIMESTAMP 値は、日付部分が January 1, 1753 以降に制限され、時刻部分が秒の精度の 300 分の 1 に制限されています。ただし、クライアントが Open Client 15.5 以降を使用している場合は、TIMESTAMP 値に制限はありません。

Open Client アプリケーションでの SQL

この項では、SAP Sybase IQ 特有の問題に特に注目しながら、Open Client アプリケーションで SQL を使用する方法を簡潔に説明します。

詳細については、http://www.sybase.com/products/databasemanagement/openserver にある Open Client のマニュアルを参照してください。

Open Client SQL 文の実行

SQL 文を Client Library 関数呼び出しに入れてデータベースサーバに送ります。たとえば、次の一組の呼び出しは DELETE 文を実行します。

```
ret = ct_command(cmd, CS_LANG_CMD,
                  "DELETE FROM Employees
                   WHERE EmployeeID=105"
                 CS_NULLTERM,
                 CS_UNUSED);
ret = ct_send(cmd);
```

Open Client の準備文

ct_dynamic 関数を使用して準備文を管理します。この関数には、実行するアクションを *type* パラメータで指定します。

Open Client で準備文を使用するには、次のタスクを実行します。

1. CS_PREPARE を *type* パラメータに指定した ct_dynamic 関数を使用して文を準備します。
2. ct_param を使用して文のパラメータを設定します。
3. CS_EXECUTE を *type* パラメータに指定した ct_dynamic を使用して文を実行します。
4. CS_DEALLOC を *type* パラメータに指定した ct_dynamic を使用して、文に関連付けられたリソースを解放します。

Open Client で準備文を使用する方法の詳細については、Open Client のマニュアルを参照してください。

Open Client カーソルの管理

ct_cursor 関数を使用してカーソルを管理します。この関数には、実行するアクションを *type* パラメータで指定します。

サポートされるカーソルタイプ
SAP Sybase IQ でサポートされるすべてのタイプのカーソルを、Open Client インタフェースを通じて使用できるわけではありません。スクロールカーソル、動的ス

クロールカーソル、または insensitive カーソルは、Open Client を通じて使用できません。

一意性と更新可能性が、カーソルの 2 つの特性です。カーソルは一意 (各ローが、アプリケーションに使用されるかどうかに関係なく、プライマリキーまたは一意性情報を持つ) でも一意でなくても構いません。カーソルは読み込み専用にも更新可能にもできます。カーソルが更新可能で一意でない場合は、CS_CURSOR_ROWS の設定に関係なく、ローのプリフェッチが行われないので、パフォーマンスが低下する可能性があります。

カーソルを使用する手順

Embedded SQL などの他のインタフェースとは異なり、Open Client はカーソルを、文字列として表現された SQL 文に対応させます。Embedded SQL の場合は、まず文を作成し、次にステートメントハンドルを使用してカーソルを宣言します。

Open Client でカーソルを使用するには、次のタスクを実行します。

1. Open Client のカーソルを宣言するには、CS_CURSOR_DECLARE を *type* パラメータに指定した ct_cursor を使用します。
2. カーソルを宣言したら、CS_CURSOR_ROWS を *type* パラメータに指定した ct_cursor を使用して、サーバからローをフェッチするたびにクライアント側にプリフェッチするローの数を制御できます。
   プリフェッチしたローをクライアント側に格納すると、サーバに対する呼び出し数を減らし、全体的なスループットとターンアラウンドタイムを改善できます。プリフェッチしたローは、すぐにアプリケーションに渡されるわけではなく、いつでも使用できるようにクライアント側のバッファに格納されます。
   prefetch データベースオプションの設定によって、他のインタフェースに対するローのプリフェッチを制御します。この設定は、Open Client 接続では無視されます。CS_CURSOR_ROWS 設定は、一意でない更新可能なカーソルについては無視されます。
3. Open Client のカーソルを開くには、CS_CURSOR_OPEN を *type* パラメータに指定した ct_cursor を使用します。
4. 各ローをアプリケーションにフェッチするには、ct_fetch を使用します。
5. カーソルを閉じるには、CS_CURSOR_CLOSE を指定した ct_cursor を使用します。
6. Open Client では、カーソルに対応するリソースの割り付けを解除する必要もあります。CS_CURSOR_DEALLOC を指定した ct_cursor を使用してください。CS_CURSOR_CLOSE とともに追加パラメータ CS_DEALLOC を指定して、これらの処理を 1 ステップで実行することもできます。

カーソルによる Open Client のローの変更

Open Client では、カーソルが 1 つのテーブル用であればカーソル内でローを削除または更新できます。テーブルを更新するパーミッションを持っている必要があり、そのカーソルは更新のマークが付けられている必要があります。

フェッチを実行する代わりに、CS_CURSOR_DELETE または CS_CURSOR_UPDATE を指定した ct_cursor を使用してカーソルのローを削除または更新できます。

Open Client アプリケーションではカーソルからのローの挿入はできません。

Open Client の結果セット

Open Client が結果セットを処理する方法は、他の SAP Sybase IQ インタフェースの方法とは異なります。

Embedded SQL と ODBC では、結果を受け取る変数の適切な数と型を設定するために、クエリまたはストアドプロシージャを*記述*します。記述は文自体を対象に行います。

Open Client では、文を記述する必要はありません。代わりに、サーバから戻される各ローは内容に関する記述を持つことができます。ct_command と ct_send を使用して文を実行した場合、クエリに戻されたローのあらゆる処理に ct_results 関数を使用できます。

このようなロー単位の結果セット処理方式を使用したくない場合は、ct_dynamic を使用して SQL 文を作成し、ct_describe を使用してその結果セットを記述できます。この方式は、他のインタフェースにおける SQL 文の記述方式と密接に対応しています。

SAP Sybase IQ における Open Client の既知の制限

Open Client インタフェースを使用すると、SAP Sybase IQ データベースを、Adaptive Server Enterprise データベースとほとんど同じ方法で使用できます。ただし、次に示すような制限があります。

- SAP Sybase IQ は Adaptive Server Enterprise のコミットサービスをサポートしません。
- クライアント／サーバ接続の*機能*によって、その接続で許可されているクライアント要求とサーバ応答のタイプが決まります。次の機能はサポートされていません。

  CS_CSR_ABS

CS_CSR_FIRST
CS_CSR_LAST
CS_CSR_PREV
CS_CSR_REL
CS_DATA_BOUNDARY
CS_DATA_SENSITIVITY
CS_OPT_FORMATONLY
CS_PROTO_DYNPROC
CS_REG_NOTIF
CS_REQ_BCP

- SSL などのセキュリティオプションはサポートされていません。ただし、パスワードの暗号化はサポートされています。
- Open Client アプリケーションは TCP/IP を使用して SAP Sybase IQ に接続できます。
  機能の詳細については、『Open Server Server-Library C リファレンスマニュアル』を参照してください。
- CS_DATAFMT を CS_DESCRIBE_INPUT とともに使用すると、パラメータが指定された変数を入力データとして SAP Sybase IQ に送信した場合、カラムのデータ型は返されません。

HTTP Web サービス

SAP Sybase IQ を HTTP Web サーバとして使用する Web サービスアプリケーションを開発します。

HTTP Web サーバとしての SAP Sybase IQ

SAP Sybase IQ には、SAP Sybase IQ データベースでのオンライン Web サービスの作成を可能にする組み込み HTTP Web サーバが含まれています。SAP Sybase IQ Web サーバでは、Web ブラウザとクライアントアプリケーションから送信される HTTP 要求と SOAP over HTTP 要求がサポートされてます。Web サービスはデータベースに組み込まれているため、Web サーバのパフォーマンスが最適化されます。

SAP Sybase IQ Web サービスは、クライアントアプリケーションに JDBC や ODBC などの従来のインタフェースの代わりになる方法を提供します。追加のコンポーネントが必要なく、Perl や Python などのスクリプト言語を含む各種の言語で記述されたマルチプラットフォームクライアントアプリケーションからアクセスできるため、これらは簡単に配備されます。

HTTP Web サーバを介した Web サービスの提供に加え、インターネットや他の SAP Sybase IQ HTTP Web サーバで使用できる標準の Web サービスにアクセスするため、SAP Sybase IQ は SOAP または HTTP クライアントアプリケーションとして機能できます。

SAP Sybase IQ を HTTP Web サーバとして使用するためのクイックスタート

この項では、SAP Sybase IQ HTTP Web サーバの起動方法、Web サービスの作成方法、Web ブラウザからのアクセス方法について説明します。SOAP over HTTP のサポートやアプリケーション開発など、SAP Sybase IQ Web サービスの機能については詳しく説明していません。このマニュアルで説明するもの以外にも、数多くの SAP Sybase IQ Web サービスの機能を使用できます。

SAP Sybase IQ HTTP Web サーバおよび HTTP Web サービスを作成するには、次のタスクを実行します。

1. SAP Sybase IQ HTTP Web サーバを起動し、SAP Sybase IQ データベースをロードします。
   コマンドプロンプトで次のコマンドを実行します。
   ```
   iqsrv16 -xs http(port=8082) iqdemo.db
   ```

注意： ネットワーク上でアクセスできるデータベースサーバを起動するには、iqsrv16 コマンドを使用します。

-xs http(port=8082) オプションは、ポート 8082 で HTTP 要求を受信することをサーバに指示します。Web サーバがポート 8082 ですでに稼働中の場合は、異なるポート番号を使用します。

2. CREATE SERVICE 文を使用して、着信 Web ブラウザ要求に応答する Web サービスを作成します。
   a. 次のコマンドを実行することで Interactive SQL を使用して `demo.db` データベースに接続します。
      ```
      dbisql -c "dbf=iqdemo.db;uid=<user_id>;pwd=<password>"
      ```
   b. データベースに新しい Web サービスを作成します。
      Interactive SQL で次の SQL 文を実行します。
      ```
      CREATE SERVICE SampleWebService
          TYPE 'web-service-type-clause'
          AUTHORIZATION OFF
          USER DBA
          AS SELECT 'Hello world!';
      ```
      web-service-type-clause を目的の Web サービスタイプに置き換えます。Web ブラウザの互換性を保つため、`HTML` type 句をおすすめします。汎用 HTTP Web サービスの type 句には、他に、`XML`、`RAW`、`JSON` があります。
      CREATE SERVICE 文により、SELECT 文の結果セットを返す `SampleWebService` Web サービスが作成されます。この例では、この文から "Hello world!" が返されます。
      AUTHORIZATION OFF 句は、Web サービスのアクセスに認証が不要であることを示します。
      USER DBA 文は、サービス文を DBA ログイン名で実行する必要があることを示します。
      AS SELECT 句を指定すると、テーブルまたは関数からサービスを選択したり、データを直接表示したりできます。AS CALL は、ストアドプロシージャを呼び出す代替句として使用します。
3. Web ブラウザで Web サービスを表示します。
   SAP Sybase IQ HTTP Web サーバが実行されているコンピュータで、Internet Explorer や Firefox などの Web ブラウザを開き、次の URL にアクセスします。
   ```
   http://localhost:8082/demo/SampleWebService
   ```
   Web ブラウザは、この URL からポート 8082 上の HTTP Web サーバにリダイレクトされます。`SampleWebService` から "Hello world" と表示されます。結果セットの出力は、手順 2 の *web-service-type-clause* で指定したフォーマットで表示されます。

他のサンプルリソース
サンプルプログラムは %ALLUSERSPROFILE%¥SybaseIQ¥samples ¥SQLAnywhere¥http ディレクトリにあります。

その他の例は、CodeXchange (http://www.sybase.com/developer/codexchange) から入手できる場合があります。

HTTP Web サーバを起動する方法

-xs サーバオプションを指定してデータベースサーバを起動すると、SAP Sybase IQ HTTP Web サーバが自動的に起動します。このオプションでは、次のタスクを実行できます。

Web サービス要求を受信する Web サービスプロトコルの有効化
サーバポート、ロギング、タイムアウト条件、最大要求サイズなどのネットワークプロトコルオプションの設定

コマンドの全般的なフォーマットを次に示します。

```
iqsrv16 -xs protocol-type(protocol-options) your-database-name.db
```

protocol-type および *protocol-options* を、サポートされている次のいずれかのプロトコルと適切なプロトコルオプションに置き換えます。

- **HTTP –** このプロトコルは、HTTP 接続を受信する場合に使用します。次に例を示します。
  ```
  iqsrv16 -xs HTTP(PORT=8082) services.db
  ```
- **HTTPS –** このプロトコルは、HTTPS 接続を受信する場合に使用します。SSL バージョン 3.0 と TLS バージョン 1.0/1.1 がサポートされています。次に例を示します。
  ```
  iqsrv16 -xs "HTTPS(FIPS=N;PORT=8082;IDENTITY="%ALLUSERSPROFILE
  %"¥SybaseIQ¥samples¥Certificates
  ¥rsaserver.id;IDENTITY_PASSWORD=test)" services.db
  ```

注意： サポートされている各プロトコルでは、ネットワークプロトコルオプションを使用できます。これらのオプションを使用すると、プロトコルの動作を制御でき、データベースサーバの起動時にコマンドラインで設定できます。

ネットワークプロトコルオプションの設定

ネットワークプロトコルオプションは、指定された Web サービスプロトコルを制御するための省略可能な設定です。-xs データベースサーバオプションを指定してデータベースサーバを起動すると、これらの設定をコマンドラインで実行できます。

たとえば、次のコマンドラインは、PORT、FIPS、Identity、Identity_Password ネットワークプロトコルオプションを指定した HTTPS リスナを設定します。

```
iqsrv16 -xs https(PORT=544;FIPS=YES;
    IDENTITY=certificate.id;IDENTITY_PASSWORD=password) your-
database-name.db
```

このコマンドは、`your-database-name.db` データベースに対して、HTTPS Web サービスプロトコルを有効にするデータベースサーバを起動します。ネットワークプロトコルオプションは、Web サーバが次のタスクを実行することを示します。

デフォルトの HTTPS ポート (443) ではなくポート 544 で受信する。
FIPS 認定セキュリティアルゴリズムを有効にして、通信を暗号化する。
パブリック証明書とプライベートキーが格納されている、指定された ID ファイル `certificate.id` を検索する。
指定された ID パスワード `password` に対してプライベートキーを検証する。

次の表に、Web サービスプロトコルで一般的に使用されるネットワークプロトコルオプションを示します。

ネットワークプロトコルオプション	使用可能な Web サービスプロトコル	説明
DatabaseName (DBN) プロトコルオプション	HTTP、HTTPS	Web 要求を処理するときに使用するデータベースの名前を指定する。また、REQUIRED や AUTO キーワードを使用して URL の一部としてデータベース名が必要かどうかを指定する。
FIPS プロトコルオプション	HTTPS	データベースファイルを暗号化したり、データベースクライアント／サーバ通信や Web サービスにおける通信を暗号化するために、FIPS 認定のセキュリティアルゴリズムを有効にする。
Identity プロトコルオプション	HTTPS	セキュア HTTPS 接続に使用する ID ファイルの名前を指定する。
Identity_Password プロトコルオプション	HTTPS	暗号化証明書のパスワードを指定する。
LocalOnly (LO) プロトコルオプション	HTTP、HTTPS	クライアントがローカルコンピュータ上のサーバ (存在する場合) にのみ接続できるようにする。

ネットワークプロトコルオプション	使用可能な Web サービスプロトコル	説明
LogFile (LOG) プロトコルオプション	HTTP、HTTPS	データベースサーバが Web 要求に関する情報を書き込むファイル名を指定する。
LogFormat (LF) プロトコルオプション	HTTP、HTTPS	データベースサーバが Web 要求に関する情報を書き込むログファイルに書き込まれるメッセージのフォーマットと、メッセージに表示されるフィールドを制御する。
LogOptions (LOPT) プロトコルオプション	HTTP、HTTPS	データベースサーバが Web 要求に関する情報を書き込むログに記録されるメッセージタイプを指定する。
ServerPort (PORT) プロトコルオプション	HTTP、HTTPS	データベースサーバが受信しているポートを指定する。

複数の HTTP Web サーバを起動する方法

複数の HTTP Web サーバ構成では、データベース間で Web サービスを作成し、それを 1 つの Web サイトの一部として表示できます。-xs データベースサーバオプションの複数のインスタンスを使用して、複数の HTTP Web サーバを起動できます。このタスクは、HTTP Web サーバごとにユニークなポート番号を指定することによって実行されます。

例

この例のコマンドラインでは、2 つの HTTP Web サービスを起動します。1 つは `your-first-database.db` 用、もう 1 つは `your-second-database.db` 用です。

```
iqsrv16 -xs http(port=80;dbn=your-first-
database),http(port=8800;dbn=your-second-database)
    your-first-database.db your-second-database.db
```

Web サービスとは？

Web サービスは、コンピュータ間のデータ転送と相互運用性を支援するソフトウェアを指します。Web サービスはビジネスロジックのセグメントをインターネットを介して使用できるようにします。HTTP Web サーバで Web サービスを管理すると、URL がクライアントで使用可能になります。URL の指定時に使用される表記規則によって、サーバと Web クライアント間の通信方法が決まります。

Web サービスの管理には、次のタスクが関係します。

管理する Web サービスのタイプの選択
対象となる Web サービスの作成と管理

Web サービスは、SQL Anywhere データベースで作成し、格納できます。

Web サービスタイプ

Web ブラウザまたはクライアントアプリケーションから SAP Sybase IQ Web サービスに対して Web サービス要求を行うと、要求が処理され、結果セットが応答で返されます。SAP Sybase IQ では、結果セットのフォーマットと結果セットを返す方法を制御するいくつかの Web サービスタイプがサポートされています。適切な Web サービスタイプを選択してから、CREATE SERVICE 文または ALTER SERVICE 文の TYPE 句を使用して Web サービスタイプを指定します。

サポートされる Web サービスタイプは次のとおりです。

- **HTML** – 文、関数、またはプロシージャの結果セットは、テーブルが含まれる HTML ドキュメントにフォーマットされます。Web ブラウザに、HTML ドキュメントの本文が表示されます。
- **XML** – 文、関数、またはプロシージャの結果セットは、XML ドキュメントとして返されます。XML でフォーマットされていない結果セットは、自動的に XML にフォーマットされます。Web ブラウザに、タグと属性を含む未加工の XML コードが表示されます。

  XML フォーマットは、次の SQL 文の例に示すように、SELECT 文で FOR XML RAW 句を指定することと同じです。

  ```
  SELECT * FROM table-name FOR XML RAW
  ```

- **RAW** – 文、関数、またはプロシージャの結果セットは、自動フォーマットなしで返されます。

  このサービスタイプでは、結果セットを最大限に制御できます。ただし、ストアドプロシージャ内で必要なマークアップ (HTML、XML) を明示的に作成することによって応答を生成する必要があります。SA_SET_HTTP_HEADER システムプロシージャを使用して、MIME タイプを指定する HTTP Content-Type ヘッダを設定すると、Web ブラウザで結果セットを正しく表示できます。
- **JSON** – 文、関数、またはプロシージャの結果セットは、JSON (JavaScript Object Notation) で返されます。JavaScript Object Notation (JSON) は、言語に依存しないテキストベースのデータ交換フォーマットで、JavaScript データの直列化のために開発されました。JSON には、次の 4 つの基本型があります。文字列、数値、ブール、および NULL です。また、JSON には、次の 2 つの構造化型もあります。オブジェクトと配列です。JSON の詳細については、http://www.json.org/ を参照してください。

  このサービスは、Web アプリケーションに対して HTTP 呼び出しを行うために AJAX で使用されます。JSON タイプの例については、%ALLUSERSPROFILE%

¥SybaseIQ¥samples¥SQLAnywhere¥HTTP¥json_sample.sql を参照してください。

- **SOAP** – 文、関数、またはプロシージャの結果セットは、SOAP 応答として返されます。SOAP サービスには、SOAP をサポートする異なるクライアントアプリケーションに対してデータアクセスを提供する、共通データ交換標準が備えられています。SOAP 要求と応答のエンベロープは、HTTP (SOAP over HTTP) を使用して XML ペイロードとして転送されます。SOAP サービスへの要求は、汎用 HTTP 要求ではなく有効な SOAP 要求である必要があります。SOAP サービスの出力は、CREATE 文または ALTER SERVICE 文の FORMAT 属性と DATATYPE 属性を使用することで調節できます。
- **DISH** – DISH サービス (SOAP ハンドラを決定) は、SAP Sybase IQ の SOAP 終了ポイントです。DISH サービスからアクセスできるすべての SOAP 操作 (SAP Sybase IQ SOAP サービス) を記述する WSDL (Web Services Description Language) ドキュメントを公開します。SOAP クライアントツールキットは、WSDL に基づいたインタフェースを使用してクライアントアプリケーションを構築します。SOAP クライアントアプリケーションは、すべての SOAP 要求を SOAP 終了ポイント (SAP Sybase IQ DISH サービス) にリダイレクトします。

例

次の例は、RAW サービスタイプを使用した汎用 HTTP Web サービスの作成を示しています。

```
CREATE PROCEDURE sp_echotext(str LONG VARCHAR)
BEGIN
    CALL sa_set_http_header( 'Content-Type', 'text/plain' );
    SELECT str;
END;

CREATE SERVICE SampleWebService
    TYPE 'RAW'
    AUTHORIZATION OFF
    USER DBA
    AS CALL sp_echotext ( :str );
```

Web サービスの管理

Web サービスの管理には、次のタスクが関係します。

- **Web サービスの作成または変更** – Web ブラウザインタフェースをサポートする Web アプリケーションを提供し、REST および SOAP 手法を使用して Web を介したデータ交換を提供するために、Web サービスを作成または変更します。
- **Web サービスの削除** – Web サービスを削除すると、そのサービスに対して作成された後続の要求によって 404 Not Found HTTP ステータスメッセージが返されます。root Web サービスが存在する場合、すべての未解決の要求 (意図的なまたは意図的でない) が処理されます。

- **Web サービスのコメント付け** – コメント付けはオプションであり、Web サービスに関するドキュメントを提供できます。
- **root Web サービスの作成とカスタマイズ** – `root` Web サービスを作成して、他の Web サービス要求と一致しない HTTP 要求を処理できます。
- **Web サービスの有効化と無効化** – 無効化された Web サービスからは、`404 Not Found` HTTP ステータスメッセージが返されます。METHOD 句は、特定の Web サービスに対して呼び出しできる HTTP メソッドを指定します。

Web サービスを作成または変更する方法

Web サービスの作成または変更には、CREATE SERVICE 文または ALTER SERVICE 文をそれぞれ使用する必要があります。この項では、これらの文を Interactive SQL で実行して異なるタイプの Web サービスを作成する方法について説明します。この項の例では、次のコマンドを使用して、Interactive SQL 経由で SAP Sybase IQ データベース *your-database.db* に接続済みであることが前提となります。

```
dbisql -c "dbf=your-database.db;uid=your-userid;pwd=your-password"
```

HTTP Web サービスを作成する方法

HTTP Web サービスは、HTML、XML、または RAW に分類されます。すべての HTTP Web サービスは、CREATE SERVICE 文および ALTER SERVICE 文と同じ構文を使用して作成、変更できます。

例

Interactive SQL で次の文を実行して、HTTP Web サーバでサンプルの汎用 HTTP Web サービスを作成します。

```
CREATE SERVICE SampleWebService
    TYPE 'web-service-type-clause'
    URL OFF
    USER DBA
    AUTHORIZATION OFF
    AS sql-statement;
```

CREATE SERVICE 文は、`SampleWebService` という新しい Web サービスを作成し、*sql-statement* の結果セットを返します。*sql-statement* を SELECT 文に置き換えてテーブルまたはビューからデータを直接選択するか、CALL 文に置き換えてデータベース内のストアドプロシージャを呼び出すことができます。

web-service-type-clause を目的の Web サービスタイプに置き換えます。HTTP Web サービスに有効な句には、`HTML`、`XML`、`RAW`、`JSON` があります。

Web ブラウザで `SampleWebService` サービスにアクセスすることで、このサービスに対して生成された結果セットを表示できます。

SOAP over HTTP サービスを作成する方法
SOAP は、多くの開発環境でサポートされているデータ交換標準です。

SOAP ペイロードは、SOAP エンベロープと呼ばれる XML ドキュメントで構成されます。SOAP 要求エンベロープには、SOAP 操作 (SOAP サービス) が含まれ、適切なパラメータをすべて指定します。SOAP サービスは、要求を解析してパラメータを取得し、他のサービスと同様にストアドプロシージャまたはファンクションの呼び出しや選択を行います。SOAP サービスの表示レイヤは、DISH サービスの WSDL で指定された事前定義のフォーマットで、SOAP エンベロープ内のクライアントに結果セットのストリームを返します。SOAP 規格の詳細については、http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ を参照してください。

デフォルトでは、SOAP サービスのパラメータと結果セットは XmlSchema 文字列型のパラメータになります。DATATYPE ON は、入力パラメータと応答データで TRUE 型を使用することを指定します。DATATYPE を指定すると、WSDL 仕様が適宜変更され、クライアント SOAP ツールキットで適切な型のパラメータと応答オブジェクトを使用してインタフェースが生成されるようになります。

FORMAT 句は、異なる機能で特定の SOAP ツールキットを対象とするために使用します。DNET には、SOAP サービス応答を System.Data.DataSet オブジェクトとして処理する Microsoft .NET クライアントアプリケーションが備えられています。CONCRETE では、.NET や Java などのオブジェクト指向アプリケーションで、ローとカラムをパッケージ化する応答オブジェクトを生成できるようにする、より汎用的な構造が公開されます。XML からは、応答全体が XML ドキュメントとして返され、文字列として公開されます。クライアントは、XML パーサを使用して、データをさらに処理できます。CREATE SERVICE 文の FORMAT 句では、複数のクライアントアプリケーションタイプがサポートされています。

注意： DATATYPE 句は、SOAP サービスにのみ関係します (HTML でのデータ入力はありません)。FORMAT 句は、SOAP サービスまたは DISH サービスのいずれかで指定できます。SOAP サービスの FORMAT 指定は、DISH サービスの指定よりも優先されます。

例

Interactive SQL で次の文を実行して、SOAP over HTTP サービスを作成します。

```
CREATE SERVICE SampleSOAPService
    TYPE 'SOAP'
    DATATYPE ON
    FORMAT 'CONCRETE'
    USER DBA
    AUTHORIZATION OFF
    AS sql-statement;
```

DISH サービスを作成する方法

SAP Sybase IQ では、SOAP サービスのグループに対して SOAP 終了ポイントとして動作する DISH サービスを作成できます。また、DISH サービスでは、WSDL (Web Services Description Language) ドキュメントも自動的に構築され、WSDL で記述される SOAP サービスとのデータ交換に必要なインタフェースを SOAP クライアントツールキットで生成できるようになります。

SOAP over HTTP サービスの現在のワーキングセットは常に公開されるため、SOAP サービスの追加と削除には、DISH サービスの管理は必要ありません。

例

Interactive SQL で次の SQL 文を実行して、HTTP Web サーバでサンプルの SOAP サービスと DISH サービスを作成します。

```
CREATE SERVICE "Samples/TestSoapOp"
    TYPE 'SOAP'
    DATATYPE ON
    USER DBA
    AUTHORIZATION OFF
    AS CALL sp_echo(:i, :f, :s);

CREATE PROCEDURE sp_echo(i INTEGER, f REAL, s LONG VARCHAR)
RESULT( ret_i INTEGER, ret_f REAL, ret_s LONG VARCHAR )
BEGIN
    SELECT i, f, s;
END;

CREATE SERVICE "dnet_endpoint"
    TYPE 'DISH'
    GROUP "Samples"
    FORMAT 'DNET';
```

最初の CREATE SERVICE 文は、`Samples/TestSoapOp` という新しい SOAP サービスを作成します。

2 番目の CREATE SERVICE 文は、`dnet_endpoint` という新しい DISH サービスを作成します。GROUP 句の `Samples` 部分は、公開する SOAP サービスのグループを識別します。DISH サービスによって生成される WSDL ドキュメントを表示できます。SAP Sybase IQ Web サーバがコンピュータで実行中の場合、`http://localhost:`*`port-number`*`/dnet_endpoint` URL を使用してサービスにアクセスできます。*port-number* は、サーバが実行されているポート番号です。

この例では、SOAP 応答フォーマットを示す FORMAT 句は、SOAP サービスに含まれていません。したがって、SOAP 応答フォーマットは DISH サービスで指定され、SOAP サービスの FORMAT 句は上書きされません。この機能により、各 DISH

終了ポイントが異なる機能のサービスを SOAP クライアントに提供する、同種の DISH サービスを作成できます。

同種の DISH サービスの作成
SOAP サービス定義が DISH サービスに対する FORMAT 句の指定と異なる場合、フォーマットを定義する DISH サービス内で SOAP サービスのセットをグループ化することができます。このようにすると、異なる FORMAT 指定を使用して、複数の DISH サービスで同じグループの SOAP サービスを公開できます。`TestSoapOp` の例を展開すると、次の SQL 文を使用して `java_endpoint` という別の DISH サービスを作成できます。

```
CREATE SERVICE "java_endpoint"
    TYPE 'DISH'
    GROUP "Samples"
    FORMAT 'CONCRETE';
```

この例では、SOAP クライアントが `java_endpoint` DISH サービスを介して `TestSoapOp` 操作用の Web サービス要求を行うと、SOAP クライアントは `TestSoapOp_Dataset` という応答オブジェクトを受信します。WSDL を調べて、`dnet_endpoint` と `java_endpoint` の違いを比較できます。この手法を使用して、特定の SOAP クライアントツールキットの条件を満たす SOAP 終了ポイントをすばやく構築できます。

Web サービスを削除する方法
Web サービスを削除すると、そのサービスに対して作成された後続の要求によって `404 Not Found` HTTP ステータスメッセージが返されます。`root` Web サービスが存在する場合、すべての未解決の要求 (意図的なまたは意図的でない) が処理されます。

例

次の SQL 文を実行して、`SampleWebService` という名前の Web サービスを削除します。

```
DROP SERVICE SampleWebService;
```

Web サービスにコメントを付ける方法
Web サービスのドキュメントを提供するには、COMMENT ON SERVICE 文を使用する必要があります。コメントは、*statement* 句を NULL に設定すると削除できます。

例

たとえば、次の SQL 文を実行して、`SampleWebService` という Web サービスに新しいコメントを作成します。

```
COMMENT ON SERVICE SampleWebService
    IS "This is a comment on my web service.";
```

root Web サービスを作成し、カスタマイズする方法

どの Web サービス要求とも一致しない HTTP クライアント要求は、root Web サービスが定義されていると、root Web サービスによって処理されます。

root Web サービスには、任意の HTTP 要求 (アプリケーションを構築する時点で URL が判明している必要はない) や認識されない要求を処理する、簡単で柔軟な方法が備えられています。

例

この例では、データベース内のテーブルに格納されている root Web サービスを使用して、Web ブラウザや他の HTTP クライアントにコンテンツを提供する方法を示します。ポート 80 で受信するローカルの HTTP Web サーバを単一のデータベースで起動していることが前提となります。すべてのスクリプトは、Web サーバで実行されます。

Interactive SQL を介してデータベースサーバに接続し、次の SQL 文を実行して、クライアントで指定された url ホスト変数を PageContent というプロシージャに渡す root Web サービスを作成します。

```
CREATE SERVICE root
    TYPE 'RAW'
    AUTHORIZATION OFF
    SECURE OFF
    URL ON
    USER DBA
    AS CALL PageContent(:url);
```

URL ON 部分は、URL という名前の HTTP 変数からフルパスコンポーネントにアクセスできることを指定します。

次の SQL 文を実行して、ページコンテンツの格納に使用するテーブルを作成します。この例では、ページコンテンツは、その URL、MIME タイプ、コンテンツ自体によって定義されます。

```
CREATE TABLE Page_Content (
    url          VARCHAR(1024) NOT NULL PRIMARY KEY,
    content_type VARCHAR(128)  NOT NULL,
    image        LONG VARCHAR  NOT NULL
);
```

次の SQL 文を実行して、テーブルを移植します。この例では、index.html ページが要求されたときに HTTP クライアントに提供するコンテンツを定義します。

```
INSERT INTO Page_Content
VALUES(
    'index.html',
    'text/html',
```

```
    '<html><body><h1>Hello World</h1></body></html>'
);
COMMIT;
```

次の SQL 文を実行して、root Web サービスに渡される url ホスト変数を受け入れる PageContent プロシージャを実装します。

```
CREATE PROCEDURE PageContent(IN @url LONG VARCHAR)
RESULT ( html_doc LONG VARCHAR )
BEGIN
    DECLARE @status CHAR(3);
    DECLARE @type   VARCHAR(128);
    DECLARE @image  LONG VARCHAR;

    SELECT content_type, image INTO @type, @image
        FROM Page_Content
        WHERE url = @url;

    IF @image is NULL THEN
        SET @status = '404';
        SET @type = 'text/html';
        SET @image =  '<html><body><h1>404 - Page Not Found</h1>'
            || '<p>There is no content located at the URL "'
            || html_encode( @url ) || '" on this server.<p>'
            || '</body></html>';
    ELSE
        SET @status = '200';
    END IF;
    CALL sa_set_http_header( '@HttpStatus', @status );
    CALL sa_set_http_header( 'Content-Type', @type );
    SELECT @image;
END;
```

HTTP サーバへの要求が定義済みの他の Web サービスの URL と一致しない場合、root Web サービスは PageContent プロシージャを呼び出します。クライアントで指定された URL が Page_Content テーブルの url と一致するかどうかが、プロシージャによってチェックされます。SELECT 文は、応答をクライアントに送信します。クライアントで指定された URL がテーブルで検出されないと、汎用 404 - Page Not Found html ページが作成され、クライアントに送信されます。

ブラウザの中には、ブラウザ自体のページで 404 ステータスに対応するものもあるので、汎用ページが表示される保証はありません。

エラーメッセージでは、HTML_ENCODE 関数を使用して、クライアントで指定された URL 内の特殊文字がエンコードされます。

@HttpStatus ヘッダは、要求で返されるステータスコードを設定するために使用します。404 ステータスは Not Found エラーを示し、200 ステータスは OK を示します。'Content-Type' ヘッダは、要求で返されるコンテンツタイプを設定するために

使用します。この例では、index.html ページのコンテンツ (MIME) タイプは text/html です。

Web サービスの SQL 文

次の SQL 文を使用して、Web サービスの開発を支援できます。

Web サーバ関連の SQL 文	説明
CREATE SERVICE 文 [HTTP Web サービス]	新しい HTTP Web サービスを作成する。
CREATE SERVICE 文 [SOAP Web サービス]	HTTP または DISH サービス経由の新しい SOAP を作成する。
ALTER SERVICE 文 [HTTP Web サービス]	既存の HTTP Web サービスを変更する。
ALTER SERVICE 文 [SOAP Web サービス]	HTTP または DISH サービス経由の既存の SOAP を変更する。

COMMENT 文	データベースオブジェクトに対するコメントをシステムテーブルに格納する。
	Web サービスにコメントを付けるには、次の構文を使用する。
	`COMMENT ON SERVICE 'web-service-name'`
	`    IS 'your comments'`

DROP SERVICE 文	Web サービスを削除する。

Web サービスの接続プーリング

Web サービスを公開する各データベースでは、データベース接続のプールにアクセスできます。特定の USER 句で定義されるすべてのサービスで同じ接続プールグループを共有するように、プールはユーザ名ごとにグループ化されます。

クエリを初めて実行するサービス要求は、最適化フェーズを経由して実行プランを確立する必要があります。プランのキャッシュと再使用が可能な場合、以降の実行では最適化フェーズを省略できます。HTTP 接続プーリングでは、プランを可能なかぎり再使用することによって、データベース接続におけるプランのキャッシュが活用されます。各サービスでは、再使用を最適化するために独自の接続リストが管理されます。ただし、負荷のピーク時には、サービスは同じユーザグループの接続のうち使用率の最も低い接続を横取りすることがあります。

いくつかのサービスの実行に対してパフォーマンスを最適化できるキャッシュされたプランが、一定期間に特定の接続で取得されることがあります。

接続プールでは、特定のサービスに対する HTTP 要求は、データベース接続をプールから取得しようとします。HTTP セッションと異なり、プールされた接続は、接続スコープ変数やテンポラリテーブルなどの接続スコープ環境をリセットすることによって削除されます。

HTTP 接続プール内にプールされたデータベース接続は、ライセンス目的に使用される接続としてはカウントされません。プールから取得された接続は、ライセンスされた接続としてカウントされます。プールから接続を取得するときに HTTP 要求がライセンスの制限を越えると、`503 Service Temporarily Unavailable` ステータスが返されます。

接続プールは、Web サービスが AUTHORIZATION OFF を指定して定義されている場合にのみ利用できます。

プール内のデータベース接続は、データベースと接続のオプションが変更されても更新されません。

HTTP Web サーバで Web サービスアプリケーションを開発する方法

この項では、Web ページの作成とカスタマイズの概要について説明します。HTTP Web サーバ用のストアドプロシージャを開発する方法について説明します。SAP Sybase IQ HTTP Web サーバの起動方法と、ストアドプロシージャを呼び出す Web サービスの作成方法に関する知識があることが前提となります。

Web サービスアプリケーションの例の詳細については、`%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥HTTP` ディレクトリを参照してください。

Web ページをカスタマイズする方法

Web ページをカスタマイズするには、まず、HTTP Web サーバから呼び出される Web サービスのフォーマットを評価する必要があります。たとえば、Web サービスで `HTML` タイプを指定すると、Web ページは HTML でフォーマットされます。

`RAW` Web サービスタイプは、HTML や XML などの必要なマークアップを提供するために Web サービスのプロシージャと関数を明示的にコーディングする必要があるため、最もカスタマイズしやすいタイプです。`RAW` タイプを使用する場合に Web ページをカスタマイズするには、次のタスクを実行してください。

- HTTP コンテンツタイプヘッダフィールドを、呼び出されるストアドプロシージャで、text/html などの適切な MIME タイプに設定する。
- 呼び出されるストアドプロシージャから Web ページ出力を生成するときに、MIME タイプに適切なマークアップを適用する。

例

次の例は、`RAW` タイプを指定して、新しい Web サービスを作成する方法を示しています。

```
CREATE SERVICE WebServiceName
    TYPE 'RAW'
    AUTHORIZATION OFF
    URL ON
    USER DBA
    AS CALL HomePage( :url );
```

この例では、Web サービスが HomePage ストアドプロシージャを呼び出します。これには、URL の PATH コンポーネントを受信する 1 つの URL パラメータを定義する必要があります。

コンテンツタイプヘッダフィールドの設定

Web ブラウザでコンテンツを正しく表示するには、sa_set_http_header システムプロシージャを使用して HTTP コンテンツタイプヘッダを定義します。

次の例は、sa_set_http_header システムプロシージャで text/html MIME タイプを使用して、Web ページ出力を HTML でフォーマットする方法を示しています。

```
CREATE PROCEDURE HomePage (IN url LONG VARCHAR)
    RESULT (html_doc XML)
    BEGIN
        CALL sa_set_http_header ( 'Content-Type', 'text/html' );
        -- Your SQL code goes here.
        ...
    END
```

MIME タイプのタグ付け規則の適用

ストアドプロシージャの Content-Type ヘッダで指定された MIME タイプのタグ付け規則を適用する必要があります。SAP Sybase IQ には、タグを作成するための関数がいくつか備えられています。

次の例は、XMLCONCAT 関数と XMLELEMENT 関数を使用して HTML コンテンツを作成する方法を示しています。sa_set_http_header システムプロシージャを使用して、Content-Type ヘッダを text/html MIME タイプに設定していることが前提となります。

```
XMLCONCAT(
   CAST('<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">' AS XML),
    XMLELEMENT(
         'HTML',
         XMLELEMENT(
             'HEAD',
             XMLELEMENT('TITLE', 'My Home Page')
         ),
         XMLELEMENT(
             'BODY',
             XMLELEMENT('H1', 'My home on the web'),
             XMLELEMENT('P', 'Thank you for visiting my web site!')
         )
    )
)
```

データ型が XML ではない場合、要素内容は必ずエスケープされるので、前述の例は CAST 関数を使用します。それ以外の場合、特殊文字はエスケープされます (例：< には <)。

クライアント指定の HTTP 変数とヘッダにアクセスする方法

HTTP クライアント要求の変数とヘッダには、次のいずれかの方法でアクセスできます。

- ストアド関数またはストアドプロシージャ呼び出しのホストパラメータとして変数とヘッダを渡す、Web サービス文の宣言。
- ストアド関数またはストアドプロシージャでの HTTP_VARIABLE、NEXT_HTTP_VARIABLE、HTTP_HEADER、NEXT_HTTP_HEADER 関数の呼び出し。

ホストパラメータを使用して HTTP 変数にアクセスする方法

関数またはプロシージャ呼び出しのホストパラメータとしてクライアント指定の変数を渡すと、クライアント指定の変数を参照できます。

例

次の例は、ShowTable という Web サービスで使用されるホストパラメータにアクセスする方法を示しています。

```
CREATE SERVICE ShowTable
    TYPE 'RAW'
    AUTHORIZATION ON
    AS CALL ShowTable( :user_name, :table_name );

CREATE PROCEDURE ShowTable(IN username VARCHAR(128), IN tblname
VARCHAR(128))
BEGIN
    -- write SQL code utilizing the username and tblname variables
here.
END;
```

サービスのホストパラメータは、プロシージャパラメータの宣言順にマッピングされます。上記の例では、user_name と table_name の各ホストパラメータが、username と tblname パラメータにそれぞれマッピングされます。

Web サービス関数を使用して HTTP 変数とヘッダにアクセスする方法
HTTP_VARIABLE、NEXT_HTTP_VARIABLE、HTTP_HEADER、NEXT_HTTP_HEADER 関数は、クライアントが指定した変数とヘッダに対して反復するために使用できます。

HTTP_VARIABLE と HTTP_NEXT_VARIABLE を使用した変数へのアクセス
ストアドプロシージャ内で NEXT_HTTP_VARIABLE と HTTP_VARIABLE 関数を使用して、クライアントが指定したすべての変数に対して反復処理を実行できます。

HTTP_VARIABLE 関数では、変数名の値を取得できます。

NEXT_HTTP_VARIABLE 関数を使用すると、クライアントから送信されたすべての変数に対する反復処理を実行できます。最初の変数名を取得するために関数を初めて呼び出す場合は、NULL 値を渡します。返された変数名を HTTP_VARIABLE 関数呼び出しのパラメータとして使用して、変数の値を取得します。前の変数名を next_http_variable 呼び出しで渡すと、次の変数名が取得されます。最後の変数名が渡されると、NULL が返されます。

変数名に対する反復処理を行うことによって各変数名が確実に 1 回のみ返されますが、変数名の順序はクライアント要求での順序とは異なる場合があります。

次の例は、ShowDetail サービスにアクセスするクライアント要求で指定されたパラメータから値を検索する、HTTP_VARIABLE 関数の使用方法を示しています。

```
CREATE SERVICE ShowDetail
    TYPE 'HTML'
    URL PATH OFF
    AUTHORIZATION OFF
    USER DBA
    AS CALL ShowDetail();

CREATE PROCEDURE ShowDetail()
BEGIN
    DECLARE v_customer_id LONG VARCHAR;
    DECLARE v_product_id LONG VARCHAR;
    SET v_customer_id = HTTP_VARIABLE( 'customer_id' );
    SET v_product_id = HTTP_VARIABLE( 'product_id' );
    CALL ShowSalesOrderDetail( v_customer_id, v_product_id );
END;
```

次の例は、image 変数に関連するヘッダフィールド値から 3 つの属性を検索する方法を示しています。

```
SET v_name = HTTP_VARIABLE( 'image', NULL, 'Content-Disposition' );
SET v_type = HTTP_VARIABLE( 'image', NULL, 'Content-Type' );
SET v_image = HTTP_VARIABLE( 'image', NULL, '@BINARY' );
```

2 番目のパラメータに整数を指定すると、追加の値を検索できます。3 番目のパラメータでは、ヘッダフィールド値をマルチパート要求から検索できます。ヘッダフィールド名を指定してこの値を検索します。

HTTP_HEADER と NEXT_HTTP_HEADER を使用したヘッダへのアクセス
NEXT_HTTP_HEADER 関数と HTTP_HEADER 関数を使用して、HTTP 要求のヘッダを要求から取得できます。

HTTP_HEADER 関数は、指定された HTTP ヘッダフィールドの値を返します。

NEXT_HTTP_HEADER 関数は、HTTP ヘッダに対して反復され、次の HTTP ヘッダ名を返します。NULL を指定してこの関数を呼び出すと、最初のヘッダ名が返されます。後続のヘッダは、前のヘッダ名を関数に渡すことによって取得されます。最後のヘッダ名が呼び出されると、NULL が返されます。

次の表は、いくつかの共通の HTTP 要求ヘッダと典型的な値を示します。

ヘッダ名	ヘッダ値
Accept	image/gif、image/x-xbitmap、image/jpeg、image/pjpeg、application/x-shockwave-flash、application/vnd.ms-excel、application/vnd.ms-powerpoint、application/msword、*/*
Accept-Language	en-us
Accept-Charset	utf-8、iso-8859-5;q=0.8
Accept-Encoding	gzip、deflate
User-Agent	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.2; WOW64; SV1; .NET CLR 2.0.50727)
Host	localhost:8080
Connection	Keep-Alive

次の表は、特殊なヘッダと典型的な値を示します。

ヘッダ名	ヘッダ値
@HttpMethod	GET
@HttpURI	/demo/ShowHTTPHeaders
@HttpVersion	HTTP/1.1
@HttpQueryString	id=-123&version=109&lang=en

処理中の要求のステータスコードを設定するには、特別なヘッダ @HttpStatus を使用します。

次の例は、ヘッダの名前と値を HTML テーブルにフォーマットする方法を示しています。

`ShowHTTPHeaders` Web サービスを作成します。

```
CREATE SERVICE ShowHTTPHeaders
    TYPE 'RAW'
    AUTHORIZATION OFF
    USER DBA
    AS CALL HTTPHeaderExample();
```

NEXT_HTTP_HEADER 関数を使用する `HTTPHeaderExample` プロシージャを作成してヘッダ名を取得してから、HTTP_HEADER 関数を使用してその値を取得します。

```
CREATE PROCEDURE HTTPHeaderExample()
RESULT ( html_string LONG VARCHAR )
BEGIN
    declare header_name LONG VARCHAR;
    declare header_value LONG VARCHAR;
    declare header_query LONG VARCHAR;
    declare table_rows XML;
    set header_name = NULL;
    set table_rows = NULL;
header_loop:
    LOOP
        SET header_name = NEXT_HTTP_HEADER( header_name );
        IF header_name IS NULL THEN
            LEAVE header_loop
        END IF;
        SET header_value = HTTP_HEADER( header_name );
        SET header_query = HTTP_HEADER( '@HttpQueryString' );
        -- Format header name and value into an HTML table row
        SET table_rows = table_rows ||
            XMLELEMENT( name "tr",
                XMLATTRIBUTES( 'left' AS "align",
                               'top' AS "valign" ),
                XMLELEMENT( name "td", header_name ),
                XMLELEMENT( name "td", header_value ),
                XMLELEMENT( name "td", header_query ) );

    END LOOP;
    SELECT XMLELEMENT( name "table",
                XMLATTRIBUTES( '' AS "BORDER",
                            '10' AS "CELLPADDING",
                             '0' AS "CELLSPACING" ),
                XMLELEMENT( name "th",
                    XMLATTRIBUTES( 'left' AS "align",
                                   'top'  AS "valign" ),
                                   'Header Name' ),
                XMLELEMENT( name "th",
                    XMLATTRIBUTES( 'left' AS "align",
                                   'top'  AS "valign" ),
                                   'Header Value' ),
                XMLELEMENT( name "th",
```

```
                    XMLATTRIBUTES( 'left' AS "align",
                                   'top'  AS "valign" ),
                                   'HTTP Query String' ),
                table_rows );
END;
```

Web ブラウザで ShowHTTPHeaders にアクセスして、HTML テーブルに配置された要求ヘッダを確認します。

クライアント指定の SOAP 要求ヘッダにアクセスする方法

SOAP 要求のヘッダは、NEXT_SOAP_HEADER 関数と SOAP_HEADER 関数を組み合わせて使用することによって取得できます。

NEXT_SOAP_HEADER 関数は、SOAP 要求エンベロープに含まれる SOAP ヘッダに対して反復され、次の SOAP ヘッダ名を返します。NULL を指定して呼び出すと、最初のヘッダの名前が返されます。後続のヘッダは、NEXT_SOAP_HEADER 関数に前のヘッダの名前を渡すことによって取得されます。最後のヘッダの名前を指定して呼び出すと、NULL が返されます。

次の例は、SOAP ヘッダの取得を示しています。

```
SET hd_key = NEXT_SOAP_HEADER( hd_key );
    IF hd_key IS NULL THEN
      -- no more header entries
      LEAVE header_loop;
    END IF;
```

この関数を繰り返し呼び出すと、すべてのヘッダフィールドが一度だけ返されます。ただし、必ずしも SOAP 要求での表示順に表示されるとはかぎりません。

SOAP_HEADER 関数は、名前付きの SOAP ヘッダフィールドの値を返します。SOAP サービスから呼び出されていない場合は NULL を返します。Web サービスを介して SOAP 要求を処理する場合に使用します。指定したフィールド名のヘッダが存在しない場合、戻り値は NULL です。

この例は、Authentication という SOAP ヘッダを探します。このヘッダが見つかると、SOAP ヘッダ全体の値を抽出し、さらに @namespace 属性と mustUnderstand 属性の値を抽出します。SOAP ヘッダの値は、次の XML 文字列のようになります。

```
<Authentication xmlns="CustomerOrderURN" mustUnderstand="1">
  <userName pwd="none">
    <first>John</first>
    <last>Smith</last>
  </userName>
</Authentication>
```

このヘッダの場合、@namespace 属性値は CustomerOrderURN になります。

また、mustUnderstand 属性値は 1 になります。

この XML 文字列の内部構造を、XPath 文字列に `/*:Authentication/*:userName` を設定した OPENXML 関数を使用して解析します。

```
SELECT * FROM OPENXML( hd_entry, xpath )
    WITH ( pwd LONG VARCHAR '@*:pwd',
        first_name LONG VARCHAR '*:first/text()',
        last_name LONG VARCHAR '*:last/text()' );
```

上記のサンプル SOAP ヘッダ値を使用した場合、SELECT 文は次のような結果セットを作成します。

pwd	first_name	last_name
none	John	Smith

この結果セットに対してカーソルが宣言され、3 つのカラム値が 3 つの変数にフェッチされます。この時点で、Web サービスに渡された関連性のある情報すべてを取得しています。

例

次の例は、パラメータを含む SOAP 要求と SOAP ヘッダが Web サーバでどのように処理されるかを示しています。この例では、次の 2 つのパラメータを取る `addItem` SOAP 操作を実行します。int 型の `amount` と文字列型の `item` です。`sp_addItems` プロシージャは、ユーザの姓と名を抽出する Authentication という SOAP ヘッダを処理します。値は sa_set_soap_header システムプロシージャで SOAP 応答検証ヘッダを移植するために使用されます。応答は、それぞれ INT、LONG VARCHAR、LONG VARCHAR 型の `quantity`、`item`、`status` の 3 つのカラムの結果です。

```
// create the SOAP service
CREATE SERVICE addItems
    TYPE 'SOAP'
    FORMAT 'CONCRETE'
    AUTHORIZATION OFF
    USER DBA
    AS CALL sp_addItems( :amount, :item );

// create SOAP endpoint for related services
CREATE SERVICE itemStore
    TYPE 'DISH'
    AUTHORIZATION OFF
    USER DBA;

// create the procedure that will process the SOAP requests for the
addItems service
CREATE PROCEDURE sp_addItems(count INT, item LONG VARCHAR)
RESULT(quantity INT, item LONG VARCHAR, status LONG VARCHAR)
BEGIN
    DECLARE hd_key LONG VARCHAR;
    DECLARE hd_entry LONG VARCHAR;
    DECLARE pwd LONG VARCHAR;
```

```
    DECLARE first_name LONG VARCHAR;
    DECLARE last_name LONG VARCHAR;
    DECLARE xpath LONG VARCHAR;
    DECLARE authinfo LONG VARCHAR;
    DECLARE namespace LONG VARCHAR;
    DECLARE mustUnderstand LONG VARCHAR;

  header_loop:
    LOOP
        SET hd_key = next_soap_header( hd_key );
        IF hd_key IS NULL THEN
            // no more header entries.
            leave header_loop;
        END IF;
        IF hd_key = 'Authentication' THEN
            SET hd_entry = soap_header( hd_key );
            SET xpath = '/*:' || hd_key || '/*:userName';
            SET namespace = soap_header( hd_key, 1, '@namespace' );
            SET mustUnderstand = soap_header( hd_key, 1,
'mustUnderstand' );
            BEGIN
                // parse for the pieces that you are interested in
                DECLARE crsr CURSOR FOR SELECT * FROM
                    OPENXML( hd_entry, xpath )
                        WITH ( pwd LONG VARCHAR '@*:pwd',
                           first_name LONG VARCHAR '*:first/text()',
                            last_name LONG VARCHAR '*:last/text()');
                OPEN crsr;
                FETCH crsr INTO pwd, first_name, last_name;
                CLOSE crsr;
            END;
           // build a response header, based on the pieces from the
request header
            SET authinfo = XMLELEMENT( 'Validation',
                XMLATTRIBUTES(
                    namespace as xmlns,
                    mustUnderstand as mustUnderstand ),
                    XMLELEMENT( 'first', first_name ),
                    XMLELEMENT( 'last', last_name ) );
            CALL sa_set_soap_header( 'authinfo', authinfo);
        END IF;
    END LOOP header_loop;
    //  code to validate user/session and check item goes here...
    SELECT count, item, 'available';
END;
```

HTTP サーバでの HTTP セッションの管理

Web アプリケーションでは、セッションをさまざまな方法でサポートできます。HTML フォーム内の非表示フィールドを使用して、複数の要求にまたがってクライアント／サーバデータを保持できます。別の方法として、AJAX 対応のクライアント側 JavaScript などの Web 2.0 手法では、クライアントステータスに基づいて非同期の HTTP 要求を行うことができます。SAP Sybase IQ には、セッション化さ

れた HTTP 要求を排他的に使用するために、データベース接続を保持する追加機能があります。

HTTP セッション内で作成または変更された接続スコープ変数とテンポラリテーブルには、特定の SessionID を指定する後続の HTTP 要求からアクセスできます。SessionID は、GET または POST HTTP 要求メソッドで指定するか、HTTP cookie ヘッダ内で指定できます。サーバは、SessionID 変数を指定した HTTP 要求を受信すると、一致するコンテキストをセッションレポジトリから検索します。セッションが検出されると、そのデータベース接続を使用して要求を処理します。セッションが使用中の場合は、HTTP 要求がキューに追加され、セッションが解放された時点で HTTP 要求をアクティブにします。

セッション ID の作成、削除、変更には、sa_set_http_option を使用できます。

HTTP セッションには、セッション条件を管理するために特殊な処理が必要です。1 つの SessionID で使用できるデータベース接続は 1 つしかないため、その SessionID に対する連続したクライアント要求は、サーバによってシリアル化されます。特定の SessionID に対して、16 個までの要求をキューに追加できます。セッションキューが満杯になると、その SessionID に対する後続の要求は `503 Service Unavailable` ステータスで拒否されます。

SessionID を初めて作成すると、SessionID がシステムによって即時に登録されます。SessionID を変更または削除する後続の要求は、HTTP 要求が終了した時点でのみ適用されます。この方法を使用すると、要求の処理がロールバックで終了する場合や、アプリケーションで SessionID を削除してリセットする場合に、動作の一貫性を維持できます。

HTTP 要求によって SessionID が変更されると、現在のセッションは削除され、保留中のセッションに置き換えられます。セッションによってキャッシュされたデータベース接続は、効率的に新しいセッションコンテキストに移動され、テンポラリテーブルや作成された変数などのステータスデータはすべて保持されます。

HTTP セッションの詳しい使用例については、`%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥HTTP¥session.sql` を参照してください。

注意： 各クライアントアプリケーション接続ではライセンスシートが保持されるため、未処理の接続数を最小限に抑えるために、古いセッションを削除し、適切なタイムアウトを設定する必要があります。HTTP セッションに関連する接続では、接続期間を通じてサーバデータベースでライセンスシートの保持が管理されます。

ライセンスの詳細については、http://www.sybase.com/detail?id=1056242 を参照してください。

HTTP セッションを作成する方法

セッションは、sa_set_http_option システムプロシージャで SessionID オプションを使用して作成できます。セッション ID の定義には、NULL 以外の任意の文字列を使用できます。

セッションステータス管理は、URL と cookie によってサポートされています。HTTP セッションには、HTTP cookie または GET 要求の URL を使用してアクセスでき、POST (x-www-form-urlencoded) 要求の本文からもアクセスできます。たとえば、次の URL では、実行時に XYZ データベース接続が使用されます。

```
http://localhost/sa_svc?SESSIONID=XYZ
```

XYZ データベース接続が存在しない場合は、標準のセッションレス要求として処理されます。

例

次のコードは、セッションの作成と削除を行う RAW Web サービスを作成する方法を示しています。有効な SessionID を指定して HTTP 要求を行うたびに、request_count という接続スコープ変数が増分されます。

```
CREATE SERVICE mysession
    TYPE 'RAW'
    AUTHORIZATION OFF
    USER DBA
    AS CALL mysession_proc();

CREATE PROCEDURE mysession_proc()
BEGIN
    DECLARE body LONG VARCHAR;
    DECLARE hostname LONG VARCHAR;
    DECLARE svcname LONG VARCHAR;
    DECLARE sesid LONG VARCHAR;

    CALL sa_set_http_header ( 'Content-Type', 'text/html' );
    SELECT CONNECTION_PROPERTY('SessionID') INTO sesid;
    SELECT CONNECTION_PROPERTY('HttpServiceName') INTO svcname;
    SELECT HTTP_HEADER( 'Host' ) INTO hostname;
    IF HTTP_VARIABLE('delete') IS NOT NULL THEN
        CALL sa_set_http_option( 'SessionID', NULL );
        SET body = '<html><body>Deleted ' || sesid
          || '</BR><a href="http://' || hostname || '/' || svcname ||
'">Start Again</a>';
        SELECT body;
    END IF;
    IF sesid = '' THEN
        SET sesid = set_session_url();
        CREATE VARIABLE request_count INT;
        SET request_count = 0;

        SET body = '<html><body> Created session ID ' || sesid
```

```
            || '</br><a href="http://' || hostname || '/' || svcname
            || '?SessionID=' || sesid || '"> Enter into Session</a>';
    ELSE
        SELECT CONNECTION_PROPERTY('SessionID') INTO sesid;
        SET request_count = request_count +1;
        SET body = '<html><body>Session ' || sesid || '</br>'
          || 'created ' || CONNECTION_PROPERTY('SessionCreateTime')
|| '</br>'
            || 'last access ' ||
CONNECTION_PROPERTY('SessionLastTime') || '</br>'
            || 'connection ID ' || CONNECTION_PROPERTY('Number') ||
'</br>'
          || '<h3>REQUEST COUNT is '|| request_count || '</h3><hr></
br>'
            || '<a href="http://' || hostname || '/' || svcname
            || '?SessionID=' || sesid || '">Enter into Session</a></
br>'
            || '<a href="http://' || hostname || '/' || svcname
            || '?SessionID=' || sesid || '&delete">Delete Session</
a>';
    END IF;

    SELECT body;
END;
```

URL を使用してセッションを管理する方法

URL セッションステータス管理システムでは、クライアントアプリケーションまたは Web ブラウザにより URL 内でセッション ID が指定されます。

例

次の例は、セッション ID を URL でのみ指定できる場合に、HTTP Web サーバの SQL 関数内でユニークなセッション ID を作成する方法を示しています。

```
CREATE FUNCTION set_session_url()
RETURNS LONG VARCHAR
BEGIN
    DECLARE session_id LONG VARCHAR;
    DECLARE tm TIMESTAMP;
    SET tm = NOW(*);
    SET session_id = 'session_' ||
       CONVERT( VARCHAR, SECONDS(tm) * 1000 + DATEPART( MILLISECOND,
tm ) );
    CALL sa_set_http_option( 'SessionID', session_id );
    SELECT CONNECTION_PROPERTY( 'SessionID' ) INTO session_id;
    RETURN( session_id );
END;
```

接続に対して session_id が定義されていないと、接続がセッションレスとなり、SessionID は空の文字列で示されます。

別の HTTP 要求が session_id を所有していると、sa_set_http_option システムプロシージャからエラーが返されます。

cookie を使用してセッションを管理する方法
cookie セッションステータス管理システムでは、クライアントアプリケーションまたは Web ブラウザにより、URL ではなく HTTP cookie ヘッダ内でセッション ID が指定されます。cookie によるセッション管理は、sa_set_http_header システムプロシージャに 'Set-Cookie' HTTP 要求を指定することでサポートされます。

注意： クライアントアプリケーションまたは Web ブラウザで cookie を無効にできる場合、cookie ステータス管理に依存することはできません。URL と cookie の両方のステータス管理をサポートすることをおすすめします。URL で指定されたセッション ID は、セッション ID が URL と cookie の両方で指定された場合に使用されます。

例

次の例は、セッション ID を URL または cookie で指定できる場合に、HTTP Web サーバの SQL 関数内でユニークなセッション ID を作成する方法を示しています。

```
CREATE FUNCTION set_session_cookie()
RETURNS LONG VARCHAR
BEGIN
    DECLARE session_id LONG VARCHAR;
    DECLARE tm TIMESTAMP;
    SET tm = NOW(*);
    SET session_id = 'session_' ||
       CONVERT( VARCHAR, SECONDS(tm) * 1000 + DATEPART( MILLISECOND,
tm ) );
    CALL sa_set_http_option( 'SessionID', session_id );
    CALL sa_set_http_header( 'Set-Cookie',
        'sessionid=' || session_id || ';' ||
        'max-age=60;' ||
        'path=/session;' );
    SELECT CONNECTION_PROPERTY( 'SessionID' ) INTO session_id;
    RETURN( session_id );
END;
```

非アクティブな HTTP セッションを検出する方法
SessionCreateTime および SessionLastTime 接続プロパティを使用して、現在の接続がセッションコンテキスト内にあるかどうかを確認できます。いずれかの接続プロパティクエリから空の文字列が返された場合、HTTP 要求はセッションコンテキスト内で実行されていません。

SessionCreateTime 接続プロパティは、指定したセッションがいつ作成されたかを確認する基準になります。このプロパティは、SessionID を確立するために sa_set_http_option システムプロシージャを呼び出したときに、初めて定義されます。

SessionLastTime 接続プロパティは、最後に処理されたセッション要求が、その前の要求の終了時にデータベース接続を解放した時刻を示します。セッションが初めて作成されたときから、そのセッションを作成した要求が接続を解放するまでの間は、このプロパティに空の文字列が返されます。

注意： セッションのタイムアウト期間は、http_session_timeout オプションを使用して調整できます。

例

次の例は、SessionCreateTime と SessionLastTime 接続プロパティを使用したセッションの検出を示しています。

```
SELECT CONNECTION_PROPERTY( 'sessioncreatetime' ) INTO ses_create;
SELECT CONNECTION_PROPERTY( 'sessionlasttime' ) INTO ses_last;
```

HTTP セッションを削除するかセッション ID を変更する方法

セッションコンテキスト内でキャッシュされているデータベース接続を明示的に切断すると、セッションが削除されます。この方法によるセッションの削除はキャンセル操作と見なされ、そのセッションキューから解放された要求はすべてキャンセルステータスになります。この操作によって、セッションキューで待ち状態になっている未処理の要求は、すべて確実に終了します。同様に、サーバまたはデータベースが停止した場合も、データベース接続がすべてキャンセルされます。

セッションを削除するには、sa_set_http_option システムプロシージャで SessionID オプションを NULL または空の文字列に設定します。

次のコードは、セッションの削除に使用できます。

```
CALL sa_set_http_option( 'SessionID', null );
```

HTTP セッションを削除するか SessionID を変更すると、セッションキューで待ち状態になっている保留中の HTTP 要求が解放され、セッションコンテキスト外で実行できるようになります。保留中の要求では、同じデータベース接続は再使用されません。

セッション ID を既存のセッション ID に設定することはできません。SessionID を変更すると、古い SessionID を参照する保留中の要求は解放され、セッションレス要求として実行されます。新しい SessionID を参照する後続の要求では、古い SessionID によってインスタンス化された同じデータベース接続が再使用されます。

HTTP セッションを削除または変更すると、次の状態になります。

- セッションに属するデータベース接続が取得されたセッションが、現在の要求に継承されているかどうか、または、セッションレス要求によって新しいセッションがインスタンス化されたかどうかによって、動作が異なります。要求がセッションレスとして開始された場合は、セッションの作成または削除操作が

即座に行われます。要求がセッションを継承している場合は、セッションの削除や SessionID の変更などのセッションステータスに関する変更は、要求が終了し変更内容がコミットされた後でのみ発生します。この動作の違いによって、1 つのクライアントで同じ SessionID を使用して同時要求を行ったとき、異常事態が発生する可能性があります。このような場合の対処を検討する必要があります。

- セッションを (保留中のセッションがない) 現在のセッションの SessionID に変更することは、エラーではなく、影響はありません。
- セッションを別の HTTP 要求によって使用されている SessionID に変更することは、エラーになります。
- 変更がすでに保留中のときにセッションを変更すると、保留中のセッションが削除され、新しい保留中のセッションが作成されます。保留中のセッションは、要求が正常に終了した場合にのみアクティブ化されます。
- 保留中のセッションがあるセッションを元の SessionID に戻すと、現在のセッションには変更を加えずに保留中のセッションが削除されます。

HTTP セッションの管理

HTTP 要求によって作成されたセッションはすぐにインスタンス化されるので、このセッションコンテキストを必要とする後続の HTTP 要求は作成されたセッションによってキューに追加されます。

この例では、sa_set_http_option プロシージャを使用してセッションがサーバで作成されると、ローカルホストクライアントは、データベース dbname で実行され、サービス session_service を実行する、指定されたセッション ID (session_63315422814117) のセッションに、次の URL を指定してアクセスできます。

```
http://localhost/dbname/session_service?
sessionid=session_63315422814117
```

Web アプリケーションでは、HTTP Web サーバ内のアクティブなセッションの使用率を追跡する手段が必要になることがあります。セッションデータを取得するには、NEXT_CONNECTION 関数をアクティブなデータベース接続に対して繰り返し呼び出し、SessionID などのセッション関連のプロパティを確認します。

次の SQL 文は、アクティブなセッションの追跡方法を示します。

```
CREATE VARIABLE conn_id LONG VARCHAR;
CREATE VARIABLE the_sessionID LONG VARCHAR;
SELECT NEXT_CONNECTION( NULL, NULL ) INTO conn_id;
conn_loop:
    LOOP
        IF conn_id IS NULL THEN
            LEAVE conn_loop;
        END IF;
        SELECT CONNECTION_PROPERTY( 'SessionID', conn_id )
            INTO the_sessionID;
```

```
        IF the_sessionID != '' THEN
            PRINT 'conn_id = %1!, SessionID = %2!', conn_id,
the_sessionID;
        ELSE
            PRINT 'conn_id = %1!', conn_id;
        END IF;
        SELECT NEXT_CONNECTION( conn_id, NULL ) INTO conn_id;
    END LOOP conn_loop;
    PRINT '¥n';
```

データベースサーバメッセージウィンドウには、次の出力のようなデータが表示されます。

```
conn_id = 30
conn_id = 29, SessionID = session_63315442223323
conn_id = 28, SessionID = session_63315442220088
conn_id = 25, SessionID = session_63315441867629
```

セッションに属する接続を明示的に切断すると、接続が閉じられ、セッションが削除されます。切断される接続が HTTP 要求の処理において現在アクティブになっている場合、その要求は削除対象としてマーク付けされ、要求を終了するキャンセル通知が送信されます。要求が終了すると、セッションは削除され、接続は閉じられます。セッションを削除すると、そのセッションのキューで保留中だったすべての要求が、キューに再度追加されます。

接続が現在アクティブでない場合は、セッションは削除対象としてマーク付けされ、セッションタイムアウトキューの先頭に再度追加されます。セッションと接続は次回のタイムアウトサイクルで削除されます (通常は 5 秒以内)。削除対象としてマーク付けされたセッションはいずれも、新しい HTTP 要求では使用できません。

データベースが停止すると、すべてのセッションが失われます。

HTTP セッションのエラーコード

新しい要求がアクセスしようとしたセッションで 16 を超える要求が保留になっていた場合、またはセッションをキューに追加しているときにエラーが発生した場合は、`503 Service Unavailable` エラーが発生します。

クライアントの IP アドレスまたはホスト名がセッション作成者の IP アドレスまたはホスト名と一致しない場合は、`403 Forbidden` エラーが発生します。

存在しないセッションが指定された要求は、暗黙的にはエラーを生成しません。この状況を (SessionID、SessionCreateTime、または SessionLastTime 接続プロパティを確認することで) 検出して、適切なアクションを実行するのは、Web アプリケーションで行う必要があります。

文字セットの変換に関する考慮事項

文字セット変換は、デフォルトによりテキストタイプの出力結果セットで自動的に実行されます。バイナリオブジェクトなどの他のタイプの結果セットは、影響

を受けません。要求の文字セットは HTTP Web サーバの文字セットに変換され、結果セットはクライアントアプリケーションの文字セットに変換されます。複数の文字セットがリストされている場合、サーバではリスト内の最初の適切な文字セットが使用されます。

文字セット変換は、sa_set_http_option システムプロシージャの HTTP オプション (CharsetConversion オプション) を設定することで有効または無効にできます。

次の例は、文字セットの自動変換をオフにする方法を示しています。

```
CALL sa_set_http_option('CharsetConversion', 'OFF');
```

sa_set_http_option システムプロシージャの 'AcceptCharset' オプションを使用して、文字セット変換が有効になっている場合の文字セットエンコードの設定を指定できます。

次の例は、Web サービスの文字セットエンコード優先度を ISO-8859-5 (サポートされていない場合は UTF-8) に指定する方法を示します。

```
CALL sa_set_http_option('AcceptCharset', 'iso-8859-5, utf-8');
```

文字セットはサーバ設定によって優先順位が付けられます。ただし、この選択には、クライアントの Accept-Charset 基準も考慮されます。クライアントに最も適し、かつ、このオプションでも指定されている文字セットが使用されます。

クロスサイトスクリプティングに関する考慮事項

Web アプリケーションを開発する場合、そのアプリケーションがクロスサイトスクリプティング (XSS) に対して脆弱でないことを確認する必要があります。このタイプの脆弱性は、攻撃者が Web ページにスクリプトを注入しようとすると発生します。

Web アプリケーションコードが運用される前に、アプリケーション開発者やデータベース管理者がそのセキュリティの脆弱性の可能性について確認することを強くおすすめします。Open Web Application Security Project (https://www.owasp.org) には、Web アプリケーションのセキュリティを保護する方法に関する情報が含まれています。

Web サービスシステムプロシージャ

次のシステムプロシージャは、Web サービスとともに使用します。

- sa_http_header_info システムプロシージャ
- sa_http_php_page システムプロシージャ
- sa_http_php_page_interpreted システムプロシージャ
- sa_http_variable_info システムプロシージャ
- sa_set_http_header システムプロシージャ
- sa_set_http_option システムプロシージャ

sa_set_soap_header システムプロシージャ

Web サービス関数

Web サービス関数は、Web サービス内の HTTP 要求と SOAP 要求の処理を支援します。

次の関数を使用できます。

HTML_DECODE 関数 [その他]
HTML_ENCODE 関数 [その他]
HTTP_BODY 関数 [Web サービス]
HTTP_DECODE 関数 [Web サービス]
HTTP_ENCODE 関数 [Web サービス]
HTTP_HEADER 関数 [Web サービス]
HTTP_RESPONSE_HEADER 関数 [Web サービス]
HTTP_VARIABLE 関数 [Web サービス]
NEXT_HTTP_HEADER 関数 [Web サービス]
NEXT_HTTP_RESPONSE_HEADER 関数 [Web サービス]
NEXT_HTTP_VARIABLE 関数 [Web サービス]
NEXT_SOAP_HEADER 関数 [SOAP]
SOAP_HEADER 関数 [SOAP]

また、Web サービスで使用できるシステムプロシージャも多数あります。

Web サービス接続プロパティ

Web サービス接続プロパティには、CONNECTION_PROPERTY 関数を使用してアクセスできるデータベースプロパティを指定できます。

次の構文を使用して、HTTP サーバの接続プロパティ値を SQL 関数またはプロシージャのローカル変数に格納します。

```
SELECT CONNECTION_PROPERTY('connection-property-name') INTO
variable_name;
```

次に、Web サービスアプリケーションで一般的に使用される、ランタイム HTTP 要求の便利な接続プロパティを示します。

- **HttpServiceName –** Web アプリケーションのサービス名オリジンを返します。
- **AuthType –** 接続時に使用される認証のタイプを返します。
- **ServerPort –** データベースサーバの TCP/IP ポート番号または 0 を返します。
- **ClientNodeAddress –** クライアント／サーバ接続のクライアント側に対応するノードを返します。

- **ServerNodeAddress –** クライアント／サーバ接続のサーバ側に対応するノードを返します。
- **BytesReceived –** クライアント／サーバ通信中に受信したバイト数を返します。

Web サービスオプション

Web サービスオプションは、HTTP サーバ動作をさまざまな面から制御します。

次の構文を使用して、HTTP サーバでパブリックオプションを設定します。

```
SET TEMPORARY OPTION PUBLIC.http_session_timeout=100;
```

次に、HTTP サーバでアプリケーション設定用に一般的に使用されるオプションを示します。

- **http_connection_pool_basesize –** データベース接続の公称スレッショルドサイズを指定します。
- **http_connection_pool_timeout –** 未使用の接続を接続プールに保持できる時間の上限を指定します。
- **http_session_timeout –** 非アクティブ状態の HTTP セッションが維持されるデフォルトのタイムアウト時間 (分単位) を指定します。
- **request_timeout –** 1 つの要求を実行できる最大時間を設定します。
- **webservice_namespace_host –** DISH サービスの仕様内で XML ネームスペースとして使用するホスト名を指定します。

SAP Sybase IQ HTTP Web サーバを参照する方法

Web サービスの命名と設計の方法によって、使用可能な URL 名が定義されます。各 Web サービスには、独自の Web コンテンツがあります。通常、このコンテンツは、データベース内のカスタム関数とプロシージャによって生成されますが、SQL 文を指定する URL を使用してコンテンツを生成することもできます。

別の方法として、またはこの方法と組み合わせて、専用サービスで処理されないすべての HTTP 要求を処理する `root` Web サービスを定義することもできます。通常、`root` Web サービスは、要求の URL とヘッダから、要求の処理方法を判断します。

URL によって、HTTP 要求または保護された HTTPS 要求を介して使用できる html コンテンツなどのリソースがユニークに指定されます。この項では、SAP Sybase IQ HTTP Web サーバで定義された Web サービスにアクセスできるようにするため、Web ブラウザの URL 構文をフォーマットする方法について説明します。

注意： この項で記載する情報は、RAW、XML、HTML、DISH サービスなどの汎用 HTTP Web サービスタイプを使用する HTTP Web サーバに適用されます。ブラウザを使用して、SOAP 要求を発行することはできません。JSON サービスから

は、AJAX を使用した Web サービスアプリケーションで使用される結果セットが返されます。

構文

```
{http|https}://host-name[:port-number][/dbn]/service-name[/path-name|?url-query]
```

パラメータ

- **host-name と port-number –** Web サーバのロケーションを指定します。オプションとして、デフォルトの HTTP ポート番号または HTTPS ポート番号として定義されていない場合は、ポート番号を指定します。*host-name* には、Web サーバを実行中のコンピュータの IP アドレスを指定できます。*port-number* は、Web サーバを起動したときに使用されているポート番号に一致している必要があります。
- **dbn –** データベース名を指定します。このデータベースは Web サーバで実行中であり、Web サービスが含まれている必要があります。

  Web サーバで 1 つのデータベースしか実行されていない場合、またはプロトコルオプションの特定の HTTP/HTTPS リスナに対してデータベース名が指定されている場合は、*dbn* を指定する必要はありません。
- **service-name –** アクセスする Web サービスの名前を指定します。この Web サービスは、*dbn* で指定されたデータベースに存在している必要があります。Web サービスを作成または変更する場合は、スラッシュ文字 (/) が有効であり、*service-name* の一部として使用できます。SAP Sybase IQ は、URL の残りの部分と定義されたサービスを一致させます。

  service-name が未指定で、`root` Web サービスが定義されている場合は、クライアント要求が処理されます。要求を処理する適切なサービスをサーバで識別できない場合は、`404 Not Found` エラーが返されます。関連動作として、`root` Web サービスが存在せず、URL 基準に基づいて要求を処理できない場合は、`404 Not Found` エラーが生成されます。
- **path-name –** サービス名の解決後、残りのスラッシュで区切られたパスは、Web サービスプロシージャからアクセスできます。URL ON を指定して作成されたサービスの場合は、指定された `URL` HTTP 変数を使用してパス全体にアクセスできます。URL ELEMENTS を指定して作成されたサービスの場合は、指定された HTTP 変数 `URL1` ～ `URL10` を使用して各パス要素にアクセスできます。

  パス要素変数は、サービス文定義のパラメータ宣言でホスト変数として定義できます。別の方法として、またはこの方法と組み合わせて、HTTP_VARIABLE 関数呼び出しを使用してストアドプロシージャ内から HTTP 変数にアクセスすることもできます。

次の例は、Web サービスの作成に使用する、URL 句を ELEMENTS に設定した SQL 文を示しています。

```
CREATE SERVICE TestWebService
    TYPE 'HTML'
    URL ELEMENTS
    AUTHORIZATION OFF
    USER DBA
    AS CALL TestProcedure ( :url1, :url2 );
```

この TestWebService Web サービスは、url1 と url2 の各ホスト変数を明示的に参照するプロシージャを呼び出します。

この Web サービスには、次の URL を使用してアクセスできます。この場合、デフォルトのポートを介した localhost から demo データベースで TestWebService が実行されていることが前提となります。

```
http://localhost/demo/TestWebService/Assignment1/Assignment2/
Assignment3
```

この URL は、TestProcedure を実行し、Assignment1 値を url1 に割り当て、Assignment2 値を url2 に割り当てる TestWebService にアクセスします。オプションとして、HTTP_VARIABLE 関数を使用して、TestProcedure から他のパス要素にアクセスできます。たとえば、HTTP_VARIABLE('url3') 関数呼び出しは Assignment3 を返します。

- **url-query –** HTTP GET 要求は、HTTP 変数を指定するクエリコンポーネントがあるパスをたどることがあります。同様に、標準の application/x-www-form-urlencoded Content-Type を使用する POST 要求の本文は、要求本文内の HTTP 変数を渡すことができます。いずれの場合も、HTTP 変数は名前／値ペアとして渡されます。変数名は、等号で値から区切られます。変数は、アンパサンドで区切られます。

  HTTP 変数は、SERVICE 文のパラメータリスト内でホスト変数として明示的に宣言したり、SERVICE 文のストアドプロシージャから HTTP_VARIABLE 関数を使用してアクセスしたりできます。

  たとえば、次の SQL 文は、2 つのホスト変数を必要とする Web サービスを作成します。ホスト変数は、コロン (:) のプレフィクスで識別されます。

```
CREATE SERVICE ShowSalesOrderDetail
    TYPE 'HTML'
    URL OFF
    AUTHORIZATION OFF
    USER DBA
    AS CALL ShowSalesOrderDetail( :customer_id, :product_id );
```

デフォルトのポートを介した localhost からデモデータベースで ShowSalesOrderDetail が実行されていると仮定した場合、次の URL を使用して Web サービスにアクセスできます。

```
http://localhost/demo/ShowSalesOrderDetail?
customer_id=101&product_id=300
```

この URL は、ShowSalesOrderDetail にアクセスし、値 101 を customer_id に割り当て、値 300 を product_id に割り当てます。結果は、Web ブラウザに HTML フォーマットで表示されます。

備考
サーバへの接続にユーザ名とパスワードが必要な場合は、Web ブラウザから入力が求められます。ブラウザの base64 によりユーザ入力が認証要求ヘッダ内でエンコードされ、要求が再送信されます。

Web サービスの URL 句が ON または ELEMENTS に設定されている場合、*path-name* と *url-query* の URL 構文プロパティを同時に使用すると、異なるフォーマットオプションのいずれかを使用して Web サービスにアクセスできるようになります。これらの構文プロパティを同時に使用する場合は、*path-name* フォーマットを最初に使用し、その後に *url-query* フォーマットを使用する必要があります。

次の例に示す SQL 文は、URL 句が ON に設定され、url 変数を定義する Web サービスを作成します。

```
CREATE SERVICE ShowSalesOrderDetail
    TYPE 'HTML'
    URL ON
    AUTHORIZATION OFF
    USER DBA
    AS CALL ShowSalesOrderDetail( :product_id, :url );
```

次に、url 値 101 と product_id 値 300 を割り当てる有効な URL のサンプルを示します。

```
http://localhost:80/demo/ShowSalesOrderDetail2/101?
product_id=300
http://localhost:80/demo/ShowSalesOrderDetail2?
url=101&product_id=300
http://localhost:80/demo/ShowSalesOrderDetail2?
product_id=300&url=101
```

path-name と *url-query* のコンテキストでホスト変数名が複数回割り当てられていると、最後の割り当てが常に優先されます。たとえば、次のサンプル URL は、url 値 101 と product_id 値 300 を割り当てます。

```
http://localhost:80/demo/ShowSalesOrderDetail2/302?
url=101&product_id=300
http://localhost:80/demo/ShowSalesOrderDetail2/String?
product_id=300&url=101
```

例

次の URL 構文は、gallery_image サービスが URL ON で定義されていると仮定した場合に、デフォルトのポートを介してローカル HTTP サーバ上の demo というデータベースで実行されている gallery_image という Web サービスにアクセスするために使用されます。

```
http://localhost/demo/gallery_image/sunset.jpg
```

この URL は、従来の Web サーバのディレクトリでグラフィックファイルを要求するように見えますが、HTTP Web サーバの入力パラメータとして sunset.jpg が指定された gallery_image サービスにアクセスします。

次の SQL 文は、この動作を実行するために、gallery サービスを HTTP サーバで定義する方法を示しています。

```
CREATE SERVICE gallery_image
    TYPE 'RAW'
    URL ON
    AUTHORIZATION OFF
    USER DBA
    AS CALL gallery_image ( :url );
```

gallery_image サービスは、同じ名前のプロシージャを呼び出して、クライアントで指定された URL を渡します。この Web サービス定義によってアクセスできる gallery_image プロシージャの実装例については、%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥HTTP¥gallery.sql を参照してください。

Web クライアントを使用した Web サービスへのアクセス

SAP Sybase IQ を Web クライアントとして使用して、SAP Sybase IQ Web サーバによって実行される Web サービスまたは Apache や IIS などのサードパーティの Web サーバにアクセスできます。

SAP Sybase IQ を Web クライアントとして使用できるだけでなく、SAP Sybase IQ Web サービスは、クライアントアプリケーションに JDBC や ODBC などの従来のインタフェースの代替を提供します。追加のコンポーネントが必要なく、Perl や Python などのスクリプト言語を含む各種の言語で記述されたマルチプラットフォームクライアントアプリケーションからアクセスできるため、これらは簡単に配備されます。

SAP Sybase IQ を Web クライアントとして使用するためのクイックスタート

この項では、SAP Sybase IQ を Web クライアントアプリケーションとして使用して SAP Sybase IQ HTTP サーバに接続し、一般的な HTTP Web サービスにアクセスする方法について説明します。SAP Sybase IQ Web クライアントの機能全般については説明しません。このトピックで説明するもの以外にも、数多くの SAP Sybase IQ Web クライアントの機能を使用できます。

あらゆるタイプのオンライン Web サーバに接続する SAP Sybase IQ Web クライアントアプリケーションを開発できますが、この項では、ローカル SAP Sybase IQ HTTP サーバをポート 8082 で起動し、次の SQL 文を使用して作成された `SampleHTMLService` という Web サービスに接続することを前提としています。

```
CREATE SERVICE SampleHTMLService
    TYPE 'HTML'
    USER DBA
    AUTHORIZATION OFF
    AS CALL sp_echo(:i, :f, :s);

CREATE PROCEDURE sp_echo(i INTEGER, f REAL, s LONG VARCHAR)
RESULT(ret_i INTEGER, ret_f REAL, ret_s LONG VARCHAR)
BEGIN
    SELECT i, f, s;
END;
```

SAP Sybase IQ Web クライアントアプリケーションを作成するには、次のタスクを実行します。

1. SAP Sybase IQ クライアントデータベースが存在しない場合、次のコマンドを実行して作成します。
   ```
   iqinit -dba DBA,sql client-database-name
   ```
   client-database-name をクライアントデータベースの新しい名前に置き換えます。
2. 次のコマンドを実行して、クライアントデータベースを起動します。
   ```
   iqsrv16 client-database-name.db
   ```
3. 次のコマンドを実行して、Interactive SQL を通じてクライアントデータベースに接続します。
   ```
   dbisql -c "UID=DBA;PWD=sql;SERVER=client-database-name"
   ```
4. 次の SQL 文を使用して、`SampleHTMLService` Web サービスに接続する新しいクライアントプロシージャを作成します。
   ```
   CREATE PROCEDURE client_post(f REAL, i INTEGER, s VARCHAR(16), x
   VARCHAR(16))
       URL 'http://localhost:8082/SampleHTMLService'
   ```

```
    TYPE 'HTTP:POST'
    HEADER 'User-Agent:SATest';
```

5. 次の SQL 文を実行して、クライアントプロシージャを呼び出し、Web サーバに HTTP 要求を送信します。

```
CALL client_post(3.14, 9, 's varchar', 'x varchar');
```

client_post によって作成された HTTP POST 要求は、次のような出力になります。

```
POST /SampleHTMLService HTTP/1.0
ASA-Id: ea1746b01cd0472eb4f0729948db60a2
User-Agent: SATest
Accept-Charset: windows-1252, UTF-8, *
Date: Wed, 9 Jun 2010 21:55:01 GMT
Host: localhost:8082
Connection: close
Content-Type: application/x-www-form-urlencoded;
charset=windows-1252
Content-Length: 58

&f=3.1400001049041748&i=9&s=s%20varchar&x=x%20varchar
```

Web サーバ上で実行される Web サービス SampleHTMLService によって i、f、s の各パラメータ値が POST 要求から抽出され、パラメータとして sp_echo プロシージャに渡されます。パラメータ値 x は無視されます。sp_echo プロシージャで結果セットが作成されて Web サービスに返されます。正しく対応付けられるように、クライアントと Web サービスの間でパラメータ名を決めておくことが重要です。

Web サービスで応答が作成されてクライアントに送り返されます。Interactive SQL に表示される出力は次のようになります。

属性	値
Status	HTTP/1.1 200 OK
Body	<html> <head> <title>/SampleHTMLService</title></head> <body> /SampleHTMLService <table border=1> <tr class="header"><th>ret_i</th> <th>ret_f</th> <th>ret_s</th> </tr> <tr><td>9</td><td>3.1400001049041748</td><td>s var- char</td>
Date	Wed, 09 Jun 2010 21:55:01 GMT

属性	値
Connection	close
Expires	Wed, 09 Jun 2010 21:55:01 GMT
Content-Type	text/html; charset=windows-1252
Server	Sybase IQ/16.0.0

SAP Sybase IQ HTTP Web サーバにアクセスするためのクイックスタート

この項では、Python と C# の 2 つの異なるタイプのクライアントアプリケーションを使用して SAP Sybase IQ HTTP Web サーバにアクセスする方法について説明します。SAP Sybase IQ Web サービスアプリケーションの機能全般については説明しません。このトピックで説明するもの以外にも、数多くの SAP Sybase IQ Web サービスの機能を使用できます。

あらゆるタイプのオンライン Web サーバに接続する SAP Sybase IQ Web クライアントアプリケーションを開発できますが、このマニュアルでは、ローカル SAP Sybase IQ HTTP サーバをポート 8082 で起動し、次の SQL 文を使用して作成された `SampleXMLService` という Web サービスに接続することを前提としています。

```
CREATE SERVICE SampleXMLService
    TYPE 'XML'
    USER DBA
    AUTHORIZATION OFF
    AS CALL sp_echo2(:i, :f, :s);

CREATE PROCEDURE sp_echo2(i INTEGER, f NUMERIC(6,2), s LONG VARCHAR )
RESULT( ret_i INTEGER, ret_f NUMERIC(6,2), ret_s LONG VARCHAR )
BEGIN
    SELECT i, f, s;
END;
```

C# または Python を使用して XML Web サービスにアクセスするには、次のタスクを実行します。

1. HTTP サーバ上で Web サービスに接続するプロシージャを作成します。
   `SampleXMLService` Web サービスにアクセスするコードを作成します。
   - C# の場合は、次のコードを入力します。

     ```
     using System;
     using System.Xml;

     public class WebClient
     {
         static void Main(string[] args)
         {
             XmlTextReader reader = new XmlTextReader(
                 "http://localhost:8082/SampleXMLService?
     ```

```
i=5&f=3.14&s=hello");
        while (reader.Read())
        {
            switch (reader.NodeType)
            {
                case XmlNodeType.Element:
                    if (reader.Name == "row")
                    {
                    Console.Write(reader.GetAttribute("ret_i")
+ " ");
                    Console.Write(reader.GetAttribute("ret_s")
+ " ");

Console.WriteLine(reader.GetAttribute("ret_f"));
                    }
                    break;
            }
        }
        reader.Close();
    }
}
```

このコードを DocHandler.cs というファイルに保存します。

プログラムをコンパイルするには、コマンドプロンプトで次のコマンドを実行します。

```
csc /out:DocHandler.exe DocHandler.cs
```

- Python の場合は、次のコードを入力します。

```
import xml.sax

class DocHandler( xml.sax.ContentHandler ):
    def startElement( self, name, attrs ):
        if name == 'row':
            table_int = attrs.getValue( 'ret_i' )
            table_string = attrs.getValue( 'ret_s' )
            table_numeric = attrs.getValue( 'ret_f' )
            print('%s %s %s' % ( table_int, table_string,
table_numeric ))

parser = xml.sax.make_parser()
parser.setContentHandler( DocHandler() )
parser.parse('http://localhost:8082/SampleXMLService?
i=5&f=3.14&s=hello')
```

このコードを DocHandler.py というファイルに保存します。

2. HTTP サーバによって送信された結果セットに対する操作を実行します。
   - C# の場合は、次のコマンドを実行します。

     ```
     DocHandler
     ```

   - Python の場合は、次のコマンドを実行します。

     ```
     python DocHandler.py
     ```

アプリケーションによって次の出力が表示されます。

```
5 hello 3.14
```

Web クライアントアプリケーションの開発

SAP Sybase IQ データベースを、Web クライアントアプリケーションとして使用して、SAP Sybase IQ によって実行される Web サービスまたはサードパーティの Web サーバ上で実行される Web サービスにアクセスできます。SAP Sybase IQ Web クライアントアプリケーションを作成するには、Web サービスのターゲット終了ポイントを指定する URL 句などの設定句を使用して、ストアドプロシージャや関数を記述します。Web クライアントプロシージャは、本文がないことを除き、その他のストアドプロシージャと同様に使用します。Web クライアントプロシージャは、呼び出されると、アウトバウンド HTTP または SOAP 要求を作成します。Web クライアントプロシージャは、それ自体へのアウトバウンド HTTP 要求を作成できないようになっています。Web クライアントプロシージャでは、同じデータベース上で実行されているローカルホスト SAP Sybase IQ Web サービスを呼び出すことができません。

Web サービスアプリケーションの例の詳細については、`%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥HTTP` ディレクトリを参照してください。

Web クライアント関数とプロシージャの要件と推奨事項

Web サービスクライアントのプロシージャと関数には、Web サービスの終了ポイントを識別するために、URL 句を定義する必要があります。Web サービスクライアントのプロシージャまたは関数は、設定用の特殊な句があることを除き、その他のストアドプロシージャや関数と同様に使用します。

CREATE PROCEDURE 文と CREATE FUNCTION 文を使用して、Web サーバに SOAP 要求または HTTP 要求を送信する Web クライアント関数とプロシージャを作成できます。

次のリストに、Web クライアント関数とプロシージャの作成または変更の要件と推奨事項を示します。Web クライアント関数またはプロシージャを作成または変更する場合、次の情報を指定できます。

- Web サービスの終了ポイントを指定する絶対 URL を必要とする URL 句 (必須)
- 要求が HTTP であるか、HTTP を介した SOAP であるかを指定する TYPE 句 (推奨)
- クライアントアプリケーションにアクセスできるポート (省略可)
- HTTP 要求ヘッダを指定する HEADER 句 (省略可)
- SOAP 要求エンベロープ内の SOAP ヘッダ基準を指定する SOAPHEADER 句 (省略可。SOAP 要求の場合のみ)
- ネームスペース URI (SOAP 要求の場合のみ)

Web クライアント URL 句

Web サービスの終了ポイントのロケーションを指定して、Web クライアント関数またはプロシージャがアクセスできるようにします。CREATE PROCEDURE 文と CREATE FUNCTION 文の URL 句により、アクセスする Web サービスの URL が提供されます。

HTTP サービスの URL の指定

URL 句内で HTTP スキームを指定すると、プロシージャまたは関数が HTTP プロトコルを使用したセキュリティ保護されていない通信に対応できるように設定されます。

次の文は、ポート 8082 の `localhost` にある HTTP Web サーバによって実行される `dbname` というデータベースにある `SampleHTMLService` という Web サービスに要求を送信するプロシージャを作成する方法を示します。

```
CREATE PROCEDURE client_sender(f REAL, i INTEGER, s VARCHAR(16))
    URL 'http://localhost:8082/dbname/SampleHTMLService'
    TYPE 'HTTP:POST'
    HEADER 'User-Agent:SATest';
```

データベース名は、HTTP サーバが複数のデータベースをホストする場合にのみ必要です。`localhost` を HTTP サーバのホスト名または IP アドレスに置き換えることができます。

HTTPS サービスの URL の指定

URL 句内で HTTPS スキームを指定すると、プロシージャまたは関数が SSL (セキュアソケットレイヤ) 経由のセキュリティ保護された通信に対応できるように設定されます。

Web クライアントアプリケーションは、RSA サーバ証明書、またはセキュア HTTPS 要求を発行するためにサーバ証明書に署名した証明書にアクセスできる必要があります。サーバを認証し、中間者攻撃を防ぐために、クライアントプロシージャには証明書が必要です。

CREATE PROCEDURE 文と CREATE FUNCTION 文の CERTIFICATE 句を使用してサーバを認証し、セキュリティ保護されたデータチャネルを確立します。証明書をファイルに保存してファイル名を指定するか、証明書全体を文字列値として提供できます。両方を行うことはできません。

次の文は、ポート 8082 の `localhost` にある HTTPS サーバ内の `dbname` というデータベースにある `SecureHTMLService` という Web サービスに要求を送信するプロシージャを作成する方法を示します。

```
CREATE PROCEDURE client_sender(f REAL, i INTEGER, s VARCHAR(16))
    URL 'HTTPS://localhost:8082/dbname/SecureHTMLService'
    CERTIFICATE 'file=%ALLUSERSPROFILE%/SybaseIQ/demo¥¥Certificates¥
```

```
¥rsaroot.crt'
    TYPE 'HTTP:POST'
    HEADER 'User-Agent:SATest';
```

この例の CERTIFICATE 句は、RSA サーバ証明書が %ALLUSERSPROFILE% ¥SybaseIQ¥samples¥Certificates¥rsaroot.crt ファイルにあることを示します。

注意： HTTPS_FIPS を指定すると、強制的に FIPS ライブラリが使用されます。HTTPS_FIPS を指定したときに、FIPS ライブラリがない場合は、代わりに FIPS 以外のライブラリが使用されます。

プロキシサーバの URL の指定

プロキシサーバを使用して送信する必要がある要求もあります。CREATE PROCEDURE 文と CREATE FUNCTION 文の PROXY 句を使用して、プロキシサーバの URL を指定し、その URL に要求をリダイレクトします。プロキシサーバは、最終送信先へ要求を転送し、応答を取得し、SAP Sybase IQ へそれを返します。

Web サービスの要求タイプ

Web クライアント関数またはプロシージャを作成するとき、Web サーバに送信するクライアント要求のタイプを指定できます。CREATE PROCEDURE 文と CREATE FUNCTION 文の TYPE 句は、要求をフォーマットしてから Web サーバに送信します。

HTTP 要求フォーマットの指定

Web クライアント関数とプロシージャは、TYPE 句で指定されたフォーマットが HTTP プレフィックスで始まる場合、HTTP 要求を送信します。

たとえば、指定した URL に HTTP 要求を送信する PostOperation という HTTP プロシージャを作成するには、Web クライアントデータベースで次の SQL 文を実行します。

```
CREATE PROCEDURE PostOperation(a INTEGER, b CHAR(128))
    URL 'HTTP://localhost:8082/dbname/SampleWebService'
    TYPE 'HTTP:POST';
```

この例では、要求は HTTP:POST 要求としてフォーマットされます。これにより、次のような要求が生成されます。

```
POST /dbname/SampleWebService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SQLAnywhere/16.0.0.3600
Accept-Charset: windows-1252, UTF-8, *
Date: Fri, 03 Feb 2012 15:02:49 GMT
Host: localhost:8082
Connection: close
Content-Type: application/x-www-form-urlencoded;
```

```
charset=windows-1252
Content-Length: 12

a=123&b=data
```

SOAP 要求フォーマットの指定
Web クライアント関数とプロシージャは、TYPE 句で指定されたフォーマットが SOAP プレフィックスで始まる場合、HTTP 要求を送信します。

たとえば、指定した URL に SOAP 要求を送信する SoapOperation という SOAP プロシージャを作成するには、Web クライアントデータベースで次の文を実行します。

```
CREATE PROCEDURE SoapOperation(intVariable INTEGER, charVariable
CHAR(128))
    URL 'HTTP://localhost:8082/dbname/SampleSoapService'
    TYPE 'SOAP:DOC';
```

この例では、このプロシージャを呼び出すと、URL に SOAP:DOC 要求が送信されます。これにより、次のような要求が生成されます。

```
POST /dbname/SampleSoapService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SQLAnywhere/16.0.0.3600
Accept-Charset: windows-1252, UTF-8, *
Date: Fri, 03 Feb 2012 15:05:13 GMT
Host: localhost:8082
Connection: close
Content-Type: text/xml; charset=windows-1252
Content-Length: 428
SOAPAction: "HTTP://localhost:8082/SoapOperation"

<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="HTTP://localhost:8082">
  <SOAP-ENV:Body>
    <m:SoapOperation>
      <m:intVariable>123</m:intVariable>
      <m:charVariable>data</m:charVariable>
    </m:SoapOperation>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

プロシージャ名は、本文内の <m:SoapOperation> タグに表示されます。プロシージャに対する 2 つのパラメータ intVariable と charVariable は、それぞれ <m:intVariable> と <m:charVariable> になります。

デフォルトでは、SOAP 要求が構築されるとき、SOAP 操作名としてストアドプロシージャ名が使用されます。パラメータ名は、SOAP エンベロープのタグ名内に

示されます。サーバでは SOAP 要求内にこれらの名前があることを予期しているため、SOAP ストアドプロシージャを定義する場合は、これらの名前を正確に参照してください。SET 句を使用すると、対象となるプロシージャの代替 SOAP 操作名を指定できます。WSDLC を使用すると、ファイルや URL の指定から WSDL を読み込み、SQL スタブ関数またはプロシージャを生成できます。最も簡単な場合 (1 つの文字列値を返す SOAP RPC など) を除き、プロシージャではなく関数の定義を使用することをおすすめします。SOAP 関数では、OPENXML を使用して解析できる、完全な SOAP 応答エンベロープを返します。

Web クライアントポート

ファイアウォールを介してサーバへの接続を確立するときに使用するポートを指定する必要がある場合があります。CREATE PROCEDURE 文と CREATE FUNCTION 文の CLIENTPORT 句を使用して、クライアントアプリケーションが TCP/IP を使用して通信するポート番号を指定できます。ファイアウォールによって特定範囲のポートへのアクセスが制限されていないかぎり、この機能を使用しないことをおすすめします。

たとえば、Web クライアントデータベースで次の SQL 文を実行して、5050 ～ 5060 の範囲のいずれかのポートまたはポート 5070 を使用して指定した URL に要求を送信する `SomeOperation` というプロシージャを作成します。

```
CREATE PROCEDURE SomeOperation()
    URL 'HTTP://localhost:8082/dbname/SampleWebService'
    CLIENTPORT '5050-5060,5070';
```

必要な場合は、ポート番号の範囲を指定することをおすすめします。ポート番号を 1 つ指定すると、一度に 1 つの接続のみが維持されます。クライアントアプリケーションは、いずれかのポート番号と接続が確立するまで、指定されたすべてのポート番号へのアクセスを試行します。接続を閉じた後、同じサーバとポートを使って新しい接続が作成されないように、数分のタイムアウト期間が開始されます。

この機能は、ClientPort ネットワークプロトコルオプションの設定と類似しています。

HTTP 要求ヘッダの管理

CREATE PROCEDURE 文と CREATE FUNCTION 文の HEADER 句を使用して、HTTP 要求ヘッダを追加、変更、または削除できます。名前を参照することによって、HTTP 要求ヘッダを抑制できます。ヘッダ名の後にコロン、値の順に指定して、HTTP 要求ヘッダ値を追加または変更します。ヘッダ値の指定は省略可能です。

たとえば、HTTP 要求ヘッダを制限する指定した URL に要求を送信する SomeOperation2 というプロシージャを作成するには、Web クライアントデータベースで次の SQL 文を実行します。

```
CREATE PROCEDURE SomeOperation2()
    URL 'HTTP://localhost:8082/dbname/SampleWebService'
    TYPE 'HTTP:GET'
    HEADER 'SOAPAction¥nDate¥nFrom:¥nCustomAlias:John Doe';
```

この例では、SAP Sybase IQ によって自動的に生成される Date ヘッダが表示されなくなります。From ヘッダは、含まれているものの、値が割り当てられていません。HTTP 要求には CustomAlias という新しいヘッダが含められ、値 John Doe が割り当てられます。GET 要求は次のようになります。

```
GET /dbname/SampleWebService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SybaseIQ/16.0.0.3600
Accept-Charset: windows-1252, UTF-8, *
From:
Host: localhost:8082
Connection: close
CustomAlias: John Doe
```

長いヘッダ値の折り返しがサポートされています。折り返すには、¥n の直後に 1 つ以上のスペースが必要です。

次の例は、長いヘッダ値のサポートを示します。

```
CREATE PROCEDURE SomeOperation3()
    URL 'HTTP://localhost:8082/dbname/SampleWebService'
    TYPE 'HTTP:POST'
    HEADER 'heading1: This long value¥n is really long for a header.
¥n
    heading2:shortvalue';
```

POST 要求は次のようになります。

```
POST /dbname/SampleWebService HTTP/1.0
ASA-Id: e88a416e24154682bf81694feaf03052
User-Agent: SybaseIQ/16.0.0.3600
Accept-Charset: windows-1252, UTF-8, *
Date: Fri, 03 Feb 2012 15:26:04 GMT
heading1: This long value is really long for a header.
heading2:shortvalue
Host: localhost:8082
Connection: close
Content-Type: application/x-www-form-urlencoded;
charset=windows-1252
Content-Length: 0
```

注意： SOAP 関数またはプロシージャの作成時に WSDL に指定された SOAP サービス URI に SOAPAction HTTP 要求ヘッダを設定してください。

自動生成される HTTP 要求ヘッダ
自動生成されるヘッダを修正すると、予期しない結果になる可能性があります。次の HTTP 要求ヘッダは、事前の対策を行わずに変更しないでください。

HTTP ヘッダ	説明
Accept-Charset	常に自動的に生成される。このヘッダを変更または削除すると、予期しないデータ変換エラーが発生する可能性がある。
ASA-Id	常に自動的に生成される。このヘッダは、デッドロックを防ぐため、クライアントアプリケーションがそれ自体に接続しないようにする。
Authorization	URL にクレデンシャルが含まれるときに自動生成される。このヘッダを変更または削除すると、要求がエラーになる可能性がある。BASIC 認証だけがサポートされる。ユーザとパスワードの情報は、HTTPS を使用した接続時だけ含めるようにする。
Connection	Connection: close は常に自動的に生成される。クライアントアプリケーションは、永続的接続をサポートしない。変更した場合、接続がハングする可能性がある。
Host	常に自動的に生成される。HTTP/1.1 クライアントが Host ヘッダを提供しない場合、400 Bad Request で応答するには HTTP/1.1 サーバが必要。
Transfer-Encoding	チャンクモードで要求を通知するときに自動生成される。このヘッダやチャンク値を削除すると、クライアントが CHUNK モードを使用しているときにエラーになる。
Content-Length	チャンクモード以外で要求を通知するときに自動生成される。このヘッダは、本文のコンテンツ長をサーバに通知するために必要。コンテンツ長が不正な場合、接続がハングするか、データが失われる可能性がある。

SOAP 要求ヘッダの管理
SOAP 要求ヘッダは、SOAP エンベロープ内の XML フラグメントです。SOAP 操作とそのパラメータはリモートプロシージャコール (RPC) であると考えることができますが、SOAP 要求ヘッダを使用して、特定の要求や応答内のメタ情報を転送できます。SOAP 要求ヘッダによって、認証情報やセッション基準などのアプリケーションのメタデータが転送されます。

SOAPHEADER 句は、SOAP 要求ヘッダエントリに準拠する有効な XML フラグメントである必要があります。複数の SOAP 要求ヘッダエントリを指定できます。ストアドプロシージャまたは関数によって、SOAP ヘッダ要素 (`SOAP-ENV:Header`) 内に SOAP 要求ヘッダエントリが自動的に注入されます。

SOAPHEADER 値では、静的定数として宣言したり、代入パラメータメカニズムを使用して動的に設定したりできる SOAP ヘッダが指定されます。次に、サンプルの SOAP 要求の一部を示します。これにはそれぞれ `Authentication` と `Session` という 2 つの XML ヘッダが含まれています。

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="HTTP://localhost:8082">
  <SOAP-ENV:Header>
    <Authentication xmlns="CustomerOrderURN">
      <userName pwd="none" mustUnderstand="1">
        <first>John</first>
        <last>Smith</last>
      </userName>
    </Authentication>
    <Session xmlns="SomeSession">123456789</Session>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:SoapOperation>
      <m:intVariable>123</m:intVariable>
      <m:charVariable>data</m:charVariable>
    </m:SoapOperation>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP 呼び出しによって返される SOAP 応答ヘッダの処理は、関数とプロシージャによって異なります。最も柔軟で推奨される方法である関数を使用すると、SOAP 応答エンベロープ全体が受信されます。その後、OPENXML 演算子を使用して応答エンベロープを処理し、SOAP ヘッダと SOAP 本文のデータを抽出できます。プロシージャを使用すると、IN または INOUT 変数にマッピングする代入パラメータを使用する場合にのみ SOAP 応答ヘッダを抽出できます。SOAP プロシージャで許容される IN または INOUT パラメータは、1 つだけです。

Web サービス関数は、応答 SOAP エンベロープを解析して、ヘッダエントリを取得する必要があります。

例

次の例は、パラメータと SOAP ヘッダを送信する SOAP プロシージャと関数を作成する方法を示します。ラッパープロシージャは、Web サービスプロシージャコールにデータを取り込み、応答を処理するために使用されます。`soapAddItemProc` プロシージャは SOAP Web サービスプロシージャの使用方法を示し、`soapAddItemFunc` 関数は SOAP Web サービス関数の使用方法を示し、`httpAddItemFunc` 関数は SOAP ペイロードが HTTP Web サービスプロシージャに渡される方法を示します。

次の例は、代入パラメータを使用して SOAP ヘッダを送信する SOAP クライアントプロシージャを示しています。1 つの INOUT パラメータを使用して SOAP ヘッダを受信しています。ラッパーストアドプロシージャ addItemProcWrapper から soapAddItemProc を呼び出して、パラメータを含む SOAP ヘッダを送受信する方法を示しています。

```
CREATE PROCEDURE soapAddItemProc(amount INT, item LONG VARCHAR,
        INOUT inoutheader LONG VARCHAR, IN inheader LONG VARCHAR)
    URL 'http://localhost:8082/itemStore'
    SET 'SOAP( OP=addItems )'
    TYPE 'SOAP:DOC'
    SOAPHEADER '!inoutheader!inheader';

CREATE PROCEDURE addItemProcWrapper(amount INT, item LONG VARCHAR,
        first_name LONG VARCHAR, last_name LONG VARCHAR)
BEGIN
    DECLARE io_header LONG VARCHAR;     // inout (write/read) soap
header
    DECLARE resxml LONG VARCHAR;
    DECLARE soap_header_sent LONG VARCHAR;
   DECLARE i_header LONG VARCHAR;      // in (write) only soap header
    DECLARE err int;
    DECLARE crsr CURSOR FOR
        CALL soapAddItemProc( amount, item, io_header, i_header );

    SET io_header = XMLELEMENT( 'Authentication',
                        XMLATTRIBUTES('CustomerOrderURN' as xmlns),
                        XMLELEMENT('userName', XMLATTRIBUTES(
                                        'none' as pwd,
                                        '1' as mustUnderstand ),
                            XMLELEMENT( 'first', first_name ),
                            XMLELEMENT( 'last', last_name ) ) );
    SET i_header = '<Session xmlns="SomeSession">123456789</
Session>';
    SET soap_header_sent = io_header || i_header;
    OPEN crsr;
    FETCH crsr INTO resxml, err;
    CLOSE crsr;

    SELECT resxml, err, soap_header_sent, io_header AS
soap_header_received;
END;

/* example call to addItemProcWrapper */
CALL addItemProcWrapper( 5, 'shirt', 'John', 'Smith' );
```

次の例は、代入パラメータを使用して SOAP ヘッダを送信する SOAP クライアント関数を示しています。SOAP 応答エンベロープ全体が返されます。SOAP ヘッダは OPENXML 演算子を使用して解析できます。ラッパー関数 addItemFuncWrapper から soapAddItemFunc を呼び出して、パラメータを含む SOAP ヘッダを送受信する方法を示しています。また、OPENXML 演算子を使用して応答を処理する方法も示しています。

```
CREATE FUNCTION soapAddItemFunc(amount INT, item LONG VARCHAR,
        IN inheader1 LONG VARCHAR, IN inheader2 LONG VARCHAR )
    RETURNS XML
    URL 'http://localhost:8082/itemStore'
    SET 'SOAP(OP=addItems)'
    TYPE 'SOAP:DOC'
    SOAPHEADER '!inheader1!inheader2';

CREATE PROCEDURE addItemFuncWrapper(amount INT, item LONG VARCHAR,
        first_name LONG VARCHAR, last_name LONG VARCHAR )
BEGIN
    DECLARE i_header1 LONG VARCHAR;
    DECLARE i_header2 LONG VARCHAR;
    DECLARE res LONG VARCHAR;
    DECLARE ns LONG VARCHAR;
    DECLARE xpath LONG VARCHAR;
    DECLARE header_entry LONG VARCHAR;
    DECLARE localname LONG VARCHAR;
    DECLARE namespaceuri LONG VARCHAR;
    DECLARE r_quantity int;
    DECLARE r_item LONG VARCHAR;
    DECLARE r_status LONG VARCHAR;

    SET i_header1 = XMLELEMENT( 'Authentication',
                        XMLATTRIBUTES('CustomerOrderURN' as xmlns),
                        XMLELEMENT('userName', XMLATTRIBUTES(
                                        'none' as pwd,
                                        '1' as mustUnderstand ),
                            XMLELEMENT( 'first', first_name ),
                            XMLELEMENT( 'last',  last_name ) ) );
    SET i_header2 = '<Session xmlns="SessionURN">123456789</
Session>';

    SET res = soapAddItemFunc( amount, item, i_header1, i_header2 );

    SET ns = '<ns xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope/"'
            || ' xmlns:mp="urn:ianywhere-com:sa-xpath-metaprop"'
            || ' xmlns:customer="CustomerOrderURN"'
            || ' xmlns:session="SessionURN"'
            || ' xmlns:tns="http://localhost:8082"></ns>';

    // Process headers...
    SET xpath = '//SOAP-ENV:Header/*';
    BEGIN
         DECLARE crsr CURSOR FOR SELECT * FROM
                OPENXML( res, xpath, 1, ns )
                  WITH (  "header_entry" LONG VARCHAR '@mp:xmltext',
                              "localname"    LONG VARCHAR
'@mp:localname',
                              "namespaceuri" LONG VARCHAR
'@mp:namespaceuri' );
        OPEN crsr;
        FETCH crsr INTO "header_entry", "localname", "namespaceuri";
        CLOSE crsr;
    END;
```

```
    // Process body...
    SET xpath = '//tns:row';
    BEGIN
         DECLARE crsr1 CURSOR FOR SELECT * FROM
                 OPENXML( res, xpath, 1, ns )
                     WITH (  "r_quantity" INT 'tns:quantity/text()',
                              "r_item"     LONG VARCHAR 'tns:item/
text()',
                              "r_status"   LONG VARCHAR 'tns:status/
text()' );
        OPEN crsr1;
        FETCH crsr1 INTO "r_quantity", "r_item", "r_status";
        CLOSE crsr1;
    END;

    SELECT r_item, r_quantity, r_status, header_entry, localname,
namespaceuri;
END;

/* example call to addItemFuncWrapper */
CALL addItemFuncWrapper( 6, 'shorts', 'Jack', 'Smith' );
```

次の例は、HTTP:POST を SOAP ペイロード全体の転送手段として使用する方法を示しています。この方法では、Web サービス SOAP クライアントプロシージャを作成するのではなく、SOAP ペイロードを転送する Web サービス HTTP プロシージャを作成しています。ラッパープロシージャ `addItemHttpWrapper` から `httpAddItemFunc` を呼び出して、POST 関数の使用例を示しています。パラメータを含む SOAP ヘッダを送受信する方法と、応答を受け取る方法を示しています。

```
CREATE FUNCTION httpAddItemFunc(soapPayload XML)
    RETURNS XML
    URL 'http://localhost:8082/itemStore'
    TYPE 'HTTP:POST:text/xml'
    HEADER 'SOAPAction: "http://localhost:8082/addItems"';

CREATE PROCEDURE addItemHttpWrapper(amount INT, item LONG VARCHAR)
RESULT(response XML)
BEGIN
    DECLARE payload XML;
    DECLARE response XML;

    SET payload =
'<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:addItems>
      <m:amount>' || amount || '</m:amount>
      <m:item>' || item || '</m:item>
```

```
    </m:addItems>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>';

    SET response = httpAddItemFunc( payload );
    /* process response as demonstrated in addItemFuncWrapper */
    SELECT response;
END;

/* example call to addItemHttpWrapper */
CALL addItemHttpWrapper( 7, 'socks' );
```

制限事項

サーバ側 SOAP サービスは、現在は input および output SOAP ヘッダ要件を定義できません。したがって、SOAP ヘッダメタデータを DISH サービスの WSDL 出力で使用することはできません。SOAP クライアントツールキットは、SAP Sybase IQ SOAP サービスの終了ポイント用に SOAP ヘッダインタフェースを自動生成することができません。
SOAP ヘッダフォールトはサポートされていません。

SOAP ネームスペース URI の要件
ネームスペース URI では、指定された SOAP 操作の SOAP 要求エンベロープを作成するために使用される XML ネームスペースを指定します。ネームスペース URI が定義されていない場合は、URL 句のドメインコンポーネントが使用されます。

サーバ側の SOAP プロセッサはこの URI を使用して、要求のメッセージ本文内にあるさまざまなエンティティの名前を解釈します。CREATE PROCEDURE 文と CREATE FUNCTION 文の NAMESPACE 句では、ネームスペース URI を指定します。

ネームスペース URI を指定しないと、プロシージャコールが成功しない可能性があります。通常、この情報は一般の Web サーバのマニュアルに示されていますが、必要なネームスペース URI は、Web サーバから使用できる WSDL から取得できます。SAP Sybase IQ Web サーバと通信しようとしている場合は、DISH サービスにアクセスして、WSDL を生成できます。

通常、NAMESPACE は、`wsdl:definition` 要素内の WSDL ドキュメントの最初で指定された `targetNamespace` 属性からコピーできます。後続の '/' は重要であるため、これらを含める場合は注意してください。次に、指定された SOAP 操作の `soapAction` 属性を確認します。この属性は、後述するように生成される `SOAPAction` HTTP ヘッダに対応している必要があります。

NAMESPACE 句は 2 つの役割を果たします。NAMESPACE 句は、SOAP エンベロープの本文のネームスペースを指定し、プロシージャに TYPE 'SOAP:DOC' が指定さ

れている場合は、SOAPAction HTTP ヘッダのドメインコンポーネントとして使用されます。

次の例は、NAMESPACE 句の使用方法を示します。

```
CREATE FUNCTION an_operation(a_parameter LONG VARCHAR)
    RETURNS LONG VARCHAR
    URL 'http://wsdl.domain.com/fictitious.asmx'
    TYPE 'SOAP:DOC'
    NAMESPACE 'http://wsdl.domain.com/'
```

Interactive SQL で次の SQL 文を実行します。

```
SELECT an_operation('a_value');
```

この文では、次の出力のような SOAP 要求が生成されます。

```
POST /fictitious.asmx HTTP/1.0
SOAPAction: "http://wsdl.domain.com/an_operation"
Host: wsdl.domain.com
Content-Type: text/xml
Content-Length: 387
Connection: close

<?xml version="1.0"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:m="http://wsdl.domain.com/">
  <SOAP-ENV:Body>
   <m:an_operation>
    <m:a_parameter>a_value</m:a_parameter>
   </m:an_operation>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

プレフィクス 'm' のネームスペースが http://wsdl.domain.com/ に設定され、SOAPAction HTTP ヘッダによって、SOAP 操作の完全に修飾された URL が指定されます。

後続のスラッシュは SAP Sybase IQ が正しく動作するための必要条件ではありませんが、診断が難しい応答障害の原因となる可能性があります。SOAPAction HTTP ヘッダは、後続のスラッシュに関係なく、正しく生成されます。

NAMESPACE が指定されていない場合、URL 句のドメインコンポーネントが SOAP 本文のネームスペースとして使用され、プロシージャが TYPE 'SOAP:DOC' の場合は、HTTP SOAPAction HTTP ヘッダの生成に使用されます。上記の例で NAMESPACE 句が省略された場合は、http://wsdl.domain.com がネームスペースとして使用されます。後続のスラッシュ '/' がないことが、わずかに異なります。SOAPAction HTTP ヘッダを含め、SOAP 要求のそれ以外のすべての点は、上記の例と同じです。

上記の SOAP:DOC で説明したように、NAMESPACE 句は SOAP 本文のネームスペースを指定するときに使用します。ただし、SOAPAction HTTP ヘッダは、空の値 SOAPAction: "" で生成されます。

SOAP:DOC 要求タイプを使用する場合は、SOAPAction HTTP ヘッダを作成するためにネームスペースも使用されます。

Web クライアント SQL 文

次の SQL 文を使用して、Web クライアントの開発を支援できます。

Web クライアント関連の SQL 文	説明
CREATE FUNCTION 文 [Web サービス]	HTTP 要求または SOAP over HTTP 要求を行う Web クライアント関数を作成する。
ALTER FUNCTION 文	関数を変更する。
CREATE PROCEDURE 文 [Web サービス]	HTTP サーバへの HTTP 要求または SOAP 要求を作成するユーザ定義の Web クライアントプロシージャを作成する。
ALTER PROCEDURE 文	プロシージャを変更する。

Web サービスへの変数の指定

Web サービスのタイプに応じて、さまざまな方法で Web サービスに変数を指定できます。

Web クライアントアプリケーションは、次のいずれかの方法を使用して、一般的な HTTP Web サービスに変数を指定できます。

URL のサフィックス
HTTP 要求の本文

標準 SOAP エンベロープの一部として変数を含めることによって、SOAP サービスタイプに変数を指定できます。

URL による Web サービスへの変数の指定

HTTP Web サーバは、Web ブラウザによって URL で指定された変数を管理できます。これらの変数は、次のいずれかの表記規則で表すことができます。

- 次の例のように、URL の末尾に変数を追加し、各パラメータ値をスラッシュ (/) で区切ります。

  ```
  http://localhost/database-name/param1/param2/param3
  ```

- 次の例のように、変数を URL パラメータリストで明示的に定義します。

```
http://localhost/database-name/?
arg1=param1&arg2=param2&arg3=param3
```

- 次の例のように、変数の URL への追加とパラメータリストでの定義を組み合わせます。

```
http://localhost/database-name/param4/param5?
arg1=param1&arg2=param2&arg3=param3
```

Web サーバによる URL の解釈は、Web サービス URL 句の指定方法によって異なります。

HTTP 要求の本文での変数の指定

Web クライアント関数またはプロシージャの TYPE 句で HTTP:POST を指定して、HTTP 要求の本文で変数を指定できます。

デフォルトで、TYPE HTTP:POST では、application/x-www-form-urlencoded MIME タイプが使用されます。すべてのパラメータは urlencoded であり、要求の本文内で渡されます。オプションで、メディアタイプが指定されている場合、Content-Type 要求ヘッダは指定されたメディアタイプに自動的に調整され、要求の本文内で 1 つのパラメータ値がアップロードされます。

例

次の例では、`XMLService` という Web サービスが `localhost` Web サーバ上に存在すると想定しています。SAP Sybase IQ クライアントデータベースを設定して Interactive SQL を通じて接続し、次の SQL 文を実行します。

```
CREATE PROCEDURE SendXMLContent(xmlcode LONG VARCHAR)
    URL 'http://localhost/XMLService'
    TYPE 'HTTP:POST:text/xml';
```

この文で作成されるプロシージャを使用すると、HTTP 要求の本文で text/xml フォーマットで変数を送信できます。

Interactive SQL で次の SQL 文を実行して、HTTP 要求を `XMLService` Web サービスに送信します。

```
CALL SendXMLContent('<title>Hello World!</title>');
```

プロシージャコールによって `xmlcode` パラメータに値が割り当てられ、Web サービスに送信されます。

SOAP エンベロープでの変数の指定

Web クライアント関数またはプロシージャの SET SOAP オプションを使用して SOAP エンベロープで変数を指定し、SOAP 操作を設定できます。

次のコードは、Web クライアント関数で SOAP 操作を設定する方法を示しています。

```
CREATE FUNCTION soapAddItemFunc(amount INT, item LONG VARCHAR)
    RETURNS XML
```

```
    URL 'http://localhost:8082/itemStore'
    SET 'SOAP(OP=addItems)'
    TYPE 'SOAP:DOC';
```

この例では、addItems は、soapAddItemFunc 関数にパラメータとして渡される amount 値と item 値を含む SOAP 操作です。

次のサンプルスクリプトを実行して要求を送信できます。

```
SELECT soapAddItemFunc(5, 'shirt');
```

soapAddItemFunc ファンクションコールの呼び出しによって、次のような SOAP エンベロープが生成されます。

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:addItems>
      <m:amount>5</m:amount>
      <m:item>shirt</m:item>
    </m:addItems>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

前の方法の代わりに、独自の SOAP ペイロードを作成し、HTTP ラッパーでサーバに送信できます。

SOAP サービスに対する変数は、標準 SOAP 要求の一部として含める必要があります。これ以外の方法で提供される値は無視されます。

次のコードは、カスタマイズされた SOAP エンベロープを構築する HTTP ラッパープロシージャを作成する方法を示しています。

```
CREATE PROCEDURE addItemHttpWrapper(amount INT, item LONG VARCHAR)
RESULT(response XML)
BEGIN
    DECLARE payload XML;
    DECLARE response XML;

    SET payload =
'<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:addItems>
      <m:amount>' || amount || '</m:amount>
      <m:item>' || item || '</m:item>
    </m:addItems>
```

```
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>';

    SET response = httpAddItemFunc( payload );
    /* process response as demonstrated in addItemFuncWrapper */
    SELECT response;
END;
```

次のコードは、要求の送信に使用する Web クライアント関数を示しています。

```
CREATE FUNCTION httpAddItemFunc(soapPayload XML)
    RETURNS XML
    URL 'http://localhost:8082/itemStore'
    TYPE 'HTTP:POST:text/xml'
    HEADER 'SOAPAction: "http://localhost:8082/addItems"';
```

次のサンプルスクリプトを実行して要求を送信できます。

```
CALL addItemHttpWrapper( 7, 'socks' );
```

結果セットからの変数へのアクセス

Web サービスクライアント呼び出しは、ストアド関数またはプロシージャで実行できます。関数を使用した場合、戻り値のタイプは CHAR、VARCHAR、LONG VARCHAR などの文字データ型です。返される値は、HTTP 応答の本文です。ヘッダ情報は含まれません。HTTP ステータス情報を含む要求に関する追加情報は、プロシージャによって返されます。したがって、追加情報にアクセスする場合は、プロシージャの使用をおすすめします。

SOAP プロシージャ

SOAP 関数からは SOAP 応答を含んだ XML ドキュメントが返されます。

SOAP 応答は構造化された XML ドキュメントであるため、デフォルトでは SAP Sybase IQ はこの情報を利用してさらに役立つ結果セットを作成しようとします。返された応答ドキュメント内の最上位レベルの各タグが抽出され、カラム名として使用されます。これらのタグのそれぞれの下にあるサブツリーの内容は、そのカラムのローの値として使用されます。

たとえば、次の SOAP 応答が返される場合、SAP Sybase IQ は次のデータセットを作成します。

```
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
     <ElizaResponse xmlns:SOAPSDK4="SoapInterop">
        <Eliza>Hi, I'm Eliza. Nice to meet you.</Eliza>
     <ElizaResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Eliza
Hi, I'm Eliza.Nice to meet you.

この例では、応答ドキュメントは <SOAP-ENV:Body> タグ内にある <ElizaResponse> タグによって区切られています。

結果セットには、最上位レベルのタグの数だけのカラムが含まれます。SOAP 応答には最上位レベルのタグが 1 つしかないため、この結果セットのカラムは 1 つだけです。この最上位レベルタグである Eliza が、カラム名となります。

XML 処理機能

SOAP 応答を含む XML 結果セット内の情報には、OPENXML プロシージャを使用してアクセスできます。

次の例では、OPENXML プロシージャを使用して SOAP 応答の一部を抽出します。この例は、SYSWEBSERVICE テーブルの内容を公開するために SOAP サービスとして Web サービスを使用しています。

```
CREATE SERVICE get_webservices
    TYPE 'SOAP'
    AUTHORIZATION OFF
    USER DBA
    AS SELECT * FROM SYSWEBSERVICE;
```

2 番目の SAP Sybase IQ データベースで作成する次の Web クライアント関数は、この Web サービスへの呼び出しを発行します。この関数の戻り値は、SOAP 応答ドキュメント全体です。`DNET` がデフォルトの SOAP サービスフォーマットであるため、応答は .NET `DataSet` フォーマットになります。

```
CREATE FUNCTION get_webservices()
    RETURNS LONG VARCHAR
    URL 'HTTP://localhost/get_webservices'
    TYPE 'SOAP:DOC';
```

次の文は、OPENXML プロシージャを使用して結果セットの 2 つのカラムを抽出する方法を示しています。`service_name` カラムおよび `secure_required` カラムは、セキュアな SOAP サービスと、HTTPS を必要としている場所をそれぞれ示します。

```
SELECT *
FROM OPENXML( get_webservices(), '//row' )
WITH ("Name"    CHAR(128) 'service_name',
      "Secure?" CHAR(1)   'secure_required' );
```

この文は、`row` ノードの子孫を選択することによって機能します。WITH 句は、目的の 2 つの要素に基づき、結果セットを作成します。`get_webservices` Web サービスのみが存在すると想定し、この関数は以下の結果セットを返します。

Name	Secure?
get_webservices	N

Web サービスからの結果セットの取得

タイプ HTTP の Web サービスプロシージャは、応答に関する全情報を 2 つのカラムから成る結果セットで返します。この結果セットには、応答ステータス、ヘッダ情報、および本文が含まれます。最初のカラムには `Attribute`、2 番目のカラムには `Value` という名前が付けられています。どちらも LONG VARCHAR データ型です。

結果セットには、応答ヘッダフィールドごとに 1 ロー、HTTP ステータス行 (Status 属性) に対して 1 ロー、応答本文 (Body 属性) に対して 1 ローが含まれます。

次の例は、一般的な応答を示します。

属性	値
Status	HTTP /1.0 200 OK
Body	<!DOCTYPE HTML ...><HTML> ...</HTML>
Content-Type	text/html
Server	GWS/2.1
Content-Length	2234
Date	Mon, 18 Oct 2004, 16:00:00 GMT

例として使用する次の Web サービスストアドプロシージャを作成します。

```
CREATE OR REPLACE PROCEDURE SybaseWebPage()
URL 'http://www.sybase.com/mobilize'
TYPE 'HTTP';
```

結果セットとして Web サービスから応答を取得するには、次の SELECT クエリを実行します。

```
SELECT * FROM SybaseWebPage()
    WITH (Attribute LONG VARCHAR, Value LONG VARCHAR);
```

Web サービスプロシージャでは結果セットの形式を記述しないので、テンポラリビューを定義するには WITH 句が必要です。

クエリの結果をテーブルに格納できます。テーブルを作成して結果セットの値を格納するには、次の SQL 文を実行します。

```
CREATE TABLE StoredResults(
     Attribute LONG VARCHAR,
     Value     LONG VARCHAR
);
```

結果セットは、次のように StoredResults テーブルに挿入します。

```
INSERT INTO StoredResults
    SELECT * FROM SybaseWebPage()
    WITH (Attribute LONG VARCHAR, Value LONG VARCHAR);
```

SELECT 文の通常の構文に従い、句を追加できます。たとえば、結果セットの特定のローのみが必要な場合は、WHERE 句を追加して SELECT の結果を 1 つのローに限定することができます。

```
SELECT * FROM SybaseWebPage()
    WITH (Attribute LONG VARCHAR, Value LONG VARCHAR)
    WHERE Attribute = 'Status';
```

この SELECT 文は、結果セットからステータス情報のみを取得します。この文は呼び出しが成功したことを確認するために使用できます。

SOAP のデータ型

デフォルトでは、パラメータ入力の XML エンコードは String 型であり、SOAP サービスフォーマットの結果セット出力には、結果セット内のカラムのデータ型について具体的に記述する情報がまったく含まれていません。すべてのフォーマットで、パラメータのデータ型は String です。DNET フォーマットの場合、応答のスキーマセクション内ですべてのカラムは String として型指定されています。CONCRETE フォーマットと XML フォーマットの場合は、応答にデータ型情報が含まれません。このデフォルトの動作は、DATATYPE 句を使用して操作できます。

SAP Sybase IQ では、DATATYPE 句を使用してデータ型指定を有効にします。データ型情報は、すべての SOAP サービスフォーマットでパラメータ入力と結果セット出力 (応答) の XML エンコードに含めることができます。これにより、パラメータを String に明示的に変換するクライアントコードが不要になるため、SOAP ツールキットからのパラメータ受け渡しが簡単になります。たとえば整数は int として渡すことができます。XML コード化されたデータ型では SOAP ツールキットを使用してデータを解析し、適切な型にキャストします。

String データ型を排他的に使用する場合、アプリケーションでは結果セット内の各カラムのデータ型を暗黙的にわかっている必要があります。データ型指定が Web サーバで要求される場合は、必要ありません。データ型情報が含まれるかどうかを制御するために、Web サービスの定義時に DATATYPE 句を使用できます。

結果セット応答にデータ型指定を含めるようにする Web サービス定義の例を次に示します。

```
CREATE SERVICE "SASoapTest/EmployeeList"
    TYPE 'SOAP'
    AUTHORIZATION OFF
    SECURE OFF
    USER DBA
```

```
    DATATYPE OUT
    AS SELECT * FROM Employees;
```

この例では、サービスにはパラメータがないため、データ型情報は結果セット応答のみに対して要求されます。

型指定は、'SOAP' 型として定義されているすべての SAP Sybase IQ Web サービスに適用できます。

入力パラメータのデータ型指定
入力パラメータの型指定は、パラメータのデータ型を実際のデータ型として DISH サービスで生成される WSDL で公開するだけでサポートされます。

一般的な String パラメータ定義 (または型指定されていないパラメータ) は次のようになります。

```
<s:element minOccurs="0" maxOccurs="1" name="a_varchar"
nillable="true" type="s:string" />
```

String パラメータは nil 可能な場合があります。つまり、出現することもしないこともあります。

整数などの型指定されたパラメータの場合、そのパラメータは出現する必要があり、nil 可能ではありません。次はその例です。

```
<s:element minOccurs="1" maxOccurs="1" name="an_int"
nillable="false" type="s:int" />
```

出力パラメータのデータ型指定
'SOAP' 型であるすべての SAP Sybase IQ Web サービスでは、応答データ内のデータ型情報を公開できます。データ型は、ローセットカラム要素内の属性として公開されます。

SOAP FORMAT 'CONCRETE' Web サービスからの型指定された SimpleDataSet 応答の例を次に示します。

```
<SOAP-ENV:Body>
  <tns:test_types_concrete_onResponse>
   <tns:test_types_concrete_onResult xsi:type='tns:SimpleDataset'>
    <tns:rowset>
      <tns:row>
       <tns:lvc xsi:type="xsd:string">Hello World</tns:lvc>
       <tns:i xsi:type="xsd:int">99</tns:i>
       <tns:ii xsi:type="xsd:long">99999999</tns:ii>
       <tns:f xsi:type="xsd:float">3.25</tns:f>
       <tns:d xsi:type="xsd:double">.555555555555555582</tns:d>
       <tns:bin xsi:type="xsd:base64Binary">AAAAZg==</tns:bin>
       <tns:date xsi:type="xsd:date">2006-05-29-04:00</tns:date>
      </tns:row>
     </tns:rowset>
   </tns:test_types_concrete_onResult>
  <tns:sqlcode>0</tns:sqlcode>
```

```
  </tns:test_types_concrete_onResponse>
</SOAP-ENV:Body>
```

XML データを String として返す SOAP FORMAT 'XML' Web サービスからの応答の例を次に示します。内部ローセットは、コード化された XML で構成されます。ここでは、わかりやすいようにデコードされた形式で示されています。

```
<SOAP-ENV:Body>
 <tns:test_types_XML_onResponse>
  <tns:test_types_XML_onResult xsi:type='xsd:string'>
   <tns:rowset
     xmlns:tns="http://localhost/satest/dish"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <tns:row>
     <tns:lvc xsi:type="xsd:string">Hello World</tns:lvc>
     <tns:i xsi:type="xsd:int">99</tns:i>
     <tns:ii xsi:type="xsd:long">99999999</tns:ii>
     <tns:f xsi:type="xsd:float">3.25</tns:f>
     <tns:d xsi:type="xsd:double">.55555555555555582</tns:d>
     <tns:bin xsi:type="xsd:base64Binary">AAAAZg==</tns:bin>
     <tns:date xsi:type="xsd:date">2006-05-29-04:00</tns:date>
    </tns:row>
   </tns:rowset>
  </tns:test_types_XML_onResult>
  <tns:sqlcode>0</tns:sqlcode>
 </tns:test_types_XML_onResponse>
</SOAP-ENV:Body>
```

要素のネームスペースと XML スキーマでは、データ型情報だけでなく、XML パーサによる後処理に必要なすべての情報を提供します。データ型情報が結果セットに存在しない場合 (DATATYPE OFF または IN)、xsi:type と XML スキーマのネームスペース宣言は省略されます。

型指定された SimpleDataSet を返す SOAP FORMAT 'DNET' Web サービスの例を次に示します。

```
<SOAP-ENV:Body>
  <tns:test_types_dnet_outResponse>
   <tns:test_types_dnet_outResult
xsi:type='sqlresultstream:SqlRowSet'>
    <xsd:schema id='Schema2'
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'
       xmlns:msdata='urn:schemas-microsoft.com:xml-msdata'>
     <xsd:element name='rowset' msdata:IsDataSet='true'>
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element name='row' minOccurs='0' maxOccurs='unbounded'>
         <xsd:complexType>
          <xsd:sequence>
          <xsd:element name='lvc' minOccurs='0' type='xsd:string' />
           <xsd:element name='ub' minOccurs='0'
type='xsd:unsignedByte' />
           <xsd:element name='s' minOccurs='0' type='xsd:short' />
           <xsd:element name='us' minOccurs='0'
```

```
type='xsd:unsignedShort' />
           <xsd:element name='i' minOccurs='0' type='xsd:int' />
           <xsd:element name='ui' minOccurs='0'
type='xsd:unsignedInt' />
           <xsd:element name='l' minOccurs='0' type='xsd:long' />
           <xsd:element name='ul' minOccurs='0'
type='xsd:unsignedLong' />
           <xsd:element name='f' minOccurs='0' type='xsd:float' />
           <xsd:element name='d' minOccurs='0' type='xsd:double' />
           <xsd:element name='bin' minOccurs='0'
type='xsd:base64Binary' />
           <xsd:element name='bool' minOccurs='0'
type='xsd:boolean' />
         <xsd:element name='num' minOccurs='0' type='xsd:decimal' />
          <xsd:element name='dc' minOccurs='0' type='xsd:decimal' />
           <xsd:element name='date' minOccurs='0' type='xsd:date' />
          </xsd:sequence>
         </xsd:complexType>
        </xsd:element>
       </xsd:sequence>
      </xsd:complexType>
     </xsd:element>
    </xsd:schema>
    <diffgr:diffgram xmlns:msdata='urn:schemas-microsoft-com:xml-
msdata' xmlns:diffgr='urn:schemas-microsoft-com:xml-diffgram-v1'>
     <rowset>
      <row>
       <lvc>Hello World</lvc>
       <ub>128</ub>
       <s>-99</s>
       <us>33000</us>
       <i>-2147483640</i>
       <ui>4294967295</ui>
       <l>-9223372036854775807</l>
       <ul>18446744073709551615</ul>
       <f>3.25</f>
       <d>.555555555555555582</d>
       <bin>QUJD</bin>
       <bool>1</bool>
       <num>123456.123457</num>
       <dc>-1.756000</dc>
       <date>2006-05-29-04:00</date>
      </row>
     </rowset>
    </diffgr:diffgram>
   </tns:test_types_dnet_outResult>
  <tns:sqlcode>0</tns:sqlcode>
  </tns:test_types_dnet_outResponse>
</SOAP-ENV:Body>
```

SAP Sybase IQ の型から XML スキーマの型へのマッピング

SAP Sybase IQ の型	XML スキーマの型	XML の例
CHAR	string	Hello World
VARCHAR	string	Hello World
LONG VARCHAR	string	Hello World
TEXT	string	Hello World
NCHAR	string	Hello World
NVARCHAR	string	Hello World
LONG NVARCHAR	string	Hello World
NTEXT	string	Hello World
UNIQUEIDENTIFIER	string	12345678-1234-5678-9012-123456789012
UNIQUEIDENTI-FIERSTR	string	12345678-1234-5678-9012-123456789012
XML	これはユーザ定義の型である。パラメータは、複合型 (base64Binary、SOAP 配列、struct など) を表す有効な XML であることが想定される。	<inputHexBinary xsi:type="xsd:hexBinary"> 414243 </inputHexBinary> ('ABC' と解釈される)
BIGINT	long	-9223372036854775807
UNSIGNED BIGINT	unsignedLong	18446744073709551615
BIT	boolean	1
VARBIT	string	11111111
LONG VARBIT	string	00000000000000001000000000000000
DECIMAL	decimal	-1.756000
DOUBLE	double	.555555555555555582
FLOAT	float	12.3456792831420898

SAP Sybase IQ の型	XML スキーマの型	XML の例
INTEGER	int	-2147483640
UNSIGNED INTEGER	unsignedInt	4294967295
NUMERIC	decimal	123456.123457
REAL	float	3.25
SMALLINT	short	-99
UNSIGNED SMALLINT	unsignedShort	33000
TINYINT	unsignedByte	128
MONEY	decimal	12345678.9900
SMALLMONEY	decimal	12.3400
DATE	date	2006-11-21-05:00
DATETIME	dateTime	2006-05-21T09:00:00.000-08:00
SMALLDATETIME	dateTime	2007-01-15T09:00:00.000-08:00
TIME	time	14:14:48.980-05:00
TIMESTAMP	dateTime	2007-01-12T21:02:14.420-06:00
TIMESTAMP WITH TIME ZONE	dateTime	2007-01-12T21:02:14.420-06:00
BINARY	base64Binary	AAAAZg==
IMAGE	base64Binary	AAAAZg==
LONG BINARY	base64Binary	AAAAZg==
VARBINARY	base64Binary	AAAAZg==

1 つまたは複数のパラメータが NCHAR、NVARCHAR、LONG NVARCHAR、NTEXT のいずれかの型の場合、応答は UTF8 で出力されます。クライアントデータベースが UTF-8 文字コードを使用している場合は動作に変更はありません (NCHAR と CHAR のデータ型は同一であるため)。ただし、データベースが UTF-8 文字コードを使用していない場合は、NCHAR 以外のデータ型のパラメータはすべて UTF8 に変換されます。XML 宣言エンコードおよび HTTP ヘッダ Content-Type の値は、使用される文字コードに対応します。

XML スキーマの型から Java の型へのマッピング

XML スキーマの型	Java データ型
xsd:string	java.lang.String
xsd:integer	java.math.BigInteger
xsd:int	int
xsd:long	long
xsd:short	short
xsd:decimal	java.math.BigDecimal
xsd:float	float
xsd:double	double
xsd:boolean	boolean
xsd:byte	byte
xsd:QName	javax.xml.namespace.QName
xsd:dateTime	javax.xml.datatype.XMLGregorianCalendar
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]
xsd:unsignedInt	long
xsd:unsignedShort	int
xsd:unsignedByte	short
xsd:time	javax.xml.datatype.XMLGregorianCalendar
xsd:date	javax.xml.datatype.XMLGregorianCalendar
xsd:g	javax.xml.datatype.XMLGregorianCalendar
xsd:anySimpleType	java.lang.Object
xsd:anySimpleType	java.lang.String
xsd:duration	javax.xml.datatype.Duration
xsd:NOTATION	javax.xml.namespace.QName

SOAP の構造化されたデータ型

Web サービスクライアントとしての SAP Sybase IQ サーバは、関数やプロシージャを使用して Web サービスに対するインタフェースになることがあります。

XML 戻り値

単純な戻り値のデータ型には、結果セット内の文字列表現で十分な場合があります。この場合、ストアドプロシージャの使用が可能になります。

配列や構造体などの複雑なデータを返すときは、Web サービス関数を使用する方が適しています。関数の宣言では、RETURN 句で XML データ型を指定できます。目的の要素を抽出するために、返された XML は OPENXML を使用して解析することができます。

dateTime などの XML データの戻り値は、結果セット内にそのままで現れます。たとえば TIMESTAMP カラムが結果セットに含まれる場合は、文字列 (2006-12-25 12:00:00.000) ではなく、XML dateTime 文字列 (2006-12-25T12:00:00.000-05:00) のようにフォーマットされます。

XML パラメータ値

SAP Sybase IQ XML データ型は、Web サービス関数とプロシージャ内のパラメータとして使用できます。単純な型の場合、SOAP 要求の本文が生成されるときに、パラメータ要素が自動的に構成されます。しかし、XML パラメータタイプの場合、要素の XML 表現で追加のデータを提供する属性が必要になることがあるため、自動的に構成できません。そのため、データ型が XML のパラメータに対して XML を生成するときは、ルート要素の名前をパラメータ名と一致させる必要があります。

```
<inputHexBinary xsi:type="xsd:hexBinary">414243</inputHexBinary>
```

この XML 型は、パラメータを hexBinary XML 型として送信する方法の例を示しています。SOAP 終了ポイントは、パラメータ名 (XML 用語ではルート要素名) が inputHexBinary であることを想定しています。

Cookbook 定数

複雑な構造体や配列を構築するには、SAP Sybase IQ がネームスペースを参照する方法を知ることが必要です。次に示すプレフィクスは、SAP Sybase IQ SOAP 要求エンベロープ用に生成されるネームスペース宣言に対応します。

SAP Sybase IQ の XML プレフィクス	ネームスペース
xsd	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance

SAP Sybase IQ の XML プレフィクス	ネームスペース
SOAP-ENC	http://schemas.xmlsoap.org/soap/encoding/
m	NAMESPACE 句で定義されたネームスペース

複雑なデータ型の例
配列、構造体、構造体の配列をそれぞれ表すパラメータを取る Web サービスクライアント関数を作成する方法を次の 3 つの例で示します。Web サービス関数は、それぞれ echoFloatArray、echoStruct、echoStructArray という SOAP 操作 (または RPC 関数名) と通信します。相互運用性テストで共通で使用されるネームスペースは http://soapinterop.org/ で、URL 句を目的の SOAP 終了ポイントに変更するだけで、関数を代替相互運用サーバに対してテストすることができます。

これらの例は、Microsoft SOAP ToolKit 3.0 Round 2 相互運用性テストサーバ (http://mssoapinterop.org/stkV3) に対して要求を発行するように設計されています。

これらの例では、テーブルを使用して XML データを生成します。このテーブルを設定する方法は次のとおりです。

```
CREATE LOCAL TEMPORARY TABLE SoapData
(
    seqno INT DEFAULT AUTOINCREMENT,
    i INT,
    f FLOAT,
    s LONG VARCHAR
) ON COMMIT PRESERVE ROWS;

INSERT INTO SoapData (i,f,s)
VALUES (99,99.999,'Ninety-Nine');

INSERT INTO SoapData (i,f,s)
VALUES (199,199.999,'Hundred and Ninety-Nine');
```

次の 3 つの関数は、SOAP 要求を相互運用サーバに送信します。このサンプルでは、Microsoft の Interop サーバに対して要求を発行します。

```
CREATE FUNCTION echoFloatArray(inputFloatArray XML)
RETURNS XML
URL 'http://mssoapinterop.org/stkV3/Interop.wsdl'
HEADER 'SOAPAction:"http://soapinterop.org/"'
NAMESPACE 'http://soapinterop.org/';

CREATE FUNCTION echoStruct(inputStruct XML)
RETURNS XML
URL 'http://mssoapinterop.org/stkV3/Interop.wsdl'
HEADER 'SOAPAction:"http://soapinterop.org/"'
NAMESPACE 'http://soapinterop.org/';

CREATE FUNCTION echoStructArray(inputStructArray XML)
RETURNS XML
```

```
URL 'http://mssoapinterop.org/stkV3/Interop.wsdl'
HEADER 'SOAPAction:"http://soapinterop.org/"'
NAMESPACE 'http://soapinterop.org/';
```

最後に、例の文を 3 つ示します。それぞれのパラメータは XML 表現で表されています。

1. 次の例のパラメータは、配列を表します。

```
SELECT echoFloatArray(
    XMLELEMENT( 'inputFloatArray',
       XMLATTRIBUTES( 'xsd:float[2]' as "SOAP-ENC:arrayType" ),
        (
        SELECT XMLAGG( XMLELEMENT( 'number', f ) ORDER BY seqno )
        FROM SoapData
        )
    )
);
```

ストアドプロシージャ echoFloatArray は、次の XML を相互運用サーバに送信します。

```
<inputFloatArray SOAP-ENC:arrayType="xsd:float[2]">
<number>99.9990005493164</number>
<number>199.998992919922</number>
</inputFloatArray>
```

相互運用サーバからの応答は次のようになります。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/">
    <SOAPSDK4:echoFloatArrayResponse
      xmlns:SOAPSDK4="http://soapinterop.org/">
      <Result SOAPSDK3:arrayType="SOAPSDK1:float[2]"
        SOAPSDK3:offset="[0]"
        SOAPSDK2:type="SOAPSDK3:Array">
        <SOAPSDK3:float>99.9990005493164</SOAPSDK3:float>
        <SOAPSDK3:float>199.998992919922</SOAPSDK3:float>
      </Result>
    </SOAPSDK4:echoFloatArrayResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

応答が変数に格納された場合は、OPENXML を使用して解析できます。

```
SELECT * FROM OPENXML( resp,'//*:Result/*' )
WITH ( varFloat FLOAT 'text()' );
```

varFloat
99.9990005493
199.9989929199

2. 次の例のパラメータは、構造体を表します。

```
SELECT echoStruct(
    XMLELEMENT('inputStruct',
        (
        SELECT XMLFOREST( s as varString,
                          i as varInt,
                          f as varFloat )
        FROM SoapData
        WHERE seqno=1
        )
    )
);
```

ストアドプロシージャ echoStruct は、次の XML を相互運用サーバに送信します。

```
<inputStruct>
  <varString>Ninety-Nine</varString>
  <varInt>99</varInt>
  <varFloat>99.9990005493164</varFloat>
</inputStruct>
```

相互運用サーバからの応答は次のようになります。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/">
    <SOAPSDK4:echoStructResponse
      xmlns:SOAPSDK4="http://soapinterop.org/">
      <Result href="#id1"/>
    </SOAPSDK4:echoStructResponse>
    <SOAPSDK5:SOAPStruct
      xmlns:SOAPSDK5="http://soapinterop.org/xsd"
      id="id1"
      SOAPSDK3:root="0"
      SOAPSDK2:type="SOAPSDK5:SOAPStruct">
      <varString>Ninety-Nine</varString>
      <varInt>99</varInt>
      <varFloat>99.9990005493164</varFloat>
    </SOAPSDK5:SOAPStruct>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

応答が変数に格納された場合は、OPENXML を使用して解析できます。

```
SELECT * FROM OPENXML( resp,'//*:Body/*:SOAPStruct' )
WITH (
varString LONG VARCHAR 'varString',
varInt INT 'varInt',
varFloat FLOAT 'varFloat' );
```

varString	varInt	varFloat
Ninety-Nine	99	99.9990005493

3. 次の例のパラメータは、構造体の配列を表します。

```
SELECT echoStructArray(
    XMLELEMENT( 'inputStructArray',
       XMLATTRIBUTES( 'http://soapinterop.org/xsd' AS "xmlns:q2",
                      'q2:SOAPStruct[2]' AS "SOAP-ENC:arrayType" ),
            (
            SELECT XMLAGG(
                XMLElement('q2:SOAPStruct',
                XMLFOREST( s as varString,
                           i as varInt,
                           f as varFloat )
                )
            ORDER BY seqno
            )
         FROM SoapData
         )
    )
);
```

ストアドプロシージャ echoFloatArray は、次の XML を相互運用サーバに送信します。

```
<inputStructArray xmlns:q2="http://soapinterop.org/xsd"
  SOAP-ENC:arrayType="q2:SOAPStruct[2]">
  <q2:SOAPStruct>
    <varString>Ninety-Nine</varString>
    <varInt>99</varInt>
    <varFloat>99.9990005493164</varFloat>
  </q2:SOAPStruct>
  <q2:SOAPStruct>
    <varString>Hundred and Ninety-Nine</varString>
    <varInt>199</varInt>
    <varFloat>199.998992919922</varFloat>
  </q2:SOAPStruct>
</inputStructArray>
```

相互運用サーバからの応答は次のようになります。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
  xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
```

```
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/">
    <SOAPSDK4:echoStructArrayResponse
      xmlns:SOAPSDK4="http://soapinterop.org/">
      <Result xmlns:SOAPSDK5="http://soapinterop.org/xsd"
        SOAPSDK3:arrayType="SOAPSDK5:SOAPStruct[2]"
        SOAPSDK3:offset="[0]" SOAPSDK2:type="SOAPSDK3:Array">
        <SOAPSDK5:SOAPStruct href="#id1"/>
        <SOAPSDK5:SOAPStruct href="#id2"/>
      </Result>
    </SOAPSDK4:echoStructArrayResponse>
    <SOAPSDK6:SOAPStruct
      xmlns:SOAPSDK6="http://soapinterop.org/xsd"
      id="id1"
      SOAPSDK3:root="0"
      SOAPSDK2:type="SOAPSDK6:SOAPStruct">
      <varString>Ninety-Nine</varString>
      <varInt>99</varInt>
      <varFloat>99.9990005493164</varFloat>
    </SOAPSDK6:SOAPStruct>
    <SOAPSDK7:SOAPStruct
      xmlns:SOAPSDK7="http://soapinterop.org/xsd"
      id="id2"
      SOAPSDK3:root="0"
      SOAPSDK2:type="SOAPSDK7:SOAPStruct">
      <varString>Hundred and Ninety-Nine</varString>
      <varInt>199</varInt>
      <varFloat>199.998992919922</varFloat>
    </SOAPSDK7:SOAPStruct>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

応答が変数に格納された場合は、OPENXML を使用して解析できます。

```
SELECT * FROM OPENXML( resp,'//*:Body/*:SOAPStruct' )
WITH (
varString LONG VARCHAR 'varString',
varInt INT 'varInt',
varFloat FLOAT 'varFloat' );
```

varString	varInt	varFloat
Ninety-Nine	99	99.9990005493
Hundred and Ninety-Nine	199	199.9989929199

句の値に使用する代入パラメータ

ストアドプロシージャまたは関数の宣言済みパラメータは、そのプロシージャまたは関数が実行されるたびに、句の定義内のプレースホルダを自動的に置き換えます。代入パラメータを使用すると、実行時に動的に句を設定する一般的な Web サービスプロシージャを作成できます。感嘆符 '!' の後に宣言されたパラメータの 1 つの名前が続いている部分文字列はすべて、パラメータの値で置換されます。

こうして、実行時に 1 つ以上のパラメータ値が代入され、1 つ以上の句の値が抽出されます。

パラメータの代入を行うには、次の規則を順守する必要があります。

- 代入に使用するパラメータはすべて英数字にします。アンダースコアは使用できません。
- 代入パラメータの直後は、英数字以外の文字が続くか終了である必要があります。たとえば、`!sizeXL` は、X が英数字であるため、size というパラメータの値で置換されません。
- パラメータ名に一致しない代入パラメータは無視されます。
- 感嘆符 (!) は別の感嘆符でエスケープできます。

たとえば、次のプロシージャは、代入パラメータの使用方法を示します。URL と HTTP ヘッダの定義は、パラメータとして渡します。

```
CREATE PROCEDURE test(uid CHAR(128), pwd CHAR(128), headers LONG
VARCHAR)
    URL 'http://!uid:!pwd@localhost/myservice'
    HEADER '!headers';
```

次の文を使用して `test` プロシージャを呼び出し、HTTP 要求を開始できます。

```
CALL test('dba', 'sql', 'NewHeader1:value1¥nNewHeader2:value2');
```

このプロシージャが呼び出されるたびに、異なる値を使用できます。

暗号化証明書の例
代入パラメータを使用してファイルからストアドプロシージャまたはストアド関数に暗号化証明書を渡すことができます。

次の例は、証明書を代入文字列として渡す方法を示します。

```
CREATE PROCEDURE secure(cert LONG VARCHAR)
URL 'https://localhost/secure'
TYPE 'HTTP:GET'
CERTIFICATE 'cert=!cert;company=test;unit=test;name=RSA Root';
```

証明書は次の呼び出しでファイルから読み込まれて `secure` に渡されます。

```
CALL secure( xp_read_file('%ALLUSERSPROFILE%/SybaseIQ/demo¥
¥Certificates¥¥rsaroot.crt) );
```

この例は説明でのみ使用されます。CERTIFICATE 句の `file=` キーワードを使用して、証明書をファイルから直接読み込むことができます。

一致するパラメータ名がない場合の例
一致するパラメータ名のないプレースホルダは、自動的に削除されます。

たとえば、次のプロシージャでは、パラメータ size はプレースホルダを置換しません。

```
CREATE PROCEDURE orderitem (size CHAR(18))
    URL 'HTTP://localhost/salesserver/order?size=!sizeXL'
    TYPE 'SOAP:RPC';
```

この例では、!sizeXL は一致するパラメータがないプレースホルダであるため、常に削除されます。

パラメータはストアド関数またはストアドプロシージャの呼び出し時に、それらの本文内のプレースホルダを置換するためにも使用できます。特定の変数のプレースホルダが存在しない場合、パラメータとその値が要求の一部として渡されます。このようにして代入に使用されるパラメータと値は、要求の一部として渡されません。

HTTP 要求と SOAP 要求の構造

パラメータの代入中に使用する場合を除き、関数またはプロシージャのすべてのパラメータは、Web サービス要求の一部として渡されます。渡されるときのフォーマットは、Web サービス要求のタイプによって異なります。

文字またはバイナリデータ型でないパラメータ値は、要求に追加する前に文字列表現に変換されます。この処理は、値を文字型にキャストすることに相当します。変換は、関数またはプロシージャの呼び出し時に、データ型のフォーマットオプションの設定に従って行われます。具体的には、変換は precision、scale、timestamp_format などのオプションによって影響されます。

HTTP 要求の構造

HTTP:GET タイプのパラメータは URL コード化され、URL 内に配置されます。パラメータ名は、HTTP 変数の名前としてそのまま使用されます。たとえば、次のプロシージャは 2 つのパラメータを宣言します。

```
CREATE PROCEDURE test(a INTEGER, b CHAR(128))
    URL 'HTTP://localhost/myservice'
    TYPE 'HTTP:GET';
```

123 と 'xyz' という値を使ってこのプロシージャを呼び出す場合、要求に使用する URL は次に示したものと同等になります。

```
HTTP://localhost/myservice?a=123&b=xyz
```

タイプが HTTP:POST である場合、パラメータとその値は URL コード化され、要求の本文内に配置されます。2 つのパラメータと値の場合、ヘッダの後に、次のテキストが HTTP 要求の本文に表示されます。

```
a=123&b=xyz
```

SOAP 要求の構造

SOAP 要求に渡されたパラメータは、SOAP 仕様で指定されているように、要求本文の一部としてひとまとめにされます。

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Body>
    <m:test>
      <m:a>123</m:a>
      <m:b>abc</m:b>
    </m:test>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Web クライアント要求のロギング方法

HTTP 要求やトランスポートデータを含む Web サービスクライアントの情報は、Web サービスクライアントログファイルに記録できます。Web サービスクライアントログファイルは、-zoc サーバオプションまたは sa_server_option システムプロシージャを使用して指定できます。

```
CALL sa_server_option( 'WebClientLogFile', 'clientinfo.txt' );
```

-zoc サーバオプションを指定すると、ロギングが自動的に有効になります。このファイルへのロギングの有効／無効を切り替えるには、sa_server_option システムプロシージャを使用します。

```
CALL sa_server_option( 'WebClientLogging', 'ON' );
```

Web サービスリファレンス

この項では、Web サービスリファレンスについて説明します。

Web サービスエラーコードリファレンス

要求に失敗すると、HTTP サーバで標準の Web サービスエラーが生成されます。これらのエラーには、プロトコル標準と一貫性のある番号が割り当てられています。

発生する可能性のある一般的なエラーは次のとおりです。

番号	名前	SOAP フォールト	説明
301	Moved permanently	Server	要求されたページは永続的に移動された。サーバは自動的に、新しいロケーションに要求をリダイレクトする。

番号	名前	SOAP フォールト	説明
304	Not Modified	Server	サーバは、要求の情報に基づき、要求されたデータは前回の要求の後変更されていないため、再度送信する必要はないと判断した。
307	Temporary Redirect	Server	要求されたページは移動されたが、この変更は永続的なものではない可能性がある。サーバは自動的に、新しいロケーションに要求をリダイレクトする。
400	Bad Request	Client.BadRequest	HTTP 要求が正しくないか不正である。
401	Authorization Required	Client.Authorization	サービスを使用するのに認証が必要であるが、有効なユーザ名とパスワードが入力されていない。
403	Forbidden	Client.Forbidden	データベースにアクセスするパーミッションがない。
404	Not Found	Client.NotFound	指定したデータベースがサーバで実行されていないか、指定した Web サービスが存在しない。
408	Request Timeout	Server.RequestTimeout	要求の受信中に最大接続アイドル時間が超過した。
411	HTTP Length Required	Client.LengthRequired	サーバは、クライアントが要求に Content-Length の指定を含めることを必要とする。通常、このエラーはデータをサーバにアップロードしているときに発生する。
413	Entity Too Large	Server	要求が最大許可サイズを超過した。
414	URI Too Large	Server	URI の長さが最大長を超過した。
500	Internal Server Error	Server	内部エラーが発生した。要求が処理できなかった。
501	Not Implemented	Server	HTTP 要求メソッドが GET、HEAD、または POST ではない。
502	Bad Gateway	Server	要求されたドキュメントがサードパーティのサーバにあり、サーバがサードパーティのサーバからエラーを受け取った。

番号	名前	SOAP フォールト	説明
503	Service Unavailable	Server	接続数が最大数を超過した。

SOAP サービスが失敗すると、次の SOAP バージョン 1.1 標準で定義されているように、フォールトがクライアントに対して SOAP フォールトとして返されます。

- 要求を処理するアプリケーションのエラーによって SQLCODE が生成されると、クライアントの faultcode により SOAP フォールトが返されます。その場合、Procedure などのサブカテゴリが含まれることもあります。SOAP フォールト内の faultstring 要素には、エラーの詳しい説明が設定され、detail 要素には、数値の SQLCODE 値が指定されます。
- トランスポートプロトコルエラーが発生した場合、faultcode はエラーに応じて Client または Server に設定され、faultstring には「`404 Not Found`」などの HTTP トランスポートメッセージが設定され、detail 要素には数値の HTTP エラー値が設定されます。
- SQLCODE 値を返すアプリケーションエラーのために生成された SOAP フォールトメッセージは、「`200 OK`」という HTTP ステータスで返されます。

クライアントを SOAP クライアントとして識別できない場合は、生成された HTML ドキュメントで適切な HTTP エラーが返されます。

HTTP Web サービスの例

Web サービスの実装サンプルのいくつかは、`%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥HTTP` フォルダにあります。サンプルの詳細については、`%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥HTTP¥readme.txt` を参照してください。

チュートリアル： Web サーバを作成して Web クライアントからアクセス

このチュートリアルでは、SQL Anywhere データベースサーバを使用して Web サーバを作成し、Web クライアントデータベースサーバから要求を送信する方法について説明します。

必須ソフトウェア

- SAP Sybase IQ

前提知識と経験

- XML の知識
- MIME (Multipurpose Internet Mail Extensions) タイプの知識
- SAP Sybase IQ Web サービスの基本的な知識

目的

- 新しい SAP Sybase IQ Web サーバデータベースを作成し、起動します。
- Web サービスを作成します。
- HTTP 要求に含まれている情報を返すプロシージャを設定します。
- 新しい SAP Sybase IQ Web クライアントデータベースを作成し、起動します。
- HTTP:POST 要求を、Web クライアントからデータベースサーバに送信します。
- HTTP 応答を、Web サーバから Web クライアントに送信します。

権限
このチュートリアルのレッスンを実行するには、次の権限が必要です。

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

レッスン 1： 要求を受信して応答を送信する Web サーバの設定

このレッスンの目標は、Web サービスが実行されている SAP Sybase IQ Web サーバを設定することです。

前提条件

このレッスンでは、このチュートリアル (Web サーバを作成して Web クライアントからアクセス) の開始時に、権限のセクションで一覧されているロールと権限を持っていることを前提としています。

手順

1. Web サービス定義を含めるために使用する SAP Sybase IQ データベースを作成します。
   ```
   iqinit -dba <user_id>,<password> echo
   ```
2. このデータベースを使用して、ネットワークデータベースサーバを起動します。このサーバは Web サーバとして動作します。
   ```
   iqsrv16 -xs http(port=8082) -n echo echo.db
   ```

HTTP Web サーバは、ポート 8082 で要求を受信するように設定されます。ネットワークで 8082 が許可されない場合は、異なるポート番号を使用します。

3. Interactive SQL を使用して、データベースサーバに接続します。

```
dbisql -c "UID=<user_id>;PWD=<password>;SERVER=echo"
```

4. 着信要求を受け入れる新しい Web サービスを作成します。

```
CREATE SERVICE EchoService
TYPE 'RAW'
USER DBA
AUTHORIZATION OFF
SECURE OFF
AS CALL Echo();
```

この文は、Web クライアントがサービスに要求を送ると、Echo という名前のストアドプロシージャを呼び出す EchoService という名前の新しいサービスを作成します。Web クライアントのフォーマットがない (RAW) HTTP 応答本文を生成します。

5. 着信要求を処理する Echo プロシージャを作成します。

```
CREATE OR REPLACE PROCEDURE Echo()
BEGIN
    DECLARE request_body LONG VARCHAR;
    DECLARE request_mimetype LONG VARCHAR;

    SET request_mimetype = http_header( 'Content-Type' );
    SET request_body = isnull( http_variable('text'),
http_variable('body') );
    IF request_body IS NULL THEN
        CALL sa_set_http_header('Content-Type', 'text/plain' );
        SELECT 'failed'
    ELSE
        CALL sa_set_http_header('Content-Type',
request_mimetype );
        SELECT request_body;
    END IF;
END
```

このプロシージャは、Content-Type ヘッダと Web クライアントに送信される応答の本文をフォーマットします。

Web サーバは要求を受信して応答を送信するように設定されます。

次のステップ

「レッスン 2: Web クライアントからの要求の送信と応答の受信」に進みます。

レッスン 2: Web クライアントからの要求の送信と応答の受信

このレッスンでは、POST メソッドを使用して要求を Web サーバに送信し、Web サーバの応答を受信するデータベースクライアントを設定します。

前提条件

このレッスンは、レッスン 1 で説明したように Web サーバが設定されていることを前提としています。

このレッスンでは、このチュートリアル (Web サーバを作成して Web クライアントからアクセス) の開始時に、権限のセクションで一覧されているロールと権限を持っていることを前提としています。

手順

このレッスンには、`localhost` への複数の参照が含まれています。Web クライアントを Web サーバと同じコンピュータで実行していない場合は、`localhost` の代わりにレッスン 1 の Web サーバのホスト名または IP アドレスを使用します。

1. Web クライアントのプロシージャを含めるために使用する SAP Sybase IQ データベースを作成します。

   ```
   iqinit -dba <user_id>,<password> echo_client
   ```

2. このデータベースを使用して、ネットワークデータベースサーバを起動します。このサーバは Web クライアントとして動作します。

   ```
   iqsrv16 echo_client.db
   ```

3. Interactive SQL を使用して、データベースサーバに接続します。

   ```
   dbisql -c "UID=<user_id>;PWD=<password>;SERVER=echo_client"
   ```

4. 要求を Web サービスに送信する新しいストアドプロシージャを作成します。

   ```
   CREATE OR REPLACE PROCEDURE SendWithMimeType(
       value LONG VARCHAR,
       mimeType LONG VARCHAR,
       urlSpec LONG VARCHAR
   )
   URL '!urlSpec'
   TYPE 'HTTP:POST:!mimeType';
   ```

   `SendWithMimeType` プロシージャには 3 つのパラメータがあります。`value` パラメータは、Web サービスに送信する要求の本文を表します。`urlSpec` パラメータは、Web サービスに接続するために使用する URL を示します。`mimeType` は、HTTP:POST に使用する MIME タイプを示します。

5. 要求を Web サーバに送信し、応答を取得します。

   ```
   CALL SendWithMimeType('<hello>this is xml</hello>',
       'text/xml',
   ```

```
    'http://localhost:8082/EchoService'
);
```

`http://localhost:8082/EchoService` 文字列は、`localhost` で実行されポート 8082 で受信する Web サーバを示します。対象となる Web サービスの名前は `EchoService` です。

6. 異なる MIME タイプを試し、応答を確認します。

```
CALL SendWithMimeType('{"menu": {  "id": "file", "value": "File",
"popup": {
    "menuitem": [{"value": "New", "onclick": "CreateNew()"},
                 {"value": "Open", "onclick": "Open()"},
                {"value": "Close", "onclick": "Close()"} ] } } }',
    'application/json',
    'http://localhost:8082/EchoService'
);
```

Web クライアントは、POST メソッドを使用して HTTP 要求を Web サーバに送信し、Web サーバの応答を受信するように設定されます。

チュートリアル：SAP Sybase IQ を使用した SOAP/DISH サービスへのアクセス

このチュートリアルでは、Web クライアントが指定した華氏の値を摂氏に変換する SOAP サーバの作成方法について説明します。

必須ソフトウェア

- SAP Sybase IQ

前提知識と経験

- SOAP の知識
- SAP Sybase IQ Web サービスの基本的な知識

目的

- 新しい SAP Sybase IQ Web サーバデータベースを作成し、起動します。
- SOAP Web サービスを作成します。
- クライアントが指定した華氏の値を摂氏の値に変換するプロシージャを設定します。
- 新しい SAP Sybase IQ Web クライアントデータベースを作成し、起動します。
- SOAP 要求を、Web クライアントからデータベースサーバに送信します。
- SOAP 応答を、データベースサーバから Web クライアントに送信します。

権限

このチュートリアルのレッスンを実行するには、次の権限が必要です。

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

レッスン 1: SOAP 要求を受信し SOAP 応答を送信する Web サーバの設定

このレッスンでは、新しいデータベースサーバを設定し、着信 SOAP 要求を処理する SOAP サービスを作成します。サーバは対応する摂氏の値に変換される華氏の値を提供する SOAP 要求を予測します。

前提条件

このレッスンでは、このチュートリアル (SAP Sybase IQ を使用した SOAP/DISH サービスへのアクセス) の開始時に、権限のセクションで一覧されているロールと権限を持っていることを前提としています。

手順

1. Web サービス定義を含めるために使用する SAP Sybase IQ データベースを作成します。

   ```
   iqinit -dba <user_id>,<password> ftc
   ```

2. このデータベースを使用して、データベースサーバを起動します。このサーバは Web サーバとして動作します。

   ```
   iqsrv16 -xs http(port=8082) -n ftc ftc.db
   ```

   HTTP Web サーバは、ポート 8082 で要求を受信するように設定されます。ネットワークで 8082 が許可されない場合は、異なるポート番号を使用します。

3. Interactive SQL を使用して、データベースサーバに接続します。

   ```
   dbisql -c "UID=<user_id>;PWD=<password>;SERVER=ftc"
   ```

4. 着信要求を受け入れる新しい DISH サービスを作成します。

   ```
   CREATE SERVICE soap_endpoint
       TYPE 'DISH'
       AUTHORIZATION OFF
       SECURE OFF
       USER DBA;
   ```

   この文は、着信 SOAP サービス要求を処理する `soap_endpoint` という新しい DISH サービスを作成します。

5. 華氏から摂氏への変換を処理する新しい SOAP サービスを作成します。

   ```
   CREATE SERVICE FtoCService
       TYPE 'SOAP'
       FORMAT 'XML'
       AUTHORIZATION OFF
       USER DBA
       AS CALL FToCConverter( :fahrenheit );
   ```

   この文は、XML 形式の文字列を出力として生成する `FtoCService` という新しい SOAP サービスを作成します。Web クライアントがサービスに SOAP 要求

を送信すると、FToCConverter というストアドプロシージャが呼び出されます。

6. 着信 SOAP 要求を処理する FToCConverter プロシージャを作成します。このプロシージャは、クライアントが指定した華氏の値を対応する摂氏の値に変換するための必要な計算を実行します。

```
CREATE OR REPLACE PROCEDURE FToCConverter( temperature FLOAT )
BEGIN
    DECLARE hd_key LONG VARCHAR;
    DECLARE hd_entry LONG VARCHAR;
    DECLARE alias LONG VARCHAR;
    DECLARE first_name LONG VARCHAR;
    DECLARE last_name LONG VARCHAR;
    DECLARE xpath LONG VARCHAR;
    DECLARE authinfo LONG VARCHAR;
    DECLARE namespace LONG VARCHAR;
    DECLARE mustUnderstand LONG VARCHAR;
header_loop:
    LOOP
        SET hd_key = NEXT_SOAP_HEADER( hd_key );
        IF hd_key IS NULL THEN
            -- no more header entries
            LEAVE header_loop;
        END IF;
        IF hd_key = 'Authentication' THEN
            SET hd_entry = SOAP_HEADER( hd_key );
            SET xpath = '/*:' || hd_key || '/*:userName';
           SET namespace = SOAP_HEADER( hd_key, 1, '@namespace' );
            SET mustUnderstand = SOAP_HEADER( hd_key, 1,
'mustUnderstand' );
            BEGIN
                -- parse the XML returned in the SOAP header
                DECLARE crsr CURSOR FOR
                    SELECT * FROM OPENXML( hd_entry, xpath )
                        WITH ( alias LONG VARCHAR '@*:alias',
                         first_name LONG VARCHAR '*:first/text()',
                         last_name LONG VARCHAR '*:last/text()' );
                OPEN crsr;
                FETCH crsr INTO alias, first_name, last_name;
                CLOSE crsr;
            END;

            -- build a response header
            -- based on the pieces from the request header
            SET authinfo =
                XMLELEMENT( 'Authentication',
                    XMLATTRIBUTES(
                        namespace as xmlns,
                        alias,
                        mustUnderstand ),
                        XMLELEMENT( 'first', first_name ),
                        XMLELEMENT( 'last', last_name ) );
            CALL SA_SET_SOAP_HEADER( 'authinfo', authinfo );
        END IF;
```

```
    END LOOP header_loop;
    SELECT ROUND((temperature - 32.0) * 5.0 / 9.0, 5) AS answer;
END;
```

NEXT_SOAP_HEADER 関数は、SOAP 要求に含まれるすべてのヘッダ名を繰り返すために LOOP 構造で使用され、NEXT_SOAP_HEADER 関数から NULL が返されるとループが終了します。

注意： この関数は、必ずしも SOAP 要求に表示される順序でヘッダを繰り返すとはかぎりません。

SOAP_HEADER 関数は、ヘッダ値またはヘッダ名が存在しない場合は NULL を返します。`FToCConverter` プロシージャは、`Authentication` というヘッダ名を検索し、`@namespace` 属性と `mustUnderstand` 属性を含めてヘッダ構造を抽出します。`@namespace` ヘッダ属性は、特定のヘッダエントリの名前空間 (`xmlns`) にアクセスするときに使用する特殊な SAP Sybase IQ 属性です。

次に、可能な `Authentication` ヘッダ構造の XML 文字列表現を示します。`@namespace` 属性の値は `"SecretAgent"`、`mustUnderstand` の値は 1 です。

```
<Authentication xmlns="SecretAgent" mustUnderstand="1">
    <userName alias="99">
        <first>Susan</first>
        <last>Hilton</last>
    </userName>
</Authentication>
```

SELECT 文の OPENXML システムプロシージャでは、XPath 文字列 `"/*:Authentication/*:userName"` を使用して XML ヘッダを解析し、`alias` 属性値と、`first` および `last` タグの内容を抽出します。カーソルを使用して結果セットを処理し、3 つのカラム値をフェッチします。

この時点で、Web サービスに渡された関連性のある情報すべてを取得しています。華氏表現された温度が取得され、Web サービスに渡されたいくつかの追加属性が SOAP ヘッダから取得されています。たとえば、取得した名前と別名 (alias) を照会して、該当人物が Web サービスの使用を許可されているかどうかを確認できます。ただし、この演習でその例は取り上げていません。

SET 文は、クライアントに送信する SOAP 応答を XML 形式で構築するために使用されます。次に、可能な SOAP 応答の XML 文字列表現を示します。これは、上記の `Authentication` ヘッダ構造の例に基づいています。

```
<Authentication xmlns="SecretAgent" alias="99"
mustUnderstand="1">
  <first>Susan</first>
  <last>Hilton</last>
</Authentication>
```

SA_SET_SOAP_HEADER システムプロシージャは、クライアントに送信される SOAP 応答ヘッダを設定するために使用されます。

最後の SELECT 文は、指定された華氏の値を摂氏の値に変換するために使用されます。この情報は、クライアントに戻されます。

この時点で、華氏から摂氏への温度変換を行うサービスを提供する SQL Anywhere Web サーバが実行されます。このサービスは、クライアントからの SOAP ヘッダを処理して SOAP 応答をクライアントに送り返します。

次のステップ

次のレッスンでは、SOAP 要求を Web サーバに送信し、Web サーバから SOAP 応答を受信する、クライアントの例を開発します。

レッスン 2: SOAP 要求を送信し SOAP 応答を受信する Web クライアントの設定

このレッスンでは、SOAP 要求を送信し SOAP 応答を受信する Web クライアントを設定します。

前提条件

このレッスンは、前のレッスンで説明したように Web サーバが設定されていることを前提としています。

このレッスンでは、このチュートリアル (SAP Sybase IQ を使用した SOAP/DISH サービスへのアクセス) の開始時に、権限のセクションで一覧されているロールと権限を持っていることを前提としています。

手順

このレッスンには、`localhost` への複数の参照が含まれています。Web クライアントを Web サーバと同じコンピュータで実行していない場合は、`localhost` の代わりにレッスン 1 の Web サーバのホスト名または IP アドレスを使用します。

1. 次のコマンドを実行して、SAP Sybase IQ データベースを作成します。
   ```
   iqinit -dba <user_id>,<password> ftc_client
   ```
2. 次のコマンドを使用してデータベースクライアントを起動します。
   ```
   iqsrv16 ftc_client.db
   ```
3. 次のコマンドを使用して Interactive SQL でデータベースに接続します。
   ```
   dbisql -c "UID=<user_id>;PWD=<password>;SERVER=ftc_client"
   ```
4. SOAP 要求を DISH サービスに送信する新しいストアドプロシージャを作成します。

   Interactive SQL で次の SQL 文を実行します。

```
CREATE OR REPLACE PROCEDURE FtoC( fahrenheit FLOAT,
    INOUT inoutheader LONG VARCHAR,
    IN inheader LONG VARCHAR )
  URL 'http://localhost:8082/soap_endpoint'
  SET 'SOAP(OP=FtoCService)'
  TYPE 'SOAP:DOC'
  SOAPHEADER '!inoutheader!inheader';
```

URL 句の `http://localhost:8082/soap_endpoint` 文字列は、`localhost` で実行されポート 8082 で受信する Web サーバを示します。対象となる DISH Web サービスの名前は `soap_endpoint` であり、SOAP 終了ポイントとして動作します。

SET 句は、呼び出す SOAP 処理の名前、またはサービス `FtoCService` を指定します。

Web サービス要求作成時のデフォルトフォーマットは 'SOAP:RPC' です。この例で使用されているフォーマットは 'SOAP:DOC' です。これは 'SOAP:RPC' と似ていますが、より多くのデータ型を使用できます。SOAP 要求は必ず XML ドキュメントとして送信されます。SOAP 要求の送信メカニズムは 'HTTP:POST' です。

FtoC のような Web サービスクライアントプロシージャの代入変数 (`inoutheader`、`inheader`) は英数字である必要があります。Web サービスクライアントが関数として宣言された場合、すべてのパラメータは IN モードのみになります (呼び出された側の関数では代入できません)。したがって、SOAP 応答ヘッダ情報を抽出するには、OPENXML またはその他の文字列関数を使用する必要があります。

5. 2 つの特殊な SOAP 要求ヘッダエントリを構築するラッパープロシージャを作成し、それらを `FtoC` プロシージャに渡して、サーバ応答を処理します。

   Interactive SQL で次の SQL 文を実行します。

```
CREATE OR REPLACE PROCEDURE FahrenheitToCelsius( Fahrenheit
FLOAT )
BEGIN
  DECLARE io_header LONG VARCHAR;
  DECLARE in_header LONG VARCHAR;
  DECLARE result LONG VARCHAR;
  DECLARE err INTEGER;
  DECLARE crsr CURSOR FOR
    CALL FtoC( Fahrenheit io_header, in_header );
  SET io_header =
    '<Authentication xmlns="SecretAgent" ' ||
    'mustUnderstand="1">' ||
    '<userName alias="99">' ||
    '<first>Susan</first><last>Hilton</last>' ||
    '</userName>' ||
    '</Authentication>';
  SET in_header =
```

```
    '<Session xmlns="SomeSession">' ||
    '123456789' ||
    '</Session>';

  MESSAGE 'send, soapheader=' || io_header || in_header;
  OPEN crsr;
  FETCH crsr INTO result, err;
  CLOSE crsr;
  MESSAGE 'receive, soapheader=' || io_header;
  SELECT Fahrenheit, Celsius
      FROM OPENXML(result, '//tns:answer', 1, result)
      WITH ("Celsius" FLOAT 'text()');
END;
```

最初の SET 文は、Web サーバにユーザクレデンシャルを通知する、SOAP ヘッダエントリの XML 表現を作成します。

```
<Authentication xmlns="SecretAgent" mustUnderstand="1">
  <userName alias="99">
    <first>Susan</first>
    <last>Hilton</last>
  </userName>
</Authentication>
```

2 番目の SET 文は、クライアントセッション ID を追跡する、SOAP ヘッダエントリの XML 表現を作成します。

```
<Session xmlns="SomeSession">123456789</Session>
```

6. OPEN 文によって FtoC プロシージャが呼び出されます。このプロシージャで、SOAP 要求が Web サーバに送信された後、Web サーバからの応答が処理されます。応答に含まれているヘッダは `inoutheader` に返されます。

この時点で、SOAP 要求を Web サーバに送信し、SOAP 応答を Web サーバから受信できるクライアントが作成されました。

レッスン 3: SOAP 要求の送信と SOAP 応答の受信

このレッスンでは、前のレッスンで作成したラッパープロシージャを呼び出します。このプロシージャは、レッスン 1 で作成した Web サーバに SOAP 要求を送信します。

前提条件

このレッスンは、レッスン 1 で説明したように Web サーバが設定されていることを前提としています。

このレッスンは、レッスン 2 で説明したように Web クライアントが設定されていることを前提としています。

このレッスンでは、このチュートリアル (SAP Sybase IQ を使用した SOAP/DISH サービスへのアクセス) の開始時に、権限のセクションで一覧されているロールと権限を持っていることを前提としています。

手順

1. レッスン 2 の接続が開かれていない場合は、Interactive SQL でクライアントデータベースに接続します。

   ```
   dbisql -c "UID=<user_id>;PWD=<password>;SERVER=ftc_client"
   ```

2. SOAP 要求と SOAP 応答のロギングを有効にします。

   Interactive SQL で次の SQL 文を実行します。

   ```
   CALL sa_server_option('WebClientLogFile', 'soap.txt');
   CALL sa_server_option('WebClientLogging', 'ON');
   ```

   これらの呼び出しによって、SOAP 要求と SOAP 応答の内容を調べることができるようになります。要求と応答のログは `soap.txt` というファイルに記録されます。

3. ラッパープロシージャを呼び出して、SOAP 要求を送信し、SOAP 応答を受信します。

   Interactive SQL で次の SQL 文を実行します。

   ```
   CALL FahrenheitToCelsius(212);
   ```

   この呼び出しでは、華氏の値 212 が `FahrenheitToCelsius` プロシージャに渡されます。このプロシージャは、カスタマイズされた 2 つの SOAP ヘッダとともに値を `FToC` プロシージャに渡します。クライアント側のどちらのプロシージャも、前のレッスンで作成されています。

`FToC` Web サービスプロシージャは、華氏の値と SOAP ヘッダを Web サーバに送信します。SOAP 要求には次の情報が含まれています。

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:m="http://localhost:8082">
  <SOAP-ENV:Header>
    <Authentication xmlns="SecretAgent" mustUnderstand="1">
      <userName alias="99">
        <first>Susan</first>
        <last>Hilton</last>
      </userName>
    </Authentication>
    <Session xmlns="SomeSession">123456789</Session>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:FtoCService>
      <m:fahrenheit>212</m:fahrenheit>
    </m:FtoCService>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

次に、FtoC プロシージャは Web サーバから応答を受信します。応答には、華氏の値に基づく結果セットが含まれています。SOAP 応答には次の情報が含まれています。

```
<SOAP-ENV:Envelope
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:tns='http://localhost:8082'>
  <SOAP-ENV:Header>
    <Authentication xmlns="SecretAgent" alias="99"
mustUnderstand="1">
      <first>Susan</first>
      <last>Hilton</last>
    </Authentication>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <tns:FtoCServiceResponse>
      <tns:FtoCServiceResult xsi:type='xsd:string'>
        &lt;tns:rowset xmlns:tns=&quot;http://localhost:8082/
ftc&quot;&gt;&#x0A;
        &lt;tns:row&gt;&#x0A;
        &lt;tns:answer&gt;100
        &lt;/tns:answer&gt;&#x0A;
        &lt;/tns:row&gt;&#x0A;
        &lt;/tns:rowset&gt;&#x0A;
      </tns:FtoCServiceResult>
      <tns:sqlcode>0</tns:sqlcode>
    </tns:FtoCServiceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

<SOAP-ENV:Header> の内容は inoutheader に返されます。

SOAP 応答を調べてみると、結果セットが FToCService Web サービスによって応答の中でエンコードされていることがわかります。結果セットはデコードされて FahrenheitToCelsius プロシージャに返されます。華氏の値 212 が Web サーバに渡された場合の結果セットは次のようになります。

```
<tns:rowset xmlns:tns="http://localhost:8082/ftc">
  <tns:row>
    <tns:answer>100
    </tns:answer>
  </tns:row>
</tns:rowset>
```

FahrenheitToCelsius プロシージャの SELECT 文では、OPENXML 関数を使用して SOAP 応答が解析され、tns:answer 構造体で定義された摂氏の値が抽出されます。

次の結果セットが Interactive SQL に生成されます。

```
Fahrenheit   Celsius
       212       100
```

チュートリアル： Visual C# を使用した SOAP/DISH Web サービスへのアクセス

このチュートリアルでは、SAP Sybase IQ Web サーバ上の SOAP/DISH サービスにアクセスする Visual C# クライアントアプリケーションの作成方法について説明します。

必須ソフトウェア

- SAP Sybase IQ
- Visual Studio

前提知識と経験

- SOAP の知識
- .NET フレームワークの知識
- SQL Anywhere Web サービスの基本的な知識

目的

- 新しい SAP Sybase IQ Web サーバデータベースを作成し、起動します。
- SOAP Web サービスを作成します。
- SOAP 要求に含まれている情報を返すプロシージャを設定します。
- WSDL ドキュメントを提供し、プロキシとして機能する DISH Web サービスを作成します。
- クライアントコンピュータで Visual C# を設定し、Web サーバから WSDL ドキュメントをインポートします。
- WSDL ドキュメントの情報を使用して SOAP サービスから情報を取得する Java クライアントアプリケーションを作成します。

権限

このチュートリアルのレッスンを実行するには、次の権限が必要です。

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

レッスン 1： SOAP 要求を受信し SOAP 応答を送信する Web サーバの設定

このレッスンでは、Visual C# クライアントアプリケーションの要求を処理する SOAP/DISH Web サービスが実行されている SAP Sybase IQ Web サーバを設定します。

前提条件

最新バージョンの Visual Studio が必要です。

このレッスンでは、このチュートリアル (Visual C# を使用した SOAP/DISH Web サービスへのアクセス) の開始時に、権限のセクションで一覧されているロールと権限を持っていることを前提としています。

手順

1. 次のコマンドを使用して SAP Sybase IQ デモデータベースを起動します。

   ```
   iqsrv16 -xs http(port=8082) iqdemo.db
   ```

   このコマンドは、HTTP Web サーバがポート 8082 で要求を受信することを指定します。ネットワークで 8082 が許可されない場合は、異なるポート番号を使用します。
2. 次のコマンドを使用して Interactive SQL でデータベースサーバに接続します。

   ```
   dbisql -c "UID=<user_id>;PWD=<password>;SERVER=demo"
   ```
3. 着信要求を受け入れる新しい SOAP サービスを作成します。

   Interactive SQL で次の SQL 文を実行します。

   ```
   CREATE SERVICE "SASoapTest/EmployeeList"
       TYPE 'SOAP'
       DATATYPE ON
       AUTHORIZATION OFF
       SECURE OFF
       USER DBA
       AS SELECT * FROM Employees;
   ```

   この文は、SOAP タイプを出力として生成する `SASoapTest/EmployeeList` という新しい SOAP Web サービスを作成します。`Employees` テーブルからすべてのカラムを選択し、結果セットをクライアントに返します。サービス名は、そのサービス名に出現するスラッシュ文字 (/) のため、引用符で囲まれています。

   DATATYPE ON は、明示的なデータ型情報が XML 結果セットの応答と入力パラメータで生成されることを示します。このオプションは、生成される WSDL ドキュメントに影響しません。

FORMAT 句は指定されていないため、SOAP サービスフォーマットは、次のステップで宣言される関連 DISH サービスフォーマットによって指定されます。

4. SOAP サービスのプロキシとして機能し、WSDL ドキュメントを生成する新しい DISH サービスを作成します。

   Interactive SQL で次の SQL 文を実行します。

```
CREATE SERVICE SASoapTest_DNET
    TYPE 'DISH'
    GROUP SASoapTest
    FORMAT 'DNET'
    AUTHORIZATION OFF
    SECURE OFF
    USER DBA;
```

   .NET からアクセスする DISH Web サービスは、FORMAT 'DNET' 句で宣言する必要があります。GROUP 句は、DISH サービスによって処理される必要がある SOAP サービスを識別します。前の手順で作成した `EmployeeList` サービスは、`SASoapTest/EmployeeList` として宣言されているため、GROUP SASoapTest の一部になります。

5. Web ブラウザで関連 WSDL ドキュメントにアクセスして、DISH Web サービスが機能していることを確認します。

   Web ブラウザを開き、http://localhost:8082/demo/SASoapTest_DNET にアクセスします。

   DISH サービスは、ブラウザのウィンドウに表示される WSDL ドキュメントを自動生成します。

Visual C# クライアントアプリケーションの要求を処理できる SOAP/DISH Web サービスが実行されている SAP Sybase IQ Web サーバが設定されました。

次のステップ

次のレッスンでは、Web サーバと通信するための Visual C# アプリケーションを作成します。

レッスン 2: Web サーバと通信するための Visual C# アプリケーションの作成

このレッスンでは、Web サーバと通信するための Visual C# アプリケーションを作成します。

前提条件

このレッスンは、レッスン 1 で説明したように Web サーバが設定されていることを前提としています。

このレッスンを終了するには、最新バージョンの Visual Studio が必要です。

このレッスンでは、このチュートリアル (Visual C# を使用した SOAP/DISH Web サービスへのアクセス) の開始時に、権限のセクションで一覧されているロールと権限を持っていることを前提としています。

手順

このレッスンには、`localhost` への複数の参照が含まれています。Web クライアントを Web サーバと同じコンピュータで実行していない場合は、`localhost` の代わりにレッスン 1 の Web サーバのホスト名または IP アドレスを使用します。

この例では、.NET Framework 2.0 の機能を使用しています。

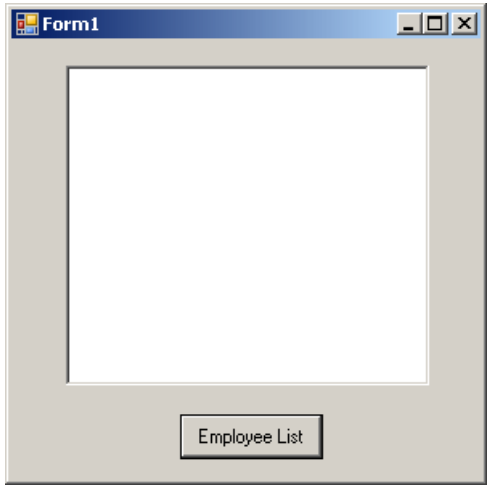
1. Visual Studio を起動します。
2. 新しい Visual C# [Windows フォームのアプリケーション]プロジェクトを作成します。

   空のフォームが表示されます。
3. オブジェクトに Web 参照を追加します。
   a. [プロジェクト] » [サービス参照の追加] をクリックします。
   b. [サービス参照の追加] ウィンドウで、[詳細] をクリックします。
   c. [サービス参照設定] ウィンドウで、[Web 参照の追加] をクリックします。
   d. [Web 参照の追加] ウィンドウで、[URL] フィールドに `http://localhost:8082/demo/SASoapTest_DNET` と入力します。
   e. [移動] (または緑の矢印) をクリックします。

      Visual Studio に、SASoapTest_DNET サービスから使用できる EmployeeList メソッドが表示されます。
   f. [参照の追加] をクリックします。

      Visual Studio は、`localhost` を [ソリューション エクスプローラー] ウィンドウ枠のプロジェクト [Web リファレンス]に追加します。
4. Web クライアントアプリケーションに適したオブジェクトを空のフォームに移植します。

   フォームが次の図のようになるように、[ツールボックス] ウィンドウ枠から ListBox オブジェクトと Button オブジェクトをフォームにドラッグし、テキスト属性を更新します。



5. Web 参照にアクセスするプロシージャを作成し、使用可能なメソッドを使用します。

   Employee List ボタンをダブルクリックし、ボタンクリックイベントに次のコードを追加します。

```
int sqlCode;

listBox1.Items.Clear();

localhost.SASoapTest_DNET proxy = new
localhost.SASoapTest_DNET();

DataSet results = proxy.EmployeeList(out sqlCode);
DataTableReader dr = results.CreateDataReader();
while (dr.Read())
{
  for (int i = 0; i < dr.FieldCount; i++)
  {
    string columnName = "(" + dr.GetDataTypeName(i)
                      + ")"
                      + dr.GetName(i);
    if (dr.IsDBNull(i))
    {
      listBox1.Items.Add(columnName + "=(null)");
    }
    else {
      System.TypeCode typeCode =
          System.Type.GetTypeCode(dr.GetFieldType(i));
      switch (typeCode)
      {
        case System.TypeCode.Int32:
          Int32 intValue = dr.GetInt32(i);
          listBox1.Items.Add(columnName + "="
              + intValue);
          break;
```

```
        case System.TypeCode.Decimal:
          Decimal decValue = dr.GetDecimal(i);
          listBox1.Items.Add(columnName + "="
              + decValue.ToString("c"));
          break;
        case System.TypeCode.String:
          string stringValue = dr.GetString(i);
          listBox1.Items.Add(columnName + "="
              + stringValue);
          break;
        case System.TypeCode.DateTime:
          DateTime dateValue = dr.GetDateTime(i);
          listBox1.Items.Add(columnName + "="
              + dateValue);
          break;
        case System.TypeCode.Boolean:
          Boolean boolValue = dr.GetBoolean(i);
          listBox1.Items.Add(columnName + "="
              + boolValue);
          break;
        case System.TypeCode.DBNull:
          listBox1.Items.Add(columnName
              + "=(null)");
          break;
        default:
          listBox1.Items.Add(columnName
              + "=(unsupported)");
          break;
      }
    }
  }
  listBox1.Items.Add("");
}
dr.Close();
```

6. アプリケーションを実行します。

   [デバッグ] » [デバッグの開始] をクリックします。

7. Web データベースサーバと通信します。

   `Employee List` をクリックします。

   `ListBox` オブジェクトに、`EmployeeList` 結果セットが「(型)名=値」のペアで表示されます。次の出力は、`ListBox` オブジェクトでのエントリの表示方法を示します。

```
(Int32)EmployeeID=102
(Int32)ManagerID=501
(String)Surname=Whitney
(String)GivenName=Fran
(Int32)DepartmentID=100
(String)Street=9 East Washington Street
(String)City=Cornwall
(String)State=New York
(String)Country=USA
```

```
(String)PostalCode=02192
(String)Phone=6175553985
(String)Status=A
(String)SocialSecurityNumber=017349033
(String)Salary=$45,700.00
(DateTime)StartDate=28/08/1984 0:00:00 AM
(DateTime)TerminationDate=(null)
(DateTime)BirthDate=05/06/1958 0:00:00 AM
(Boolean)BenefitHealthInsurance=True
(Boolean)BenefitLifeInsurance=True
(Boolean)BenefitDayCare=False
(String)Sex=F
```

Salary の値は、クライアントの通貨フォーマットに変換されます。

NULL が格納された値は、DBNull として返されます。日付が格納され時刻がない値には、時刻に 00:00:00 つまり午前 0 時が割り当てられます。

Web サーバからの XML 応答には、フォーマットされた結果セットが含まれます。すべてのカラムデータは、データの文字列表現に変換されます。次の結果セットは、クライアントに送信されるときの結果セットのフォーマット方法を示します。

```
<row>
 <EmployeeID>102</EmployeeID>
 <ManagerID>501</ManagerID>
 <Surname>Whitney</Surname>
 <GivenName>Fran</GivenName>
 <DepartmentID>100</DepartmentID>
 <Street>9 East Washington Street</Street>
 <City>Cornwall</City>
 <State>NY</State>
 <Country>USA</Country>
 <PostalCode>02192</PostalCode>
 <Phone>6175553985</Phone>
 <Status>A</Status>
 <SocialSecurityNumber>017349033</SocialSecurityNumber>
 <Salary>45700.000</Salary>
 <StartDate>1984-08-28-05:00</StartDate>
 <TerminationDate xsi:nil="true" />
 <BirthDate>1958-06-05-05:00</BirthDate>
 <BenefitHealthInsurance>1</BenefitHealthInsurance>
 <BenefitLifeInsurance>1</BenefitLifeInsurance>
 <BenefitDayCare>0</BenefitDayCare>
 <Sex>F</Sex>
 </row>
```

日付または時刻の情報が格納されたカラムには、Web サーバの UTC からのオフセットが含まれます。上記の結果セットでは、オフセットは -05:00 であり、これは UTC (アメリカ東部標準時) から西に 5 時間であることを意味します。

日付だけが格納されたカラムは、`yyyy-mm-dd-HH:MM` または `yyyy-mm-dd +HH:MM` のようにフォーマットされます。ゾーンオフセット (-HH:MM または +HH:MM) は文字列のサフィックスとして付きます。

時刻だけが格納されたカラムは、hh:mm:ss.nnn-HH:MM または hh:mm:ss.nnn+HH:MM のようにフォーマットされます。ゾーンオフセット (-HH:MM または +HH:MM) は文字列のサフィックスとして付きます。

日付と時刻の両方が格納されたカラムは、yyyy-mm-ddThh:mm:ss.nnn-HH:MM または yyyy-mm-ddThh:mm:ss.nnn+HH:MM のようにフォーマットされます。日付と時刻は、文字 T で区切られます。ゾーンオフセット (-HH:MM または +HH:MM) は文字列のサフィックスとして付きます。

前のレッスンで DATATYPE ON 句が指定されたため、XML 結果セット応答にデータ型情報が生成されます。Web サーバからの応答の一部を次に示します。型情報はデータベースカラムのデータ型に一致します。

```
<xsd:element name='EmployeeID' minOccurs='0' type='xsd:int' />
<xsd:element name='ManagerID' minOccurs='0' type='xsd:int' />
<xsd:element name='Surname' minOccurs='0' type='xsd:string' />
<xsd:element name='GivenName' minOccurs='0' type='xsd:string' />
<xsd:element name='DepartmentID' minOccurs='0' type='xsd:int' />
<xsd:element name='Street' minOccurs='0' type='xsd:string' />
<xsd:element name='City' minOccurs='0' type='xsd:string' />
<xsd:element name='State' minOccurs='0' type='xsd:string' />
<xsd:element name='Country' minOccurs='0' type='xsd:string' />
<xsd:element name='PostalCode' minOccurs='0' type='xsd:string' />
<xsd:element name='Phone' minOccurs='0' type='xsd:string' />
<xsd:element name='Status' minOccurs='0' type='xsd:string' />
<xsd:element name='SocialSecurityNumber' minOccurs='0'
type='xsd:string' />
<xsd:element name='Salary' minOccurs='0' type='xsd:decimal' />
<xsd:element name='StartDate' minOccurs='0' type='xsd:date' />
<xsd:element name='TerminationDate' minOccurs='0' type='xsd:date' />
<xsd:element name='BirthDate' minOccurs='0' type='xsd:date' />
<xsd:element name='BenefitHealthInsurance' minOccurs='0'
type='xsd:boolean' />
<xsd:element name='BenefitLifeInsurance' minOccurs='0'
type='xsd:boolean' />
<xsd:element name='BenefitDayCare' minOccurs='0'
type='xsd:boolean' />
<xsd:element name='Sex' minOccurs='0' type='xsd:string' />
```

チュートリアル： JAX-WS を使用した SOAP/DISH Web サービスへのアクセス

このチュートリアルでは、SQL Anywhere Web サーバ上の SOAP/DISH サービスにアクセスする XML Web サービス (JAX-WS) クライアントアプリケーション用の Java API の作成方法について説明します。

必須ソフトウェア

- SAP Sybase IQ
- JDK 1.7.0

- JAX-WS 2.2.7 以降のバージョン

前提知識と経験

- SOAP の知識
- Java および JAX-WS の知識
- SAP Sybase IQ Web サービスの基本的な知識

目的

- 新しい SAP Sybase IQ Web サーバデータベースを作成し、起動します。
- SOAP Web サービスを作成します。
- SOAP 要求に含まれている情報を返すプロシージャを設定します。
- WSDL ドキュメントを提供し、プロキシとして機能する DISH Web サービスを作成します。
- クライアントコンピュータで JAX-WS を使用し、Web サーバから WSDL ドキュメントを処理します。
- WSDL ドキュメントの情報を使用して SOAP サービスから情報を取得する Java クライアントアプリケーションを作成します。

権限
このチュートリアルのレッスンを実行するには、次の権限が必要です。

- CREATE ANY OBJECT
- MANAGE ANY WEB SERVICE

レッスン 1： SOAP 要求を受信し SOAP 応答を送信する Web サーバの設定

このレッスンでは、JAX-WS クライアントアプリケーションの要求を処理する SOAP/DISH Web サービスが実行されている SAP Sybase IQ Web サーバを設定します。

前提条件

このレッスンでは、このチュートリアル (JAX-WS を使用した SOAP/DISH Web サービスへのアクセス) の開始時に、権限のセクションで一覧されているロールと権限を持っていることを前提としています。

手順

このレッスンでは、次のレッスンで使用する Web サーバと単純な Web サービスを設定します。プロキシソフトウェアを使用して XML メッセージトラフィックを確認します。プロキシは、クライアントアプリケーションと Web サーバの間に挿入されます。

1. 次のコマンドを使用して SAP Sybase IQ デモデータベースを起動します。

   ```
   iqsrv16 -xs http(port=8082) iqdemo.db
   ```

   このコマンドは、HTTP Web サーバがポート 8082 で要求を受信することを指定します。ネットワークで 8082 が許可されない場合は、異なるポート番号を使用します。
2. 次のコマンドを使用して Interactive SQL でデータベースサーバに接続します。

   ```
   dbisql -c "UID=<user_id>;PWD=<password>;SERVER=demo"
   ```
3. `Employees` テーブルカラムをリストするストアドプロシージャを作成します。

   Interactive SQL で次の SQL 文を実行します。

   ```
   CREATE OR REPLACE PROCEDURE ListEmployees()
   RESULT (
       EmployeeID            INTEGER,
       Surname               CHAR(20),
       GivenName             CHAR(20),
       StartDate             DATE,
       TerminationDate       DATE )
   BEGIN
       SELECT EmployeeID, Surname, GivenName, StartDate,
   TerminationDate
       FROM Employees;
   END;
   ```

   これらの文は、結果セット出力の構造を定義する `ListEmployees` という新しいストアドプロシージャを作成し、Employees テーブルから特定のカラムを選択します。
4. 着信要求を受け入れる新しい SOAP サービスを作成します。

   Interactive SQL で次の SQL 文を実行します。

   ```
   CREATE SERVICE "WS/EmployeeList"
       TYPE 'SOAP'
       FORMAT 'CONCRETE' EXPLICIT ON
       DATATYPE ON
       AUTHORIZATION OFF
       SECURE OFF
       USER DBA
       AS CALL ListEmployees();
   ```

   この文は、SOAP タイプを出力として生成する `WS/EmployeeList` という新しい SOAP Web サービスを作成します。Web クライアントがサービスに要求を送信すると、`ListEmployees` プロシージャが呼び出されます。サービス名は、そのサービス名に出現するスラッシュ文字 (/) のため、引用符で囲まれています。

   JAX-WS からアクセスする SOAP Web サービスは、FORMAT 'CONCRETE' 句で宣言する必要があります。EXPLICIT ON 句は、`ListEmployees` プロシージャ

の結果セットに基づいて、対応する DISH サービスで明示的なデータセットオブジェクトを記述する XML スキーマを生成することを示します。EXPLICIT 句の影響を受けるのは、生成される WSDL ドキュメントのみです。

DATATYPE ON は、明示的なデータ型情報が XML 結果セットの応答と入力パラメータで生成されることを示します。このオプションは、生成される WSDL ドキュメントに影響しません。

5. SOAP サービスのプロキシとして機能し、WSDL ドキュメントを生成する新しい DISH サービスを作成します。

   Interactive SQL で次の SQL 文を実行します。

```
CREATE SERVICE WSDish
    TYPE 'DISH'
    FORMAT 'CONCRETE'
    GROUP WS
    AUTHORIZATION OFF
    SECURE OFF
    USER DBA;
```

   JAX-WS からアクセスする DISH Web サービスは、FORMAT 'CONCRETE' 句で宣言する必要があります。GROUP 句は、DISH サービスによって処理される必要がある SOAP サービスを識別します。前の手順で作成した EmployeeList サービスは、`WS/EmployeeList` として宣言されているため、GROUP `WS` の一部になります。

6. Web ブラウザで関連 WSDL ドキュメントにアクセスして、DISH Web サービスが機能していることを確認します。

   Web ブラウザを開き、http://localhost:8082/demo/WSDish にアクセスします。

   DISH サービスは、ブラウザのウィンドウに表示される WSDL ドキュメントを自動生成します。`EmployeeListDataset` オブジェクトを確認します。次のような出力になります。

```
<s:complexType name="EmployeeListDataset">
<s:sequence>
<s:element name="rowset">
  <s:complexType>
  <s:sequence>
  <s:element name="row" minOccurs="0" maxOccurs="unbounded">
    <s:complexType>
    <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="EmployeeID"
nillable="true" type="s:int" />
    <s:element minOccurs="0" maxOccurs="1" name="Surname"
nillable="true" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="GivenName"
nillable="true" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="StartDate"
nillable="true" type="s:date" />
```

```
    <s:element minOccurs="0" maxOccurs="1" name="TerminationDate"
nillable="true" type="s:date" />
    </s:sequence>
    </s:complexType>
  </s:element>
  </s:sequence>
  </s:complexType>
</s:element>
</s:sequence>
</s:complexType>
```

`EmployeeListDataset` は、`EmployeeList` SOAP サービスの FORMAT 'CONCRETE' 句と EXPLICIT ON 句で生成された明示的なオブジェクトです。この後のレッスンで、wsimport アプリケーションはこの情報を使用して、このサービス用の SOAP 1.1 クライアントインタフェースを生成します。

JAX-WS クライアントアプリケーションの要求を処理できる SOAP/DISH Web サービスが実行されている SAP Sybase IQ Web サーバが設定されました。

次のステップ

次のレッスンでは、Web サーバと通信するための Java アプリケーションを作成します。

レッスン 2: Web サーバと通信するための Java アプリケーションの作成

このレッスンでは、DISH サービスで生成された WSDL ドキュメントを処理し、WSDL ドキュメントで定義されたスキーマに基づいてテーブルデータにアクセスする Java アプリケーションを作成します。

前提条件

このレッスンの内容は、レッスン 1 で行われたステップによって異なります。

このレッスンでは、このチュートリアル (JAX-WS を使用した SOAP/DISH Web サービスへのアクセス) の開始時に、権限のセクションで一覧されているロールと権限を持っていることを前提としています。

手順

本書作成時点では、JAX-WS は JDK 1.7.0 に含まれており、JAX-WS の最新バージョンは 2.2.7 でした。次のステップはそのバージョンによって決まります。JDK に JAX-WS が存在するかどうかを確認するには、JDK `bin` ディレクトリで `wsimport` アプリケーションをチェックしてください。存在しない場合、http://jax-ws.java.net/ にアクセスし、最新バージョンの JAX-WS をダウンロードしてインストールします。

このレッスンには、localhost への複数の参照が含まれています。Web クライアントを Web サーバと同じコンピュータで実行していない場合は、localhost の代わりにレッスン 1 の Web サーバのホスト名または IP アドレスを使用します。

1. コマンドプロンプトで、Java コードと生成ファイル用に新しい作業ディレクトリを作成します。この新しいディレクトリに移動します。
2. 次のコマンドを使用して、DISH Web サービスを呼び出し WSDL ドキュメントをインポートするインタフェースを生成します。

   ```
   wsimport -keep "http://localhost:8082/demo/WSDish"
   ```

   wsimport アプリケーションは、特定の URL から WSDL ドキュメントを取得します。WSDL ドキュメント用のインタフェースを作成するために .java ファイルを生成して、.class ファイルにコンパイルします。

   keep オプションは、クラスファイルの生成後に .java ファイルを削除しないことを示します。生成された Java ソースコードを使用すると、生成されたクラスファイルについて理解できます。

   wsimport アプリケーションは、現在の作業ディレクトリに localhost¥_8082¥demo¥ws という新しいサブフォルダを作成します。次に、ディレクトリ ws の内容のリストを示します。

   - EmployeeList.class
   - EmployeeList.java
   - EmployeeListDataset$Rowset$Row.class
   - EmployeeListDataset$Rowset.class
   - EmployeeListDataset.class
   - EmployeeListDataset.java
   - EmployeeListResponse.class
   - EmployeeListResponse.java
   - FaultMessage.class
   - FaultMessage.java
   - ObjectFactory.class
   - ObjectFactory.java
   - package-info.class
   - package-info.java
   - WSDish.class
   - WSDish.java
   - WSDishSoapPort.class
   - WSDishSoapPort.java
3. 生成されたソースコードに定義されるデータセットオブジェクトスキーマに基づいて、データベースサーバからテーブルデータにアクセスする Java アプリケーションを作成します。

次に、これを実行するサンプル Java アプリケーションを示します。ソースコードを、現在の作業ディレクトリに SASoapDemo.java として保存します。現在の作業ディレクトリには、localhost サブフォルダが含まれている必要があります。

```
// SASoapDemo.java illustrates a web service client that
// calls the WSDish service and prints out the data.

import java.util.*;
import javax.xml.ws.*;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import javax.xml.datatype.*;
import localhost._8082.demo.ws.*;

public class SASoapDemo
{
  public static void main( String[] args )
  {
    try {
      WSDish service = new WSDish();

      Holder<EmployeeListDataset> response =
          new Holder<EmployeeListDataset>();
      Holder<Integer> sqlcode = new Holder<Integer>();

      WSDishSoapPort port = service.getWSDishSoap();

      // This is the SOAP service call to EmployeeList
      port.employeeList( response, sqlcode );

      EmployeeListDataset result = response.value;
      EmployeeListDataset.Rowset rowset = result.getRowset();

     List<EmployeeListDataset.Rowset.Row> rows = rowset.getRow();

      String fieldType;
      String fieldName;
      String fieldValue;
      Integer fieldInt;
      XMLGregorianCalendar fieldDate;

      for ( int i = 0; i < rows.size(); i++ ) {
        EmployeeListDataset.Rowset.Row row = rows.get( i );

        fieldType =
row.getEmployeeID().getDeclaredType().getSimpleName();
        fieldName = row.getEmployeeID().getName().getLocalPart();
        fieldInt = row.getEmployeeID().getValue();
        System.out.println( "(" + fieldType + ")" + fieldName +
                            "=" + fieldInt );

        fieldType =
row.getSurname().getDeclaredType().getSimpleName();
        fieldName = row.getSurname().getName().getLocalPart();
```

```
        fieldValue = row.getSurname().getValue();
        System.out.println( "(" + fieldType + ")" + fieldName +
                            "=" + fieldValue );

        fieldType =
row.getGivenName().getDeclaredType().getSimpleName();
        fieldName = row.getGivenName().getName().getLocalPart();
        fieldValue = row.getGivenName().getValue();
        System.out.println( "(" + fieldType + ")" + fieldName +
                             "=" + fieldValue );

        fieldType =
row.getStartDate().getDeclaredType().getSimpleName();
        fieldName = row.getStartDate().getName().getLocalPart();
        fieldDate = row.getStartDate().getValue();
        System.out.println( "(" + fieldType + ")" + fieldName +
                            "=" + fieldDate );

        if ( row.getTerminationDate() == null ) {
          fieldType = "unknown";
          fieldName = "TerminationDate";
          fieldDate = null;
        } else {
          fieldType =

row.getTerminationDate().getDeclaredType().getSimpleName();
          fieldName =
row.getTerminationDate().getName().getLocalPart();
          fieldDate = row.getTerminationDate().getValue();
        }
        System.out.println( "(" + fieldType + ")" + fieldName +
                            "=" + fieldDate );
        System.out.println();
      }
    }
    catch (Exception x) {
      x.printStackTrace();
    }
  }
}
```

このアプリケーションは、サーバが提供するすべてのカラムデータを標準のシステム出力に表示します。

注意： このアプリケーションでは、レッスン 1 で指示すように、SAP Sybase IQ Web サーバがポート 8082 で受信することを前提としています。`import localhost._8082.demo.ws.*` コード行の 8082 部分を、SAP Sybase IQ Web サーバを起動したときに指定したポート番号に置き換えます。

このアプリケーションで使用される Java メソッドの詳細については、http://docs.oracle.com/javase/ にある javax.xml.bind.JAXBElement クラス API のマニュアルを参照してください。

4. 次のコマンドを使用して、Java アプリケーションをコンパイルします。
   ```
   javac SASoapDemo.java
   ```
5. 次のコマンドを使用して、アプリケーションを実行します。
   ```
   java SASoapDemo
   ```
6. アプリケーションは、Web サーバに要求を送信し、いくつかのローエントリが含まれたローセットを持つ `EmployeeListResult` で構成される XML 結果セット応答を受信します。

次に、実行中の SASoapDemo からの出力例を示します。

```
(Integer)EmployeeID=102
(String)Surname=Whitney
(String)GivenName=Fran
(XMLGregorianCalendar)StartDate=1984-08-28
(unknown)TerminationDate=null

(Integer)EmployeeID=105
(String)Surname=Cobb
(String)GivenName=Matthew
(XMLGregorianCalendar)StartDate=1985-01-01
(unknown)TerminationDate=null
.
.
.
(Integer)EmployeeID=1740
(String)Surname=Nielsen
(String)GivenName=Robert
(XMLGregorianCalendar)StartDate=1994-06-24
(unknown)TerminationDate=null

(Integer)EmployeeID=1751
(String)Surname=Ahmed
(String)GivenName=Alex
(XMLGregorianCalendar)StartDate=1994-07-12
(XMLGregorianCalendar)TerminationDate=2008-04-18
```

`TerminationDate` カラムは、その値が NULL でない場合にのみ送信されます。この Java アプリケーションは、`TerminationDate` カラムが存在しない場合、それを検出するように設計されています。この例では、終了日に NULL 以外の値が設定され、Employees テーブルの最後のローが変更されました。

次の例は、Web サーバからの SOAP 応答を示します。応答には、クエリの実行結果の SQLCODE が含まれます。

```
<tns:EmployeeListResponse>
 <tns:EmployeeListResult xsi:type='tns:EmployeeListDataset'>
  <tns:rowset>
    <tns:row> ... </tns:row>
    .
    .
    .
    <tns:row>
```

```
      <tns:EmployeeID xsi:type="xsd:int">1751</tns:EmployeeID>
      <tns:Surname xsi:type="xsd:string">Ahmed</tns:Surname>
      <tns:GivenName xsi:type="xsd:string">Alex</tns:GivenName>
      <tns:StartDate xsi:type="xsd:dateTime">1994-07-12</
tns:StartDate>
      <tns:TerminationDate xsi:type="xsd:dateTime">2010-03-22</
tns:TerminationDate>
    </tns:row>
  </tns:rowset>
 </tns:EmployeeListResult>
 <tns:sqlcode>0</tns:sqlcode>
</tns:EmployeeListResponse>
```

各ローセットには、カラム名とデータ型が含まれます。

3 層コンピューティングと分散トランザクション

SAP Sybase IQ は、データベースとして使用するほかに、トランザクションサーバによって調整された分散トランザクションに関わる *リソースマネージャ*として使用できます。

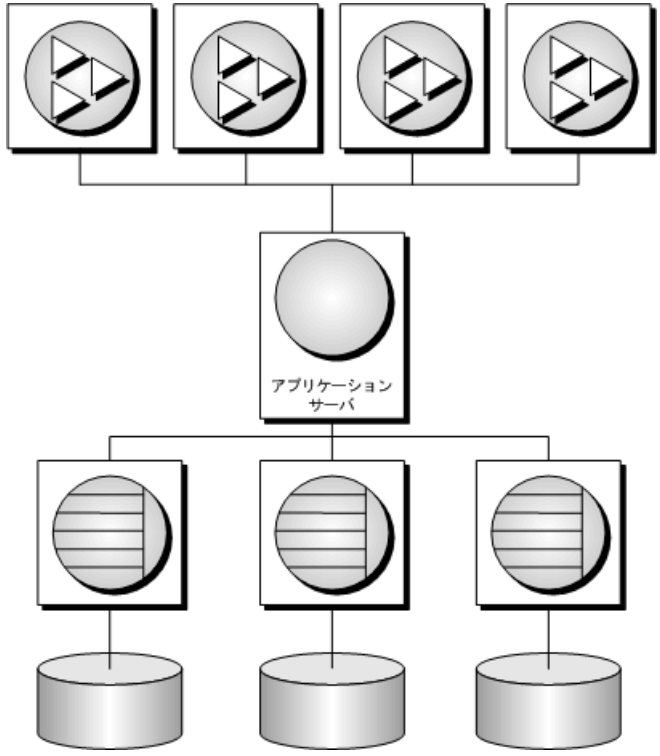
3 層環境では、クライアントアプリケーションと一連のリソースマネージャの間にアプリケーションサーバを置きますが、これが一般的な分散トランザクション環境です。Sybase EAServer と他のアプリケーションサーバの一部もトランザクションサーバです。

Sybase EAServer と Microsoft Transaction Server はともに、Microsoft DTC (分散トランザクションコーディネーター) を使用してトランザクションを調整します。SAP Sybase IQ は、DTC サービスによって制御された分散トランザクションをサポートします。そのため、前述したアプリケーションサーバのいずれかとともに、または DTC モデルに基づくその他のどのような製品とでも、SAP Sybase IQ を使用できます。

SAP Sybase IQ を 3 層環境に統合する場合、作業のほとんどをアプリケーションサーバから行う必要があります。この項では、3 層コンピューティングの概念とアーキテクチャおよび、SAP Sybase IQ の関連機能の概要について説明します。ここでは、アプリケーションサーバを設定して SAP Sybase IQ とともに動作させる方法については説明しません。詳細については、使用しているアプリケーションサーバのマニュアルを参照してください。

3 層コンピューティングの アーキテクチャ

3 層コンピューティングの場合、アプリケーション論理は Sybase EAServer などのアプリケーションサーバに格納されます。アプリケーションサーバは、リソースマネージャとクライアントアプリケーションの間に置かれます。多くの場合、1 つのアプリケーションサーバから複数のリソースマネージャにアクセスできます。インターネットの場合、クライアントアプリケーションはブラウザベースであり、アプリケーションサーバは、通常、Web サーバの拡張機能です。



Sybase EAServer は、アプリケーション論理をコンポーネントとして格納し、このコンポーネントをクライアントアプリケーションから利用できるようにします。利用できるコンポーネントは、PowerBuilder® コンポーネント、JavaBeans、または COM コンポーネントです。

詳細については、Sybase EAServer のマニュアルを参照してください。

3 層コンピューティングにおける分散トランザクション

クライアントアプリケーションまたはアプリケーションサーバが SAP Sybase IQ などの単一のトランザクション処理データベースとともに動作するときは、データベース自体の外部にトランザクション論理は必要ありません。しかし、複数のリソースマネージャとともに動作するときは、トランザクションで使用される複数のリソースにわたってトランザクション制御を行う必要があります。アプリケーションサーバは、クライアントアプリケーションにトランザクション論理を提供し、一連の操作がアトミックに実行されることを保証します。

Sybase EAServer をはじめとする多くのトランザクションサーバは、Microsoft DTC (分散トランザクションコーディネーター) を使用して、クライアントアプリケーションにトランザクションサービスを提供します。DTC は *OLE トランザクション*

を使用します。OLE トランザクションは *2 フェーズコミット*のプロトコルを使用して、複数のリソースマネージャに関わるトランザクションを調整します。この項で説明する機能を使用するには、DTC がインストールされている必要があります。

分散トランザクションにおける SAP Sybase IQ
DTC が調整するトランザクションに SAP Sybase IQ を追加できます。つまり、Sybase EAServer や Microsoft Transaction Server などのトランザクションサーバを使用して、SAP Sybase IQ データベースを分散トランザクションの中で使用できます。また、アプリケーションの中で直接 DTC を使用して、複数のリソースマネージャにわたるトランザクションを調整することもできます。

分散トランザクションに関する用語

この項は、分散トランザクションについて基本的な知識を持っている方を対象としています。詳細については、使用しているトランザクションサーバのマニュアルを参照してください。この項では、よく使用される用語をいくつか説明します。

- *リソースマネージャ*は、トランザクションに関連するデータを管理するサービスです。
  分散トランザクションの中で ADO.NET、OLE DB、または ODBC を通してアクセスする場合、SAP Sybase IQ データベースサーバはリソースマネージャとして動作します。SAP Sybase IQ .NET データプロバイダ、OLE DB プロバイダ、ODBC ドライバは、クライアントコンピュータ上のリソースマネージャプロキシとして動作します。SAP Sybase IQ .NET データプロバイダは、DbProviderFactory と TransactionScope を使用して分散トランザクションをサポートします。
- アプリケーションコンポーネントは、リソースマネージャと直接通信しないで*リソースディスペンサー*と通信できます。リソースディスペンサーは、リソースマネージャへの接続または接続プールを管理します。
  SAP Sybase IQ がサポートする 2 つのリソースディスペンサは、ODBC ドライバマネージャと OLE DB です。
- トランザクションコンポーネントが (リソースマネージャを使用して) データベースとの接続を要求すると、アプリケーションサーバはトランザクションに関わるデータベース接続を*エンリスト*します。DTC とリソースディスペンサーがエンリスト処理を実行します。

2 フェーズコミット
2 フェーズコミットを使用して、分散トランザクションを管理します。トランザクション処理が完了すると、トランザクションマネージャ (DTC) は、トランザクションにエンリストされたすべてのリソースマネージャにトランザクションをコミットする準備ができているかどうかを問い合わせます。このフェーズは、コミットの*準備*と呼ばれます。

すべてのリソースマネージャからコミット準備完了の応答があると、DTC は各リソースマネージャにコミット要求を送信し、トランザクションの完了をクライアントに通知します。1 つ以上のリソースマネージャが応答しない場合、またはトランザクションをコミットできないと応答した場合、トランザクションのすべての処理は、すべてのリソースマネージャにわたってロールバックされます。

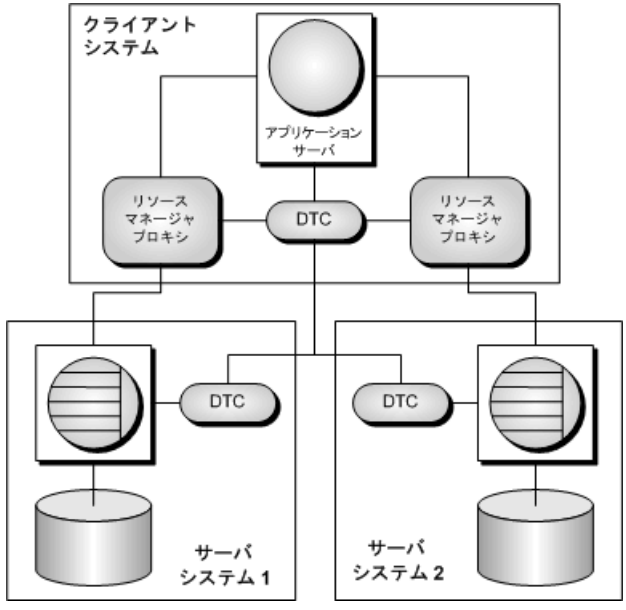
アプリケーションサーバが DTC を使用する方法

Sybase EAServer と Microsoft Transaction Server は、どちらもコンポーネントサーバです。アプリケーション論理はコンポーネントとして格納され、クライアントアプリケーションから利用できます。

各コンポーネントのトランザクション属性は、コンポーネントがどのようにトランザクションに関わるかを示します。コンポーネントを構築する場合、トランザクションの作業 (リソースマネージャとの接続、各リソースマネージャが管理するデータに対する操作など) をコンポーネントの中にプログラムする必要があります。ただし、トランザクション管理のロジックをコンポーネントに追加する必要はありません。トランザクション属性が設定され、コンポーネントにトランザクション管理が必要な場合、EAServer は、DTC を使用してトランザクションをエンリストし、2 フェーズコミット処理を管理します。

分散トランザクションのアーキテクチャ

次の図は、分散トランザクションのアーキテクチャを示しています。この場合、リソースマネージャプロキシは ADO.NET、OLE DB、または ODBC です。



この場合、単一のリソースディスペンサーが使用されています。アプリケーションサーバは、DTC にトランザクションの準備を要求します。DTC とリソースディスペンサーは、トランザクション内の各接続をエンリストします。作業を実行し、必要に応じてトランザクションステータスを DTC に通知するためには、各リソースマネージャが DTC とデータベースの両方にアクセスする必要があります。

分散トランザクションを操作するには、各コンピュータ上で分散トランザクションコーディネーター (DTC) サービスを実行中にしてください。Microsoft Windows の [サービス] ウィンドウから DTC を開始または停止できます。DTC サービスタスクは、`MSDTC` という名前で表示されます。

詳細については、DTC または EAServer のマニュアルを参照してください。

分散トランザクション

SAP Sybase IQ は、分散トランザクションにエンリストされている間は、トランザクション制御をトランザクションサーバに渡します。また、SAP Sybase IQ は、トランザクション管理を暗黙的に実行しないようにします。SAP Sybase IQ が分散トランザクションを処理する場合、自動的に次の条件が設定されます。

- オートコミットが使用されている場合は、自動的にオートコミットがオフになります。
- 分散トランザクション中は、データ定義文 (副次的な効果としてコミットされる) を使用できません。
- アプリケーションが明示的な COMMIT または ROLLBACK を発行する場合に、トランザクションコーディネーターを介さずに直接 SAP Sybase IQ に発行すると、エラーが発生します。ただし、トランザクションはアボートしません。
- 1 つの接続で処理できるのは、1 回に 1 つの分散トランザクションに限られます。
- 接続が分散トランザクションにエンリストされるときに、すべてのコミット操作が完了している必要があります。

DTC の独立性レベル

DTC には、独立性レベルのセットが定義されています。アプリケーションサーバは、この中からレベルを指定します。DTC の独立性レベルは、次のように SAP Sybase IQ の独立性レベルにマッピングされます。

DTC の独立性レベル	SAP Sybase IQ 独立性レベル
ISOLATIONLEVEL_UNSPECIFIED	0
ISOLATIONLEVEL_CHAOS	0

DTC の独立性レベル	SAP Sybase IQ 独立性レベル
ISOLATIONLEVEL_READUNCOMMITTED	0
ISOLATIONLEVEL_BROWSE	0
ISOLATIONLEVEL_CURSORSTABILITY	1
ISOLATIONLEVEL_READCOMMITTED	1
ISOLATIONLEVEL_REPEATABLEREAD	2
ISOLATIONLEVEL_SERIALIZABLE	3
ISOLATIONLEVEL_ISOLATED	3

分散トランザクションからのリカバリ

コミットされていない操作の保留中にデータベースサーバにフォールトが発生した場合、トランザクションのアトミックな状態を保つために、起動時にこれらの操作をロールバックまたはコミットする必要があります。

分散トランザクションからコミットされていない操作がリカバリ中に検出されると、データベースサーバは DTC に接続を試み、検出された操作を保留または不明のトランザクションに再エンリストするように要求します。再エンリストが完了すると、DTC は未処理操作のロールバックまたはコミットをデータベースサーバに指示します。

再エンリスト処理が失敗すると、SAP Sybase IQ は不明の操作をコミットするかロールバックするかを判断できなくて、リカバリは失敗します。データの状態が保証されないことを前提にして、リカバリに失敗したデータベースをリカバリする場合は、次のデータベースサーバオプションを使って強制リカバリします。

- **-tmf –** DTC が特定できないときは、未処理の操作をロールバックしてリカバリを続行します。
- **-tmt –** 指定した時間内に再エンリストが完了しないときは、未処理の操作をロールバックしてリカバリを続行します。

データベースツールインタフェース (DBTools)

SAP Sybase IQ は Sybase Control Center とデータベース管理用のユーティリティのセットを含みます。これらのデータベース管理ユーティリティを使用すると、データベースのバックアップ、データベースの作成、トランザクションログの SQL への変換などの作業を実行できます。

サポートするプラットフォーム

すべてのデータベース管理ユーティリティは*データベースツールライブラリ*と呼ばれる共有ライブラリを使用します。このライブラリは、Windows オペレーティングシステムと Linux および UNIX 向けに提供されています。Windows 向けのライブラリ名は `dbtool16.dll` です。Linux および UNIX 向けのライブラリ名は `libdbtool16_r.so` です。

データベースツールライブラリを呼び出すことによって、独自のデータベース管理ユーティリティを開発したり、データベース管理機能をアプリケーションに組み込んだりできます。この項では、データベースツールライブラリに対するインタフェースについて説明します。この項の説明は、使用中の開発環境からライブラリルーチンを呼び出す方法に精通しているユーザを対象にしています。

データベースツールライブラリは、各データベース管理ユーティリティに対してそれぞれ関数、またはエントリポイントを持ちます。また、他のデータベースツール関数の使用前と使用後に、関数を呼び出す必要があります。

dbtools.h ヘッダファイル

dbtools ヘッダファイルは、DBTools ライブラリへのエントリポイントと、DBTools ライブラリとの間で情報をやりとりするために使用する構造体をリストします。`dbtools.h` ファイルはすべて、SAP Sybase IQ インストールディレクトリの `SDK¥Include` サブディレクトリにインストールされています。エントリポイントと構造体メンバーの最新情報については、`dbtools.h` ファイルを参照してください。

`dbtools.h` ヘッダファイルには、他に次のようなファイルが含まれています。

- **`sqlca.h`** – SQLCA 自身ではなく、さまざまなマクロの解析のために使用するものです。
- **`dllapi.h`** – オペレーティングシステムと言語に依存するマクロのためのプリプロセッサマクロを定義します。
- **`dbtlvers.h`** – DB_TOOLS_VERSION_NUMBER プリプロセッサマクロとその他のバージョン固有のマクロを定義します。

sqldef.h ヘッダファイル
`sqldef.h` ヘッダファイルはエラー戻り値も含みます。

dbrmt.h ヘッダファイル
SAP Sybase IQ に含まれる `dbrmt.h` ヘッダファイルは、DBTools ライブラリへの DBRemoteSQL エントリポイントと、DBRemoteSQL エントリポイントとの間で情報をやりとりするために使用する構造体をリストします。`dbrmt.h` ファイルはすべて、SAP Sybase IQ インストールディレクトリの `SDK¥Include` サブディレクトリにインストールされています。DBRemoteSQL エントリポイントと構造体メンバーの最新情報については、`dbrmt.h` ファイルを参照してください。

DBTools インポートライブラリ

DBTools 関数を使用するには、必要な関数定義を含む DBTools *インポートライブラリ*にアプリケーションをリンクする必要があります。

UNIX システムでは、インポートライブラリは不要です。`libdbtool16.so` (非スレッド) または `libdbtool16_r.so` (スレッド) に対して直接リンクします。

インポートライブラリ
DBTools インタフェース用のインポートライブラリは、Windows 用の SAP Sybase IQ に用意されています。Windows の場合、インポートライブラリは SAP Sybase IQ インストールディレクトリの `SDK¥Lib¥x86` サブフォルダと `SDK¥Lib¥x64` サブフォルダにあります。提供される DBTools インポートライブラリは次のとおりです。

コンパイラ	ライブラリ
Microsoft Windows	`dbtlstm.lib`

DBTools ライブラリの初期化とファイナライズ

他の DBTools 関数を使用する前に、DBToolsInit を呼び出す必要があります。DBTools ライブラリを使い終わったときは、DBToolsFini を呼び出してください。

DBToolsInit と DBToolsFini 関数の主な目的は、DBTools ライブラリが SAP Sybase IQ メッセージライブラリをロードおよびアンロードできるようにすることです。メッセージライブラリには、DBTools が内部的に使用する、ローカライズされたバージョンのすべてのエラーメッセージとプロンプトが含まれています。DBToolsFini を呼び出さないと、メッセージライブラリのリファレンスカウントが

減分されず、アンロードされません。そのため、DBToolsInit と DBToolsFini の呼び出し回数が等しくなるよう注意してください。

次のコードは、DBTools を初期化してファイナライズする方法を示しています。

```
// Declarations
a_dbtools_info  info;
short           ret;

//Initialize the a_dbtools_info structure
memset( &info, 0, sizeof( a_dbtools_info) );
info.errorrtn = (MSG_CALLBACK)MyErrorRtn;

// initialize the DBTools library
ret = DBToolsInit( &info );
if( ret != EXIT_OKAY ) {
    // library initialization failed
    ...
}
// call some DBTools routines ...
...
// finalize the DBTools library
DBToolsFini( &info );
```

DBTools 関数呼び出し

すべてのツールは、まず構造体に値を設定し、次に DBTools ライブラリの関数 (または*エントリポイント*) を呼び出すことによって実行します。各エントリポイントには、引数として単一構造体へのポインタを渡します。

次の例は、Windows オペレーティングシステムでの DBBackup 関数の使い方を示しています。

```
// Initialize the structure
a_backup_db backup_info;
memset( &backup_info, 0, sizeof( backup_info ) );

// Fill out the structure
backup_info.version = DB_TOOLS_VERSION_NUMBER;
backup_info.output_dir = "c:¥¥backup";
backup_info.connectparms
="UID=<user_id>;PWD=<password>;DBF=iqdemo.db";

backup_info.confirmrtn = (MSG_CALLBACK) ConfirmRtn ;
backup_info.errorrtn = (MSG_CALLBACK) ErrorRtn ;
backup_info.msgrtn = (MSG_CALLBACK) MessageRtn ;
backup_info.statusrtn = (MSG_CALLBACK) StatusRtn ;
backup_info.backup_database = TRUE;

// start the backup
DBBackup( &backup_info );
```

コールバック関数

DBTools 構造体には MSG_CALLBACK 型の要素がいくつかあります。それらはコールバック関数へのポインタです。

コールバック関数の使用
コールバック関数を使用すると、DBTools 関数はオペレーションの制御をユーザの呼び出し側アプリケーションに戻すことができます。DBTools ライブラリはコールバック関数を使用して、DBTools 関数から、次の 4 つの目的を持ってユーザに送られたメッセージを処理します。

- **確認** – ユーザがアクションを確認する必要がある場合に呼び出されます。たとえば、バックアップディレクトリが存在しない場合、ツールライブラリはディレクトリを作成する必要があるか確認を求めます。
- **エラーメッセージ** – オペレーション中にディスク容量が足りなくなった場合など、エラーが発生したときにメッセージを処理するために呼び出されます。
- **情報メッセージ** – ツールがユーザにメッセージを表示するときに呼び出されます (アンロード中の現在のテーブル名など)。
- **ステータス情報** – ツールがオペレーションのステータス (テーブルのアンロード処理の進捗率など) を表示するときに呼び出されます。

コールバック関数の構造体への割り当て
コールバックルーチンを構造体に直接割り当てることができます。次の文は、バックアップ構造体を使用した例です。

```
backup_info.errorrtn = (MSG_CALLBACK) MyFunction
```

MSG_CALLBACK は、SAP Sybase IQ に付属する `dllapi.h` ヘッダファイルに定義されています。ツールルーチンは、呼び出し側アプリケーションにメッセージを付けてコールバックできます。このメッセージは、ウィンドウ環境でも、文字ベースのシステムの標準出力でも、またはそれ以外のユーザインタフェースであっても、適切なユーザインタフェースに表示されます。

確認コールバック関数の例
次の確認ルーチンの例では、YES または NO をプロンプトに答えるようユーザに求め、ユーザの選択結果を戻します。

```
extern short _callback ConfirmRtn(
        char * question )
{
    int ret = IDNO;
    if( question != NULL ) {
        ret = MessageBox( HwndParent, question,
        "Confirm", MB_ICONEXCLAMATION|MB_YESNO );
    }
```

```
    return( ret == IDYES );
}
```

エラーコールバック関数の例
次はエラーメッセージ処理ルーチンの例です。エラーメッセージをウィンドウに表示します。

```
extern short _callback ErrorRtn(
        char * errorstr )
{
    if( errorstr != NULL ) {
        MessageBox( HwndParent, errorstr, "Backup Error",
MB_ICONSTOP|MB_OK );
    }
    return( 0 );
}
```

メッセージコールバック関数の例
メッセージコールバック関数の一般的な実装では、メッセージを画面に表示します。

```
extern short _callback MessageRtn(
        char * messagestr )
{
    if( messagestr != NULL ) {
                    OutputMessageToWindow( messagestr );
    }
    return( 0 );
}
```

ステータスコールバック関数の例
ステータスコールバックルーチンは、ツールがオペレーションのステータス (テーブルのアンロード処理の進捗率など) を表示する必要がある場合に呼び出されます。一般的な実装では、メッセージを画面に表示するだけです。

```
extern short _callback StatusRtn(
        char * statusstr )
{
    if( statusstr != NULL ) {
        OutputMessageToWindow( statusstr );
    return( 0 );
}
```

バージョン番号と互換性

各構造体にはバージョン番号を示すメンバーがあります。DBTools の関数を呼び出す前に、このバージョンフィールドに、アプリケーション開発に使用した DBTools ライブラリのバージョン番号を設定しておきます。DBTools ライブラリの現在のバージョンは、`dbtools.h` ヘッダファイルをインクルードするときに定義されます。

次の例では、現在のバージョンを a_backup_db 構造体のインスタンスに割り当てます。

```
backup_info.version = DB_TOOLS_VERSION_NUMBER;
```

バージョン番号を使用することによって、DBTools ライブラリのバージョンが新しくなってもアプリケーションを継続して使用できます。DBTools 関数は、DBTools 構造体に新しいメンバーが追加されても、アプリケーションが提示するバージョン番号を使用してアプリケーションが作動できるようにします。

DBTools 構造体が更新されたり、新しいバージョンのソフトウェアがリリースされると、バージョン番号が大きくなります。DB_TOOLS_VERSION_NUMBER を使用し、新しいバージョンの DBTools ヘッダファイルを使用してアプリケーションを再構築する場合は、新しいバージョンの DBTools ライブラリを配備してください。

ビットフィールド

DBTools 構造体の多くは、ビットフィールドを使用してブール情報を効率よく格納しています。たとえば、バックアップ構造体には次のビットフィールドがあります。

```
a_bit_field     backup_database : 1;
a_bit_field     backup_logfile  : 1;
a_bit_field     no_confirm  : 1;
a_bit_field     quiet       : 1;
a_bit_field     rename_log  : 1;
a_bit_field     truncate_log : 1;
a_bit_field     rename_local_log: 1;
a_bit_field     server_backup   : 1;
```

各ビットフィールドは 1 ビット長です。これは、構造体宣言のコロンの右側の 1 によって示されています。a_bit_field に割り当てられている値に応じて、特定のデータ型が使用されます。a_bit_field は dbtools.h の先頭で設定され、設定値はオペレーティングシステムに依存します。

0 または 1 の値をビットフィールドに割り当てて、構造体のブール情報を渡します。

DBTools の例

このサンプルとコンパイル手順は、%ALLUSERSPROFILE%¥SybaseIQ¥samples¥SQLAnywhere¥DBTools ディレクトリにあります。サンプルプログラム自体は main.cpp にあります。このサンプルは、DBTools ライブラリを使用してデータベースのバックアップを作成する方法を示しています。

```
#define WIN32

#include <stdio.h>
#include <string.h>
#include "windows.h"
#include "sqldef.h"
#include "dbtools.h"
extern short _callback ConfirmCallBack( char * str )
{
    if( MessageBox( NULL, str, "Backup",
         MB_YESNO|MB_ICONQUESTION ) == IDYES )
    {
        return 1;
    }
    return 0;
}
extern short _callback MessageCallBack( char * str )
{
    if( str != NULL )
    {
        fprintf( stdout, "%s¥n", str );
    }
    return 0;
}
extern short _callback StatusCallBack( char * str )
{
    if( str != NULL )
    {
        fprintf( stdout, "%s¥n", str );
    }
    return 0;
}
extern short _callback ErrorCallBack( char * str )
{
    if( str != NULL )
    {
        fprintf( stdout, "%s¥n", str );
    }
    return 0;
}
typedef void (CALLBACK *DBTOOLSPROC)( void * );
typedef short (CALLBACK *DBTOOLSFUNC)( void * );

// Main entry point into the program.
int main( int argc, char * argv[] )
{
    a_dbtools_info  dbt_info;
    a_backup_db     backup_info;
    char            dir_name[ _MAX_PATH + 1];
    char            connect[ 256 ];
    HINSTANCE       hinst;
    DBTOOLSFUNC     dbbackup;
    DBTOOLSFUNC     dbtoolsinit;
    DBTOOLSPROC     dbtoolsfini;
    short           ret_code;
```

```
// Always initialize to 0 so new versions
// of the structure will be compatible.
memset( &dbt_info, 0, sizeof( a_dbtools_info ) );
dbt_info.errorrtn = (MSG_CALLBACK)MessageCallBack;;

memset( &backup_info, 0, sizeof( a_backup_db ) );
backup_info.version = DB_TOOLS_VERSION_NUMBER;
backup_info.quiet = 0;
backup_info.no_confirm = 0;
backup_info.confirmrtn = (MSG_CALLBACK)ConfirmCallBack;
backup_info.errorrtn = (MSG_CALLBACK)ErrorCallBack;
backup_info.msgrtn = (MSG_CALLBACK)MessageCallBack;
backup_info.statusrtn = (MSG_CALLBACK)StatusCallBack;
if( argc > 1 )
{
    strncpy( dir_name, argv[1], _MAX_PATH );
}
else
{
    // DBTools does not expect (or like) a trailing slash
    strcpy( dir_name, "c:¥¥temp" );
}
backup_info.output_dir = dir_name;
if( argc > 2 )
{
    strncpy( connect, argv[2], 255 );
}
else
{
    strcpy( connect, "DSN=Sybase IQ Demo" );
}
backup_info.connectparms = connect;
backup_info.quiet = 0;
backup_info.no_confirm = 0;
backup_info.backup_database = 1;
backup_info.backup_logfile = 1;
backup_info.rename_log = 0;
backup_info.truncate_log = 0;
hinst = LoadLibrary( "dbtool16.dll" );
if( hinst == NULL )
{
    // Failed
    return EXIT_FAIL;
}
dbbackup = (DBTOOLSFUNC) GetProcAddress( (HMODULE)hinst,
    "_DBBackup@4" );
dbtoolsinit = (DBTOOLSFUNC) GetProcAddress( (HMODULE)hinst,
    "_DBToolsInit@4" );
dbtoolsfini = (DBTOOLSPROC) GetProcAddress( (HMODULE)hinst,
    "_DBToolsFini@4" );
ret_code = (*dbtoolsinit)( &dbt_info );
if( ret_code != EXIT_OKAY ) {
    return ret_code;
}
```

```
    ret_code = (*dbbackup)( &backup_info );
    (*dbtoolsfini)( &dbt_info );
    FreeLibrary( hinst );
    return ret_code;
}
```

ソフトウェアコンポーネントの終了コード

すべてのデータベースツールライブラリのエントリポイントで次の終了コードが使用されます。SAP Sybase IQ のユーティリティでもこれらの終了コードを使用します。

コード	状態	説明
0	EXIT_OKAY	成功
1	EXIT_FAIL	一般的な失敗
2	EXIT_BAD_DATA	無効なファイルフォーマット
3	EXIT_FILE_ERROR	ファイルが見つからない、開くことができない
4	EXIT_OUT_OF_MEMORY	メモリ不足
5	EXIT_BREAK	ユーザによる終了
6	EXIT_COMMUNICATIONS_FAIL	通信失敗
7	EXIT_MISSING_DATABASE	必要なデータベース名なし
8	EXIT_PROTOCOL_MISMATCH	クライアントとサーバのプロトコルが一致しない
9	EXIT_UNABLE_TO_CONNECT	データベースサーバと接続できない
10	EXIT_ENGINE_NOT_RUNNING	データベースサーバが起動されない
11	EXIT_SERVER_NOT_FOUND	データベースサーバが見つからない
12	EXIT_BAD_ENCRYPT_KEY	暗号化キーが見つからないか、不正である
13	EXIT_DB_VER_NEWER	データベースを実行するためにサーバをアップグレードする必要がある
14	EXIT_FILE_INVALID_DB	ファイルがデータベースでない
15	EXIT_LOG_FILE_ERROR	ログファイルが見つからないか、その他のエラーが発生した
16	EXIT_FILE_IN_USE	ファイルが使用中

コード	状態	説明
17	EXIT_FATAL_ERROR	致命的なエラーが発生した
18	EXIT_MISSING_LICENSE_FILE	サーバライセンスファイルが見つからない
19	EXIT_BACKGROUND_SYNC_ABORTED	優先度の高い操作を続行するためにバックグラウンド同期がアボートされた
20	EXIT_FILE_ACCESS_DENIED	アクセスが拒否されたためデータベースを起動できない
255	EXIT_USAGE	コマンドラインで無効なパラメータ
21	EXIT_SERVER_NAME_IN_USE	同じ名前をもつ別のサーバが現在実行中

これらの終了コードは、`%IQDIR16%¥sdk¥include¥sqldef.h` ファイルで定義されています。

データベースツール C API リファレンス

ヘッダファイルは `dbtools.h` と `dbrmt.h` です。

付録： OLAP の使用

オンライン分析処理 (OLAP: Online Analytical Processing) は、リレーショナルデータベースに格納されている情報を効率的にデータ分析するための手法です。

OLAP を使用すると、データをさまざまな次元で分析し、小計ローを含んだ結果セットを取得し、データを多次元キューブに編成するという処理をすべて 1 つの SQL クエリで行うことができます。また、フィルタを使用してデータを絞り込み、結果セットを迅速に返すことができます。この章では、SAP Sybase IQ がサポートする SQL/OLAP 関数について説明します。

注意： OLAP の例に示されているテーブルは、`iqdemo` データベースにあります。

OLAP について

この分析関数を使用すると、複雑なデータ分析を 1 つの SQL 文で実行できます。これは、オンライン分析処理 (OLAP) と呼ばれるソフトウェアテクノロジ分類に基づいています。OLAP の関数には、次のようなものが含まれています。

- **GROUP BY** 句の拡張 - **CUBE** と **ROLLUP**
- 分析関数：
  - 単純な集合関数 - **AVG**、**COUNT**、**MAX**、 **MIN**、**SUM**、**STDDEV**、**VARIANCE**

    注意： **Grouping()** 以外の単純な集合関数は OLAP ウィンドウ関数と併用できます。
  - ウィンドウ関数
    - ウィンドウ集合関数 - **AVG**、**COUNT**、**MAX**、 **MIN**、**SUM**
    - ランク付け関数 - **RANK**、**DENSE_RANK**、**PERCENT_RANK**、**NTILE**
    - 統計関数 – **STDDEV**、**STDDEV_SAMP**、**STDDEV_POP**、**VARIANCE**、**VAR_POP**、 **VAR_SAMP**、 **REGR_AVGX**、**REGR_AVGY**、**REGR_COUNT**、**REGR_INTERCEPT**、**REGR_R2**、**REGR_SLOPE**、**REGR_SXX**、**REGR_SXY**、**REGR_SYY**、**CORR**、**COVAR_POP**、**COVAR_SAMP**、**CUME_DIST**、**EXP_WEIGHTED_AVG**、および **WEIGHTED_AVG**。
    - 分散統計関数 - **PERCENTILE_CONT** と **PERCENTILE_DISC**
  - 数値関数 - **WIDTH_BUCKET**、**CEIL**、**LN**、**EXP**、**POWER**、**SQRT**、**FLOOR**

1999 年の SQL 標準の改正で、ANSI SQL 標準に複雑なデータ分析機能を含めるための拡張が導入されました。SAP Sybase IQ SQL の強化で、これらの拡張をサポートしています。

データベース製品によっては、OLAP モジュールが独立しており、分析前にデータをデータベースから OLAP モジュールに移動しなければならないものもあります。一方、SAP Sybase IQ では OLAP 機能がデータベースそのものに組み込まれているため、ストアドプロシージャなどのデータベース機能との統合や配備を簡単かつシームレスに行うことができます。

OLAP の利点

OLAP 関数を **GROUPING**、**CUBE**、**ROLLUP** という拡張機能と組み合わせて使用すると、2 つの大きな利点があります。

第一に、多次元のデータ分析、データマイニング、時系列分析、傾向分析、コストの割り当て、ゴールシーク、一時的な多次元構造変更、非手続き型モデリング、例外の警告を多くの場合 1 つの SQL 文で実行できます。第二に、OLAP のウィンドウおよびレポート集合関数では、*ウィンドウ*という関係演算子を使用することができ、これはセルフジョインや相関サブクエリを使用するセマンティック的に等価なクエリよりも効率的に実行できます。 OLAP を使用して取得した結果セットには小計ローを含めることができ、この結果セットを多次元キューブに編成することもできます。

さまざまな期間での移動平均と移動和を計算したり、選択したカラムの値が変化したときに集計とランクをリセットしたり、複雑な比率を単純な言葉で表現したりできます。1 つのクエリ式のスコープ内で、それぞれ独自の分割ルールを持ついくつかの異なる OLAP 関数を定義することができます。

OLAP の評価

OLAP の評価は、最終的な結果に影響を及ぼすクエリ実行のいくつかのフェーズとして概念化できます。

OLAP の実行フェーズは、クエリ内の対応する句によって識別されます。 たとえば、SQL クエリの指定にウィンドウ関数が含まれている場合は、**WHERE**、**JOIN**、**GROUP BY**、**HAVING** 句が先に処理されます。 **GROUP BY** 句でグループが定義された後、クエリの **ORDER BY** 句に含まれる最後の **SELECT** リストが評価される前に、パーティションが作成されます。

グループ化の際には、NULL 値はすべて同じグループと見なされます (それぞれの NULL 値が等しくない場合でも同様です)。

HAVING 句は、**WHERE** 句に類似しており、**GROUP BY** 句の結果に対するフィルタとして機能します。

ANSI SQL 標準に基づく SQL 文と **SELECT**、**FROM**、**WHERE**、**GROUP BY**、**HAVING** 句を含んだ単純なクエリ仕様のセマンティックを考えてみます。

1. クエリにより、**FROM** 句のテーブル式を満たすローセットが取得されます。

2. **WHERE** 句の述部が、テーブルから取得したローに適用されます。 **WHERE** 句の条件を満たさない (条件が true にならない) ローが除外されます。
3. 残りの各ローについて、**SELECT** リストおよび **GROUP BY** 句に含まれている式 (集合関数を除く) が評価されます。
4. **GROUP BY** 句の式の重複しない値に基づいて、結果のローがグループ化されます (NULL は各ドメインで特殊な値として扱われます)。 **PARTITION BY** 句がある場合、**GROUP BY** 句の式はパーティションキーとして使用されます。
5. 各パーティションについて、**SELECT** リストまたは **HAVING** 句の集合関数が評価されます。いったん集合関数を適用すると、中間の結果セットには個々のテーブルローが含まれなくなります。 新しい結果セットには、**GROUP BY** の式と、各パーティションについて計算した集合関数の値が含まれます。
6. **HAVING** 句の条件が結果グループに適用されます。 **HAVING** 句の条件を満たさないグループが除外されます。
7. **PARTITION BY** 句で定義された境界に基づいて結果が分割されます。 結果ウィンドウについて、OLAP ウィンドウ関数 (ランク付け関数および集合関数) が計算されます。

図 1 : OLAP の SQL 処理

GROUP BY 句の拡張

GROUP BY 句を拡張することで、次のような処理を行う複雑な SQL 文を記述できます。

- 入力ローを複数の次元に分割し、結果グループの複数のサブセットを組み合わせる。
- 「データキューブ」を作成し、データマイニング分析のための疎密度の多次元結果セットを用意する。
- 元のグループを含んだ結果セットを作成する (必要に応じて、小計ローと合計ローを含める場合もある)。

ROLLUP や **CUBE** など、OLAP の Grouping() 操作は、プレフィクスや小計ローとして概念化できます。

プレフィクス

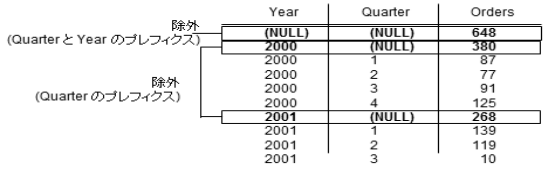
GROUP BY 句を含むクエリでは、*プレフィクス*のリストが作成されます。プレフィクスとは、**GROUP BY** 句の項目のサブセットであり、クエリの **GROUP BY** 句の項目のうち最も右にある 1 つ以上の項目を除外することで作成されます。残りのカラムは*プレフィクスカラム*と呼ばれます。

ROLLUP の例 1 - 次の **ROLLUP** のクエリ例では、**GROUP BY** リストに 2 つの変数 (*Year* と *Quarter*) が含まれています。

```
SELECT year (OrderDate) AS Year, quarter(OrderDate)
  AS Quarter, COUNT(*) Orders
FROM SalesOrders
GROUP BY ROLLUP(Year, Quarter)
ORDER BY Year, Quarter
```

このクエリには次の 2 つのプレフィクスがあります。

- Quarter を除外するプレフィクス ― 一連のプレフィクスカラムには 1 つのカラム `Year` が含まれます。
- Quarter と Year の両方を除外するプレフィクス ― プレフィクスカラムは存在しません。



Year	Quarter	Orders
(NULL)	**(NULL)**	**648**
2000	**(NULL)**	**380**
2000	1	87
2000	2	77
2000	3	91
2000	4	125
2001	**(NULL)**	**268**
2001	1	139
2001	2	119
2001	3	10

注意： GROUP BY リストには、項目と同じ数のプレフィクスが含まれます。

GROUP BY での ROLLUP と CUBE

ROLLUP と **CUBE** は、一般的なグループ化プレフィクスを指定する構文簡略化パターンです。

GROUP BY ROLLUP

ROLLUP 演算子では、グループ化式が順に並べられたリストを引数として指定する必要があります。

ROLLUP 構文。

```
SELECT … [ GROUPING (column-name) … ] …
GROUP BY [ expression [, …]
| ROLLUP ( expression [, …] ) ]
```

GROUPING は、カラム名をパラメータとして受け取り、次の表に示すようにブール値を返します。

表 1 : ROLLUP 演算子が指定された GROUPING によって返される値

結果値の種類	GROUPING の戻り値
ROLLUP 処理によって作成された NULL	1 (真)
ローが小計であることを示す NULL	1 (真)
ROLLUP 処理によって作成されたもの以外の NULL	0 (偽)
格納されていた NULL	0 (偽)

ROLLUP は、**GROUP BY** 句に指定された標準の集合値を最初に計算します。 次に、**ROLLUP** はグループ化を行うカラムのリストを右から左に移動し、より高いレベルの小計を連続して作成します。最後に総計が作成されます。 グループ化するカラムの数が *n* 個の場合、**ROLLUP** は *n* に 1 を加えたレベルの小計を作成します。

SQL 構文の例	定義されるセット
`GROUP BY ROLLUP (A, B, C);`	(A, B, C) (A, B) (A) ( )

ROLLUP と小計ロー

ROLLUP は、一連の **GROUP BY** クエリの **UNION** に等しくなります。次の 2 つのクエリの結果セットは等しくなります。 **GROUP BY** (A, B) の結果セットは、A と B に定数が含まれているすべてのローについての小計から成ります。UNION を可能にするために、カラム C には NULL が割り当てられます。

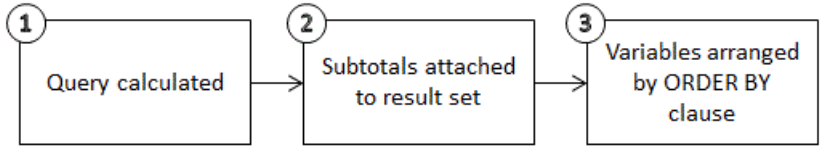
ROLLUP クエリの例	ROLLUP を使用せずに記述した同じ内容のクエリ
`select year(orderdate) as year, quarter(orderdate) as Quarter, count(*) Ordersfrom SalesOrdersgroup by Rollup (year, quarter)order by year, quarter`	`Select null,null, count(*) Orders from SalesOrdersunion allSELECT year(orderdate) AS YEAR, NULL, count(*) Orders from SalesOrdersGROUP BY year(orderdate) union allSELECT year(orderdate) as YEAR, quarter(orderdate) as QUATER, count(*) Orders from SalesOrdersGROUP BY year(orderdate), quarter(orderdate)`

小計ローはデータの分析に役立ちます。特に、データが大量にある場合、データにさまざまな次元がある場合、データがさまざまなテーブルに含まれている場合、

またはまったく異なるデータベースに含まれている場合に威力を発揮します。たとえば販売マネージャが、売上高についてのレポートを営業担当者別、地域別、四半期別に整理して、売上パターンの理解に役立てることができます。データの小計は、販売マネージャが売上高の全体像をさまざまな視点から分析するのに役立ちます。販売マネージャが比較したいと考える基準に基づいて要約情報が提供されていれば、データの分析を容易に行うことができます。

OLAP を使用すると、ローおよびカラムの小計を分析して計算する処理をユーザの目から隠すことができます。

図 2 : 小計



1. このステップで、まだ **ROLLUP** とは見なされない中間の結果セットが生成されます。
2. 小計が評価され、結果セットに付加されます。
3. クエリ内の **ORDER BY** 句に従ってローが並べられます。

NULL 値と小計ロー

GROUP BY 操作に対する入力でローに NULL が含まれている場合、**ROLLUP** または **CUBE** 操作によって追加された小計ローと、最初の入力データの一部として NULL 値を含んでいるローが混在している可能性があります。

Grouping() 関数は、小計ローをその他のローから区別します。具体的には、**GROUP BY** リストのカラムを引数として受け取り、そのカラムが小計ローであるために NULL になっている場合は 1 を返し、それ以外の場合は 0 を返します。

次の例では、結果セットの中に Grouping() カラムが含まれています。強調表示されているローは、小計ローであるために NULL を含んでいるのではなく、入力データの結果として NULL を含んでいるローです。Grouping() カラムは強調表示されています。このクエリは、`Employees` テーブルと `SalesOrders` テーブルの間の外部ジョインです。このクエリでは、テキサス、ニューヨーク、またはカリフォルニアに住んでいる女性従業員を選択しています。営業担当者でない (したがって売上がない) 女性従業員については、カラムに NULL が表示されます。

注意： たとえば、SAP Sybase IQ デモデータベース `iqdemo.db` を使用します。

```
SELECT Employees.EmployeeID as EMP, year(OrderDate) as
  YEAR, count(*) as ORDERS, grouping(EMP) as
  GE, grouping(YEAR) as GY
  FROM Employees LEFT OUTER JOIN SalesOrders on
  Employees.EmployeeID = SalesOrders.SalesRepresentative
```

```
  WHERE Employees.Sex IN ('F') AND Employees.State
  IN ('TX', 'CA', 'NY')
GROUP BY ROLLUP (YEAR, EMP)
ORDER BY YEAR, EMP
```

前述のクエリは、以下を返します。

```
EMP          YEAR       ORDERS      GE      GY
------       ----       ------      --      --
NULL         NULL            5       1       0
NULL         NULL          169       1       1
 102         NULL            1       0       0
 309         NULL            1       0       0
1062         NULL            1       0       0
1090         NULL            1       0       0
1507         NULL            1       0       0
NULL         2000           98       1       0
 667         2000           34       0       0
 949         2000           31       0       0
1142         2000           33       0       0
NULL         2001           66       1       0
 667         2001           20       0       0
949          2001           22       0       0
1142         2001           24       0       0
```

個々のプレフィクスについて、プレフィクスカラムに同じ値が含まれているすべてのローに関する*小計ロー*が作成されます。

ROLLUP の結果を具体的に説明するために、前述のクエリの例をもう一度詳しく見ていきます。

```
SELECT year (OrderDate) AS Year, quarter
  (OrderDate) AS Quarter, COUNT (*) Orders
FROM SalesOrders
  GROUP BY ROLLUP (Year, Quarter)
  ORDER BY Year, Quarter
```

このクエリでは、`Year` カラムを含んでいるプレフィクスにより、Year=2000 の合計ローと Year=2001 の合計ローが作成されます。このプレフィクスに関する 1 つの合計ローはカラムを含んでいません。これは、中間の結果セットに含まれているすべてのローの小計です。

小計ローの各カラムの値は、次のようになっています。

- プレフィクスに含まれているカラム ― そのカラムの値です。たとえば、前述のクエリでは、Year=2000 のローに関する小計の `Year` カラムの値は 2000 になります。
- プレフィクスから除外されたカラム ― NULL です。たとえば、Year カラムから成るプレフィクスにより生成された小計ローでは、`Quarter` カラムの値は NULL になります。
- 集合関数 ― 除外されているカラムの値を計算した結果です。

小計値は、集約されたローではなく、基本データのローに対して計算されます。 多くの場合、たとえば **SUM** や **COUNT** などでは結果は等しくなりますが、**AVG**、**STDDEV**、**VARIANCE** などの統計関数では結果が異なってくるため、この区別は重要です。

ROLLUP 演算子には次の制限があります。

- **ROLLUP** 演算子は、**GROUP BY** 句で使用できるすべての集合関数をサポートしています (**COUNT DISTINCT** と **SUM DISTINCT** は除く)。
- **ROLLUP** は、**SELECT** 文のみで使用できます。サブクエリでは **ROLLUP** を使用できません。
- 複数の **ROLLUP**、**CUBE**、**GROUP BY** カラムを同じ **GROUP BY** 句で組み合わせたグループ化の指定は、現在サポートされていません。
- **GROUP BY** のキーに定数式を指定することはできません。

ROLLUP 例 2 - 次は、**ROLLUP** と **GROUPING** の使用例です。**GROUPING** によって作成される一連のマスクカラムを表示します。 カラム S、N、C に表示されている数字 0 と 1 は、**GROUPING** からの戻り値であり、**ROLLUP** の結果の値を表現しています。マスクが "011" であれば小計のローであり、"111" であれば総計のローであると特定できます。これを利用して、クエリの結果をプログラムで分析することが可能です。

```
SELECT size, name, color, SUM(quantity),
  GROUPING(size) AS S,
  GROUPING(name) AS N,
  GROUPING(color) AS C
FROM Products
GROUP BY ROLLUP(size, name, color) HAVING (S=1 or N=1 or C=1)
ORDER BY size, name, color;
```

前述のクエリは、以下を返します。

```
size                name           color     SUM  S  N  C
----                -----          ------     ---  -  -  -
(NULL)              (NULL)          (NULL)    496  1  1  1
Large               (NULL)          (NULL)     71  0  1  1
Large               Sweatshirt      (NULL)     71  0  0  1
Medium              (NULL)          (NULL)    134  0  1  1
Medium              Shorts          (NULL)     80  0  0  1
Medium              Tee Shirt       (NULL)     54  0  0  1
One size fits all   (NULL)          (NULL)    263  0  1  1
One size fits all   Baseball Cap    (NULL)    124  0  0  1
One size fits all   Tee Shirt       (NULL)     75  0  0  1
One size fits all   Visor           (NULL)     64  0  0  1
Small               (NULL)          (NULL)     28  0  1  1
Small               Tee Shirt       (NULL)     28  0  0  1
```

注意： ROLLUP 例 2 の結果では、SUM カラムは `SUM(products.quantity)` で表示します。

ROLLUP 例 3 - 次の例は、**GROUPING** を使用して、最初から格納されていた NULL 値と **ROLLUP** 操作によって生成された “NULL” 値を区別する方法を示しています。

このクエリで指定されているとおり、最初から格納されていた NULL 値はカラム prod_id に [NULL] として表示され、**ROLLUP** によって生成された “NULL” 値はカラム PROD_IDS で ALL に置き換えられます。

```
SELECT year(ShipDate) AS Year,
                ProductID, SUM(quantity)AS OSum,
CASE
        WHEN GROUPING(Year) = 1
        THEN 'ALL'
        ELSE
        CAST(Year AS char(8))
END,
CASE
        WHEN GROUPING(ProductID) = 1
        THEN 'ALL'
        ELSE
        CAST(ProductID as char(8))
END
FROM SalesOrderItems
GROUP BY ROLLUP(Year, ProductID) HAVING OSum > 36
ORDER BY Year, ProductID;
```

前述のクエリは、以下を返します。

```
Year  ProductID  OSum  ...(Year)...  ...(ProductID)...
---------     -------   ---   ----------   --------
NULL           NULL   28359   ALL          ALL
2000           NULL   17642   2000         ALL
2000           300     1476   2000         300
2000           301     1440   2000         301
2000           302     1152   2000         302
2000           400     1946   2000         400
2000           401     1596   2000         401
2000           500     1704   2000         500
2000           501     1572   2000         501
2000           600     2124   2000         600
2000           601     1932   2000         601
2000           700     2700   2000         700
2001           NULL   10717   2001         ALL
2001           300      888   2001         300
2001           301      948   2001         301
2001           302      996   2001         302
2001           400     1332   2001         400
2001           401     1105   2001         401
2001           500      948   2001         500
2001           501      936   2001         501
2001           600      936   2001         600
2001           601      792   2001         601
2001           700     1836   2001         700
```

ROLLUP 例 4 - 次のクエリ例は、注文数を年別および四半期別に集計したデータを返します。

```
SELECT year (OrderDate) AS Year,
quarter(OrderDate) AS Quarter, COUNT (*) Orders
```

```
FROM SalesOrders
GROUP BY ROLLUP (Year, Quarter)
ORDER BY Year, Quarter
```

次の図は、このクエリの結果を示しています。結果セット内の小計ローは強調表示されています。各小計ローでは、その小計の計算対象になったカラムに NULL 値が格納されています。

	Year	Quarter	Orders
①	**(NULL)**	**(NULL)**	**648**
②	**2000**	**(NULL)**	**380**
③	2000	1	87
	2000	2	77
	2000	3	91
	2000	4	125
②	**2001**	**(NULL)**	**268**
③	2001	1	139
	2001	2	119
	2001	3	10

ロー①は、両方の年 (2000 年および 2001 年) のすべての四半期の注文数の合計を示しています。このローは、`Year` カラムと `Quarter` カラムの両方が NULL であり、すべてのカラムがプレフィクスから除外されています。

注意： すべての **ROLLUP** 操作によって返される結果セットには、集合カラムを除くすべてのカラムが NULL であるローが 1 つ含まれています。このローは、集合関数に対する全カラムの要約を表しています。たとえば、集合関数として SUM を使用している場合は、このローはすべての値の総計を表します。

ロー②は、2000 年および 2001 年の注文数の合計をそれぞれ示しています。どちらのローも、`Quarter` カラムは NULL になっています。これは、このカラムの値を加算して、`Year` の小計を出しているためです。結果セットにこのようなローが含まれる数は、**ROLLUP** クエリで使用されている変数の数に応じて異なります。

③としてマークされている残りのローは要約情報を示し、それぞれの年の各四半期の注文数の合計を表しています。

ROLLUP 例 5 - **ROLLUP** 操作の例では、年別、四半期別、地域別の注文数を集計する、少し複雑な結果セットを返します。この例では、第 1 および第 2 四半期と 2 つの地域 (カナダと東部地区) だけを分析します。

```
SELECT year(OrderDate) AS Year, quarter(OrderDate)AS Quarter,
region, COUNT(*) AS Orders
FROM SalesOrders WHERE region IN ('Canada','Eastern') AND quarter IN
(1, 2)
GROUP BY ROLLUP (Year, Quarter, Region)ORDER BY Year, Quarter, Region
```

次の図は、このクエリの結果セットを示しています。各小計ローでは、その小計の計算対象になったカラムに NULL が格納されています。

	Year	Quarter	Region	Orders
①	**(NULL)**	**(NULL)**	**(NULL)**	**183**
	2000	**(NULL)**	**(NULL)**	**68**
	2000	1	(NULL)	36
	2000	1	Canada	3
②	2000	1	Eastern	33
	2000	2	(NULL)	32
	2000	2	Canada	3
	2000	2	Eastern	29
	2001	**(NULL)**	**(NULL)**	**115**
	2001	1	(NULL)	57
	2001	1	Canada	11
	2001	1	Eastern	46
	2001	2	(NULL)	58
	2001	2	Canada	4
	2001	2	Eastern	54

ロー①はすべてのローの集約結果であり、`Year`、`Quarter`、`Region` カラムに NULL が含まれています。このローの `Orders` カラムの値は、カナダと東部地区における、2000 年と 2001 年の第 1 および第 2 四半期の注文数の合計を示しています。

②としてマークされているローは、それぞれの年 (2000 年と 2001 年) におけるカナダと東部地区の第 1 および第 2 四半期の注文数の合計を示しています。ロー②の値を足すと、ロー①に示されている総計に等しくなります。

③としてマークされているローは、特定の年および四半期の全地域の注文数の合計を示しています。

Year	Quarter	Region	Orders
(NULL)	(NULL)	(NULL)	183
2000	(NULL)	(NULL)	68
2000	**1**	**(NULL)**	**36**
2000	1	Canada	3
2000	1	Eastern	33
2000	**2**	**(NULL)**	**32**
2000	2	Canada	3
2000	2	Eastern	29
2001	(NULL)	(NULL)	**115**
2001	**1**	**(NULL)**	57
2001	1	Canada	11
2001	1	Eastern	46
2001	**2**	**(NULL)**	**58**
2001	2	Canada	4
2001	2	Eastern	54

③

④としてマークされているローは、結果セット内のそれぞれの年の各四半期の各地域の注文の合計数を示しています。

Year	Quarter	Region	Orders
(NULL)	(NULL)	(NULL)	183
2000	(NULL)	(NULL)	68
2000	1	(NULL)	36
2000	**1**	**Canada**	**3**
2000	**1**	**Eastern**	**33**
2000	2	(NULL)	32
2000	**2**	**Canada**	**3**
2000	**2**	**Eastern**	**29**
2001	(NULL)	(NULL)	115
2001	1	(NULL)	57
2001	**1**	**Canada**	**11**
2001	**1**	**Eastern**	**46**
2001	2	(NULL)	58
2001	**2**	**Canada**	**4**
2001	**2**	**Eastern**	**54**

④

GROUP BY CUBE

GROUP BY 句の **CUBE** 演算子は、データを複数の次元 (グループ化式) でグループ化することでデータを分析します。

CUBE では、次元の順序リストを引数として指定する必要があります。これにより、**SELECT** 文の中で、そのクエリに指定した次元グループの考えられるすべての組み合わせの小計を計算し、選択した複数のカラムのすべての値の組み合わせに関する要約を示す結果セットを生成できます。

CUBE 構文：

```
            SELECT … [ GROUPING (column-name) … ] …
GROUP BY [ expression [,…]
| CUBE ( expression [,…] ) ]
```

GROUPING は、カラム名をパラメータとして受け取り、次の表に示すようにブール値を返します。

表 2 : CUBE 演算子が指定された GROUPING によって返される値

結果値の種類	GROUPING が返す値
CUBE 処理によって作成された NULL	1 (真)
ローが小計であることを示す NULL	1 (真)
CUBE 処理によって作成されたもの以外の NULL	0 (偽)
格納されていた NULL	0 (偽)

CUBE は、同じ階層の一部ではない次元を扱うときに特に有用です。

SQL 構文の例	定義されるセット
`GROUP BY CUBE (A, B, C);`	(A, B, C) (A, B) (A, C) (A) (B, C) (B) (C) ( )

CUBE 演算子には次の制限があります。

- **CUBE** 演算子は、**GROUP BY** 句で使用できるすべての集合関数をサポートしていますが、**COUNT DISTINCT** または **SUM DISTINCT** では、**CUBE** は現在サポートされていません。
- **CUBE** は、逆分散統計関数 (**PERCENTILE_CONT** と **PERCENTILE_DISC**) では現在サポートされていません。
- **CUBE** は、**SELECT** 文のみで使用できます。**SELECT** サブクエリでは **CUBE** を使用できません。
- **ROLLUP**、**CUBE**、**GROUP BY** カラムを同じ GROUP BY 句で組み合わせた **GROUPING** の指定は、現在サポートされていません。
- **GROUP BY** のキーに定数式を指定することはできません。

注意： キューブのサイズがテンポラリキャッシュのサイズを超えると、**CUBE** のパフォーマンスが低下します。

GROUPING と **CUBE** 演算子を併用すると、格納されていた NULL 値と **CUBE** によって作成されたクエリ結果の NULL 値を区別できます。

GROUPING 関数を使用して結果を分析する方法については、**ROLLUP** 演算子の説明で示されている例を参照してください。

すべての **CUBE** 操作が返す結果セットには、集合カラムを除くすべてのカラムの値が NULL であるローが、少なくとも 1 つは含まれています。このローは、集合関数に対する全カラムの要約を表しています。

CUBE 例 1 - 次の例は、対象者の州 (地理的な位置)、性別、教育レベル、および収入などで構成される調査データを使用したクエリです。 最初に紹介するクエリには **GROUP BY** 句が指定されています。この句は、クエリの結果を、`census` テーブルの `state`、`gender`、`education` カラムの値に応じてローグループに分類し、収入の平均とローの合計数をグループごとに計算します。 このクエリには **GROUP BY** 句のみを使用し、ローのグループ化に **CUBE** 演算子を使用していません。

```
SELECT State, Sex as gender, DepartmentID,
COUNT(*),CAST(ROUND(AVG(Salary),2) AS NUMERIC(18,2))AS AVERAGEFROM
employees WHERE state IN ('MA' , 'CA')GROUP BY State, Sex,
DepartmentIDORDER BY 1,2;
```

このクエリの結果セットを次に示します。

```
state  gender  DepartmentID  COUNT()     AVERAGE
-----  ------  ------------  --------    ---------
CA       F       200         2          58650.00
CA       M       200             1          39300.00
```

GROUP BY 句の **CUBE** 拡張機能を使用すると、調査データを 1 回参照するだけで、調査データ全体における州別、性別、教育別の平均収入を計算し、`state`、`gender`、`education` カラムの考えられるすべての組み合わせでの平均収入を計

算できます。 **CUBE** 演算子を使用すると、たとえば、すべての州における全女性の平均収入を計算したり、調査対象者全員の平均収入を、各自の教育別および州別に計算したりすることができます。

CUBE でグループを計算する場合、計算されたグループのカラムに NULL 値が生成されます。 最初からデータベース内に格納されていた NULL であるのか、**CUBE** の結果として生成された NULL であるのかを区別するには、**GROUPING** 関数を使用します。 **GROUPING** 関数は、指定されたカラムが上位レベルのグループにマージされている場合は 1 を返します。

CUBE 例 2 - 次のクエリは、**GROUPING** 関数と **GROUP BY CUBE** を併用した例を示しています。

```
SELECT case grouping(State) WHEN 1 THEN 'ALL' ELSE StateEND AS
c_state, case grouping(sex) WHEN 1 THEN 'ALL'ELSE Sex end AS
c_gender, case grouping(DepartmentID)WHEN 1 THEN 'ALL' ELSE
cast(DepartmentID as char(4)) endAS c_dept, COUNT(*),
CAST(ROUND(AVG(salary),2) ASNUMERIC(18,2))AS AVERAGEFROM employees
WHERE state IN ('MA' , 'CA')GROUP BY CUBE(state, sex,
DepartmentID)ORDER BY 1,2,3;
```

このクエリの結果は次のとおりです。 **CUBE** が生成した小計ローを示す NULL 値が、クエリ内の指定によって小計ローで ALL に置き換えられています。

```
c_state   c_gender  c_dept    COUNT()   AVERAGE
-------   --------  -------    -----     --------
ALL       ALL        200         3       52200.00
ALL       ALL        ALL         3       52200.00
ALL       F          200         2       58650.00
ALL       F          ALL         2       58650.00
ALL       M         200          1       39300.00
ALL       M         ALL          1       39300.00
CA        ALL       200          3       52200.00
CA        ALL       ALL          3       52200.00
CA        F         200          2       58650.00
CA        F         ALL          2       58650.00
CA        M         200          1       39300.00
CA        M         ALL          1       39300.00
```

CUBE 例 3 - この例のクエリは、注文数の合計を要約する結果セットを返し、次に、年別および四半期別の注文数の小計を計算します。

注意： 比較する変数の数が増えると、キューブの計算コストが急激に増大します。

```
SELECT year (OrderDate) AS Year, quarter(OrderDate) AS Quarter, COUNT
(*) OrdersFROM SalesOrdersGROUP BY CUBE (Year, Quarter)ORDER BY Year,
Quarter
```

次の図は、このクエリの結果セットを示しています。この結果セットでは、小計ローが強調表示されています。各小計ローでは、その小計の計算対象になったカラムに NULL が格納されています。

	Year	Quarter	Orders
①	(NULL)	(NULL)	648
②	(NULL)	1	226
	(NULL)	2	196
	(NULL)	3	101
	(NULL)	4	125
③	2000	(NULL)	380
	2000	1	87
	2000	2	77
	2000	3	91
	2000	4	125
③	2001	(NULL)	268
	2001	1	139
	2001	2	119
	2001	3	10

先頭のロー①は、両方の年のすべての四半期の注文数の合計を示しています。`Orders` カラムの値は、③としてマークされている各ローの値の合計です。これは、②としてマークされている 4 つのローの値の合計でもあります。

②としてマークされている一連のローは、両方の年の四半期別の注文数の合計を示しています。③としてマークされている 2 つのローは、それぞれ 2000 年および 2001 年のすべての四半期の注文数の合計を示しています。

分析関数

SAP Sybase IQ では、1 つの SQL 文内で複雑なデータ分析を実行できる機能を備えた単純な集合関数とウィンドウ集合関数の両方を提供しています。

これらの関数を使用して、たとえば「ダウ工業株 30 種平均の四半期の移動平均はどうなっているか」または「各部署のすべての従業員とその累積給与を一覧表示せよ」というクエリの結果を計算できます。 さまざまな期間における移動平均と累積和を計算し、集計とランクを分割できるため、パーティション値が変化したときに集合計算がリセットされます。1 つのクエリ式のスコープ内で、それぞれ独自の分割ルールを持ついくつかの異なる OLAP 関数を定義することができます。分析関数は 2 つのカテゴリに分けられます。

- 単純な集合関数 (**AVG**、**COUNT**、**MAX**、 **MIN**、**SUM** など) は、データベースに含まれるローのグループのデータを要約します。 **SELECT** 文の **GROUP BY** 句を使用して、グループを作成します。

- 1 つの引数を取る単項の統計集合関数には、**STDDEV**、**STDDEV_SAMP**、**STDDEV_POP**、**VARIANCE**、**VAR_SAMP**、**VAR_POP** があります。

単純な集合関数でも単項の集合関数でも、データベース内のローのグループに関するデータを要約することができ、ウィンドウ指定と組み合わせて、処理の際に結果セットに対する移動ウィンドウを計算することができます。

注意： 集合関数 **AVG**、**SUM**、**STDDEV**、**STDDEV_POP**、**STDDEV_SAMP**、**VAR_POP**、**VAR_SAMP**、**VARIANCE** は、バイナリデータ型 BINARY と VARBINARY をサポートしていません。

単純な集合関数

単純な集合関数 (**AVG**、**COUNT**、**MAX**、 **MIN**、**SUM** など) は、データベースに含まれるローのグループのデータを要約します。

SELECT 文の **GROUP BY** 句を使用して、グループを作成します。 集合関数は、SELECT リストと、**SELECT** 文の **HAVING** および **ORDER BY** 句の中のみで使用できます。

注意： Grouping() 関数を除き、単純な集合関数と単項の集合関数はどちらも、SQL クエリの指定に「ウィンドウ句」(*ウィンドウ*) を組み込むウィンドウ関数として使用できます。これにより、処理時に結果セットに対して概念的に移動ウィンドウを作成できます。

ウィンドウ

OLAP に関する ANSI SQL 拡張で導入された主な機能は、「*ウィンドウ*」という名前の構成体です。 このウィンドウ拡張により、ユーザはクエリの結果セット (クエリの論理パーティション) をパーティションと呼ばれるローのグループに分割し、現在のローについて集計するローのサブセットを決定できます。

1 つのウィンドウで 3 つのウィンドウ関数クラス (ランク付け関数、ローナンバリング関数、ウィンドウ集合関数) を使用できます。

```
<WINDOWED TABLE FUNCTION TYPE> ::=
  <RANK FUNCTION TYPE> <LEFT PAREN> <RIGHT PAREN>
  | ROW_NUMBER <LEFT PAREN> <RIGHT PAREN>
  | <WINDOW AGGREGATE FUNCTION>
```

ウィンドウ拡張は、ウィンドウ名または指定に対するウィンドウ関数の種類を指定し、1 つのクエリ式のスコープ内のパーティション化された結果セットに適用されます。 ウィンドウパーティションは、特殊な **OVER** 句の 1 つ以上のカラムで定義されている、クエリから返されるローのサブセットです。

```
olap_function() OVER (PARTITION BY col1, col2...)
```

ウィンドウ操作では、パーティション内の各ローのランク付け、パーティション内のローの値の分布、および類似の操作などの情報を設定できます。また、データの移動平均や合計を計算し、データおよびそのデータの操作に対する影響を評価する機能を拡張することもできます。

OLAP ウィンドウの 3 つの重要な側面

OLAP ウィンドウは、ウィンドウパーティション、ウィンドウ順序、ウィンドウフレームという 3 つの重要な側面から成ります。それぞれの要素は、その時点でウィンドウ内で可視となるデータローに大きな影響を与えます。 また、OLAP の **OVER** 句は、次の 3 つの特徴的な機能により、OLAP 関数を他の分析関数やレポート関数から区別します。

- ウィンドウパーティションの定義 (**PARTITION BY** 句)。
- パーティション内でのローの順序付け (**ORDER BY** 句)。
- ウィンドウフレームの定義 (ROWS/RANGE 指定)。

複数のウィンドウ関数を指定したり、冗長なウィンドウ定義を避けたりするために、OLAP ウィンドウ指定に関して名前を指定できます。 その場合は、キーワード **WINDOW** の後に少なくとも 1 つのウィンドウ定義を指定します (複数指定する場合はカンマで区切ります)。ウィンドウ定義には、クエリ内でウィンドウを識別するための名前と、ウィンドウパーティション、順序、フレームを定義するためのウィンドウ指定の詳細を含めます。

```
<WINDOW CLAUSE> ::= <WINDOW DEFINITION LIST>
```

```
<WINDOW DEFINITION LIST> ::=
  <WINDOW DEFINITION> [ { <COMMA> <WINDOW DEFINITION>
  } . . . ]
```

```
<WINDOW DEFINITION> ::=
  <NEW WINDOW NAME> AS <WINDOW SPECIFICATION>
```

```
<WINDOW SPECIFICATION DETAILS> ::=
  [ <EXISTING WINDOW NAME> ]
  [ <WINDOW PARTITION CLAUSE> ]
  [ <WINDOW ORDER CLAUSE> ]
  [ <WINDOW FRAME CLAUSE> ]
```

ウィンドウパーティション内の各ローについて、ウィンドウフレームを定義できます。ウィンドウフレームにより、パーティションの現在のローに対して計算を実行するときに使用される、ローの指定範囲を変更できます。現在のローは、ウィンドウフレームの開始ポイントと終了ポイントを決定するための参照ポイントとなります。

ウィンドウ指定は、物理的なローの数 (ウィンドウフレーム単位 ROWS を定義するウィンドウ指定を使用) または論理的な数値の間隔 (ウィンドウフレーム単位 RANGE を定義するウィンドウ指定を使用) に基づきます。

OLAP のウィンドウ操作では、次のカテゴリの関数を使用できます。

- ランク付け関数
- ウィンドウ集合関数
- 統計集合関数
- 分散統計関数

ウィンドウパーティション

ウィンドウパーティションとは、 **PARTITION BY** 句を使用して、ユーザ指定の結果セット (入力ロー) を分割することです。

パーティションは、カンマで区切られた 1 つ以上の値の式によって定義されます。パーティションに分割されたデータは暗黙的にソートされ、デフォルトのソート順序は昇順 (ASC) になります。

```
<WINDOW PARTITION CLAUSE> ::=
  PARTITION BY <WINDOW PARTITION EXPRESSION LIST>
```

ウィンドウパーティション句を指定しなかった場合は、入力が 1 つのパーティションとして扱われます。

注意： 統計関数に対して*「パーティション」*という用語を使用した場合は、結果セットのローを **PARTITION BY** 句に基づいて分割することのみを意味します。

ウィンドウパーティションは任意の式に基づいて定義できます。また、ウィンドウパーティションの処理は GROUPING の後に行われるため (**GROUP BY** 句が指定されている場合)、**SUM**、**AVG**、**VARIANCE** などの集合関数の結果をパーティションの式で使用できます。したがって、パーティションを使用すると、**GROUP BY** 句や **ORDER BY** 句とはまた別に、グループ化と順序付けの操作を実行できます。たとえば、ある数量の最大 **SUM** を求めるなど、集合関数に対して集合関数を計算するクエリを記述できます。

GROUP BY 句がなくても、**PARTITION BY** 句を指定できます。

ウィンドウ順序

ウィンドウ順序とは、各ウィンドウパーティション内の結果 (ロー) を WINDOW ORDER 句に基づいて並べることです。この句には、1 つ以上の値の式をカンマ区切りで指定します。

WINDOW ORDER 句を指定しなかった場合は、入力ローが任意の順序で処理されることがあります。

```
<WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>
```

OLAP の WINDOW ORDER 句は、非ウィンドウクエリの式に指定できる **ORDER BY** 句とは異なります。

OLAP 関数で使用する **ORDER BY** 句は、通常はウィンドウパーティション内のローをソートするための式を定義しますが、**PARTITION BY** 句がなくても **ORDER BY** 句を使用できます。この場合、このソート指定によって、意味のある (かつ目的どおりの) 順序で並べられた中間の結果セットに OLAP 関数を確実に適用できます。

OLAP のランク付け関数には順序の指定が必須であり、ランキング値の基準は、ランク付け関数の引数ではなく **ORDER BY** 句で指定します。 OLAP の集合関数では、通常は **ORDER BY** 句の指定は必須ではありませんが、ウィンドウフレームを定義するときには必須です。これは、各フレームの適切な集合値を計算する前に、パーティション内のローをソートしなければならないためです。

この **ORDER BY** 句には、昇順および降順のソートを定義するためのセマンティックと、NULL 値の取り扱いに関する規則を指定します。OLAP 関数は、デフォルトでは昇順 (最も小さい値が 1 番目にランク付けされる) を使用します。

これは **SELECT** 文の最後に指定する **ORDER BY** 句のデフォルト動作と同じですが、連続的な計算を行う場合にはわかりにくいかもしれません。 ほとんどの OLAP の計算では、降順 (最も大きい値が 1 番目にランク付けされる) でのソートが必要になります。この要件を満たすには、**ORDER BY** 句に明示的に DESC キーワードを指定する必要があります。

注意： ランク付け関数は、ソートされた入力のみを扱うように定義されているため、「WINDOW ORDER 句」を指定する必要があります。「クエリ指定」の「ORDER BY 句」と同様に、デフォルトのソート順序は昇順です。

「ウィンドウフレーム単位」で RANGE を使用する場合も、「WINDOW ORDER 句」を指定する必要があります。 RANGE の場合は、「WINDOW ORDER 句」に 1 つの式のみを指定します。

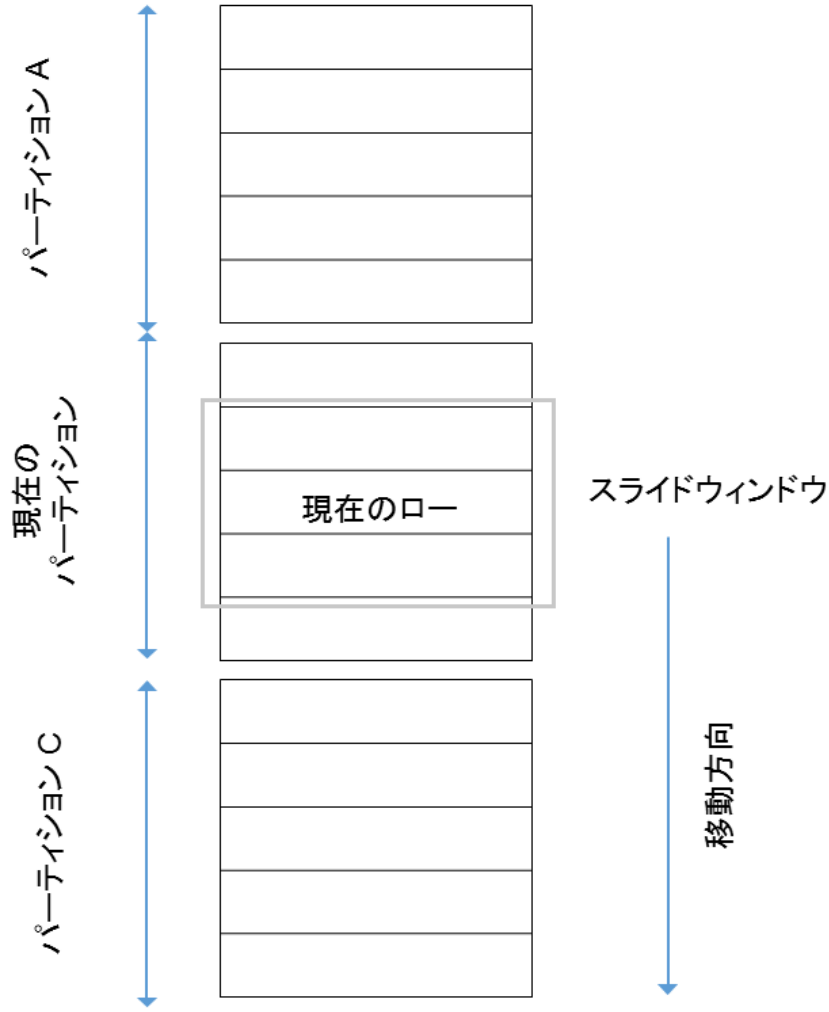
ウィンドウフレーム

ランク付け関数を除く OLAP 集合関数では、WINDOW FRAME 句を使用してウィンドウフレームを定義できます。この句には、現在のローを基準としてウィンドウの開始位置と終了位置を指定します。

```
<WINDOW FRAME CLAUSE> ::=
  <WINDOW FRAME UNIT>
  <WINDOW FRAME EXTENT>
```

これにより、パーティション全体の固定的な内容ではなく、移動するフレームの内容に対して OLAP 関数を計算できます。定義にもよりますが、パーティションには開始ローと終了ローがあり、ウィンドウフレームは開始ポイントからパーティションの終了位置に向けてスライドします。

図 3 : 分割された入力と、3 ロー分の移動ウィンドウ



UNBOUNDED PRECEDING と UNBOUNDED FOLLOWING
ウィンドウフレームは、パーティションの先頭 (UNBOUNDED PRECEDING)、最後 (UNBOUNDED FOLLOWING)、または両方まで到達する無制限の集合グループによって定義されます。

UNBOUNDED PRECEDING には、パーティション内の現在のロー以前にあるすべてのローが含まれており、ROWS または RANGE で指定できます。 UNBOUNDED FOLLOWING には、パーティション内の現在のロー以後にあるすべてのローが含まれており、ROWS または RANGE で指定できます。

FOLLOWING の値では、現在のロー以降にあるローの範囲または数を指定します。ROWS を指定する場合、その値には、ローの数を表す正の数を指定します。RANGE を指定する場合、そのウィンドウには、現在のローに指定の数値を足した数よりも少ないローが含まれます。 RANGE を指定する場合、そのウィンドウ値のデータ型は、**ORDER BY** 句のソートキー式の型に対応している必要があります。 指定できるソートキー式は 1 つのみで、このソートキー式のデータ型は「加算」を許可している必要があります。

PRECEDING の値では、現在のロー以前にあるローの範囲または数を指定します。ROWS を指定する場合、その値には、ローの数を表す正の数を指定します。RANGE を指定する場合、そのウィンドウには、現在のローから指定の数値を引いた数よりも少ないローが含まれます。 RANGE を指定する場合、そのウィンドウ値のデータ型は、**ORDER BY** 句のソートキー式の型に対応している必要があります。 指定できるソートキー式は 1 つのみで、このソートキー式のデータ型は「減算」を許可している必要があります。1 つ目のバインドされたグループで CURRENT ROW または FOLLOWING の値を指定している場合は、2 つ目のバインドされたグループにこの句を指定することはできません。

BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING の組み合わせを使用すると、グループ化したクエリとのジョインを構築しなくても、パーティション全体についての集合を計算できます。パーティション全体についての集合は、レポート集合とも呼ばれます。

CURRENT ROW の概念
物理的な集合グループでは、現在のローに対する相対位置に基づき、隣接するローの数に応じて、ローを含めるか除外するかが判断されます。現在のローは、クエリの中間結果における次のローへの参照にすぎません。現在のローが前に進むと、ウィンドウ内に含まれる新しいローセットに基づいてウィンドウが再評価されます。現在のローをウィンドウ内に含めるという要件はありません。

WINDOW FRAME 句を指定しなかった場合のデフォルトのウィンドウフレームは、WINDOW ORDER 句を指定しているかどうかによって異なります。

- ウィンドウ指定に WINDOW ORDER 句が含まれている場合、ウィンドウの開始ポイントは **UNBOUNDED PRECEDING**、終了ポイントは **CURRENT ROW** になり、累積値の計算に適した可変サイズのウィンドウが定義されます。
- ウィンドウ指定に WINDOW ORDER 句が含まれていない場合、ウィンドウの開始ポイントは **UNBOUNDED PRECEDING**、終了ポイントは **UNBOUNDED FOLLOWING** になり、現在のローに関係なく固定サイズのウィンドウが定義されます。

注意： WINDOW FRAME 句はランク付け関数とは併用できません。

ローベース (ロー指定) または値ベース (範囲指定) のウィンドウフレーム単位を指定してウィンドウを定義することもできます。

```
<WINDOW FRAME UNIT> ::= ROWS | RANGE
```

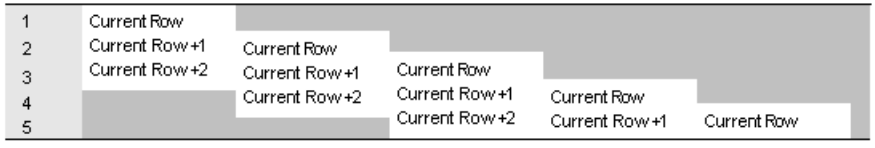
```
<WINDOW FRAME EXTENT> ::= <WINDOW FRAME START> | <WINDOW FRAME
BETWEEN>
```

WINDOW FRAME 句で **BETWEEN** を使用するときは、ウィンドウフレームの開始ポイントと終了ポイントを明示的に指定します。

WINDOW FRAME 句でこの 2 つの値のどちらか一方のみ指定した場合は、他方の値がデフォルトで **CURRENT ROW** になります。

ローベースのウィンドウフレーム - この例では、ロー [1] ～ [5] は 1 つのパーティションを表しています。それぞれのローは、OLAP のウィンドウフレームが前にスライドするにつれて現在のローになります。このウィンドウフレームは Between Current Row And 2 Following として定義されているため、各フレームには、最大で 3 つ、最小で 1 つのローが含まれます。フレームがパーティションの終わりに到達したときは、現在のローだけがフレームに含まれます。 網掛けの部分は、各ステップでフレームから除外されているローを表しています。

図 4 : ローベースのウィンドウフレーム

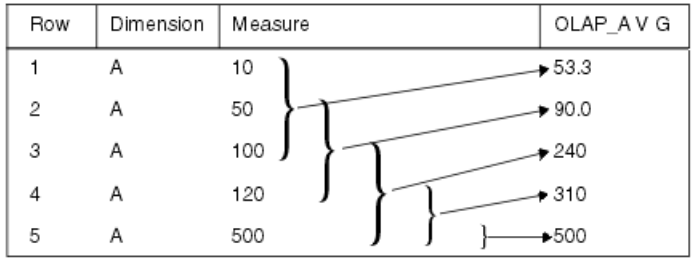


ウィンドウフレームは、次のような規則で機能しています。

- ロー [1] が現在のローであるときは、ロー [4] および [5] が除外される。
- ロー [2] が現在のローであるときは、ロー [5] および [1] が除外される。
- ロー [3] が現在のローであるときは、ロー [1] および [2] が除外される。
- ロー [4] が現在のローであるときは、ロー [1]、[2]、[3] が除外される。

- ロー [5] が現在のローであるときは、ロー [1]、[2]、[3]、[4] が除外される。

次の図では、この規則を特定の値セットに適用し、OLAP の **AVG** 関数を使用して各ローを計算しています。スライド計算により、現在のローの位置に応じて、3 つまたはそれ以下のローを範囲として移動平均を算出しています。



次のクエリは、移動ウィンドウの定義の例を示しています。

```
SELECT dimension, measure,
  AVG(measure) OVER(partition BY dimension
    ORDER BY measure
    ROWS BETWEEN CURRENT ROW and 2 FOLLOWING)
    AS olap_avg
FROM ...
```

平均値は次のようにして計算されています。

- ロー [1] = (10 + 50 + 100)/3
- ロー [2] = (50+ 100 + 120)/3
- ロー [3] = (100 + 120 + 500)/3
- ロー [4] = (120 + 500 + NULL)/3
- ロー [5] = (500 + NULL + NULL)/3

結果セット内の以降のすべてのパーティション (たとえば B、C など) についても、同様の計算が実行されます。

現在のウィンドウにローが含まれていない場合、**COUNT** 以外のケースでは、結果は NULL になります。

ROWS

ウィンドウフレーム単位 ROWS では、現在のローの前後に指定の数のローを含むウィンドウを定義します (現在のローは、ウィンドウの開始ポイントと終了ポイントを決定するための参照ポイントになります)。

それぞれの分析計算は、パーティション内の現在のローに基づいて行われます。ローで表現されるウィンドウを使用して限定的な結果を生成するには、ユニークな順序付けの式を指定する必要があります。

どのウィンドウフレームでも、現在のローが参照ポイントになります。SQL/OLAP の構文には、ローベースのウィンドウフレームを、現在のローの前または

後にある任意の数のロー (または現在のローの前および後ろにある任意の数のロー) として定義するためのメカニズムが用意されています。

ウィンドウフレーム単位の代表的な例を次に示します。

- Rows Between Unbounded Preceding and Current Row ― 各パーティションの先頭を開始ポイントとし、現在のローを終了ポイントとするウィンドウを指定します。累積和など、累積的な結果を計算するためのウィンドウを構築するときによく使用されます。
- Rows Between Unbounded Preceding and Unbounded Following ― 現在のローに関係なく、パーティション全体についての固定ウィンドウを指定します。そのため、ウィンドウ集合関数の値は、パーティションのすべてのローで等しくなります。
- Rows Between 1 Preceding and 1 Following ― 3 つの隣接するロー (現在のローとその前後のロー) を含む固定サイズの移動ウィンドウを指定します。このウィンドウフレーム単位を使用して、たとえば 3 日間または 3 か月間の移動平均を計算できます。
  ウィンドウ値にギャップがあると、ROWS を使用した場合に意味のない結果が生成されることがあるので注意してください。値セットが連続していない場合は、ROWS の代わりに RANGE を使用することを検討してください。RANGE に基づくウィンドウ定義では、重複する値を含んだ隣接ローが自動的に処理され、範囲内にギャップがあるときに他のローが含まれません。

  注意： 移動ウィンドウでは、入力の最初のローの前、および入力の最後のローの後ろには、NULL 値を含むローが存在することが想定されます。つまり、3 つのローから成る移動ウィンドウの場合は、入力の最後のローを現在のローとして計算するときに、直前のローと NULL 値が計算に含まれます。
- Rows Between Current Row and Current Row ― ウィンドウを現在のローのみに制限します。
- Rows Between 1 Preceding and 1 Preceding ― 現在のローの直前のローのみを含む単一ローのウィンドウを指定します。この指定を、現在のローのみに基づく値を計算する別のウィンドウ関数と組み合わせると、隣接するロー同士のデルタ (値の差分) を簡単に計算することができます。

RANGE

範囲ベースのウィンドウフレーム - SQL/OLAP 構文では、別の種類のウィンドウフレームとして、物理的なローのシーケンスではなく、値ベース (または範囲ベース) のローセットに基づいて境界を定義する方法がサポートされています。

値ベースのウィンドウフレームは、ウィンドウパーティション内で、特定の範囲の数値を含んでいるローを定義します。 OLAP 関数の **ORDER BY** 句では、範囲指定を適用する数値カラムを定義します。このカラムの現在のローの値が、範囲指定の基準となります。範囲指定ではロー指定と同じ構文を使用しますが、構文の解釈の仕方は異なります。

ウィンドウフレーム単位 RANGE では、特定の順序付けカラムについて現在のローを基準とする値範囲を指定し、その範囲内の値を持つローを検索して、ウィンドウフレームに含めます。これは論理的なオフセットに基づくウィンドウフレームと呼ばれ、"3 preceding" などの定数を指定することも、評価結果が数値定数となる任意の式を指定することもできます。 RANGE で定義されているウィンドウを使用するときは、**ORDER BY** 句に数値式を 1 つのみ指定できます。

注意： ORDER BY キーは、RANGE ウィンドウフレーム内の数値データである必要があります。

たとえば、次のように指定すると、カラムに現在のローの前後数年に当たる *year* 値を含むローセットをフレームとして定義できます。

```
  ORDER BY year ASC range BETWEEN 1 PRECEDING AND CURRENT ROW
```

1 PRECEDING という部分は、現在のローの *year* 値から 1 を減算することを意味しています。

このような範囲指定は内包的です。 現在のローの *year* 値が 2000 である場合は、ウィンドウパーティション内で、year 値が 2000 および 1999 であるすべてのローがこのフレームに含まれることになります。パーティション内での各ローの物理的な位置は問われません。値ベースのフレームでは、ローを含めたり除外したりする規則が、ローベースのフレームの規則とは大きく異なります (ローベースのフレームの規則は、ローの物理的なシーケンスに完全に依存しています)。

OLAP の **AVG()** 関数の例で考えてみます。次の部分的な結果セットは、値ベースのウィンドウフレームの概念を具体的に表しています。前述のように、このフレームには次のローが含まれます。

- 現在のローと同じ year 値を持つロー
- 現在のローから 1 を減算したものと同じ year 値を持つロー

Row	Dimension	Year	Measure	Olap_avg
1	A	1999	10000	10000
2	A	2001	5000	3000
3	A	2001	1000	3000
4	A	2002	12000	5250
5	A	2002	3000	5250

次のクエリは、範囲ベースのウィンドウフレーム定義の例を示しています。

```
SELECT dimension, year, measure,
  AVG(measure) OVER(PARTITION BY dimension
    ORDER BY year ASC
    range BETWEEN CURRENT ROW and 1 PRECEDING)
    as olap_avg
FROM ...
```

平均値は次のようにして計算されています。

- ロー [1] = 1999 のため、ロー [2] ～ [5] は除外。したがって AVG = 10,000/1
- ロー [2] = 2001 のため、ロー [1]、[4]、[5] は除外。したがって AVG = 6,000/2
- ロー [3] = 2001 のため、ロー [1]、[4]、[5] は除外。したがって AVG = 6,000/2
- ロー [4] = 2002 のため、ロー [1] は除外。したがって AVG = 21,000/4
- ロー [5] = 2002 のため、ロー [1] は除外。したがって AVG = 21,000/4

値ベースのフレームの昇順と降順 - 値ベースのウィンドウフレームを使用する OLAP 関数の **ORDER BY** 句では、範囲指定の対象となる数値カラムを特定するだけではなく、**ORDER BY** 値のソート順序も宣言できます。次の指定により、直前の部分のソート順序 (ASC または DESC) を設定できます。

```
RANGE BETWEEN CURRENT ROW AND n FOLLOWING
```

n FOLLOWING の指定には、次のような意味があります。

- パーティションがデフォルトの昇順 (ASC) でソートされている場合、*n* は正の値として解釈されます。
- パーティションが降順 (DESC) でソートされている場合は、*n* は負の値として解釈されます。

たとえば、`year` カラムに 1999 ～ 2002 の 4 種類の値が含まれているとします。次の表は、これらの値をデフォルトの昇順でソートした場合 (左側) と降順でソートした場合 (右側) を示しています。

ORDER BY year ASC	ORDER BY year DESC
1999	2002
2000	2001
2001	2000
2002	1999

現在のローが 1999 で、フレームが次のように指定されている場合、このフレームには値 1999 のローと値 1998 のロー (このテーブルには存在しません) が含まれます。

```
ORDER BY year DESC range BETWEEN CURRENT ROW and 1 FOLLOWING
```

注意： ORDER BY 値のソート順序は、値ベースのフレームに含まれるローの条件をテストするときに重要な要素です。フレームに含まれるか除外されるかは、数値だけでは決まりません。

無制限ウィンドウの使用 - 次のクエリでは、すべての製品と、すべての製品の総数から成る結果セットが生成されます。

```
SELECT id, description, quantity,
  SUM(quantity) OVER () AS total
FROM products;
```

隣接ロー間のデルタの計算 - 現在のローと前のローをそれぞれ 1 つのウィンドウとして定義し、この 2 つのウィンドウを使用すると、隣接するロー間のデルタ (つまり差分) を直接計算できます。

```
SELECT EmployeeID, Surname, SUM(salary)
OVER(ORDER BY BirthDate rows between current row and current row)
AS curr, SUM(Salary)
OVER(ORDER BY BirthDate rows between 1 preceding and 1 preceding)
AS prev, (curr-prev) as delta
FROM Employees
WHERE State IN ('MA', 'AZ', 'CA', 'CO') AND DepartmentID>10
ORDER BY EmployeeID, Surname;
```

このクエリの結果セットを次に示します。

```
EmployeeID  Surname      curr         prev         delta
---------   ---------    --------     ---------    -----
148         Jordan       51432.000191
209         Bertrand              29800.000
39300.000             -9500.000278
225         Melkisetian     48500.000       42300.000
6200.000299
657         Overbey                   39300.000
41700.750             -2400.750318
902         Crow                      41700.750
45000.000             -3299.250586
949         Coleman                   42300.000
46200.000             -3900.000690
1053        Poitras                   46200.000
29800.000             16400.000703
1090       Martinez              55500.800           51432.000
4068.800949
1154        Savarino                  72300.000
55500.800             16799.2001101
1420        Preston      37803.000             48500.000
-10697.0001142
1507        Clark        45000.000             72300.000
-27300.000
```

ここではウィンドウ関数 **SUM()** を使用していますが、ウィンドウの指定方法により、この合計には現在のローまたは前のローの salary 値のみが含まれています。また、結果セットの最初のローには前のローが存在しないため、最初のローの `prev` 値は NULL になります。したがって、`delta` も NULL になります。

ここまでの例では、**OVER()** 句と一緒に **SUM()** 集合関数を使用しました。

明示的なウィンドウ句とインラインのウィンドウ句

SQL OLAP では、クエリ内でウィンドウを指定する方法が 2 とおり用意されています。

- 明示的なウィンドウ句。**HAVING** 句の後でウィンドウを定義します。 OLAP 関数を呼び出すときには、このようなウィンドウ句で定義したウィンドウを、ウィンドウの名前を指定して参照します。たとえば、次のようにします。
  ```
  SUM ( ...) OVER w2
  ```
- インラインのウィンドウ指定。クエリ式の **SELECT** リスト内でウィンドウを定義します。 これにより、**HAVING** 句の後のウィンドウ句でウィンドウを定義し、それをウィンドウ関数呼び出しから名前で参照する方法に加えて、関数呼び出しと同時にウィンドウを定義する方法が可能になります。

  注意： インラインのウィンドウ指定を使用する場合は、ウィンドウの名前を指定できません。 1 つの **SELECT** リスト内で複数のウィンドウ関数呼び出しが同じウィンドウを使用する場合は、ウィンドウ句で定義した名前付きウィンドウを参照するか、インラインのウィンドウ定義を繰り返します。

ウィンドウ関数の例 ― 次は、ウィンドウ関数の例です。このクエリでは、データを部署別のパーティションに分け、在社年数が最も長い従業員を基点とした従業員の累積給与を計算して、結果セットを返します。この結果セットには、マサチューセッツ在住の従業員だけが含まれます。sum_salary カラムには、従業員の給与の累積和が含まれます。

```
SELECT DepartmentID, Surname, StartDate, Salary, SUM(Salary) OVER
(PARTITION BY DepartmentID ORDER BY startdate
rows between unbounded preceding and current row)
AS sum_salary FROM Employees
WHERE State IN ('CA') AND DepartmentID IN (100, 200)
ORDER BY DepartmentID;
```

次の結果セットは部署別に分割されています。

```
DepartmentID  Surname   start_date  salary     sum_salary
------------  --------  ----------  ---------  ----------
200           Overbey   1987-02-19  39300.000   39300.000
200           Savarino  1989-11-07  72300.000  111600.000
200           Clark     1990-07-21  45000.000  156600.000
```

ランク付け関数

ランク付け関数を使用すると、データセットの値をランク付けされた順序のリストにまとめ、「今年度出荷された製品の中で売上合計が上位 10 位の製品名」または「15 社以上から受注した営業部員の上位 5%」といった要求を満たすクエリを、1 つの SQL 文で作成できます。

SQL/OLAP では、次の 5 つの関数がランク付け関数として分類されています。

```
<RANK FUNCTION TYPE> ::=
  RANK | DENSE_RANK | PERCENT_RANK | ROW_NUMBER | NTILE
```

ランキング関数を使用すると、クエリで指定された順序に基づいて、結果セット内の各ローのランク値を計算することができます。たとえば販売マネージャが、営業成績が最高または最低の営業部員、販売成績が最高または最低の販売地域、または売上が最高または最低の製品を調べたい場合があります。この情報はランキング関数によって入手できます。

RANK

RANK 関数は、**ORDER BY** 句で定義されたカラムについて、ローのパーティション内での現在のローのランクを表す数値を返します。

パーティション内の最初のローが 1 位となり、25 のローを含むパーティションでは、パーティション内の最後のローが 25 位となります。**RANK** は構文変換として指定されており、実際に **RANK** を同じ構文に変換することも、変換を行った場合に返される値と同じ結果を返すこともできます。

次の例に出てくる ws1 は、w1 という名前のウィンドウを定義するウィンドウ指定を表しています。

```
RANK() OVER ws
```

これは次の指定に相当します。

```
( COUNT (*) OVER ( ws RANGE UNBOUNDED PRECEDING )
- COUNT (*) OVER ( ws RANGE CURRENT ROW ) + 1 )
```

この **RANK** 関数の変換では、論理的な集合 (RANGE) を使用しています。この結果、同位のロー (順序付けカラムに同じ値が含まれているロー) が複数ある場合は、それらに同じランクが割り当てられます。パーティション内で異なる値を持つ次のグループには、同位のローのランクよりも 1 以上大きいランクが割り当てられます。たとえば、順序付けカラムに 10、20、20、20、30 という値を含むローがある場合、1 つ目のローのランクは 1 になり、2 つ目のローのランクは 2 になります。3 つ目と 4 つ目のローのランクも 2 になりますが、5 つ目のローのランクは 5 になります。ランクが 3 または 4 のローは存在しません。このアルゴリズムは非連続型ランキング (sparse ranking) とも呼ばれます。

RANK 関数 [統計]

グループ内の項目をランク付けします。

構文

RANK () **OVER** ([**PARTITION BY**] **ORDER BY** *expression* [**ASC** | **DESC**])

パラメータ

パラメータ	説明
expression	ソートを指定する。カラムの参照、集合関数、またはこれらの項目を起動する式など、有効な式を何でも指定できる。

戻り値
INTEGER

備考
RANK はランク付け統計関数です。ロー R のランクは、R 以前にあり R と同等でないローの数で決まります。**OVER** 句で指定されたグループどうしで、2 つ以上のローが同等な場合、または結果セット全体で同等な場合は、ランク付けの順番に 1 つ以上の隔たりが生じます。**RANK** と **DENSE_RANK** の相違点は、順位が同じである場合に、**DENSE_RANK** ではランク順に隔たりが置かれないことです。**RANK** では隔たりが置かれます。

RANK には **OVER (ORDER BY)** 句が必須です。**ORDER BY** 句は、ランク付けを実行するパラメータと、各グループでローをソートする順序を指定します。この **ORDER BY** 句は、**OVER** 句の内部のみで使用するもので、**SELECT** の **ORDER BY** とは異なります。ランク付けクエリ内の集合関数に DISTINCT を指定することはできません。

OVER (ORDER BY) 句内の **PARTITION BY** ウィンドウパーティション句はオプションです。

ASC と DESC のパラメータでは、昇順または降順の順序付けシーケンスを指定します。昇順がデフォルトです。

OVER 句は、関数がクエリの結果セットに対して処理を行うことを示します。結果セットは、**FROM**、**WHERE**、**GROUP BY**、**HAVING** の各句がすべて評価された後で返されるローです。**OVER** 句は、ランク付け統計関数の計算の対象となるローのデータセットを定義します。

RANK は、**SELECT** または **INSERT** 文の SELECT リスト、および **SELECT** 文の **ORDER BY** 句でのみ使用できます。**RANK** は、ビューまたは union に含めることができます。**RANK** 関数は、サブクエリ、**HAVING** 句、**UPDATE** 文や **DELETE** 文の SELECT リストでは使用できません。1 つのクエリで使用可能なランク付け統計関数は、1 つだけです。

標準と互換性

- SQL — ISO/ANSI SQL 文法のベンダ拡張。
- Sybase — Adaptive Server または SQL Anywhere でサポートされていません。

例

RANK 関数は、次の文のように使用します。

```
SELECT Surname, Sex, Salary, RANK() OVER (PARTITION BY Sex
ORDER BY Salary DESC) AS RANK FROM Employees
WHERE State IN ('CA', 'AZ') AND DepartmentID IN (200, 300)
ORDER BY Sex, Salary DESC;
```

このクエリの結果セットを次に示します。

```
Surname            Sex      Salary     RANK
-------            ---      ------     ----
Savarino            F        72300.000  1
Jordan              F        51432.000  2
Clark               F        45000.000  3
Coleman             M        42300.000  1
Overbey             M        39300.000  2
```

DENSE_RANK

DENSE_RANK 関数は、抜けのないランキング値を返します。

同位のローに対しては同じように等しいランク値が割り当てられますが、このローのランクは、個々のローの順位ではなく、順序付けカラムに等しい値を含んでいるローの集まりの順位を表しています。 **RANK** の例と同様に、順序付けカラムに 10、20、20、20、30 という値を含むローがある場合、1 つ目のローのランクは同じく 1 となり、2 つ目のローおよび 3 つ目、4 つ目のローのランクも同じく 2 となります。しかし、最後のローのランクは 5 ではなく 3 になります。

DENSE_RANK も、構文変換を通じて計算されます。

```
DENSE_RANK() OVER ws
```

これは次の指定に相当します。

```
COUNT ( DISTINCT ROW ( expr_1, . . ., expr_n ) )
 OVER ( ws RANGE UNBOUNDED PRECEDING )
```

この例では、*expr_1*から *expr_n*の部分が、ウィンドウ w1 のソート指定リストに含まれている値の式のリストを表しています。

DENSE_RANK 関数 [統計]

グループ内の項目をランク付けします。

構文

```
DENSE_RANK () OVER ( ORDER BY  expression [ ASC | DESC ] )
```

パラメータ

表 3 : パラメータ

パラメータ	説明
expression	ソートを指定する。カラムの参照、集合関数、またはこれらの項目を起動する式など、有効な式を何でも指定できる。

戻り値
INTEGER

備考
DENSE_RANK はランク付け統計関数です。ロー R のランクは、R 以前のローの数のうち、**OVER** 句で指定されたグループ間で同等なロー、または結果セット全体で同等なローを引いた数になります。**DENSE_RANK** と **RANK** の相違点は、順位が同じである場合に、**DENSE_RANK** ではランク順に隔たりが置かれないことです。**RANK** では隔たりが置かれます。

DENSE_RANK には **OVER (ORDER BY)** 句が必須です。**ORDER BY** 句は、ランク付けを実行するパラメータと、各グループでローをソートする順序を指定します。この **ORDER BY** 句は、**OVER** 句の内部のみで使用するもので、**SELECT** の **ORDER BY** とは異なります。ランク付けクエリ内の集合関数で **DISTINCT** を指定することはできません。

OVER 句は、関数がクエリの結果セットに対して処理を行うことを示します。結果セットは、**FROM**、**WHERE**、**GROUP BY**、**HAVING** の各句がすべて評価された後で返されるローです。**OVER** 句は、ランク付け統計関数の計算の対象となるローのデータセットを定義します。

ASC と DESC のパラメータでは、昇順または降順の順序付けシーケンスを指定します。昇順がデフォルトです。

DENSE_RANK は、**SELECT** または **INSERT** 文の SELECT リスト、および **SELECT** 文の **ORDER BY** 句でのみ使用できます。**DENSE_RANK** は、ビューまたは union に含めることができます。**DENSE_RANK** 関数は、サブクエリ、HAVING 句、および **UPDATE** または **DELETE** 文の SELECT リストでは使用できません。1 つのクエリで使用可能なランク付け統計関数は、1 つだけです。

標準と互換性

- SQL — ISO/ANSI SQL 文法のベンダ拡張。
- Sybase — Adaptive Server または SQL Anywhere でサポートされていません。

例
次の文は、**DENSE_RANK** 関数の使い方を示しています。

```
SELECT s_suppkey, DENSE_RANK()
OVER ( ORDER BY ( SUM(s_acctBal) DESC )
AS rank_dense FROM supplier GROUP BY s_suppkey;

s_suppkey         sum_acctBal       rank_dense
supplier#011      200,000            1
supplier#002      200,000            1
supplier#013      123,000            2
supplier#004      110,000            3
supplier#035      110,000            3
supplier#006      50,000             4
supplier#021      10,000             5
```

PERCENT_RANK
PERCENT_RANK 関数は、小数ではなく、パーセンテージでランクを計算して、0 ～ 1 の 10 進値を返します。

PERCENT_RANK が返すのはローの相対的なランクであり、この数値は、該当するウィンドウパーティション内での現在のローの相対位置を表します。たとえば、順序付けカラムにそれぞれ異なる値を持つ 10 個のローがパーティションに含まれている場合、このパーティションの 3 つ目のローに対する **PERCENT_RANK** の値は 0.222... となります。これは、パーティションの 1 つ目のローに続く 2/9 (22.222...%) のローをカバーしているためです。次の例に示すとおり、ローの **PERCENT_RANK** は、「ローの **RANK** - 1」を「パーティション内のローの数 - 1」で除算したものとして定義されています (“ANT” は、`REAL` や `DOUBLE PRECISION` などの概数値の型を表します)。

```
PERCENT_RANK() OVER ws
```

これは次の指定に相当します。

```
CASE
  WHEN COUNT (*) OVER ( ws RANGE BETWEEN UNBOUNDED
  PRECEDING AND UNBOUNDED FOLLOWING ) = 1
  THEN CAST (0 AS ANT)
  ELSE
   ( CAST ( RANK () OVER ( ws ) AS ANT ) -1 /
   ( COUNT (*) OVER ( ws RANGE BETWEEN UNBOUNDED
   PRECEDING AND UNBOUNDED FOLLOWING ) - 1 )
END
```

PERCENT_RANK 関数 [統計]
ORDER BY 句の定義に従い、クエリから返される 1 つのローの、クエリから返されるその他のローに対する (小数による) 位置を返します。

0 ～ 1 の 10 進値を返します。

構文

```
PERCENT_RANK() OVER ( ORDER BY  expression [ ASC | DESC ] )
```

パラメータ

パラメータ	説明
expression	ソートを指定する。カラムの参照、集合関数、またはこれらの項目を起動する式など、有効な式を何でも指定できる。

戻り値

PERCENT_RANK 関数は、0 ～ 1 の DOUBLE 値を返します。

備考

PERCENT_RANK はランク付け統計関数です。ロー R のパーセントランクは、**OVER** 句に指定された各グループ内でのローのランクから 1 を引いた値を、**OVER** 句に指定された各グループのローの合計数から 1 を引いた数で割って算出します。**PERCENT_RANK** は 0 ～ 1 の値を返します。最初のローのパーセントランクはゼロになります。

ローの **PERCENT_RANK** は、以下のように計算されます。

```
(Rx - 1) / (NtotalRow - 1)
```

Rx はグループのローのランク位置で、*NtotalRow* は **OVER** 句に指定されたグループでのローの合計数です。

PERCENT_RANK には **OVER (ORDER BY)** 句が必須です。**ORDER BY** 句は、ランク付けを実行するパラメータと、各グループでローをソートする順序を指定します。この **ORDER BY** 句は、**OVER** 句の内部のみで使用するもので、**SELECT** の **ORDER BY** とは異なります。ランク付けクエリ内の集合関数で **DISTINCT** を指定することはできません。

OVER 句は、関数がクエリの結果セットに対して処理を行うことを示します。結果セットは、**FROM**、**WHERE**、**GROUP BY**、**HAVING** の各句がすべて評価された後で返されるローです。**OVER** 句は、ランク付け統計関数の計算の対象となるローのデータセットを定義します。

ASC と DESC のパラメータでは、昇順または降順の順序付けシーケンスを指定します。昇順がデフォルトです。

PERCENT_RANK は、**SELECT** または **INSERT** 文の SELECT リスト、および **SELECT** 文の **ORDER BY** 句でのみ使用できます。**PERCENT_RANK** は、ビューまたは union に含めることができます。**PERCENT_RANK** 関数は、サブクエリ、**HAVING** 句、

UPDATE 文や **DELETE** 文の SELECT リストでは使用できません。1 つのクエリで使用可能なランク付け統計関数は、1 つだけです。

標準と互換性

- SQL — ISO/ANSI SQL 文法のベンダ拡張。
- Sybase — Adaptive Server または SQL Anywhere でサポートされていません。

例
次の文は、**PERCENT_RANK** 関数の使い方を示しています。

```
SELECT s_suppkey, SUM(s_acctBal) AS sum_acctBal,
PERCENT_RANK() OVER ( ORDER BY SUM(s_acctBal) DESC )
AS percent_rank_all FROM supplier GROUP BY s_suppkey;

s_suppkey        sum_acctBal       percent_rank_all
supplier#011     200000            0
supplier#002     200000            0
supplier#013     123000            0.3333
supplier#004     110000            0.5
supplier#035     110000            0.5
supplier#006     50000             0.8333
supplier#021     10000             1
```

ROW_NUMBER
ROW_NUMBER 関数は、ローごとにユニークなロー番号を返します。

ウィンドウパーティションを定義すると、**ROW_NUMBER** は、各パーティションのローナンバリングを 1 から開始し、ローごとに 1 ずつ増やしていきます。ウィンドウパーティションを指定しない場合、**ROW_NUMBER** は、結果セット全体に対して 1 からテーブルの合計カーディナリティまでの番号を付けます。

ROW_NUMBER 関数の構文は次のとおりです。

```
ROW_NUMBER() OVER ([PARTITION BY window partition] ORDER BY
            window ordering)
```

ROW_NUMBER では引数は必須ではありませんが、カッコは指定してください。

PARTITION BY 句はオプションです。 **OVER** (**ORDER_BY**) 句に、ウィンドウフレーム **ROWS**/**RANGE** 指定を含めることはできません。

ROW_NUMBER 関数 [統計]
ウィンドウパーティション内の各ローにユニークなロー番号を返すランク付け関数です。ウィンドウパーティションごとに新しい番号を割り当てます。

ウィンドウパーティションが存在しない場合は、結果セット内のローに 1 からテーブルのカーディナリティまでの番号が割り当てられます。

構文

ROW_NUMBER() **OVER** ([**PARTITION BY** *window partition*] **ORDER BY** *window ordering*)

パラメータ

パラメータ	説明
ウィンドウパーティション	(オプション) 結果ローセットの分割方法を示す、カンマ区切りの 1 つ以上の値式。
ウィンドウ順序	ローのソートのための式を定義する。ウィンドウパーティションを指定した場合はウィンドウパーティション内でのソート、指定しなかった場合は結果セット内でのソート。

備考

ROW_NUMBER 関数は、**OVER** (**ORDER_BY**) ウィンドウ指定を必要とします。**OVER** (**ORDER_BY**) 句内のウィンドウパーティション句はオプションです。**OVER** (**ORDER_BY**) 句に、ウィンドウフレーム **ROWS**／**RANGE** 指定を含めることはできません。

標準と互換性

- SQL — ISO/ANSI SQL 準拠。SQL/OLAP 機能 T611 です。

例

次の例は、Employees テーブルから給与データを返し、部署 ID で結果セットをパーティションに分割し、入社日でデータを並べ替えます。**ROW_NUMBER** 関数は、各ローにロー番号を割り当てます。ウィンドウパーティションごとに新しいロー番号を割り当てます。

```
SELECT DepartmentID dID, StartDate, Salary,
ROW_NUMBER()OVER(PARTITION BY dID ORDER BY StartDate) FROM  Employees
ORDER BY 1,2;
```

次の結果セットが返されます。

```
dID         StartDate    Salary      Row_number()
=========  ===========  ==========  =============
100         1986-10-14  42,998.000     1
100         1987-07-23  39,875.500     2
100         1988-03-23  37,400.000     3
100         1989-04-20  42,500.000     4
100         1990-01-15  42,100.000     5
200         1985-02-03  38,500.000     1
200         1987-02-19  39,300.000     2
200         1988-11-22  39,800.000     3
200         1989-06-01  34,892.000     4
```

```
200          1990-05-13  33,890.000     5
200          1990-07-11  37,803.000     6
```

ランク付けの例

次は、ランク付け関数の一例です。

ランク付けの例 1 - 次の SQL クエリでは、カリフォルニア州在住の男性従業員と女性従業員を検索し、給与を基準として降順にランク付けしています。

```
SELECT Surname, Sex, Salary, RANK() OVER (
ORDER BY Salary DESC) as RANK FROM Employees
WHERE State IN ('CA') AND DepartmentID =200
ORDER BY Salary DESC;
```

このクエリの結果セットを次に示します。

```
Surname       Sex         Salary    RANK
-------       ---         ------    ----
Savarino       F           72300.000  1
Clark          F           45000.000  2
Overbey        M           39300.000  3
```

ランク付けの例 2 - 前述のクエリ例を基にして、データを性別のパーティションに分けることができます。次の例では、性別のパーティションに分けて、従業員の給与を降順にランク付けしています。

```
SELECT Surname, Sex, Salary, RANK() OVER (PARTITION BY Sex
ORDER BY Salary DESC) AS RANK FROM Employees
WHERE State IN ('CA', 'AZ') AND DepartmentID IN (200, 300)
ORDER BY Sex, Salary DESC;
```

このクエリの結果セットを次に示します。

```
Surname            Sex       Salary      RANK
-------            ---       ---------   ----
Savarino            F         72300.000  1
Jordan              F         51432.000  2
Clark               F         45000.000  3
Coleman             M         42300.000  1
Overbey             M         39300.000  2
```

ランク付けの例 3 - この例では、カリフォルニアおよびテキサスの女性従業員のリストを、給与を基準として降順にランク付けしています。 **PERCENT_RANK** 関数により、累積和が降順で示されます。

```
SELECT Surname, Salary, Sex, CAST(PERCENT_RANK() OVER
(ORDER BY Salary DESC) AS numeric (4, 2)) AS RANK
FROM Employees WHERE State IN ('CA', 'TX') AND Sex ='F'
ORDER BY Salary DESC;
```

このクエリの結果セットを次に示します。

```
Surname          salary    sex            RANK
---------       ---------    ---    ----------
Savarino        72300.000    F           0.00
```

```
Smith          51411.000   F           0.33
Clark          45000.000   F           0.66
Garcia         39800.000   F           1.00
```

ランク付けの例 4 - **PERCENT_RANK** 関数を使用して、データセットにおける上位または下位のパーセンタイルを調べることができます。次のクエリは、給与の額がデータセットの上位 5% に入る男性従業員を返します。

```
SELECT * FROM (SELECT Surname, Salary, Sex,
CAST(PERCENT_RANK() OVER (ORDER BY salary DESC) as
numeric (4, 2)) AS percent
FROM Employees WHERE State IN ('CA') AND sex ='F' ) AS
DT where percent > 0.5
ORDER BY Salary DESC;
```

このクエリの結果セットを次に示します。

```
Surname          salary    sex     percent
---------      ----------    ---   ---------
Clark                          45000.000     F      1.00
```

ランク付けの例 5 - この例では、**ROW_NUMBER** 関数を使用して、すべてのウィンドウパーティション内の各ローについてロー番号を返しています。 このクエリでは、Employees テーブルを部署 ID ごとにパーティションに分け、各パーティションのローを開始日に基づいて順序付けします。

```
SELECT DepartmentID dID, StartDate, Salary ,
ROW_NUMBER()OVER(PARTITION BY dID ORDER BY StartDate)
FROM  Employees ORDER BY 1,2;
```

このクエリの結果セットを次に示します。

```
dID         StartDate     Salary       Row_number()
========    ===========   ==========   =============
  100       1984-08-28    47500.000        1
  100       1985-01-01    62000.500        2
  100       1985-06-17    57490.000        3
  100       1986-06-07    72995.000        4
  100       1986-07-01    48023.690        5
  ...       ...           ...              ...
  200       1985-02-03    38500.000        1
  200       1985-12-06    54800.000        2
  200       1987-02-19    39300.000        3
  200       1987-07-10    49500.000        4
  ...       ...           ...              ...
  500       1994-02-27     24903.000       9
```

ウィンドウ集合関数

ウィンドウ集合関数を使用すると、複数のレベルの集合を 1 つのクエリで計算できます。

たとえば、支出が平均より少ない四半期をすべて列挙することができます。 集合関数 (単純な集合関数 **AVG**、**COUNT**、**MAX**、 **MIN**、**SUM** など) を使用すると、1 つの

文の中でさまざまなレベルで計算した結果を 1 つのローに書き出すことができます。これにより、ジョインや相関サブクエリを使用しなくても、集合値をグループ内のディテールローと比較することができます。

これらの関数を使用して、非集合値と集合値を比較することも可能です。たとえば、営業部員が特定の年にある製品に対して平均以上の注文を出した顧客の一覧を作成したり、販売マネージャが従業員の給与をその部署の平均給与と比較したりすることが考えられます。

クエリが **SELECT** 文の中で **DISTINCT** を指定している場合は、ウィンドウ演算子の後に **DISTINCT** 操作が適用されます。 ウィンドウ演算子は、**GROUP BY** 句が処理された後、**SELECT** リストの項目やクエリの **ORDER BY** 句が評価される前に計算されます。

ウィンドウ集合関数の例 1 - 次のクエリは、平均販売数よりも多く売れた製品の一覧を年別に分けて示す結果セットを返します。

```
SELECT * FROM (SELECT Surname AS E_name, DepartmentID AS
   Dept, CAST(Salary AS numeric(10,2) ) AS Sal,
   CAST(AVG(Sal) OVER(PARTITION BY DepartmentID) AS
   numeric(10, 2)) AS Average, CAST(STDDEV_POP(Sal)
   OVER(PARTITION BY DepartmentID) AS numeric(10,2)) AS
   STD_DEV
FROM Employees
GROUP BY Dept, E_name, Sal) AS derived_table WHERE
   Sal> (Average+STD_DEV )
ORDER BY Dept, Sal, E_name;
```

このクエリの結果セットを次に示します。

```
E_name    Dept  Sal       Average   STD_DEV
--------  ----  --------  -------   --------
Lull      100         87900.00        58736.28        16829.59
Sheffield 100         87900.00        58736.28        16829.59
Scott             100         96300.00        58736.28        16829.59
Sterling     200         64900.00        48390.94        13869.59
Savarino     200         72300.00        48390.94     13869.59
Kelly              200         87500.00        48390.94     13869.59
Shea              300        138948.00        59500.00        30752.39
Blaikie          400         54900.00        43640.67     11194.02
Morris              400         61300.00        43640.67     11194.02
Evans               400         68940.00        43640.67     11194.02
Martinez     500         55500.80        33752.20         9084.49
```

2000 年の平均注文数は 1,787 であり、4 つの製品 (700、601、600、400) が平均を上回っています。2001 年の平均注文数は 1,048 であり、3 つの製品が平均を上回っています。

ウィンドウ集合関数の例 2 - 次のクエリは、給与の額がそれぞれの部署の平均給与よりも 1 標準偏差以上高い従業員を表す結果セットを返します。標準偏差とは、そのデータが平均からどのぐらい離れているかを示す尺度です。

```
SELECT * FROM (SELECT Surname AS E_name, DepartmentID AS
  Dept, CAST(Salary AS numeric(10,2) ) AS Sal,
  CAST(AVG(Sal) OVER(PARTITION BY dept) AS
  numeric(10, 2)) AS Average, CAST(STDDEV_POP(Sal)
  OVER(PARTITION BY dept) AS numeric(10,2)) AS
  STD_DEV
FROM Employees
GROUP BY Dept, E_name, Sal) AS derived_table WHERE
  Sal> (Average+STD_DEV )
ORDER BY Dept, Sal, E_name;
```

次の結果から、どの部署にも、給与の額が平均を大きく上回っている従業員が 1 人以上いることがわかります。

```
E_name        Dept    Sal         Average      STD_DEV
--------      ----    --------    --------     --------
Lull           100    87900.00    58736.28     16829.59
Sheffield      100    87900.00    58736.28     16829.59
Scott          100    96300.00    58736.28     16829.59
Sterling       200    64900.00    48390.94     13869.59
Savarino       200    72300.00    48390.94     13869.59
Kelly          200    87500.00    48390.94     13869.59
Shea           300   138948.00    59500.00     30752.39
Blaikie        400    54900.00    43640.67     11194.02
Morris         400    61300.00    43640.67     11194.02
Evans          400    68940.00    43640.67     11194.02
Martinez       500    55500.80    33752.20      9084.49
```

従業員 Scott の給与は 96,300.00 ドルで、所属部署の平均給与は 58,736.28 ドルです。この部署の標準偏差は 16,829.00 なので、給与の額が 75,565.88 ドル (58736.28 + 16829.60 = 75565.88) 未満ならば、平均の 1 標準偏差以内の範囲に収まります。

統計集合関数

ANSI SQL/OLAP 拡張機能には、数値データの統計的分析を行うための集合関数がこの他にも数多く用意されています。これには、分散、標準偏差、相関、直線回帰を計算するための関数も含まれます。

標準偏差と分散

SQL/OLAP の一般的な関数の中には、次の構文の太字で表示されている部分のように、1 つの引数を取る関数があります。

```
<SIMPLE WINDOW AGGREGATE FUNCTION TYPE> ::=
  <BASIC AGGREGATE FUNCTION TYPE>
  | STDDEV | STDDEV_POP | STDDEV_SAMP
  | VARIANCE | VARIANCE_POP | VARIANCE_SAMP
```

- **STDDEV_POP** - グループまたはパーティションの各ロー (**DISTINCT** が指定されている場合は、重複が削除された後に残る各ロー) に対して評価される「値の式」についての母標準偏差を計算します。これは、母分散の平方根として定義されます。

- **STDDEV_SAMP** - グループまたはパーティションの各ロー (**DISTINCT** が指定されている場合は、重複が削除された後に残る各ロー) に対して評価される「値の式」についての母標準偏差を計算します。これは、標本分散の平方根として定義されます。
- **VAR_POP** - グループまたはパーティションの各ロー (**DISTINCT** が指定されている場合は、重複が削除された後に残る各ロー) に対して評価される「値の式」についての母分散を計算します。これは、「値の式」と「値の式の平均」との差の 2 乗和を、グループまたはパーティション内の残りのロー数で除算した値として定義されます。
- **VAR_SAMP** - グループまたはパーティションの各ロー (**DISTINCT** が指定されている場合は、重複が削除された後に残る各ロー) に対して評価される「値の式」についての標本分散を計算します。これは、「値の式」の差の 2 乗和を、グループまたはパーティション内の残りのロー数より 1 少ない数で除算した値として定義されます。

これらの関数と **STDDEV** および **VARIANCE** 関数は、クエリの **ORDER BY** 句の指定に従ってローのパーティションについての値を計算できる集合関数です。 **MAX** や **MIN** など、他の基本的な集合関数と同様に、これらの関数は入力データ内の NULL 値を無視します。また、分析される式のドメインに関係なく、分散と標準偏差の計算では必ず IEEE の倍精度浮動小数点数が使用されます。分散関数または標準偏差関数への入力が空のデータセットである場合、これらの関数は結果として NULL を返します。 **VAR_SAMP** 関数は 1 つのローに対して計算を行うと NULL を返しますが、**VAR_POP** は値 0 を返します。

相関
相関係数を計算する SQL/OLAP 関数は、次のとおりです。

- **CORR** - 一連の数値ペアの相関係数を返します。

CORR 関数は、ウィンドウ集合関数 (ウィンドウ名または指定に対するウィンドウ関数の種類を指定する関数) としても、OVER 句のない単純な集合関数としても使用できます。

共分散
共分散を計算する SQL/OLAP 関数は、次のとおりです。

- **COVAR_POP** - 一連の数値ペアの母共分散を返します。
- **COVAR_SAMP** - 一連の数値ペアの標本共分散を返します。

共分散関数は、式 1 または式 2 が NULL 値を持つ、すべてのペアを除外します。

共分散関数は、ウィンドウ集合関数 (ウィンドウ名または指定に対するウィンドウ関数の種類を指定する関数) としても、OVER 句のない単純な集合関数としても使用できます。

累積分布
ローのグループにおける 1 つの値の相対位置を計算する SQL/OLAP 関数は、**CUME_DIST** です。

ウィンドウ指定には ORDER_BY 句が含まれていることが必要です。

CUME_DIST 関数では、複合ソートキーは許可されません。

回帰分析
回帰分析関数は、直線回帰の方程式を使用し、独立変数と従属変数との間の関係を計算します。SQL/OLAP の直線回帰関数は、次のとおりです。

- **REGR_AVGX** - 回帰線の独立変数の平均を計算します。
- **REGR_AVGY** - 回帰線の独立変数の平均を計算します。
- **REGR_COUNT** - 回帰線の調整に使用される NULL 値以外のペアの数を表す整数を返します。
- **REGR_INTERCEPT** - 従属変数と独立変数に最適な回帰線の y 切片を計算します。
- **REGR_R2** - 回帰線の決定係数 (適合度の統計情報) を計算します。
- **REGR_SLOPE** - NULL 以外のペアに適合する直線回帰の傾きを計算します。
- **REGR_SXX** - 直線回帰モデルに使用される独立した式の 2 乗和の合計を返します。この関数は、回帰モデルの統計的な有効性を評価するときに使用できます。
- **REGR_SXY** - 従変数および独立変数の積和を返します。この関数は、回帰モデルの統計的な有効性を評価するときに使用できます。
- **REGR_SYY** - 回帰モデルの統計的な有効性を評価できる値を返します。

回帰分析関数は、ウィンドウ集合関数 (ウィンドウ名または指定に対するウィンドウ関数の種類を指定する関数) としても、OVER 句のない単純な集合関数としても使用できます。

OLAP の加重集合関数
OLAP の加重集合関数は、加重移動平均を計算します。

- **EXP_WEIGHTED_AVG** - 指数関数的に加重された移動平均を計算します。加重は、平均を構成する各数量の相対的な重要性を決定します。**EXP_WEIGHTED_AVG** の加重は、指数関数的に減少します。 指数関数的な加重は、最新の値に加重を適用し、加重を適用しつつ、古い値の加重を減少させます。
- **WEIGHTED_AVG** - 時間経過とともに等差階級的に加重が減少した、直線的加重移動平均を計算します。 加重は、最新のデータポイントの最高値から減少し、最も古いデータポイントでゼロになります。

ウィンドウ指定には **ORDER_BY** 句を使用します。

データベース業界の標準外の拡張機能
データベース業界で使用される ANSI 以外の SQL/OLAP 集合関数の拡張機能には、**FIRST_VALUE**、**MEDIAN**、**LAST_VALUE** があります。

- **FIRST_VALUE** - 一連の値の最初の値を返します。
- **MEDIAN** - 式から中央値を返します。
- **LAST_VALUE** - 一連の値の最後の値を返します。

FIRST_VALUE および **LAST_VALUE** 関数には、ウィンドウ指定が必要です。 **MEDIAN** 関数は、ウィンドウ集合関数 (ウィンドウ名または指定に対するウィンドウ関数の種類を指定する関数) としても、OVER 句のない単純な集合関数としても使用できます。

Interrow 関数の使用法

Interrow 関数 **LAG** および **LEAD** を使用すると、一連のデータ内では前後の値にアクセスでき、テーブル内では複数のローにアクセスできます。

また、Interrow 関数は、セルフジョインなしで同時にパーティション分けします。LAG では、テーブル内またはパーティション内の **CURRENT ROW** から特定の物理的オフセット分だけ前にあるローにアクセスできます。LEAD では、テーブル内またはパーティション内の **CURRENT ROW** から特定の物理的オフセット分だけ後ろにあるローにアクセスできます。

LAG と **LEAD** の構文は同じです。 どちらの関数にも **OVER** (**ORDER_BY**) ウィンドウ指定が必要です。例を示します。

```
LAG (value_expr) [, offset [, default]]) OVER ([PARTITION BY
            window partition] ORDER BY
            window ordering)
```

および

```
LEAD (value_expr) [, offset [, default]]) OVER ([PARTITION BY
            window partition] ORDER BY
            window ordering)
```

OVER (**ORDER_BY**) 句内の **PARTITION BY** 句はオプションです。 **OVER** (**ORDER_BY**) 句に、ウィンドウフレーム **ROWS**/**RANGE** 指定を含めることはできません。

value_expr は、テーブルから返すオフセットデータを定義するテーブルカラムまたは式です。 *value_expr* では他の関数を定義できます。ただし、分析関数を除きます。

両方の関数について、物理的なオフセットを入力してターゲットローを指定します。*offset* 値は、現在のローより上または下のロー数です。負でない数値式を入力してください (負の値を入力するとエラーが生成されます)。0 を入力すると、SAP Sybase IQ から現在のローが返されます。

オプションの *default* 値は、*offset* 値がテーブルのスコープを超える場合に返される値を定義します。*default* のデフォルト値は **NULL** です。*default* のデータ型は、*value_expr* 値のデータ型に暗黙的に変換可能である必要があります。変換可能でない場合、SAP Sybase IQ は変換エラーを生成します。

LAG の例 1 - Interrow 関数は、証券取引などのデータストリームに対して計算を実行する金融サービスアプリケーションで使用すると有用です。この例では、**LAG** 関数を使用して、特定の株式取引価格の変化率を計算しています。次の `stock_trades` という架空のテーブルの取引データについて考えてみます。

```
traded at             symbol   price
-------------------   ------   ------
2009-07-13 06:07:12   SQL      15.84
2009-07-13 06:07:13   TST       5.75
2009-07-13 06:07:14   TST       5.80
2009-07-13 06:07:15   SQL      15.86
2009-07-13 06:07:16   TST       5.90
2009-07-13 06:07:17   SQL      15.86
```

注意： 架空の `stock_trades` テーブルは、`iqdemo` データベースには含まれていません。

このクエリでは、銘柄記号ごとに取引をパーティションに分け、取引時間に基づいて順序付けします。また、**LAG** 関数を使用して、現在の取引価格と前の取引価格を比較し、その増加率または減少率を計算します。

```
select  stock_symbol as 'Stock',
    traded_at    as 'Date/Time of Trade',
    trade_price  as 'Price/Share',
    cast ( ( ( (trade_price
        - (lag(trade_price, 1)
        over (partition by stock_symbol
            order by traded_at)))
        / trade_price)
    * 100.0) as numeric(5, 2) )
        as '% Price Change vs Previous Price'
from stock_trades
order by 1, 2
```

このクエリは次の結果を返します。

```
Stock   Date/Time of Trade   Price/  % Price Change_vs
symbol                       Share   Previous Price
------  -------------------  -----   -----------------
SQL     2009-07-13 06:07:12  15.84   NULL
SQL     2009-07-13 06:07:15  15.86   0.13
SQL     2009-07-13 06:07:17  15.86   0.00
TST     2009-07-13 06:07:13   5.75   NULL
TST     2009-07-13 06:07:14   5.80   0.87
TST     2009-07-13 06:07:16   5.90   1.72
```

1 番目と 4 番目の出力ローの NULL は、**LAG** 関数が、2 つの各パーティションの最初のローについてはスコープ外であることを示します。比較対象となる前のローがないため、*default* 変数で指定されているように、SAP Sybase IQ は NULL を返します。

分散統計関数

SQL/OLAP には、順序付きセットを取り扱う関数がいくつか定義されています。

PERCENTILE_CONT および **PERCENTILE_DISC** という 2 つの逆分散統計関数があります。 これらの分析関数は、パーセンタイル値を引数として受け取り、**WITHIN GROUP** 句で指定されたデータのグループまたはデータセット全体に対して処理を行います。

これらの関数は、グループごとに 1 つの値を返します。 **PERCENTILE_DISC** (不連続) では、結果のデータ型は **WITHIN GROUP** 句に指定した **ORDER BY** の項目のデータ型と同じになります。 **PERCENTILE_CONT** (連続) では、結果のデータ型は、numeric (**WITHIN GROUP** 句の **ORDER BY** 項目が numeric の場合) または double (**ORDER BY** 項目が整数または浮動小数点の場合) となります。

逆分散統計関数では、**WITHIN GROUP** (**ORDER BY**) 句を指定する必要があります。例を示します。

```
PERCENTILE_CONT ( expression1 )
WITHIN GROUP ( ORDER BY
               expression2 [ ASC | DESC ] )
```

expression1 の値には、numeric データ型の定数を、0 以上 1 以下の範囲で指定します。引数が NULL であれば、"wrong argument for percentile" エラーが返されます。引数の値が 0 よりも小さいか、1 よりも大きい場合は、"data value out of range" エラーが返されます。

必須の **ORDER BY** 句には、パーセンタイル関数の実行対象となる式と、各グループ内でのローのソート順を指定します。この **ORDER BY** 句は、**WITHIN GROUP** 句の内部のみで使用するもので、**SELECT** 文の **ORDER BY** とは異なります。

WITHIN GROUP 句は、クエリ結果を順序付けられたデータセットに分配します。関数は、このデータセットに基づいて結果を計算します。

値 *expression2* には、カラム参照を含む 1 つの式でソートを指定します。このソート式に、複数の式やランク付け分析関数、set 関数、またはサブクエリを指定することはできません。

ASC と DESC のパラメータでは、昇順または降順の順序付けシーケンスを指定します。昇順がデフォルトです。

逆分散統計関数は、サブクエリ、**HAVING** 句、ビュー、union で使用できます。逆分散統計関数は、統計を行わない単純な集合関数が使用されるところであれば、どこでも使用できます。逆分散統計関数は、データセット内の NULL 値を無視します。

PERCENTILE_CONT の例 - この例では、**PERCENTILE_CONT** 関数を使用して、各地域の自動車販売の 10 番目のパーセンタイル値を求めます。次のようなデータセットを使用します。

```
sales       region        dealer_name
-----       ---------     -----------
900         Northeast     Boston
800         Northeast     Worcester
800         Northeast     Providence
700         Northeast     Lowell
540         Northeast     Natick
500         Northeast     New Haven
450         Northeast     Hartford
800         Northwest     SF
600         Northwest     Seattle
500         Northwest     Portland
400         Northwest     Dublin
500         South         Houston
400         South         Austin
300         South         Dallas
200         South         Dover
```

次のクエリ例では、**SELECT** 文に **PERCENTILE_CONT** 関数を使用しています。

```
SELECT region, PERCENTILE_CONT(0.1)
WITHIN GROUP ( ORDER BY ProductID DESC )
FROM ViewSalesOrdersSales GROUP BY region;
```

この **SELECT** 文の結果には、各地域の自動車販売の 10 番目のパーセンタイル値が一覧表示されます。

```
region           percentile_cont
---------        ---------------
Canada           601.0
Central          700.0
Eastern          700.0

South            700.0
Western             700.0
```

PERCENTILE_DISC の例 - この例では、**PERCENTILE_DISC** 関数を使用して、1 つの地域における自動車販売の 10 番目のパーセンタイル値を求めます。次のようなデータセットを使用します。

```
sales       region        dealer_name
-----       ---------     -----------
900         Northeast     Boston
800         Northeast     Worcester
800         Northeast     Providence
```

```
700          Northeast     Lowell
540          Northeast     Natick
500          Northeast     New Haven
450          Northeast     Hartford
800          Northwest     SF
600          Northwest     Seattle
500          Northwest     Portland
400          Northwest     Dublin
500          South         Houston
400          South         Austin
300          South         Dallas
200          South         Dover
```

次のクエリ例では、**SELECT** 文に **PERCENTILE_DISC** 関数を使用しています。

```
SELECT region, PERCENTILE_DISC(0.1) WITHIN GROUP
  (ORDER BY sales DESC )
FROM carSales GROUP BY region;
```

この **SELECT** 文の結果には、各地域の自動車販売の 10 番目のパーセンタイル値が一覧表示されます。

```
region           percentile_cont
 ---------        ---------------
Northeast        900
Northwest        800
South            500
```

PERCENTILE_CONT 関数 [統計]

指定されたパーセンタイルから、対応する値を返します。連続分布データモデルを前提としています。

注意： パーセンタイルを計算するだけであれば、**NTILE** 関数に値 100 を指定して使用してください。

構文

```
PERCENTILE_CONT ( expression1 )
WITHIN GROUP  ( ORDER BY  expression2 [ ASC | DESC ] )
```

パラメータ

パラメータ	説明
expression1	数値データ型の定数を、0 以上 1 以下で指定する。引数が NULL であれば、"wrong argument for percentile" エラーが返される。引数の値が 0 よりも小さいか、1 よりも大きい場合は、"data value out of range" エラーが返される。
expression2	ソートを指定する。カラム参照を含む単一の式で指定する。このソート式に、複数の式やランク付け統計関数、set 関数、またはサブクエリを指定することはできない。

備考
逆分散統計関数は、K-理論パーセンタイル値を返します。これは、データセットのスレッショルド許容値を決定する際に使用します。**PERCENTILE_CONT** 関数は、パーセンタイル値を引数として受け取り、**WITHIN GROUP** 句で指定されたデータグループか、データセット全体に対して処理を実行します。グループごとに 1 つの値を返し、 クエリの **GROUP BY** に指定したカラムが存在しなければ、結果は単一のローになります。結果のデータ型は、**WITHIN GROUP** 句に指定した **ORDER BY** の項目のデータ型と同じになります。**PERCENTILE_CONT** の **ORDER BY** 式のデータ型は、数値型である必要があります。

PERCENTILE_CONT には **WITHIN GROUP (ORDER BY)** 句を指定する必要があります。

必須の **ORDER BY** 句には、パーセンタイル関数の実行対象となる式と、各グループ内でのローのソート順を指定します。**PERCENTILE_CONT** 関数の場合、この式のデータ型は数値型であることが必要です。この **ORDER BY** 句は、**WITHIN GROUP** 句の内部のみで使用するもので、**SELECT** の **ORDER BY** とは異なります。

WITHIN GROUP 句は、クエリ結果を順序付けられたデータセットに分類します。関数はこのデータセットに基づいて結果を計算します。**WITHIN GROUP** 句には、単一のソート項目を含めてください。**WITHIN GROUP** 句に指定されたソート項目が 1 つより多い (または少ない) 場合は、エラーが報告されます。

ASC と DESC のパラメータでは、昇順または降順の順序付けシーケンスを指定します。昇順がデフォルトです。

PERCENTILE_CONT 関数は、サブクエリ、**HAVING** 句、ビュー、union で使用できます。**PERCENTILE_CONT** は、統計を行わない単純な集合関数が使用されるところであれば、どこでも使用できます。**PERCENTILE_CONT** 関数では、データセット内の NULL 値は無視されます。

標準と互換性
- SQL — ISO/ANSI SQL 文法のベンダ拡張。
- Sybase — Adaptive Server または SQL Anywhere でサポートされていません。

例
次の例では、**PERCENTILE_CONT** 関数を使用して、各地域の自動車販売の 10 番目のパーセンタイル値を求めます。

この例では、次のデータセットが使用されています。

```
sales       region        dealer_name
900         Northeast     Boston
800         Northeast     Worcester
800         Northeast     Providence
700         Northeast     Lowell
```

```
540          Northeast      Natick
500          Northeast      New Haven
450          Northeast      Hartford
800          Northwest      SF
600          Northwest      Seattle
500          Northwest      Portland
400          Northwest      Dublin
500          South          Houston
400          South          Austin
300          South          Dallas
200          South          Dover
```

次の **SELECT** 文には、**PERCENTILE_CONT** 関数が含まれています。

```
SELECT region, PERCENTILE_CONT(0.1)
WITHIN GROUP ( ORDER BY sales DESC )
FROM carSales GROUP BY region;
```

この **SELECT** 文の結果には、各地域の自動車販売の 10 番目のパーセンタイル値が一覧表示されます。

```
region           percentile_cont
Northeast        840
Northwest        740
South            470
```

PERCENTILE_DISC 関数 [統計]

指定されたパーセンタイルから、対応する値を返します。離散分布データモデルを前提としています。

注意： パーセンタイルを計算するだけであれば、**NTILE** 関数に値 100 を指定して使用してください。

構文

```
PERCENTILE_DISC ( expression1 )
WITHIN GROUP ( ORDER BY expression2 [ ASC | DESC ] )
```

パラメータ

パラメータ	説明
expression1	数値データ型の定数を、0 以上 1 以下で指定する。引数が NULL であれば、"wrong argument for percentile" エラーが返される。引数の値が 0 よりも小さいか、1 よりも大きい場合は、"data value out of range" エラーが返される。
expression2	ソートを指定する。カラム参照を含む単一の式で指定する。このソート式に、複数の式やランク付け統計関数、set 関数、またはサブクエリを指定することはできない。

備考
逆分散統計関数は、K-理論パーセンタイル値を返します。これは、データセットのスレッショルド許容値を決定する際に使用します。**PERCENTILE_DISC** 関数は、パーセンタイル値を引数として受け取り、**WITHIN GROUP** 句で指定されたデータグループか、データセット全体に対して処理を実行します。グループごとに 1 つの値を返し、クエリの **GROUP BY** に指定したカラムが存在しなければ、結果は単一のローになります。結果のデータ型は、**WITHIN GROUP** 句に指定した **ORDER BY** の項目のデータ型と同じになります。**PERCENTILE_DISC** は、SAP Sybase IQ でソート可能なすべてのデータ型をサポートします。

PERCENTILE_DISC には **WITHIN GROUP (ORDER BY)** 句を指定する必要があります。

必須の **ORDER BY** 句には、パーセンタイル関数の実行対象となる式と、各グループ内でのローのソート順を指定します。この **ORDER BY** 句は、**WITHIN GROUP** 句の内部のみで使用するもので、**SELECT** の **ORDER BY** とは異なります。

WITHIN GROUP 句は、クエリ結果を順序付けられたデータセットに分類します。関数はこのデータセットに基づいて結果を計算します。**WITHIN GROUP** 句には、単一のソート項目を含めてください。**WITHIN GROUP** 句に指定されたソート項目が 1 つより多い (または少ない) 場合は、エラーが報告されます。

ASC と DESC のパラメータでは、昇順または降順の順序付けシーケンスを指定します。昇順がデフォルトです。

PERCENTILE_DISC 関数は、サブクエリ、**HAVING** 句、ビュー、union で使用できます。**PERCENTILE_DISC** は、分析を行わない単純な集合関数が使用されるところであれば、どこでも使用できます。**PERCENTILE_DISC** 関数では、データセット内の NULL 値は無視されます。

標準と互換性

- SQL — ISO/ANSI SQL 文法のベンダ拡張。
- Sybase — Adaptive Server または SQL Anywhere でサポートされていません。

例
次の例では、**PERCENTILE_DISC** 関数を使用して、各地域の自動車販売の 10 番目のパーセンタイル値を求めます。

この例では、次のデータセットが使用されています。

```
sales       region        dealer_name
900         Northeast     Boston
800         Northeast     Worcester
800         Northeast     Providence
700         Northeast     Lowell
540         Northeast     Natick
```

```
500          Northeast      New Haven
450          Northeast      Hartford
800          Northwest      SF
600          Northwest      Seattle
500          Northwest      Portland
400          Northwest      Dublin
500          South          Houston
400          South          Austin
300          South          Dallas
200          South          Dover
```

次の **SELECT** 文には、**PERCENTILE_DISC** 関数が含まれています。

```
SELECT region, PERCENTILE_DISC(0.1)
WITHIN GROUP ( ORDER BY sales DESC )
FROM carSales GROUP BY region;
```

この **SELECT** 文の結果には、各地域の自動車販売の 10 番目のパーセンタイル値が一覧表示されます。

```
region           percentile_cont
Northeast        900
Northwest        800
South            500
```

数値関数

SAP Sybase IQ でサポートされる OLAP 数値関数には、**CEILING** (エイリアスは **CEIL**)、**EXP** (エイリアスは **EXPONENTIAL**)、**FLOOR**、**LN** (エイリアスは **LOG**)、**SQRT**、および **WIDTH_BUCKET** があります。

```
<numeric value function> :: =
 <natural logarithm>
| <exponential function>
| <power function>
| <square root>
| <floor function>
| <ceiling function>
| <width bucket function>
```

表 4 : 数値関数の構文

数値関数	構文
自然対数	**LN** (*numeric-expression*)
指数関数	**EXP** (*numeric-expression*)
累乗関数	**POWER** (*numeric-expression1*, *numeric-expression2*)
平方根	**SQRT** (*numeric-expression*)
床関数	**FLOOR** (*numeric-expression*)
天井関数	**CEILING** (*numeric-expression*)

数値関数	構文
等幅ヒストグラム作成関数	**WIDTH_BUCKET** (*expression*, *min_value*, *max_value*, *num_buckets*)

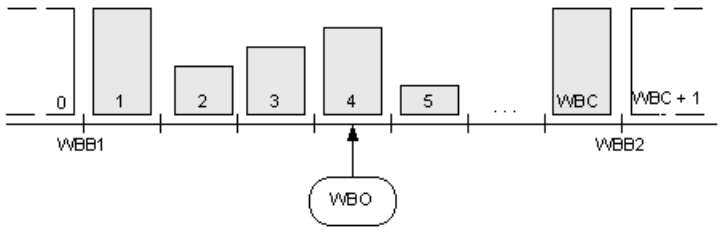
それぞれの数値関数の機能は次のとおりです。

- **LN** - 引数値の自然対数を返します。引数値がゼロまたは負の場合は、エラー状態が発生します。 LN は **LOG** の同意語です。
- **EXP** - *e* (自然対数の底) の値を、引数値で指定された指数まで累乗した結果を返します。
- **POWER** - 1 つ目の引数値を、2 つ目の引数値で指定された指数まで累乗した結果を返します。両方の引数の値が 0 の場合は、1 が返されます。1 つ目の引数が 0 で、2 つ目の引数が正の値である場合は、0 が返されます。1 つ目の引数が 0 で、2 つ目の引数が負の値である場合は、例外が発生します。1 つ目の引数が負の値で、2 つ目の引数が整数でない場合は、例外が発生します。
- **SQRT** - 引数値の平方根を返します。これは、"**POWER** (*expression*, *0.5*)" の構文変換で定義されます。
- **FLOOR** - 引数の値以下で、正の無限大に最も近い整数値を返します。
- **CEILING** - 引数の値以上で、負の無限大に最も近い整数値を返します。 CEIL は CEILING の同意語です。

WIDTH_BUCKET 関数

WIDTH_BUCKET 関数は、他の数値関数よりも少し複雑です。この関数は 4 つの引数を取ります。具体的には、「目的の値」、2 つの範囲境界、そしてこの範囲を何個の等しいサイズ (または可能なかぎり等しいサイズ) の「バケット」に分割するかを指定します。 WIDTH_BUCKET 関数は、範囲の上限から下限までの差のパーセンテージに基づき、目的の値が何番目のバケットに含まれるかを示す数値を返します。最初のバケットが、バケット番号 1 となります。

目的の値が範囲境界の外にある場合のエラーを避けるために、範囲の下限よりも小さい目的の値は、先頭の補助バケット (バケット 0) に配置されます。同様に、範囲の上限よりも大きい目的の値は、末尾の補助バケット (バケット N+1) に配置されます。



たとえば、**WIDTH_BUCKET** (*14, 5, 30, 5*) は 2 を返します。処理の内容は次のとおりです。

- (30-5)/5 = 5 であるため、指定の範囲を 5 つのバケットに分割すると、各バケットの幅は 5 になります。
- 1 つ目のバケットは 0.00 ～ 19.999 …% の値、2 つ目のバケットは 20.00 ～ 39.999 …% の値を表し、以降同様に続き、5 つ目のバケットは 80.00 ～ 100.00% の値を表します。
- 目的の値を含むバケットは、(5*(14-5)/(30-5)) + 1 という計算によって算出されます。これは、バケットの総数と、指定範囲に対する「下限から目的の値までのオフセット」の比率を乗算し、それに 1 を加算するという計算です。実際の数式は (5*9/25) + 1 となり、これを計算すると 2.8 になります。 これはバケット番号 2 (2.0 ～ 2.999...) の範囲に含まれる値であるため、バケット番号 2 が返されます。

WIDTH_BUCKET 例

次の例では、サンプルテーブル内のマサチューセッツ州の顧客の `credit_limit` カラムに 10 のバケットヒストグラムを作成し、各顧客のバケット数 (“Credit Group”) を返します。最大値を超える限度額が設定されている顧客は、オーバフローバケット 11 に割り当てられます。

注意： これは説明用の例であり、iqdemo データベースから生成したものではありません。

```
SELECT customer_id, cust_last_name, credit_limit,
  WIDTH_BUCKET(credit_limit, 100, 5000, 10) "Credit
  Group"
  FROM customers WHERE territory = 'MA'
  ORDER BY "Credit Group";
```

```
CUSTOMER_ID  CUST_LAST_NAME  CREDIT_LIMIT  Credit Group
-----------  --------------  ------------  ------------
825          Dreyfuss                 500             1
826          Barkin                   500             1
853          Palin                    400             1
827          Siegel                   500             1
843          Oates                    700             2
844          Julius                   700             2
835          Eastwood                1200             3
840          Elliott                 1400             3
842          Stern                   1400             3
841          Boyer                   1400             3
837          Stanton                 1200             3
836          Berenger                1200             3
848          Olmos                   1800             4
847           Streep                     5000                 11
```

範囲が逆の場合、バケットはオープンクローズ間隔になります。たとえば、**WIDTH_BUCKET** (*credit_limit*, *5000*, *0*, *5*) のようになります。 この例では、バケット

番号 1 は (4000, 5000)、バケット番号 2 は (3000, 4000)、およびバケット番号 5 は (0, 1000) です。オーバフローバケットには 0 (5000,+infinity) の番号が付き、アンダフローバケットには 6 (-infinity, 0) の番号が付きます。

BIT_LENGTH 関数 [文字列]

カラムパラメータのビット長を、符号なし 64 ビット値で返します。

構文

```
BIT_LENGTH ( column-name )
```

パラメータ

パラメータ	説明
column-name	カラムの名前。

戻り値

INT

備考

引数が NULL の場合は、NULL 値を返します。

BIT_LENGTH 関数は、すべての SAP Sybase IQ データ型をサポートしています。

非構造化データ分析機能の使用ライセンスを取得している場合は、この関数でラージオブジェクトデータを使用できます。

非構造化データ分析 の「関数のサポート」を参照してください。

標準と互換性

- SQL — ISO/ANSI SQL 文法のベンダ拡張。
- Sybase — SQL Anywhere または Adaptive Server でサポートされていません。

CEIL 関数 [数値]

指定された式以上の最小の整数を返します。

CEIL は **CEILING** と同意語です。

構文

```
CEIL ( numeric-expression )
```

パラメータ

パラメータ	説明
expression	カラム、変数、またはデータ型が真数値、概数値、通貨、またはこれらの型の 1 つに暗黙的に変換できる式です。他のデータ型を指定すると、**CEIL** ではエラーが返ります。戻り値のデータ型は、指定した値のデータ型と同じです。

備考
指定された式について、**CEIL** 関数は 1 つの引数を取ります。たとえば、**CEIL (-123.45)** は -123 を返し、**CEIL (123.45)** は 124 を返します。

標準と互換性

- SQL — ISO/ANSI SQL 準拠。
- Sybase － Adaptive Server Enterprise 互換。

CEILING 関数 [数値]
数値の上限値 (その値以上の最も小さい整数) を返します。

CEIL は **CEILING** と同意語です。

構文
CEILING (*numeric-expression*)

パラメータ

パラメータ	説明
numeric-expression	切り上げ値を計算する対象の数値。

戻り値
DOUBLE

標準と互換性

- SQL － ISO/ANSI SQL 文法のベンダ拡張。
- Sybase — Adaptive Server 互換。

例
次の文は、値 60.00000 を返します。

```
SELECT CEILING( 59.84567 ) FROM iq_dummy
```

次の文は、値 123 を返します。

```
SELECT CEILING( 123 ) FROM iq_dummy
```

次の文は、値 124.00 を返します。

```
SELECT CEILING( 123.45 ) FROM iq_dummy
```

次の文は、値 -123.00 を返します。

```
SELECT CEILING( -123.45 ) FROM iq_dummy
```

EXP 関数 [数値]

指数関数 (e の数値乗) を返します。

構文

```
EXP ( numeric-expression )
```

パラメータ

表 5 : パラメータ

パラメータ	説明
numeric-expression	指数。

戻り値

DOUBLE

標準と互換性

- SQL — ISO/ANSI SQL 文法のベンダ拡張。
- Sybase — Adaptive Server Enterprise 互換。

例

次の文は、値 3269017.3724721107 を返します。

```
SELECT EXP( 15 ) FROM iq_dummy
```

FLOOR 関数 [数値]

数値の下限値 (その値以下の最も大きい整数) を返します。

構文

```
FLOOR ( numeric-expression )
```

パラメータ

表 6 : パラメータ

パラメータ	説明
numeric-expression	数値。通常は float 型。

戻り値
DOUBLE

標準と互換性

- SQL － ISO/ANSI SQL 文法のベンダ拡張。
- Sybase － Adaptive Server Enterprise 互換。

例
次の文は、値 123.00 を返します。

```
SELECT FLOOR ( 123 ) FROM iq_dummy
```

次の文は、値 123 を返します。

```
SELECT FLOOR ( 123.45 ) FROM iq_dummy
```

次の文は、値 -124.00 を返します。

```
SELECT FLOOR ( -123.45 ) FROM iq_dummy
```

LN 関数 [数値]
指定された式の自然対数を返します。

構文

```
LN ( numeric-expression )
```

パラメータ

パラメータ	説明
expression	カラム、変数、またはデータ型が真数値、概数値、通貨、またはこれらの型の 1 つに暗黙的に変換できる式。他のデータ型を指定すると、**LN** 関数ではエラーが返る。戻り値は、DOUBLE データ型。

備考
LN は 1 つの引数を取ります。たとえば **LN** (*20*) は、2.995732 を返します。

LN 関数は **LOG** 関数のエイリアスです。

標準と互換性

- SQL — ISO/ANSI SQL 文法のベンダ拡張。
- Sybase — Adaptive Server Enterprise ではサポートされていません。代わりに、LOG 関数を使用してください。

POWER 関数 [数値]
ある数値を、別の数値で累乗します。

構文
```
POWER ( numeric-expression1, numeric-expression2 )
```

パラメータ

パラメータ	説明
numeric-expression1	底。
numeric-expression2	指数。

戻り値
DOUBLE

備考
numeric-expression1 を *numeric-expresson2* で累乗します。

標準と互換性

- SQL — ISO/ANSI SQL 文法のベンダ拡張。
- Sybase — Adaptive Server Enterprise 互換。

例
次の文は、値 64 を返します。
```
SELECT Power( 2, 6 ) FROM iq_dummy
```

SQRT 関数 [数値]
数値の平方根を返します。

構文
```
SQRT ( numeric-expression )
```

パラメータ

パラメータ	説明
numeric-expression	平方根が計算される数値。

戻り値
DOUBLE

標準と互換性

- SQL — ISO/ANSI SQL 文法のベンダ拡張。
- Sybase — Adaptive Server Enterprise 互換。

例
次の文は、値 3 を返します。

```
SELECT SQRT( 9 ) FROM iq_dummy
```

WIDTH_BUCKET 関数 [数値]

与えられた式に対して、**WIDTH_BUCKET** 関数は、この式の評価後の結果に割り当てられるバケット番号を返します。

構文

```
WIDTH_BUCKET ( expression, min_value, max_value, num_buckets )
```

パラメータ

パラメータ	説明
expression	ヒストグラムが作成されている式。この式は、数値または日時の値、または暗黙で数値または日時の値に変換できる値に評価される必要がある。*expr* が NULL に評価されると、式は NULL を返す。
min_value	*expr* に使用できる範囲の各ポイントに解決される式。数値または日時値にも評価される必要があり、NULL には評価できない。
max_value	*expr* に使用できる範囲の各ポイントに解決される式。数値または日時値にも評価される必要があり、NULL には評価できない。
num_buckets	バケット数を示す定数に解決される式。この式は正の整数に評価される必要がある。

備考
WIDTH_BUCKET 関数を使用して等幅ヒストグラムを生成できます。等幅ヒストグラムでは、データセットを間隔サイズ (最も高い値から最も低い値まで) の同じバ

ケットに分割します。保持されるロー数はバケットごとに異なります。関連する関数の **NTILE** は、等高バケットを作成します。

等幅ヒストグラムは数値、日付、日時データ型でのみ生成されるため、最初の 3 つのパラメータはすべて数値式またはすべて日付式にする必要があります。他の型の式は使用できません。最初のパラメータが NULL の場合、結果は NULL です。2 番目および 3 番目のパラメータが NULL の場合、エラーメッセージが返されます。これは、NULL 値は日付または数値次元の範囲のどの終了ポイント (またはあらゆるポイント) も示すことができないためです。最後のパラメータ (バケットの数) は、正の整数値に評価される数値式にする必要があります。0、NULL、または負の値にすると、エラーが発生します。

バケットには 0 から (n+1) まで番号が付けられます。バケット 0 は、最小値未満の値のカウントを保持します。バケット (n+1) は、指定された最大値以上の値のカウントを保持します。

標準と互換性

- SQL － ISO/ANSI SQL 文法のベンダ拡張。
- Sybase － Adaptive Server Enterprise ではサポートされていません。

例

次の例では、サンプルテーブル内のマサチューセッツ州の顧客の `credit_limit` カラムに 10 のバケットヒストグラムを作成し、各顧客のバケット数 (“Credit Group”) を返します。最大値を超える限度額が設定されている顧客は、オーバフローバケット 11 に割り当てられます。

```
select EmployeeID, Surname, Salary, WIDTH_BUCKET(Salary, 29000,
60000, 4) "Wages" from Employees where State = 'FL' order by "Wages"
```

```
EMPLOYEEID   SURNAME              SALARY         Wages
----------   -------              ------         -----
888          Charlton          28300.000             0
1390         Litton            58930.000             4
207          Francis           53870.000             4
266          Gowda             59840.000             4
445          Lull              87900.000             5
1021         Sterling          64900.000             5
902          Kelly             87500.000             5
1576         Evans             68940.000             5
```

範囲が逆の場合、バケットはオープンクローズ間隔になります。たとえば、**WIDTH_BUCKET** (*credit_limit*, *5000*, *0*, *5*) では、バケット番号 1 は (4000, 5000)、バケット番号 2 は (3000, 4000)、バケット番号 5 は (0, 1000) です。オーバフローバケットには 0 (5000,+infinity) の番号が付き、アンダフローバケットには 6 (-infinity, 0) の番号が付きます。

OLAP の規則と制限

以下では、OLAP 関数を制御する規則と制限の概要について説明します。

OLAP 関数を使用できる場合
SAP Sybase IQ は SQL OLAP 関数を提供しますが、規則と制限が適用されます。

- **SELECT** リストの中
- 式の中
- スカラ関数の引数として
- 最後の **ORDER BY** 句の中 (クエリ内のどこかで定義されている OLAP 関数のエイリアスまたは位置参照を使用)

OLAP 関数を使用できる場合
OLAP 関数は、次の条件下では使用できません。

- サブクエリの中。
- **WHERE** 句の検索条件の中。
- **SET** (集合) 関数の引数として。たとえば次の式は無効です。
  ```
  SUM(RANK() OVER(ORDER BY dollars))
  ```
- ウィンドウ集合を、他の集合に対する引数として使用することはできません (ただし、内側の集合がビューまたは抽出テーブル内で生成されたものである場合は例外です)。ランキング関数についても同じことが言えます。
- ウィンドウ集合関数と **RANK** 関数は、**HAVING** 句の中では使用できません。
- ウィンドウ集合関数に **DISTINCT** を指定しないでください。
- ウィンドウ関数を他のウィンドウ関数の内部にネストすることはできません。
- 逆分散統計関数は、**OVER** 句ではサポートされていません。
- ウィンドウ定義句では外部参照を使用できません。
- OLAP 関数内での相関参照は認められていますが、相関があるカラムのエイリアスは認められていません。

OLAP 関数から参照するカラムは、その OLAP 関数と **GROUP BY** 句が含まれている同じクエリブロック内のグループ化カラムまたは集合関数である必要があります。 OLAP の処理は、グループ化操作と集合操作の後、最後の **ORDER BY** 句が適用される前に行われます。そのため、中間の結果セットから OLAP 式を導出することもできます。 クエリブロック内に **GROUP BY** 句がない場合、OLAP 関数は SELECT リスト内の他のカラムを参照できます。

SAP Sybase IQ の制限事項
SAP Sybase IQ で SQL OLAP 関数を使用するときの制限事項を次に示します。

- ウィンドウフレーム定義内ではユーザ定義関数を使用できません。
- ウィンドウフレーム定義内で使用する定数は、符号なし数値である必要があります。また、最大値 BIG INT 2^{63-1} を超えることはできません。
- ウィンドウ集合関数と **RANK** 関数は、**DELETE** 文および **UPDATE** 文では使用できません。
- ウィンドウ集合関数と **RANK** 関数は、サブクエリ内では使用できません。
- CUME_DIST は、現時点ではサポートされていません。
- グループ化セットは、現時点ではサポートされていません。
- 相関関数と直線回帰関数は、現時点ではサポートされていません。

その他の OLAP の例

この項では、OLAP 関数を使用したその他の例を紹介します。

ウィンドウの開始ポイントと終了ポイントは、中間の結果ローが処理されるときに変化する可能性があります。たとえば、累積和を計算する場合には、ウィンドウの開始ポイントは各パーティションの最初のローに固定されますが、終了ポイントは現在のローを含めるためにパーティション内のローを移動していきます。

また、ウィンドウの開始ポイントと終了ポイントの両方が可変だが、パーティション全体のローの数は一定であるという例も考えられます。このようなウィンドウを使用すると移動平均を計算するクエリを作成でき、たとえば 3 日間の株価の移動平均を返す SQL クエリを作成できます。

例： クエリ内でのウィンドウ関数

次のクエリは、2005 年の 7 月と 8 月に出荷された全製品と、出荷日別の累積出荷数を一覧にして示します。

```
SELECT p.id, p.description, s.quantity, s.shipdate,

SUM(s.quantity) OVER (PARTITION BY productid ORDER BY s.shipdate rows
between unbounded preceding and current row)FROM SalesOrderItems s
JOIN Products p on(s.ProductID =p.id) WHERE s.ShipDate BETWEEN
'2001-05-01' and '2001-08-31' AND s.quantity > 40 ORDER BY p.id;
```

```
ID   description   quantity   ship_date   sum quantity
---  -----------   --------   ---------   ------------
302  Crew Neck           60  2001-07-02             60
400  Cotton Cap          60  2001-05-26             60
400  Cotton Cap          48  2001-07-05            108
401  Wool cap            48  2001-06-02             48
401  Wool cap            60  2001-06-30            108
401  Wool cap            48  2001-07-09            156
500  Cloth Visor         48  2001-06-21             48
501  Plastic Visor       60  2001-05-03             60
501  Plastic Visor       48  2001-05-18            108
501  Plastic Visor       48  2001-05-25            156
```

```
501  Plastic Visor       60  2001-07-07            216
601  Zipped Sweatshirt   60  2001-07-19             60
700  Cotton Shorts       72  2001-05-18             72
700  Cotton Shorts       48  2001-05-31            120
```

この例では、2 つのテーブルのジョインとクエリの **WHERE** 句を適用した後に、**SUM** ウィンドウ関数の計算が行われます。このクエリではインラインのウィンドウ指定を使用しており、このウィンドウ指定によって、ジョインからの入力ローが次のように処理されています。

1. `prod_id` 属性の値に基づいて入力ローがパーティション (グループ) に分けられます。
2. 各パーティション内で、ローが `ship_date` 属性に基づいてソートされます。
3. パーティション内の各ローの quantity 属性について、**SUM()** 関数が評価されます。このとき、ソートされた各パーティションの最初のローから現在のローまでを含む移動ウィンドウが使用されます。

このクエリを別の方法で記述するには、関数の外でウィンドウを定義し、そのウィンドウを関数呼び出しから参照します。この方法は、同じウィンドウに基づくウィンドウ関数を複数指定する場合に便利です。このウィンドウ関数を使用するクエリを、独立したウィンドウ句を使用する方法で記述すると次のようになります (cumulative というウィンドウを宣言しています)。

```
SELECT p.id, p.description, s.quantity, s.shipdate, SUM(s.quantity)
OVER(cumulative ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW )
cumulative FROM SalesOrderItems s JOIN Products p On (s.ProductID
=p.id)WHERE s.shipdate BETWEEN '2001-07-01' and '2001-08-31'Window
cumulative as (PARTITION BY s.productid ORDER BY s.shipdate)ORDER BY
p.id;
```

このクエリ指定では、ウィンドウ句が **ORDER BY** 句の前にあります。ウィンドウ句を使用するときには、次の制限が適用されます。

- インラインのウィンドウ指定に **PARTITION BY** 句を含めることはできません。
- ウィンドウ句で指定されるウィンドウに WINDOW FRAME 句 を含めることはできません。

  ```
  <WINDOW FRAME CLAUSE> ::=
    <WINDOW FRAME UNIT>
    <WINDOW FRAME EXTENT>
  ```

- インラインのウィンドウ指定にもウィンドウ句のウィンドウ指定にも WINDOW ORDER 句を含めることができますが、両方に含めることはできません。

  ```
  <WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>
  ```

例：複数の関数で使用されるウィンドウ

次のクエリでは、1 つの名前付きウィンドウを定義し、そのウィンドウに基づいて複数の関数を計算できます。

```
SELECT p.ID, p.Description, s.quantity, s.ShipDate, SUM(s.Quantity)
   OVER ws1, MIN(s.quantity) OVER ws1
FROM SalesOrderItems s
JOIN Products p ON (s.ProductID =p.ID)
  WHERE s.ShipDate BETWEEN '2000-01-09' AND'2000-01-17'
  AND s.Quantity > 40 window ws1
  AS(PARTITION BY productid
  ORDER BY shipdate rows between unbounded preceding and current row)
ORDER BY p.id;
```

```
ID   Description   quantity   shipDate     SUM   MIN
---  -----------   --------   -----------  ---   ---
400  Cotton Cap          48    2000-01-09   48    48
401  Wool cap            48    2000-01-09   48    48
500  Cloth Visor         60    2000-01-14   60    60
500  Cloth Visor         60    2000-01-15  120    60
501  Plastic Visor       60    2000-01-14   60    60
```

例：累積和の計算

このクエリでは、**ORDER BY** start_date の順序に従って、部署別の給与の累積和を計算します。

```
SELECT dept_id, start_date, name, salary,
  SUM(salary) OVER (PARTITION BY dept_id ORDER BY
  start_date ROWS BETWEEN UNBOUNDED PRECEDING AND
  CURRENT ROW)
FROM emp1
ORDER BY dept_id, start_date;
```

```
DepartmentID  start_date   name    salary   sum(salary)

--------     ----------    ----    ------   -----
100          1996-01-01    Anna       18000       18000
100                       1997-01-01    Mike    28000    46000
100                       1998-01-01    Scott   29000    75000
100                       1998-02-01    Antonia 22000    97000
100                       1998-03-12    Adam    25000   122000
100                       1998-12-01    Amy     18000   140000
200                       1998-01-01    Jeff    18000    18000
200                       1998-01-20    Tim     29000    47000
200                       1998-02-01    Jim     22000    69000
200                       1999-01-10    Tom     28000    97000
300                       1998-03-12    Sandy   55000    55000
300                       1998-12-01    Lisa    38000    93000
300                       1999-01-10    Peter   48000   141000
```

例：移動平均の計算

このクエリでは、連続する 3 か月間の売上の移動平均を計算します。 使用するウィンドウフレームのサイズは 3 つのローから成り、先行する 2 つのローと現在のローが含まれます。このウィンドウは、パーティションの最初から最後までスライドしていきます。

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num ROWS
  BETWEEN 2 PRECEDING AND CURRENT ROW)
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

```
prod_id   month_num      sales       avg(sales)
-------   ---------      ------      ----------
10                1        100          100.00
10                2        120          110.00
10                3        100          106.66
10                4        130          116.66
10                5        120          116.66
10                6        110          120.00
20                1         20           20.00
20                2         30           25.00
20                3         25           25.00
20                4         30           28.33
20                5         31           28.66
20                6         20           27.00
30                1         10           10.00
30                2         11           10.50
30                3         12           11.00
30                4          1            8.00
```

例：ORDER BY の結果

この例では、クエリの最上位の **ORDER BY** 句がウィンドウ関数の最終的な結果に適用されます。 ウィンドウ句に指定されている **ORDER BY** は、ウィンドウ関数の入力データに適用されます。

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num ROWS
  BETWEEN 2 PRECEDING AND CURRENT ROW)
FROM sale WHERE rep_id = 1
ORDER BY prod_id desc, month_num;
```

```
prod_id  month_num       sales   avg(sales)
-------  ---------       -----   ----------
30               1          10       10.00
30               2          11       10.50
30               3          12       11.00
30               4           1        8.00
20               1          20       20.00
20               2          30       25.00
20               3          25       25.00
```

```
20                4           30       28.33
20                5           31       28.66
20                6           20       27.00
10                1          100      100.00
10                2          120      110.00
10                3          100      106.66
10                4          130      116.66
10                5          120      116.66
10                6          110      120.00
```

例：1 つのクエリ内で複数の集合関数を使用

この例では、1 つのクエリ内で、異なるウィンドウに対して複数の集合値を計算しています。

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
  (WS1 ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS
  CAvg, SUM(sales) OVER(WS1 ROWS BETWEEN UNBOUNDED
  PRECEDING AND CURRENT ROW) AS CSum
FROM sale WHERE rep_id = 1 WINDOW WS1 AS (PARTITION BY
  prod_id
ORDER BY month_num)
ORDER BY prod_id, month_num;
```

```
prod_id  month_num    sales     CAvg     CSum
-------  ---------    -----     ----     ----
10                1      100   110.00      100
10                2      120   106.66      220
10                3      100   116.66      320
10                4      130   116.66      450
10                5      120   120.00      570
10                6      110   115.00      680
20                1       20    25.00       20
20                2       30    25.00       50
20                3       25    28.33       75
20                4       30    28.66      105
20                5       31    27.00      136
20                6       20    25.50      156
30                1       10    10.50       10
30                2       11    11.00       21
30                3       12     8.00       33
30                4        1     6.50       34
```

例：ウィンドウフレーム指定の ROWS と RANGE の比較

このクエリでは、ROWS と RANGE を比較しています。 **ORDER BY** 句ごとに、このデータには重複する ROWS が含まれています。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (ws1 RANGE BETWEEN 2 PRECEDING AND CURRENT ROW) AS
    Range_sum, SUM(sales) OVER
  (ws1 ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS
    Row_sum
FROM sale window ws1 AS (PARTITION BY prod_id ORDER BY
```

```
  month_num)
ORDER BY prod_id, month_num;
```

```
prod_id  month_num   sales   Range_sum   Row_sum
-------  ---------   -----   ----------  -------
10               1     100         250      100
10               1     150         250      250
10               2     120         370      370
10               3     100         470      370
10               4     130         350      350
10               5     120         381      350
10               5      31         381      281
10               6     110         391      261
20               1      20          20       20
20               2      30          50       50
20               3      25          75       75
20               4      30          85       85
20               5      31          86       86
20               6      20          81       81
30               1      10          10       10
30               2      11          21       21
30               3      12          33       33
30               4       1          25       24
30               4       1          25       14
```

例：現在のローを除外するウィンドウフレーム

この例では、現在のローを除外するウィンドウフレームを定義しています。このクエリは、現在のローを除く 4 つのローの合計を計算します。

```
SELECT prod_id, month_num, sales, sum(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num RANGE
  BETWEEN 6 PRECEDING AND 2 PRECEDING)
FROM sale
ORDER BY prod_id, month_num;
```

```
prod_id   month_num       sales    sum(sales)
-------   ---------       -----    ----------
10                1         100        (NULL)
10                1         150        (NULL)
10                2         120        (NULL)
10                3         100          250
10                4         130          370
10                5         120          470
10                5          31          470
10                6         110          600
20                1          20        (NULL)
20                2          30        (NULL)
20                3          25           20
20                4          30           50
20                5          31           75
20                6          20          105
30                1          10        (NULL)
30                2          11        (NULL)
30                3          12           10
```

```
30                4           1           21
30                4           1           21
```

例：RANGE のウィンドウフレーム

このクエリは、RANGE のウィンドウフレームを示しています。合計で使用されるローの数値は、変数です。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num RANGE
  BETWEEN 1 FOLLOWING AND 3 FOLLOWING)
FROM sale
ORDER BY prod_id, month_num;
```

```
prod_id   month_num       sales    sum(sales)
-------   ---------       -----    ----------
10                1         100           350
10                1         150           350
10                2         120           381
10                3         100           391
10                4         130           261
10                5         120           110
10                5          31           110
10                6         110        (NULL)
20                1          20            85
20                2          30            86
20                3          25            81
20                4          30            51
20                5          31            20
20                6          20        (NULL)
30                1          10            25
30                2          11            14
30                3          12             2
30                4           1        (NULL)
30                4           1        (NULL)
```

例：UNBOUNDED PRECEDING と UNBOUNDED FOLLOWING

この例では、パーティション内のすべてのローがウィンドウフレームに含まれます。このクエリは、パーティション全体 (各月に重複ローは含まれていません) における売上の最大値を計算します。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num ROWS
  BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

```
prod_id   month_num       sales    SUM(sales)
-------   ---------       -----    ----------
10                1         100           680
10                2         120           680
10                3         100           680
10                4         130           680
10                5         120           680
```

```
10                6          110           680
20                1           20           156
20                2           30           156
20                3           25           156
20                4           30           156
20                5           31           156
20                6           20           156
30                1           10            34
30                2           11            34
30                3           12            34
30                4            1            34
```

このクエリは、次のクエリと同じ意味になります。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id )
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

例：RANGE のデフォルトのウィンドウフレーム

このクエリは、RANGE のデフォルトのウィンドウフレームの例を示しています。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num)
FROM sale
ORDER BY prod_id, month_num;
```

```
prod_id   month_num       sales    SUM(sales)
-------   ---------       -----    ----------
10                1         100           250
10                1         150           250
10                2         120           370
10                3         100           470
10                4         130           600
10                5         120           751
10                5          31           751
10                6         110           861
20                1          20            20
20                2          30            50
20                3          25            75
20                4          30           105
20                5          31           136
20                6          20           156
30                1          10            10
30                2          11            21
30                3          12            33
30                4           1            35
30                4           1            35
```

このクエリは、次のクエリと同じ意味になります。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num RANGE
  BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
```

```
FROM sale
ORDER BY prod_id, month_num;
```

OLAP 関数の BNF 文法

次の BNF (Backus-Naur Form) 文法は、さまざまな ANSI SQL 分析関数に関する具体的な構文サポートの概要を示しています。ここに記載されている関数の多くは SAP Sybase IQ で実装されています。

文法規則 1

```
<SELECT LIST EXPRESSION> ::=
  <EXPRESSION>
  | <GROUP BY EXPRESSION>
  | <AGGREGATE FUNCTION>
  | <GROUPING FUNCTION>
  | <TABLE COLUMN>
  | <WINDOWED TABLE FUNCTION>
```

文法規則 2

```
<QUERY SPECIFICATION> ::=
  <FROM CLAUSE>
  [ <WHERE CLAUSE> ]
  [ <GROUP BY CLAUSE> ]
  [ <HAVING CLAUSE> ]
  [ <WINDOW CLAUSE> ]
[ <ORDER BY CLAUSE> ]
```

文法規則 3

```
<ORDER BY CLAUSE> ::= <ORDER SPECIFICATION>
```

文法規則 4

```
<GROUPING FUNCTION> ::=
  GROUPING <LEFT PAREN> <GROUP BY EXPRESSION>
  <RIGHT PAREN>
```

文法規則 5

```
<WINDOWED TABLE FUNCTION> ::=
  <WINDOWED TABLE FUNCTION TYPE> OVER <WINDOW NAME OR
  SPECIFICATION>
```

文法規則 6

```
<WINDOWED TABLE FUNCTION TYPE> ::=
  <RANK FUNCTION TYPE> <LEFT PAREN> <RIGHT PAREN>
  | ROW_NUMBER <LEFT PAREN> <RIGHT PAREN>
  | <WINDOW AGGREGATE FUNCTION>
```

文法規則 7

```
<RANK FUNCTION TYPE> ::=
  RANK | DENSE RANK | PERCENT RANK | CUME_DIST
```

文法規則 8

```
<WINDOW AGGREGATE FUNCTION> ::=
  <SIMPLE WINDOW AGGREGATE FUNCTION>
  | <STATISTICAL AGGREGATE FUNCTION>
```

文法規則 9

```
<AGGREGATE FUNCTION> ::=
  <DISTINCT AGGREGATE FUNCTION>
  | <SIMPLE AGGREGATE FUNCTION>
  | <STATISTICAL AGGREGATE FUNCTION>
```

文法規則 10

```
<DISTINCT AGGREGATE FUNCTION> ::=
  <BASIC AGGREGATE FUNCTION TYPE> <LEFT PAREN>
  <DISTINCT> <EXPRESSION> <RIGHT PAREN>
  | LIST <LEFT PAREN> DISTINCT <EXPRESSION>
  [ <COMMA> <DELIMITER> ]
  [ <ORDER SPECIFICATION> ] <RIGHT PAREN>
```

文法規則 11

```
<BASIC AGGREGATE FUNCTION TYPE> ::=
  SUM | MAX | MIN | AVG | COUNT
```

文法規則 12

```
<SIMPLE AGGREGATE FUNCTION> ::=
  <SIMPLE AGGREGATE FUNCTION TYPE> <LEFT PAREN>
  <EXPRESSION> <RIGHT PAREN>
  | LIST <LEFT PAREN> <EXPRESSION> [ <COMMA>
  <DELIMITER> ]
  [ <ORDER SPECIFICATION> ] <RIGHT PAREN>
```

文法規則 13

```
<SIMPLE AGGREGATE FUNCTION TYPE> ::= <SIMPLE WINDOW AGGREGATE
FUNCTION TYPE>
```

文法規則 14

```
<SIMPLE WINDOW AGGREGATE FUNCTION> ::=
  <SIMPLE WINDOW AGGREGATE FUNCTION TYPE> <LEFT PAREN>
  <EXPRESSION> <RIGHT PAREN>
| GROUPING FUNCTION
```

文法規則 15

```
<SIMPLE WINDOW AGGREGATE FUNCTION TYPE> ::=
  <BASIC AGGREGATE FUNCTION TYPE>
  | STDDEV | STDDEV_POP | STDDEV_SAMP
  | VARIANCE | VARIANCE_POP | VARIANCE_SAMP
```

文法規則 16

```
<STATISTICAL AGGREGATE FUNCTION> ::=
  <STATISTICAL AGGREGATE FUNCTION TYPE> <LEFT PAREN>
  <DEPENDENT EXPRESSION> <COMMA> <INDEPENDENT
  EXPRESSION> <RIGHT PAREN>
```

文法規則 17

```
<STATISTICAL AGGREGATE FUNCTION TYPE> ::=
  CORR | COVAR_POP | COVAR_SAMP | REGR_R2 |
  REGR_INTERCEPT | REGR_COUNT | REGR_SLOPE |
  REGR_SXX | REGR_SXY | REGR_SYY | REGR_AVGY |
  REGR_AVGX
```

文法規則 18

```
<WINDOW NAME OR SPECIFICATION> ::=
  <WINDOW NAME> | <IN-LINE WINDOW SPECIFICATION>
```

文法規則 19

```
<WINDOW NAME> ::= <IDENTIFIER>
```

文法規則 20

```
<IN-LINE WINDOW SPECIFICATION> ::= <WINDOW SPECIFICATION>
```

文法規則 21

```
<WINDOW CLAUSE> ::= <WINDOW WINDOW DEFINITION LIST>
```

文法規則 22

```
<WINDOW DEFINITION LIST> ::=
  <WINDOW DEFINITION> [ { <COMMA> <WINDOW DEFINITION>
  } . . . ]
```

文法規則 23

```
<WINDOW DEFINITION> ::=
  <NEW WINDOW NAME> AS <WINDOW SPECIFICATION>
```

文法規則 24

```
<NEW WINDOW NAME> ::= <WINDOW NAME>
```

文法規則 25

```
<WINDOW SPECIFICATION> ::=
  <LEFT PAREN> <WINDOW SPECIFICATION> <DETAILS> <RIGHT
  PAREN>
```

文法規則 26

```
<WINDOW SPECIFICATION DETAILS> ::=
  [ <EXISTING WINDOW NAME> ]
  [ <WINDOW PARTITION CLAUSE> ]
  [ <WINDOW ORDER CLAUSE> ]
  [ <WINDOW FRAME CLAUSE> ]
```

文法規則 27

```
<EXISTING WINDOW NAME> ::= <WINDOW NAME>
```

文法規則 28

```
<WINDOW PARTITION CLAUSE> ::=
  PARTITION BY <WINDOW PARTITION EXPRESSION LIST>
```

文法規則 29

```
<WINDOW PARTITION EXPRESSION LIST> ::=
  <WINDOW PARTITION EXPRESSION>
  [ { <COMMA> <WINDOW PARTITION EXPRESSION> } . . . ]
```

文法規則 30

```
<WINDOW PARTITION EXPRESSION> ::= <EXPRESSION>
```

文法規則 31

```
<WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>
```

文法規則 32

```
<WINDOW FRAME CLAUSE> ::=
  <WINDOW FRAME UNIT>
  <WINDOW FRAME EXTENT>
```

文法規則 33

```
<WINDOW FRAME UNIT> ::= ROWS | RANGE
```

文法規則 34

```
<WINDOW FRAME EXTENT> ::= <WINDOW FRAME START> | <WINDOW FRAME
BETWEEN>
```

文法規則 35

```
<WINDOW FRAME START> ::=
  UNBOUNDED PRECEDING
  | <WINDOW FRAME PRECEDING>
  | CURRENT ROW
```

文法規則 36

```
<WINDOW FRAME PRECEDING> ::= <UNSIGNED VALUE SPECIFICATION>
PRECEDING
```

文法規則 37

```
<WINDOW FRAME BETWEEN> ::=
  BETWEEN <WINDOW FRAME BOUND 1> AND <WINDOW FRAME
  BOUND 2>
```

文法規則 38

```
<WINDOW FRAME BOUND 1> ::= <WINDOW FRAME BOUND>
```

文法規則 39

```
<WINDOW FRAME BOUND 2> ::= <WINDOW FRAME BOUND>
```

文法規則 40

```
<WINDOW FRAME BOUND> ::=
  <WINDOW FRAME START>
  | UNBOUNDED FOLLOWING
  | <WINDOW FRAME FOLLOWING>
```

文法規則 41

```
<WINDOW FRAME FOLLOWING> ::= <UNSIGNED VALUE SPECIFICATION>
FOLLOWING
```

文法規則 42

```
<GROUP BY EXPRESSION> ::= <EXPRESSION>
```

文法規則 43

```
<SIMPLE GROUP BY TERM> ::=
  <GROUP BY EXPRESSION>
  | <LEFT PAREN> <GROUP BY EXPRESSION> <RIGHT PAREN>
  | <LEFT PAREN> <RIGHT PAREN>
```

文法規則 44

```
<SIMPLE GROUP BY TERM LIST> ::=
  <SIMPLE GROUP BY TERM> [ { <COMMA> <SIMPLE GROUP BY
  TERM> } . . . ]
```

文法規則 45

```
<COMPOSITE GROUP BY TERM> ::=
  <LEFT PAREN> <SIMPLE GROUP BY TERM>
  [ { <COMMA> <SIMPLE GROUP BY TERM> } . . . ]
  <RIGHT PAREN>
```

文法規則 46

```
<ROLLUP TERM> ::= ROLLUP <COMPOSITE GROUP BY TERM>
```

文法規則 47

```
<CUBE TERM> ::= CUBE <COMPOSITE GROUP BY TERM>
```

文法規則 48

```
<GROUP BY TERM> ::=
  <SIMPLE GROUP BY TERM>
  | <COMPOSITE GROUP BY TERM>
  | <ROLLUP TERM>
  | <CUBE TERM>
```

文法規則 49

```
<GROUP BY TERM LIST> ::=
  <GROUP BY TERM> [ { <COMMA> <GROUP BY TERM> } … ]
```

文法規則 50

```
<GROUP BY CLAUSE> ::= GROUP BY <GROUPING SPECIFICATION>
```

文法規則 51

```
<GROUPING SPECIFICATION> ::=
  <GROUP BY TERM LIST>
  | <SIMPLE GROUP BY TERM LIST> WITH ROLLUP
  | <SIMPLE GROUP BY TERM LIST> WITH CUBE
```

文法規則 52
サポートされていません。

文法規則 53

```
<ORDER SPECIFICATION> ::= ORDER BY <SORT SPECIFICATION LIST>
  <SORT SPECIFICATION LIST> ::= <SORT SPECIFICATION>
  [ { <COMMA> <SORT SPECIFICATION> } . . . ]
  <SORT SPECIFICATION> ::= <SORT KEY>
  [ <ORDERING SPECIFICATION> ] [ <NULL ORDERING> ]
  <SORT KEY> ::= <VALUE EXPRESSION>
  <ORDERING SPECIFICATION> ::= ASC | DESC
  <NULL ORDERING> := NULLS FIRST | NULLS LAST
```

付録： リモートデータへのアクセス

SAP Sybase IQ では、他のサーバに置かれているデータに対しても、ローカルデータにアクセスするかのような感覚でアクセスできます。これは、アクセス先が SAP Sybase のサーバでも SAP Sybase 以外のサーバでも同じです。

この機能を使用すると、データを SAP Sybase IQ データベースに移行したり、複数のデータベースにまたがってデータを問い合わせることができます。

SAP Sybase IQ とリモートデータ

SAP Sybase IQ のリモートデータアクセス機能によって、他のデータソースのデータにアクセスできます。この機能を使用すると、データを SQL Anywhere データベースに移行したり、複数のデータベースにまたがってデータを問い合わせることができます。

Sybase Open Client と jConnect 接続の特性

SAP Sybase IQ は TDS を通してアプリケーションからの要求を処理するとき、関連するデータベースオプションを、Adaptive Server のデフォルトの動作と互換性のある値に自動的に設定します。このオプションの設定は、その接続中だけの一時的なものです。オプションは、クライアントアプリケーションによっていつでも無効にできます。

デフォルトの設定値
TDS を使用する接続で設定されるデータベースオプションは、次のとおりです。

オプション	設定値
allow_nulls_by_default	Off
ansinull	Off
chained	Off
close_on_endtrans	Off
date_format	YYYY-MM-DD
date_order	MDY
escape_character	Off
isolation_level	1

オプション	設定値
on_tsql_error	Continue
quoted_identifier	Off
time_format	HH:NN:SS.SSS
timestamp_format	YYYY-MM-DD HH:NN:SS.SSS
tsql_variables	On

起動オプションの設定方法
TDS 接続用のデフォルトデータベースオプションは、システムプロシージャ sp_tsql_environment を使用して設定されます。このプロシージャでは、以下のオプションを設定します。

```
SET TEMPORARY OPTION allow_nulls_by_default='Off';
SET TEMPORARY OPTION ansinull='Off';
SET TEMPORARY OPTION chained='Off';
SET TEMPORARY OPTION close_on_endtrans='Off';
SET TEMPORARY OPTION date_format='YYYY-MM-DD';
SET TEMPORARY OPTION date_order='MDY';
SET TEMPORARY OPTION escape_character='Off';
SET TEMPORARY OPTION isolation_level='1';
SET TEMPORARY OPTION on_tsql_error='Continue';
SET TEMPORARY OPTION quoted_identifier='Off';
SET TEMPORARY OPTION time_format='HH:NN:SS.SSS';
SET TEMPORARY OPTION timestamp_format='YYYY-MM-DD HH:NN:SS.SSS';
SET TEMPORARY OPTION tsql_variables='On';
```

注意： sp_tsql_environment プロシージャは、変更しないでください。このプロシージャは、システムが専用に使用します。

このプロシージャは、TDS 通信プロトコルを使用する接続についてのみ、オプションを設定します。設定される接続には、jConnect を使用する Sybase Open Client 接続や JDBC 接続があります。その他の接続 (ODBC と Embedded SQL) は、データベース用のデフォルト設定値を持っています。

SAP Sybase IQ では長いユーザ名とパスワードが許可されていますが、TDS クライアントの名前とパスワードは最大で 30 バイトです。パスワードまたはユーザ ID が 30 バイト以上の場合、TDS 経由の接続試行 (たとえば、jConnect の使用) では無効なユーザまたはパスワードエラーが返されます。

TDS 接続用オプションの設定値の変更

SAP Sybase IQ は TDS を通してアプリケーションからの要求を処理するとき、関連するデータベースオプションを、Adaptive Server のデフォルトの動作と互換性のあ

る値に自動的に設定します。TDS 接続用のオプションは、次のようにしていつでも変更できます。

TDS 接続用オプションの設定値の変更

SAP Sybase IQ は TDS を通してアプリケーションからの要求を処理するとき、関連するデータベースオプションを、Adaptive Server のデフォルトの動作と互換性のある値に自動的に設定します。TDS 接続用のオプションは、次のようにしていつでも変更できます。

1. 目的のデータベースオプションを設定するプロシージャを作成します。
2. login_procedure オプションに新しいプロシージャ名を設定します。

今後の接続では、このプロシージャを使用します。別のユーザ ID に対して、別のプロシージャを設定できます。

リモートデータにアクセスするための要件

ここでは、リモートデータへのアクセスに必要な基本要素について説明します。

リモートテーブルのマッピング

クライアントアプリケーション側から見ると、接続している SQL Anywhere 内にすべてのテーブルがあるかのように見えます。リモートテーブルに関わるクエリが実行されると、内部では対象となるデータソースを割り出し、外部にあるそのデータソースにアクセスしてデータを取り出します。

リモートテーブルのデータにアクセスするには、次の設定を行う必要があります。

1. リモートオブジェクトが置かれるリモートサーバを定義します。これにはサーバのクラスとリモートサーバの場所が含まれます。これを行うには CREATE SERVER 文を使用します。
2. リモートサーバ上のデータベースへのアクセスに必要なクレデンシャルが、接続されているデータベースへのアクセスに必要なクレデンシャルと異なる場合は、リモートサーバのユーザログイン情報を定義します。これを行うには CREATE EXTERNLOGIN 文を使用します。
3. プロキシテーブルの定義を作成します。これによってリモートテーブルに対するローカルプロキシテーブルのマッピングを指定します。リモートテーブルが置かれているサーバ、リモートテーブルのデータベース名、所有者名、テーブル名、カラム名を指定します。これを行うには CREATE EXISTING TABLE 文を使用します。また、CREATE TABLE 文を使用しても、リモートサーバで新しいテーブルを作成できます。

リモートサーバの定義、外部ログイン、プロキシテーブルのマッピングを管理するには、Interactive SQL などのツールを使用して SQL 文を実行します。

警告！ 警告

Microsoft Access、Microsoft SQL Server、Sybase Adaptive Server Enterprise などの一部のリモートサーバでは、COMMIT や ROLLBACK を実行してもカーソルは保存されません。ただし、オートコミット機能が無効 (Interactive SQL のデフォルトの動作) になっている場合は、Interactive SQL を使用してプロキシテーブルのデータを表示および編集できます。Oracle Database、IBM DB2、SQL Anywhere などのその他の RDBMS では、この制限はありません。

リモートデータアクセスのサーバクラス

CREATE SERVER 文に指定するサーバクラスは、リモート接続の動作を決定します。サーバクラスは、サーバの機能の詳細な情報を SQL Anywhere に提供します。SQL Anywhere は、サーバの機能に合わせて SQL 文をフォーマットします。

サーバクラスはすべて ODBC ベースです。各サーバクラスには各種のユニークな特性があります。リモートデータアクセス用にサーバを設定するには、この特性を知っておく必要があります。サーバクラスのカテゴリ全般についての情報とともに、個々のサーバクラスに固有の情報を参照する必要があります。

サーバのクラスは次のとおりです。

SAODBC
ULODBC
ADSODBC
ASEODBC
DB2ODBC
HANAODBC
IQODBC
MSACCESSODBC
MSSODBC
MYSQLODBC
ODBC
ORAODBC

注意： リモートデータアクセスを使用する際に、Unicode をサポートしていない ODBC ドライバを使用すると、その ODBC ドライバから受け取るデータに対して、文字セット変換が実行されません。

ODBC 外部サーバ定義

ODBC ベースのリモートサーバを定義する最も一般的な方法は、ODBC データソースを基にすることです。これを実行するために、ODBC データソースアドミニストレータを使用して、データソースを作成することができます。

データソースを定義すると、CREATE SERVER 文の USING 句は ODBC データソース名 (DSN) を参照します。

たとえば、データソース名も mydb2 である mydb2 という名前の IBM DB2 サーバを定義するには、次の文を使用します。

```
CREATE SERVER mydb2
CLASS 'DB2ODBC'
USING 'mydb2';
```

使用するドライバはデータベースサーバのビット設定と一致する必要があります。

Windows では、ビット設定がデータベースサーバと一致するシステムデータソース名 (System DSN) を定義する必要もあります。たとえば、32 ビットのシステム DSN を作成するには、32 ビットの ODBC データソースアドミニストレータを使用します。ユーザ DSN にはビット設定はありません。

データソースの代わりに接続文字列を使用

データソース名を使用しない代わりに、CREATE SERVER 文の USING 句に接続文字列を指定します。これを実行するには、使用している ODBC ドライバの接続パラメータが必要です。たとえば、次は SQL Anywhere データベースサーバへの接続の例です。

```
CREATE SERVER TestSA
CLASS 'SAODBC'
USING 'DRIVER=Sybase IQ;HOST=myhost;Server=TestSA;DBN=sample';
```

この例で定義される接続先の TestSA という名前のデータベースサーバは、myhost というコンピュータで実行され、データベースが sample という名前で、使用するプロトコルが TCP/IP です。

CREATE SERVER 文の USING 句

アクセスするリモートの SQL Anywhere データベースごとに、個別の CREATE SERVER 文を発行してください。たとえば、TestSA という名前の SQL Anywhere サーバが Banana というコンピュータで稼働していて、3 つのデータベース (db1、db2、db3) を所有する場合は、次のようなリモートサーバを設定します。

```
CREATE SERVER TestSAdb1
CLASS 'SAODBC'
USING 'DRIVER=Sybase IQ;HOST=Banana;Server=TestSA;DBN=db1';

CREATE SERVER TestSAdb2
CLASS 'SAODBC'
USING 'DRIVER=Sybase IQ;HOST=Banana;Server=TestSA;DBN=db2';

CREATE SERVER TestSAdb3
CLASS 'SAODBC'
USING 'DRIVER=Sybase IQ;HOST=Banana;Server=TestSA;DBN=db3';
```

データベース名を指定しない場合、リモート接続にはリモートの SQL Anywhere サーバのデフォルトデータベースが使用されます。

サーバクラス SAODBC

サーバクラス SAODBC のリモートサーバは、SQL Anywhere データベースサーバです。SQL Anywhere データソースの設定には、特別な要件はありません。

複数のデータベースをサポートする SQL Anywhere データベースサーバにアクセスするには、各データベースへの接続を定義する ODBC データソース名を作成します。これらの ODBC データソース名のそれぞれに対して、CREATE SERVER 文を実行します。

例

CREATE SERVER 文の USING 句で接続文字列を指定して、SQL Anywhere データベースに接続します。

```
CREATE SERVER TestSA
CLASS 'SAODBC'
USING 'DRIVER=Sybase IQ;HOST=myhost;Server=TestSA;DBN=sample';
```

サーバクラス ADSODBC

CREATE TABLE 文を実行するときに、SQL Anywhere は、次のデータ型変換を使用して、データ型を対応する Advantage Database Server のデータ型に自動的に変換します。

SQL Anywhere データ型	ADS のデフォルトデータ型
BIT	Logical
VARBIT(*n*)	Binary(*n*)
LONG VARBIT	Binary(2G)
TINYINT	Integer
SMALLINT	Integer
INTEGER	Integer
BIGINT	Numeric(32)
UNSIGNED TINYINT	Numeric(11)
UNSIGNED SMALLINT	Numeric(11)
UNSIGNED INTEGER	Numeric(11)
UNSIGNED BIGINT	Numeric(32)
CHAR(*n*)	Character(*n*)
VARCHAR(*n*)	VarChar(*n*)
LONG VARCHAR	VarChar(65000)

SQL Anywhere データ型	ADS のデフォルトデータ型
NCHAR(*n*)	NChar(*n*)
NVARCHAR(*n*)	NVarChar(*n*)
LONG NVARCHAR	NVarChar(32500)
BINARY(*n*)	Binary(*n*)
VARBINARY(*n*)	Binary(*n*)
LONG BINARY	Binary(2G)
DECIMAL(*precision*, *scale*)	Numeric(*precision*+3)
NUMERIC(*precision*, *scale*)	Numeric(*precision*+3)
SMALLMONEY	Money
MONEY	Money
REAL	Double
DOUBLE	Double
FLOAT(*n*)	Double
DATE	Date
TIME	Time
TIMESTAMP	TimeStamp
TIMESTAMP WITH TIMEZONE	Char(254)
BIGDATE	datetime
BIGDATETIME	datetime
XML	Binary(2G)
ST_GEOMETRY	Binary(2G)
UNIQUEIDENTIFIER	Binary(2G)

サーバクラス ASEODBC

サーバクラスが ASEODBC のリモートサーバは、Adaptive Server Enterprise (バージョン 10 以降) データベースサーバです。SQL Anywhere では、クラスが ASEODBC のリモートの Adaptive Server Enterprise データベースサーバに接続するために、Adaptive Server Enterprise ODBC ドライバと Open Client 接続ライブラリのインストールが必要です。

注意

- Open Client はバージョン 11.1.1、EBF 7886 以降が必要です。Open Client をインストールして Adaptive Server Enterprise サーバへの接続を検証してから、ODBC をインストールして SQL Anywhere を設定してください。Sybase ODBC ドライバはバージョン 11.1.1、EBF 7911 以降が必要です。
- quoted_identifier オプションのローカル設定は、Adaptive Server Enterprise の引用符付き識別子の使用を制御します。たとえば、quoted_identifier オプションをローカルで Off に設定すると、Adaptive Server Enterprise に対して引用符付き識別子がオフになります。
- [Configuration Manager] でユーザデータソースを次の属性で設定します。
  - [General] **タブ –** [Data Source Name] に任意の値を入力します。この値は、CREATE SERVER 文の USING 句に使用されます。

    サーバ名は Sybase interfaces ファイルにあるサーバ名と一致させてください。
  - [Advanced] **タブ –** [Application Using Threads] オプションと [Enable Quoted Identifiers] オプションをクリックします。
  - [Connection] **タブ –** [charset] フィールドを、SQL Anywhere の文字セットに一致するように設定します。

    [language] フィールドを、エラーメッセージを表示したい言語に設定します。
  - [Performance] **タブ –** [Prepare Method] を **2-Full** に設定します。

    最高のパフォーマンスを得るには、[Fetch Array Size] をできるだけ大きな値に設定します。これはメモリ内にキャッシュされるローの数なので、この値を大きくすると必要なメモリ量が増大します。Adaptive Server Enterprise では 100 の値を使用することをおすすめします。

    [Select Method] を **0-Cursor** に設定します。

    [Packet Size] をできるだけ大きな値に設定します。Adaptive Server Enterprise では -1 の値を使用することをおすすめします。

    [Connection Cache] を 1 に設定します。

データ型変換：ODBC と Adaptive Server Enterprise

CREATE TABLE 文を実行するときに、SQL Anywhere は、データ型を対応する Adaptive Server Enterprise のデータ型に自動的に変換します。次の表に、SQL Anywhere から Adaptive Server Enterprise へのデータ型変換を示します。

SQL Anywhere データ型	Adaptive Server Enterprise のデフォルトデータ型
BIT	bit
VARBIT(*n*)	if (n <= 255) varbinary(*n*) else image
LONG VARBIT	image
TINYINT	tinyint
SMALLINT	smallint
INT, INTEGER	int
BIGINT	numeric(20,0)
UNSIGNED TINYINT	tinyint
UNSIGNED SMALLINT	int
UNSIGNED INTEGER	numeric(11,0)
UNSIGNED BIGINT	numeric(20,0)
CHAR(*n*)	if (n <= 255) char(*n*) else text
VARCHAR(*n*)	if (n <= 255) varchar(*n*) else text
LONG VARCHAR	text
NCHAR(*n*)	if (n <= 255) nchar(*n*) else ntext
NVARCHAR(*n*)	if (n <= 255) nvarchar(*n*) else ntext
LONG NVARCHAR	ntext
BINARY(*n*)	if (n <= 255) binary(*n*) else image
VARBINARY(*n*)	if (n <= 255) varbinary(*n*) else image
LONG BINARY	image
DECIMAL(*prec,scale*)	decimal(*prec,scale*)
NUMERIC(*prec,scale*)	numeric(*prec,scale*)
SMALLMONEY	numeric(10,4)
MONEY	numeric(19,4)
REAL	real
DOUBLE	float

SQL Anywhere データ型	Adaptive Server Enterprise のデフォルトデータ型
FLOAT(*n*)	float(*n*)
DATE	datetime
TIME	datetime
SMALLDATETIME	smalldatetime
TIMESTAMP	datetime
TIMESTAMP WITH TIMEZONE	varchar(254)
XML	text
ST_GEOMETRY	image
UNIQUEIDENTIFIER	binary(16)

例

CREATE SERVER 文の USING 句で接続文字列を指定して、Adaptive Server Enterprise データベースに接続します。

```
CREATE SERVER TestASE
CLASS 'ASEODBC'
USING 'DRIVER=SYBASE ASE ODBC
Driver;Server=TestASE;Port=5000;Database=testdb;UID=username;PWD=pa
ssword'
```

サーバクラス DB2ODBC

サーバクラスが DB2ODBC のリモートサーバは IBM DB2 データベースサーバです。

注意

- iAnywhere は、IBM の DB2 Connect バージョン 5 (修正パック WR09044 付き) の使用を確認しています。この製品の説明に従って、ODBC 構成の設定とテストを実行してください。SQL Anywhere には、IBM DB2 データソースの設定について特別な要件はありません。
- 以下は、mydb2 という ODBC データソースを持つ IBM DB2 サーバの CREATE EXISTING TABLE 文の例です。

  ```
  CREATE EXISTING TABLE ibmcol
  AT 'mydb2..sysibm.syscolumns';
  ```

データ型変換：IBM DB2

CREATE TABLE 文を実行するときに、SQL Anywhere は、データ型を対応する IBM DB2 のデータ型に自動的に変換します。次の表に、SQL Anywhere から IBM DB2 へのデータ型変換を示します。

SQL Anywhere データ型	IBM DB2 のデフォルトデータ型
BIT	smallint
VARBIT(*n*)	if (n <= 4000) varchar(*n*) for bit data else long varchar for bit data
LONG VARBIT	long varchar for bit data
TINYINT	smallint
SMALLINT	smallint
INTEGER	int
BIGINT	decimal(20,0)
UNSIGNED TINYINT	int
UNSIGNED SMALLINT	int
UNSIGNED INTEGER	decimal(11.0)
UNSIGNED BIGINT	decimal(20,0)
CHAR(*n*)	if (n < 255) char(*n*) else if (n <= 4000) varchar(*n*) else long varchar
VARCHAR(*n*)	if (n <= 4000) varchar(*n*) else long varchar
LONG VARCHAR	long varchar
NCHAR(*n*)	サポートされていない
NVARCHAR(*n*)	サポートされていない
LONG NVARCHAR	サポートされていない
BINARY(*n*)	if (n <= 4000) varchar(*n*) for bit data else long varchar for bit data
VARBINARY(*n*)	if (n <= 4000) varchar(*n*) for bit data else long varchar for bit data
LONG BINARY	long varchar for bit data

SQL Anywhere データ型	IBM DB2 のデフォルトデータ型
DECIMAL(*prec,scale*)	decimal(*prec,scale*)
NUMERIC(*prec,scale*)	decimal(*prec,scale*)
SMALLMONEY	decimal(10,4)
MONEY	decimal(19,4)
REAL	real
DOUBLE	float
FLOAT(*n*)	float(*n*)
DATE	date
TIME	time
TIMESTAMP	timestamp
TIMESTAMP WITH TIMEZONE	varchar(254)
XML	long varchar for bit data
ST_GEOMETRY	long varchar for bit data
UNIQUEIDENTIFIER	varchar(16) for bit data

サーバクラス HANAODBC

サーバクラスが HANAODBC のリモートサーバは SAP HANA データベースサーバです。

注意

- 以下は、mySAPHANA という ODBC データソースを持つ SAP HANA データベースサーバの CREATE EXISTING TABLE 文の例です。

```
CREATE EXISTING TABLE hanatable
AT 'mySAPHANA..dbo.hanatable';
```

データ型変換：SAP HANA

CREATE TABLE 文を実行するときに、SQL Anywhere は、データ型を対応する SAP HANA のデータ型に自動的に変換します。次の表に、SQL Anywhere から SAP HANA へのデータ型変換を示します。

SQL Anywhere データ型	SAP HANA のデフォルトデータ型
BIT	TINYINT

SQL Anywhere データ型	SAP HANA のデフォルトデータ型
VARBIT(*n*)	if (n <= 5000) VARBINARY(*n*) else BLOB
LONG VARBIT	BLOB
TINYINT	TINYINT
SMALLINT	SMALLINT
INTEGER	INTEGER
BIGINT	BIGINT
UNSIGNED TINYINT	TINYINT
UNSIGNED SMALLINT	INTEGER
UNSIGNED INTEGER	BIGINT
UNSIGNED BIGINT	DECIMAL(20,0)
CHAR(*n*)	if (n <= 5000) VARCHAR(*n*) else CLOB
VARCHAR(*n*	if (n <= 5000) VARCHAR(*n*) else CLOB
LONG VARCHAR	CLOB
NCHAR(*n*)	if (n <= 5000) NVARCHAR(*n*) else NCLOB
NVARCHAR(*n*)	if (n <= 5000) NVARCHAR(*n*) else NCLOB
LONG NVARCHAR	NCLOB
BINARY(*n*)	if (n <= 5000) VARBINARY(*n*) else BLOB
VARBINARY(*n*)	if (n <= 5000) VARBINARY(*n*) else BLOB
LONG BINARY	BLOB
DECIMAL(*precision*, *scale*)	DECIMAL(*precision*, *scale*)
NUMERIC(*precision*, *scale*)	DECIMAL(*precision*, *scale*)
SMALLMONEY	DECIMAL(13,4)
MONEY	DECIMAL(19.4)
REAL	REAL
DOUBLE	FLOAT
FLOAT(*n*)	FLOAT
DATE	DATE

SQL Anywhere データ型	SAP HANA のデフォルトデータ型
TIME	TIME
TIMESTAMP	TIMESTAMP
TIMESTAMP WITH TIMEZONE	VARCHAR(254)
XML	BLOB
ST_GEOMETRY	BLOB
UNIQUEIDENTIFIER	VARBINARY(16)

サーバクラス IQODBC

サーバクラスが IQODBC のリモートサーバは SAP Sybase IQ データベースサーバです。SAP Sybase IQ データソースの設定には、特別な要件はありません。

複数のデータベースをサポートする SAP Sybase IQ データベースサーバにアクセスするには、各データベースへの接続を定義する ODBC データソース名を作成します。これらの ODBC データソース名のそれぞれに対して、CREATE SERVER 文を実行します。

サーバクラス MSACCESSODBC

Access データベースは .mdb ファイルに格納されます。ODBC マネージャを使用して、ODBC データソースを作成し、これらのファイルの 1 つにマッピングします。新しい .mdb ファイルは、ODBC マネージャを使って作成できます。SQL Anywhere でテーブルを作成するときにデフォルトを指定しないと、このデータベースファイルがデフォルトになります。

ODBC データソースが access という名前であると仮定した場合、次のいずれかの文を使用してデータにアクセスできます。

- ```
  CREATE TABLE tab1 (a int, b char(10))
  AT 'access...tab1';
  ```
- ```
  CREATE TABLE tab1 (a int, b char(10))
  AT 'access;d:¥¥pcdb¥¥data.mdb;;tab1';
  ```
- ```
  CREATE EXISTING TABLE tab1
  AT 'access;d:¥¥pcdb¥¥data.mdb;;tab1';
  ```

Access では所有者名の修飾をサポートしないので、これはブランクのままにしてください。
  ```

データ型変換：Microsoft Access

SQL Anywhere データ型	Microsoft Access のデフォルトデータ型
BIT	TINYINT
VARBIT(*n*)	if (n <= 4000) BINARY(*n*) else IMAGE
LONG VARBIT	IMAGE
TINYINT	TINYINT
SMALLINT	SMALLINT
INTEGER	INTEGER
BIGINT	DECIMAL(19,0)
UNSIGNED TINYINT	TINYINT
UNSIGNED SMALLINT	INTEGER
UNSIGNED INTEGER	DECIMAL(11,0)
UNSIGNED BIGINT	DECIMAL(20,0)
CHAR(*n*)	if (n < 255) CHARACTER(*n*) else TEXT
VARCHAR(*n*)	if (n < 255) CHARACTER(*n*) else TEXT
LONG VARCHAR	TEXT
NCHAR(*n*)	Not supported
NVARCHAR(*n*)	Not supported
LONG NVARCHAR	Not supported
BINARY(*n*)	if (n <= 4000) BINARY(*n*) else IMAGE
VARBINARY(*n*)	if (n <= 4000) BINARY(*n*) else IMAGE
LONG BINARY	IMAGE
DECIMAL(*precision*, *scale*)	DECIMAL(*precision*, *scale*)
NUMERIC(*precision*, *scale*)	DECIMAL(*precision*, *scale*)
SMALLMONEY	MONEY
MONEY	MONEY
REAL	REAL

SQL Anywhere データ型	Microsoft Access のデフォルトデータ型
DOUBLE	FLOAT
FLOAT(*n*)	FLOAT
DATE	DATETIME
TIME	DATETIME
TIMESTAMP	DATETIME
TIMESTAMP WITH TIMEZONE	CHARACTER(254)
XML	XML
ST_GEOMETRY	IMAGE
UNIQUEIDENTIFIER	BINARY(16)

サーバクラス MSSODBC

サーバクラス MSSODBC を使用して、Microsoft SQL Server に、そのいずれかの ODBC ドライバを介してアクセスします。

注意

- これまでに使用されてきた Microsoft SQL Server ODBC ドライバのバージョンは次のとおりです。
  - Microsoft SQL Server ODBC Driver バージョン 06.01.7601
  - Microsoft SQL Server Native Client バージョン 10.00.1600
- 次に、Microsoft SQL Server の例を示します。

```
CREATE SERVER mysqlserver
CLASS 'MSSODBC'
USING 'DSN=MSSODBC_cli';

CREATE EXISTING TABLE accounts
AT 'mysqlserver.master.dbo.accounts';
```

- quoted_identifier オプションのローカル設定は、Microsoft SQL Server の引用符付き識別子の使用を制御します。たとえば、quoted_identifier オプションをローカルで Off に設定すると、Microsoft SQL Server に対して引用符付き識別子がオフになります。

データ型変換：Microsoft SQL Server

CREATE TABLE 文を実行するときに、SQL Anywhere は、次のデータ型変換を使用して、データ型を対応する Microsoft SQL Server のデータ型に自動的に変換します。

SQL Anywhere データ型	Microsoft SQLServer のデフォルトデータ型
BIT	bit
VARBIT(*n*)	if (n <= 255) varbinary(*n*) else image
LONG VARBIT	image
TINYINT	tinyint
SMALLINT	smallint
INTEGER	int
BIGINT	numeric(20,0)
UNSIGNED TINYINT	tinyint
UNSIGNED SMALLINT	int
UNSIGNED INTEGER	numeric(11,0)
UNSIGNED BIGINT	numeric(20,0)
CHAR(*n*)	if (n <= 255) char(*n*) else text
VARCHAR(*n*)	if (n <= 255) varchar(*n*) else text
LONG VARCHAR	text
NCHAR(*n*)	if (n <= 4000) nchar(*n*) else ntext
NVARCHAR(*n*)	if (n <= 4000) nvarchar(*n*) else ntext
LONG NVARCHAR	ntext
BINARY(*n*)	if (n <= 255) binary(*n*) else image
VARBINARY(*n*)	if (n <= 255) varbinary(*n*) else image
LONG BINARY	image
DECIMAL(*precision*, *scale*)	decimal(*precision*, *scale*)
NUMERIC(*precision*, *scale*)	numeric(*precision*, *scale*)
SMALLMONEY	smallmoney
MONEY	money
REAL	real
DOUBLE	float
FLOAT(*n*)	float(*n*)

SQL Anywhere データ型	Microsoft SQLServer のデフォルトデータ型
DATE	datetime
TIME	datetime
SMALLDATETIME	smalldatetime
DATETIME	datetime
TIMESTAMP	datetime
TIMESTAMP WITH TIMEZONE	varchar(254)
XML	xml
ST_GEOMETRY	image
UNIQUEIDENTIFIER	binary(16)

サーバクラス MYSQLODBC

CREATE TABLE 文を実行するときに、SQL Anywhere は、次のデータ型変換を使用して、データ型を対応する MySQL のデータ型に自動的に変換します。

SQL Anywhere データ型	MySQL のデフォルトデータ型
BIT	bit(1)
VARBIT(*n*)	if (n <= 4000) varbinary(*n*) else longblob
LONG VARBIT	longblob
TINYINT	tinyint unsigned
SMALLINT	smallint
INTEGER	int
BIGINT	bigint
UNSIGNED TINYINT	tinyint unsigned
UNSIGNED SMALLINT	int
UNSIGNED INTEGER	bigint
UNSIGNED BIGINT	decimal(20,0)
CHAR(*n*)	if (n < 255) char(*n*) else if (n <= 4000) varchar(*n*) else longtext
VARCHAR(*n*)	if (n <= 4000) varchar(*n*) else longtext
LONG VARCHAR	longtext

SQL Anywhere データ型	MySQL のデフォルトデータ型
NCHAR(*n*)	if (n < 255) national character(*n*) else if (n <= 4000) national character varying(*n*) else longtext
NVARCHAR(*n*)	if (n <= 4000) national character varying(*n*) else longtext
LONG NVARCHAR	longtext
BINARY(*n*)	if (n <= 4000) varbinary(*n*) else longblob
VARBINARY(*n*)	if (n <= 4000) varbinary(*n*) else longblob
LONG BINARY	longblob
DECIMAL(*precision*, *scale*)	decimal(*precision*, *scale*)
NUMERIC(*precision*, *scale*)	decimal(*precision*, *scale*)
SMALLMONEY	decimal(10,4)
MONEY	decimal(19,4)
REAL	real
DOUBLE	float
FLOAT(*n*)	float(*n*)
DATE	date
TIME	time
TIMESTAMP	datetime
TIMESTAMP WITH TIMEZONE	varchar(254)
XML	longblob
ST_GEOMETRY	longblob
UNIQUEIDENTIFIER	varbinary(16)

例

CREATE SERVER 文の USING 句で接続文字列を指定して、MySQL データベースに接続します。

```
CREATE SERVER TestMySQL
CLASS 'MYSQLODBC'
USING 'DRIVER=MySQL ODBC 5.1
Driver;DATABASE=mydatabase;SERVER=mySQLHost;UID=me;PWD=secret'
```

サーバクラス ODBC

独自のサーバクラスを持たない ODBC データソースでは、サーバクラス `ODBC` を使用します。任意の ODBC ドライバを使用できます。iAnywhere は次の ODBC データソースの使用を確認しています。

Microsoft Excel (Microsoft 3.51.171300)
Microsoft FoxPro (Microsoft 3.51.171300)
Lotus Notes SQL

最新バージョンの Microsoft ODBC ドライバは、Microsoft Data Access Components (MDAC) とともに、Microsoft ダウンロードセンタからダウンロードできます。MDAC 2.0 には、上に示すバージョンの Microsoft ドライバが含まれています。

Microsoft Excel (Microsoft 3.51.171300)

Excel では、それぞれの Excel ブックはいくつかのテーブルを持つデータベースであると、論理的に考えられます。テーブルはブック内のシートにマッピングされます。ODBC ドライバマネージャで ODBC データソース名を設定する場合は、そのデータソースに関連付けられたデフォルトのブック名を指定します。ただし、CREATE TABLE 文を実行する場合には、デフォルトを無効にしてロケーションの文字列にブック名を指定できます。これによって、1 つの ODBC DSN を使用してすべての Excel ブックにアクセスできます。

Microsoft Excel ODBC ドライバに接続する excel という名前のリモートサーバを作成します。

```
CREATE SERVER excel
CLASS 'ODBC'
USING 'DRIVER=Microsoft Excel Driver (*.xls);DBQ=d:¥
¥work1.xls;READONLY=0;DriverID=790'
```

mywork というシート (テーブル) を持つ `work1.xls` というブックを作成するには、次のようにします。

```
CREATE TABLE mywork (a int, b char(20))
AT 'excel;d:¥¥work1.xls;;mywork';
```

2 番目のシート (テーブル) を作成するには、次のような文を実行します。

```
CREATE TABLE mywork2 (x float, y int)
AT 'excel;d:¥¥work1.xls;;mywork2';
```

CREATE EXISTING 文を使用して、既存のシートを SQL Anywhere にインポートできます。ここでは、シートの最初のローには、カラム名が入ることを前提としています。

```
CREATE EXISTING TABLE mywork
AT'excel;d:¥¥work1;;mywork';
```

SQL Anywhere によって、テーブルが見つからないと表示された場合は、マッピングするカラムとローの範囲を明示的に指定する必要があります。次に例を示します。

```
CREATE EXISTING TABLE mywork
AT 'excel;d:¥¥work1;;mywork$';
```

シート名に $ を追加すると、ワークシート全体が選択されることを示します。

AT で指定するロケーションの文字列で、フィールドセパレータとしてピリオドの代わりにセミコロンが使用されていることに注意してください。これは、ファイル名にピリオドが使用されるためです。Excel では所有者名フィールドをサポートしないので、これはブランクのままにしてください。

削除はサポートされていません。また、Excel ドライバは位置付け更新をサポートしないため、更新も可能でない場合があります。

例

次の文は、ODBC DSN を使用して Excel ワークブック `LogFile.xlsx` にアクセスし、そのシートを SQL Anywhere にインポートする、TestExcel という名前のデータベースサーバを作成します。

```
CREATE SERVER TestExcel
CLASS 'ODBC'
USING 'DRIVER=Microsoft Excel Driver (*.xls);DBQ=c:¥¥temp¥
¥LogFile.xlsx;READONLY=0;DriverID=790'

CREATE EXISTING TABLE MyWorkbook
AT 'TestExcel;c:¥¥temp¥¥LogFile.xlsx;;Logfile$';

SELECT * FROM MyWorkbook;
```

Microsoft FoxPro (Microsoft 3.51.171300)

FoxPro の各テーブルを 1 つの FoxPro データベースファイル (`.dbc`) にまとめて格納したり、各テーブルをそれぞれの `.dbf` ファイルに格納することができます。`.dbf` ファイルを使用するときは、ファイル名がロケーションの文字列に入っていることを確認してください。入っていない場合は、SQL Anywhere が起動されたディレクトリが使用されます。

```
CREATE TABLE fox1 (a int, b char(20))
AT 'foxpro;d:¥¥pcdb;;fox1';
```

この文は、ODBC ドライバマネージャで [Free Table Directory] オプションが選択されている場合、`d:¥pcdb¥fox1.dbf` というファイルを作成します。

Lotus Notes SQL

このドライバを取得するには、Lotus NotesSQL の Web サイト (http://www.ibm.com/developerworks/lotus/products/notesdomino/notessql/) を参照してください。Notes データがリレーショナルテーブルにどのようにマッピングされるかについては、ドライバに付属のマニュアルを参照してください。SQL Anywhere テーブルは、Notes フォームに簡単にマッピングできます。

Lotus Notes の連絡先にアクセスするよう SQL Anywhere を設定する方法を次に示します。

- Lotus Notes プログラムフォルダがパスに含まれていることを確認します (C:¥Program Files (x86)¥IBM¥Lotus¥Notes など)。
- NotesSQL ODBC ドライバを使用して 32 ビットの ODBC データソースを作成します。この例では `names.nsf` データベースを使用します。[特殊文字のマップ]オプションを有効にしてください。この例では、[データソース名]は `my_notes_dsn` です。
- 32 ビットのデータベースサーバに接続された Interactive SQL を使用して、リモートデータアクセスサーバを作成します。次に例を示します。

```
CREATE SERVER NotesContacts
CLASS 'ODBC'
USING 'my_notes_dsn';
```

- Lotus Notes サーバ用の外部ログインを作成します。次に例を示します。

```
CREATE EXTERNLOGIN "DBA" TO "NotesContacts"
REMOTE LOGIN 'John Doe/SYBASE' IDENTIFIED BY 'MyNotesPassword';
```

- Person フォームの一部のカラムを SQL Anywhere テーブルにマッピングします。

```
CREATE EXISTING TABLE PersonDetails
( DisplayName CHAR(254),
  DisplayMailAddress CHAR(254),
  JobTitle CHAR(254),
  CompanyName CHAR(254),
  Department CHAR(254),
  Location CHAR(254),
  OfficePhoneNumber CHAR(254) )
AT 'NotesContacts...Person';
```

- 次のようにテーブルを問い合わせます。

```
SELECT * FROM PersonDetails
WHERE Location LIKE 'Waterloo%';
```

サーバクラス ORAODBC

サーバクラスが ORAODBC のリモートサーバは、Oracle Database バージョン 8.0 以降です。

注意

- iAnywhere は、Oracle Database バージョン 8.0.03 の ODBC ドライバの使用を確認しています。この製品の説明に従って、ODBC 構成の設定とテストを実行してください。
- 以下は、myora という Oracle Database サーバの CREATE EXISTING TABLE 文の例です。

```
CREATE EXISTING TABLE employees
AT 'myora.database.owner.employees';
```

データ型変換：Oracle Database

CREATE TABLE 文を実行するときに、SQL Anywhere は、次のデータ型変換を使用して、データ型を対応する Oracle Database のデータ型に自動的に変換します。

SQL Anywhere データ型	Oracle Database のデータ型
BIT	number(1,0)
VARBIT(*n*)	if (n <= 255) raw(*n*) else long raw
LONG VARBIT	long raw
TINYINT	number(3,0)
SMALLINT	number(5,0)
INTEGER	number(11,0)
BIGINT	number(20,0)
UNSIGNED TINYINT	number(3,0)
UNSIGNED SMALLINT	number(5,0)
UNSIGNED INTEGER	number(11,0)
UNSIGNED BIGINT	number(20,0)
CHAR(*n*)	if (n <= 255) char(*n*) else long
VARCHAR(*n*)	if (n <= 2000) varchar(*n*) else long
LONG VARCHAR	long
NCHAR(*n*)	if (n <= 255) nchar(*n*) else nclob

SQL Anywhere データ型	Oracle Database のデータ型
NVARCHAR(*n*)	if (n <= 2000) nvarchar(*n*) else nclob
LONG NVARCHAR	nclob
BINARY(*n*)	if (n > 255) long raw else raw(*n*)
VARBINARY(*n*)	if (n > 255) long raw else raw(*n*)
LONG BINARY	long raw
DECIMAL(*precision*, *scale*)	number(*precision*, *scale*)
NUMERIC(*precision*, *scale*)	number(*precision*, *scale*)
SMALLMONEY	numeric(13,4)
MONEY	number(19,4)
REAL	real
DOUBLE	float
FLOAT(*n*)	float
DATE	date
TIME	date
TIMESTAMP	date
TIMESTAMP WITH TIMEZONE	varchar(254)
XML	long raw
ST_GEOMETRY	long raw
UNIQUEIDENTIFIER	raw(16)

例

CREATE SERVER 文の USING 句で接続文字列を指定して、Oracle データベースに接続します。

```
CREATE SERVER TestOracle
CLASS 'ORAODBC'
USING 'DRIVER=Oracle ODBC
Driver;DBQ=mydatabase;UID=username;PWD=password'
```

リモートサーバ

リモートオブジェクトを配置するリモートサーバを定義してから、リモートオブジェクトをローカルプロキシテーブルにマッピングします。

リモートサーバの作成

CREATE SERVER 文を使用して、リモートサーバの定義を設定します。

SAP Sybase IQ や SQL Anywhere を含む一部のシステムでは、各データソースが 1 つのデータベースを表すため、データベースごとにリモートサーバの定義が必要になります。

参照：

- CREATE SERVER 文 (775 ページ)

リモートの Oracle データにアクセスする前に

リモートの Oracle データにアクセスするには、必須のソフトウェアを使用してシステムを設定します。

1. 前提条件の確認

   コンポーネント統合サービス (CIS) を使用して Oracle データにアクセスするために必要なソフトウェアコンポーネントがシステムに揃っていることを確認します。
2. Oracle データソース名の作成

   .odbc.ini ファイル内にエントリを作成するには、iqdsn ユーティリティを使用します。
3. Oracle データにアクセスするための環境変数の設定

   SAP Sybase IQ サーバを起動して Oracle データにアクセスする前に、いくつかの環境変数を設定する必要があります。
4. SAP Sybase IQ サーバの起動

   Oracle データにアクセスするフロントエンドとして使用する SAP Sybase IQ サーバを起動します。

前提条件の確認

コンポーネント統合サービス (CIS) を使用して Oracle データにアクセスするために必要なソフトウェアコンポーネントがシステムに揃っていることを確認します。

前提条件は次のとおりです。

- Oracle データベース。
- Oracle クライアントソフトウェア (基本パッケージ) (`network/admin/tnsnames.ora` ファイルを含む)。
- プラットフォーム固有のドライバ (SAP Sybase IQ とともにインストールされます):

プラットフォーム	ファイル
AIX 64	`$IQDIR16/libxx/libdboraodbc12_r.so`
HPiUX	`$IQDIR16/libxx/libdboraodbc12_r.so.1`
Linux64	`$IQDIR16/libxx/libdboraodbc12_r.so.1`
SunOS64	`$IQDIR16/libxx/libdboraodbc12_r.so.1`
WinAMD64	`%$IQDIR16%¥bin64¥dboraodbc12.dll`

Oracle データソース名の作成

`.odbc.ini` ファイル内にエントリを作成するには、**iqdsn** ユーティリティを使用します。

1. Oracle 接続キーワードを表示します。

   ```
   % iqdsn -cl -or
   ```

   ```
   Driver
   UserID              UID
   Password            PWD
   SID                 SID
   Encrypted Password  ENP
   ProcResults         PROC
   ArraySize           SIZE
   EnableMSDTC         EDTC
   ProcOwner           POWNER
   ```

2. `.odbc.ini` ファイルのエントリを作成します。

   ```
   % iqdsn -or -y -w "MyOra2" -c
   "UID=system;PWD=manager;SID=QAORA"
   ```

   ```
   [MyOra2]
   Driver=/Sybase/IQ-16_0/lib64/libdboraodbc12_r.so
   UserID=system
   Password=manager
   SID=QAORA
   ```

Oracle データにアクセスするための環境変数の設定

SAP Sybase IQ サーバを起動して Oracle データにアクセスする前に、いくつかの環境変数を設定する必要があります。

Oracle にアクセスするための環境変数を設定します。

- ORACLE_HOME
  `setenv ORACLE_HOME`
- ODBCINI

```
setenv ODBCINI <location of .odbc.ini file with Oracle
entry>
```

- 使用するプラットフォームに応じたライブラリパス

プラットフォーム	コマンド
AIX	`setenv LIBPATH <path to platform-specific Oracle client directory> $LIBPATH`
その他の UNIX プラットフォーム	`setenv LD_LIBRARY_PATH <path to platform-specific Oracle client directory; $LD_LIBRARY_PATH`

SAP Sybase IQ サーバの起動

Oracle データにアクセスするフロントエンドとして使用する SAP Sybase IQ サーバを起動します。

```
start_iq -n myserver
```

Oracle データベースへの接続

SAP Sybase IQ を コンポーネント統合サービス 経由でリモート Oracle データに接続します。

前提条件

dbisql または **iqisql** にログインします。

手順

1. `.odbc.ini` ファイルのデータソース名を使用してサーバを作成します。
   ```
   CREATE SERVER myora CLASS 'oraodbc' USING 'MyOra2'
   ```
2. 外部ログインを作成します。
   ```
   CREATE EXTERNLOGIN DBA TO myora REMOTE LOGIN system
   IDENTIFIED BY manager
   ```
3. 接続を確認します。
   ```
   sp_remote_tables myora
   ```
4. Oracle データのテーブルを作成します。
   ```
   CREATE EXISTING TABLE my_oratable at
   'myora..system.oratable'
   ```
5. データを選んで、接続が機能していることを確認します。
   ```
   SELECT * FROM my_oratable
   ```

Oracle データベースへのアクセスのトラブルシューティング

Oracle データにアクセスした場合にエラーが返されたときは、該当する設定コンポーネントを確認します。

1. ドライバのロードエラー

   ドライバのロードエラーは、環境変数または設定情報ファイルの問題を示していることがあります。
2. 接続 ID の解決エラー

   接続 ID の解決エラーは、Oracle の定義、環境変数、または設定情報ファイルの問題であることがあります。

ドライバのロードエラー

ドライバのロードエラーは、環境変数または設定情報ファイルの問題を示していることがあります。

`Can't load driver` エラーが返された場合、次のことを確認します。

- `.odbc.ini` のエントリに含まれるドライバが正しいことを確認します。
- Oracle クライアントソフトウェアが LD_LIBRARY_PATH 定義に追加されていることを確認します。

接続 ID の解決エラー

接続 ID の解決エラーは、Oracle の定義、環境変数、または設定情報ファイルの問題であることがあります。

`ORA-12154:TNS:could not resolve the connect identifier` エラーが返された場合、次のことを確認します。

- Oracle の定義が正しいことを確認します。
- ORACLE_HOME が正しく設定されていることを確認します。
- `.odbc.ini` に入力されているゲートウェイシステム識別子 (SID) が正しいことを確認します。

ネイティブクラスなしでのリモートデータのロード

DirectConnect™ を使用してデータをロードします。

ネイティブクラスでは、次の場所にあるリモートデータソースへのアクセスに DirectConnect を使用します。

- 64 ビット UNIX プラットフォーム
- ODBC ドライバを使用できない 32 ビットプラットフォーム (Microsoft SQL Server など)

UNIX 上の SAP Sybase IQ サーバに MS SQL Server データをロード

このリモートデータの例では、UNIX 上の SAP Sybase IQ サーバに MS SQL Server データをロードしています。

この例では次の状況を想定します。

- *mssql* という名前の Enterprise Connect Data Access (ECDA) サーバが UNIX ホスト *myhostname*、ポート 12530 上にあります。
- データは、ホスト *myhostname*、ポート 1433 上の MS SQL サーバ (名前は *2000*) から取り出されます。

1. DirectConnect のマニュアルに従って、DirectConnect をデータソースに合わせて設定します。
2. ECDA サーバ (*mssql*) が SAP Sybase IQ の interfaces ファイルにリストされているかどうかを確認します。

```
mssql
master tcp ether myhostname 12530
query tcp ether myhostname 12530
```

3. *mssql* サーバのユーザ ID とパスワードを使用して、新しいユーザを追加します。

```
isql -Udba -Psql -Stst_iqdemo
grant connect to chill identified by chill
grant dba to chill
```

4. 新しいユーザとしてログインし、SAP Sybase IQ 上にローカルテーブルを作成します。

```
isql -Uchill -Pchill -Stst_iqdemo
create table billing(status char(1), name varchar(20), telno int)
```

5. データを挿入します。

```
insert into billing location 'mssql.pubs' { select * from
billing }
```

ネイティブクラスなしでデータのクエリを実行

ネイティブクラスなしでデータのクエリを実行する場合は、次のガイドラインに従います。

1. DirectConnect 経由で接続するために、ASE/CIS、リモートサーバ、プロキシを設定します。たとえば、Oracle サーバには DirectConnect for Oracle を使用します。
2. ASEJDBC クラスを Adaptive Server サーバに使用して、SAP Sybase IQ とリモートサーバを設定します (Adaptive Server 用の 64 ビット Unix ODBC ドライバが存在しないため、ASEODBC クラスは利用できません)。

3. **CREATE EXISTING TABLE** 文を使用してプロキシテーブルを作成します。そのプロキシテーブルが指す ASE 内のプロキシテーブルを介して、最終的に Oracle を指すようにします。

UNIX 上で DirectConnect とプロキシテーブルを使用して、リモートデータのクエリを実行

DirectConnect を使用してデータのクエリを実行します。

この例は、MS SQL Server データへのアクセス方法を示します。この例では次の状況を想定します。

- ホスト *myhostname*、ポート 7594 の上に SAP Sybase IQ サーバが存在しています。
- ホスト *myhostname*、ポート 4101 の上に Adaptive Server サーバが存在しています。
- *mssql* という名前の Enterprise Connect Data Access (ECDA) サーバがホスト *myhostname*、ポート 12530 上にあります。
- データは、ホスト *myhostname*、ポート 1433 上の MS SQL サーバ (名前は *2000*) から取り出されます。

MS SQL Server のクエリを実行できるよう Adaptive Server を設定

DirectConnect を介して MS SQL Server のクエリを実行できるよう、Adaptive Server とコンポーネント統合サービス (CIS) を設定します。

たとえば、サーバ名が *jones_1207* であるとします。

1. *mssql* に接続するためのエントリを Adaptive Server の interfaces ファイルに追加します。

```
mssql

master tcp ether hostname 12530

query tcp ether hostname 12530
```

2. ASE サーバでの CIS とリモートプロシージャコールの処理を有効にします。たとえば、CIS がデフォルトで有効になっているとします。

```
sp_configure 'enable cis'

Parameter Name Default Memory Used Config Value Run Value

enable cis                          1          0
1              1

(1 row affected)
(return status=0)

sp_configure 'cis rpc handling', 1

Parameter Name Default Memory Used Config Value Run Value
```

```
enable cis                          0          0
0              1
```

```
(1 row affected)
Configuration option changed. The SQL Server need not be restarted
since the option is dynamic.
```

3. Adaptive Server サーバの SYSSERVERS システムテーブルに DirectConnect サーバを追加します。

```
sp_addserver mssql, direct_connect, mssql
```

```
Adding server 'mssql', physical name 'mssql'
Server added.
(Return status=0)
```

4. Adaptive Server に接続するために SAP Sybase IQ で使用するユーザを Adaptive Server に作成します。

```
sp_addlogin tst, tsttst
```

```
Password correctly set.
Account unlocked. New login created.
(return status = 0)
```

```
grant role sa_role to tst
use tst_db
sp_adduser tst
```

```
New user added.
(return status = 0)
```

5. master データベースから外部ログインを追加します。

```
use master
sp_addexternlogin mssql, tst, chill, chill
```

```
User 'tst' will be known as 'chill' in remote server 'mssql'.
(return status = 0)
```

6. 目的のデータベースから追加されたユーザとして、ASE プロキシテーブルを作成します。

```
isql -Utst -Ttsttst
use test_db
create proxy_table billing_tst at 'mssql.pubs..billing'
select * from billing_tst
```

```
status                name                telno
------                ------------              -----
D                BOTANICALLY               1
B                BOTANICALL                2
(2 rows affected)
```

Adaptive Server サーバに接続できるよう SAP Sybase IQ を設定

Adaptive Server データのクエリを実行するには、次の手順に従います。

1. SAP Sybase IQ の interfaces ファイルにエントリを追加します。

```
jones_1207
master tcp ether jones 4101
query tcp ether jones 4101
```

2. Adaptive Server への接続に使用するユーザを作成します。

```
GRANT CONNECT TO tst IDENTIFIED BY tsttst
GRANT dba TO tst
```

3. 追加されたユーザとしてログインし、'asejdbc' サーバクラスを作成して外部ログインを追加します。

```
isql -Utst -Ptsttst -Stst_iqdemo
CREATE SERVER jones_1207 CLASS 'asejdbc' USING 'jones:4101/tst_db'
CREATE EXISTING TABLE billing_iq AT
'jones_1207.tst_db..billing_txt'
SELECT * from billing_iq
```

```
status                name          telno
------    -----------   -----
D         BOTANICALLY   1
B         BOTANICALL    2
(2 rows affected)
```

リモートサーバの削除

ISYSSERVER システムテーブルからリモートサーバを削除するには、**DROP SERVER** 文を使用します。

このアクションを正しく実行するには、そのサーバ上で定義されているすべてのリモートテーブルが削除済みであることが必要です。

例
次の文は RemoteSA という名前のサーバを削除します。

```
DROP SERVER RemoteSA;
```

参照：

- DROP SERVER 文 (796 ページ)

リモートサーバの変更

サーバの属性を変更するには、**ALTER SERVER** 文を使用します。これらの変更は、次にリモートサーバに接続するまで有効になりません。

ALTER SERVER 文を実行します。

次の文は、RemoteASE という名前のサーバのサーバクラスを aseodbc に変更します。この例では、サーバのデータソース名は RemoteASE です。

```
ALTER SERVER RemoteASE
CLASS 'aseodbc';
```

参照：

- ALTER SERVER 文 (769 ページ)

リモートサーバ上のテーブルのリスト (SQL の場合)

システムプロシージャを使用すると、リモートサーバ上の全テーブルの制限されたリスト、または包括的なリストを表示できます。

前提条件

なし。

手順

sp_remote_tables システムプロシージャを呼び出し、リモートサーバ上のテーブルのリストを返します。

@table_name または *@table_owner* を指定すると、テーブルのリストは一致するテーブルにのみ制限されます。

すべてのテーブルのリスト、またはテーブルの制限されたリストが返されます。

リモートサーバの機能

sp_servercaps システムプロシージャは、リモートサーバの機能に関する情報を表示します。SQL Anywhere はこの機能の情報を使用して、リモートサーバに渡すことができる SQL 文の量を判断します。

また、SYSCAPABILITY と SYSCAPABILITYNAME のシステムビューを問い合わせると、リモートサーバの機能情報も参照できます。これらのシステムビューは、SQL Anywhere が最初にリモートサーバに接続するまでは空の状態です。

sp_servercaps システムプロシージャを使用する場合は、*server-name* には CREATE SERVER 文で使用した *server-name* と同じ名前を指定してください。

次のように、sp_servercaps ストアドプロシージャを実行します。

```
CALL sp_servercaps('server-name');
```

外部ログイン

SAP Sybase IQ は、クライアントに代わってリモートサーバに接続するときに、それらのクライアントの名前とパスワードを使用します。ただし、外部ログインを作成すると、この動作を無効にすることができます。

外部ログインとは、リモートサーバとの通信時に使用される代替ログイン名とパスワードのことです。

SAP Sybase IQ がリモートサーバに接続するときに、**CREATE EXTERN LOGIN** でリモートログインが作成されており、**CREATE SERVER** 文でリモートサーバが定義さ

れている場合、**INSERT...LOCATION** は現在の接続のユーザ ID にリモートログインを使用できます。

リモートサーバが定義されていないか、現在の接続のユーザ ID に対するリモートログインが作成されていない場合、SAP Sybase IQ は現在の接続のユーザ ID とパスワードを使用して接続します。

注意： デフォルトのユーザ ID とパスワードを使用している場合に、ユーザがパスワードを変更した場合は、リモートサーバで新しいパスワードが有効になる前に、サーバを停止して、再起動する必要があります。**CREATE EXTERN LOGIN** を使用して作成されたリモートログインは、デフォルトユーザ ID のパスワードが変更されても、影響を受けません。

統合化ログインを使用する場合、SAP Sybase IQ クライアントの SAP Sybase IQ 名とパスワードは、SAP Sybase IQ のユーザ ID が `syslogins` にマッピングしたデータベースのログイン ID およびパスワードと同じです。

プロキシテーブル

リモートオブジェクトに対応するローカルの**プロキシテーブル**を作成すると、意識することなくリモートデータにアクセスできるようになります。プロキシテーブルを使用すると、リモートデータベースがプロキシテーブルの候補としてエクスポートする任意のオブジェクト (テーブル、ビュー、マテリアライズドビューを含む) にアクセスできます。プロキシテーブルを作成するには、次の文のいずれかを使用します。

- リモートサーバにすでにテーブルが存在する場合は、CREATE EXISTING TABLE 文を使用します。この文は、リモートサーバの既存のテーブルのプロキシテーブルを定義します。
- リモートサーバにテーブルが存在しない場合は、CREATE TABLE 文を使用します。この文はリモートサーバに新しいテーブルを作成して、そのテーブルのプロキシテーブルを定義します。

注意： セーブポイント内では、プロキシテーブルを変更することはできません。

プロキシテーブルでトリガを起動する場合は、プロキシテーブルの所有者の権限ではなく、トリガを起動するユーザの権限を使用します。

プロキシテーブルのロケーション

AT キーワードを CREATE TABLE 文と CREATE EXISTING TABLE 文とともに使用して、既存のオブジェクトのロケーションを定義します。ロケーション文字列は、ピリオドかセミコロンで区切られた 4 つの部分からなります。セミコロンデリミタを使用すると、データベースフィールドと所有者フィールドでファイル名と拡張子を使用できます。

AT 句の構文は次のようになります。

```
... AT 'server.database.owner.table-name'
```

- **server –** CREATE SERVER 文で指定されたもので、Adaptive Server Anywhere がサーバを識別する名前です。このフィールドはすべてのリモートデータソースに必須です。
- **database –** データベースフィールドの意味は、データソースによって異なります。このフィールドは使用しないで、空にしておく場合があります。ただし、その場合でもデリミタは必要です。

  データソースが Adaptive Server Enterprise の場合は、*database* によってテーブルを保管しているデータベースが指定されます。たとえば、master または pubs2 などです。

  データソースが SQL Anywhere の場合は、このフィールドは適用されません。入力しないでおきます。

  データソースが Excel、Lotus Notes、または Access の場合は、テーブルが保管されているファイルの名前を入力します。ファイル名にピリオドがある場合には、セミコロンデリミタを使用してください。
- **owner –** データベースが所有者の概念をサポートしている場合、所有者名を表します。このフィールドは、何人かの所有者が同じ名前でテーブルを所有する場合にだけ必要です。
- **table-name –** このフィールドはテーブルの名前を指定します。Excel スプレッドシートの場合、これはブックのシートの名前になります。*table-name* を入力しない場合、リモートテーブル名はローカルのプロキシテーブル名と同じであると見なされます。

例

以下はロケーション文字列の使用例です。

- SQL Anywhere：

```
'RemoteSA..GROUPO.Employees'
```

- Adaptive Server Enterprise：

```
'RemoteASE.pubs2.dbo.publishers'
```

- Excel：

```
'RemoteExcel;d:¥pcdb¥quarter3.xls;;sheet1$'
```

- Access：

```
'RemoteAccessDB;¥¥server1¥production¥inventory.mdb;;parts'
```

プロキシテーブルの作成 (SQL の場合)

Interactive SQL で CREATE TABLE 文または CREATE EXISTING TABLE 文を使用して、プロキシテーブルを作成できます。

前提条件

ユーザ本人が所有するプロキシテーブルを作成するには、CREATE PROXY TABLE システム権限が必要です。他のユーザが所有するプロキシテーブルを作成するには、CREATE ANY TABLE または CREATE ANY OBJECT のシステム権限が必要です。

手順

AT 句とともに CREATE TABLE 文を使用すると、リモートサーバに新しいテーブルを作成し、そのテーブルに対するプロキシテーブルをローカルサーバに作成します。カラムは SQL Anywhere のデータ型を使用して定義します。SQL Anywhere は、リモートサーバのネイティブの型にデータを自動的に変換します。

CREATE TABLE 文を使用してローカルとリモートの両方のテーブルを作成してから、引き続き DROP TABLE 文を使用してプロキシテーブルを削除すると、リモートテーブルも削除されます。ただし、DROP TABLE 文を使用して、CREATE EXISTING TABLE 文を使用して作成されたプロキシテーブルを削除できます。この場合、リモートテーブル名は削除されません。

CREATE EXISTING TABLE 文は、リモートサーバ上にある既存のテーブルにマッピングするプロキシテーブルを作成します。SQL Anywhere は、リモートロケーションのオブジェクトからカラム属性とインデックス情報を導出します。

1. ホストデータベースに接続します。
2. CREATE EXISTING TABLE 文を実行します。

プロキシテーブルが作成されます。

参照：

- CREATE EXISTING TABLE 文 (772 ページ)
- CREATE TABLE 文 (777 ページ)

リモートテーブルのカラムのリスト

CREATE EXISTING TABLE 文を実行する前に、リモートテーブルのカラムのリストを取得すると便利な場合があります。sp_remote_columns システムプロシージャは、リモートテーブルにあるカラムのリストとそのデータ型の説明を生成します。sp_remote_columns システムプロシージャの構文は次のとおりです。

```
CALL sp_remote_columns( @server_name, @table_name [, @table_owner [,
@table_qualifier ] ] )
```

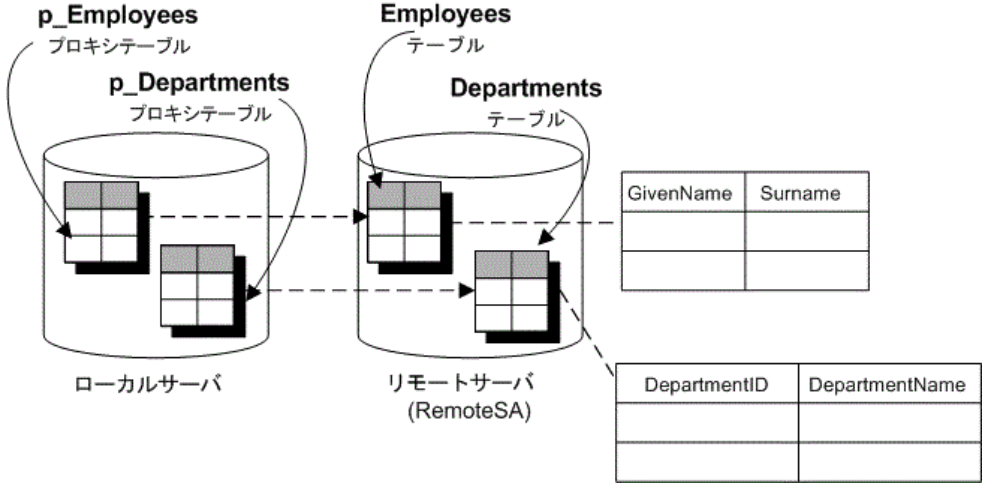
テーブル名、所有者、またはデータベース名を指定すると、カラムのリストはその指定に当てはまるものだけに限定されます。

たとえば、asetest という名前の Adaptive Server Enterprise サーバの production データベースにある sysobjects テーブルのカラムのリストを取得するには、次のように指定します。

```
CALL sp_remote_columns('asetest, 'sysobjects', null, 'production');
```

リモートテーブル間のジョイン

次の図は、リモートサーバ RemoteSA にある SQL Anywhere サンプルデータベースのリモートテーブル Employees と Departments にマッピングされている、ローカルデータベースサーバのプロキシテーブルを示しています。



異なる SQL Anywhere データベースのテーブル間にジョインを使用できます。次の例では、データベースを 1 つだけ使用した簡単なケースについて説明して、その仕組みを示します。

例

2 つのリモートテーブル間のジョインを実行します。

1. `empty.db` という名前の新しいデータベースを作成します。
   このデータベースにはデータがありません。このデータベースは、リモートオブジェクトを定義して、SQL Anywhere サンプルデータベースにアクセスするためだけに使用します。
2. `empty.db` を実行するデータベースサーバを起動します。これを行うには、次のコマンドを実行します。

```
iqsrv16 empty
```

3. Interactive SQL から DBA ユーザとして empty.db に接続します。
4. 新しいデータベースで、RemoteSA という名前のリモートサーバを作成します。このサーバのサーバクラスは SAODBC で、接続文字列は SQL Anywhere 16 Demo ODBC データソースを参照します。

```
CREATE SERVER RemoteSA
CLASS 'SAODBC'
USING 'SQL Anywhere 16 Demo';
```

5. この例では、ローカルデータベースと同じユーザ ID とパスワードをリモートデータベースで使用するので、外部ログインは必要ありません。
   場合によっては、リモートサーバのデータベースに接続するとき、ユーザ ID とパスワードを入力します。新しいデータベースの場合は、リモートサーバへの外部ログインを作成します。例では簡素化するために、ローカルのログイン名とリモートのユーザ ID はどちらも DBA とします。

```
CREATE EXTERNLOGIN DBA
TO RemoteSA
REMOTE LOGIN DBA
IDENTIFIED BY sql;
```

6. p_Employees プロキシテーブルを定義します。

```
CREATE EXISTING TABLE p_Employees
AT 'RemoteSA..GROUPO.Employees';
```

7. p_Departments プロキシテーブルを定義します。

```
CREATE EXISTING TABLE p_Departments
AT 'RemoteSA..GROUPO.Departments';
```

8. SELECT 文にプロキシテーブルを使用して、ジョインを実行します。

```
SELECT GivenName, Surname, DepartmentName
FROM p_Employees JOIN p_Departments
ON p_Employees.DepartmentID = p_Departments.DepartmentID
ORDER BY Surname;
```

複数のローカルデータベースのテーブル間のジョイン

SQL Anywhere サーバでは、複数のローカルデータベースを同時に稼働できます。他のローカル SQL Anywhere データベース内のテーブルをリモートテーブルとして定義することによって、データベース間のジョインを実行できます。

例

データベース db1 を使用しているときに、データベース db2 内のテーブルのデータにアクセスするとします。この場合は、データベース db2 のテーブルを示すプロキシテーブル定義を設定します。RemoteSA という名前の SQL Anywhere サーバ上で、db1、db2、db3 の 3 つのデータベースが使用可能だとします。

1. ODBC を使用している場合、アクセスするデータベースのそれぞれに ODBC データソース名を作成します。

2. ジョインの実行元のデータベースに接続します。たとえば db1 に接続します。
3. アクセスするその他のローカルデータベースのそれぞれに、CREATE SERVER 文を実行します。これによって、SQL Anywhere サーバへの**ループバック**接続が設定されます。

```
CREATE SERVER remote_db2
CLASS 'SAODBC'
USING 'RemoteSA_db2';
CREATE SERVER remote_db3
CLASS 'SAODBC'
USING 'RemoteSA_db3';
```

4. アクセスする他のデータベースにあるテーブルに CREATE EXISTING TABLE 文を実行して、プロキシテーブルの定義を作成します。

```
CREATE EXISTING TABLE Employees
AT 'remote_db2...Employees';
```

ネイティブ文とリモートサーバ

FORWARD TO 文を使用して、1 つ以上の文をネイティブの構文でリモートサーバに送信できます。この文は、2 つの方法で使用できます。

- 1 つの文をリモートサーバに送信する。
- SQL Anywhere をパススルーモードにして一連の文をリモートサーバに送信する。

FORWARD TO 文を使用して、サーバが正しく設定されていることを検証できます。リモートサーバに文を送信して、SQL Anywhere がエラーメッセージを返さなければ、リモートサーバは正しく設定されています。

プロシージャまたはバッチ内では FORWARD TO 文を使用できません。

指定したサーバに接続できない場合、メッセージがユーザに返されます。接続が確立された場合は、クライアントプログラムが認識できるフォームに結果が変換されます。

例 1

次の文は、バージョン文字列をセレクトすることによって、RemoteASE というサーバへの接続を検証します。

```
FORWARD TO RemoteASE {SELECT @@version};
```

例 2

次の文は、サーバ RemoteASE とのパススルーセッションを示します。

```
FORWARD TO RemoteASE;
    SELECT * FROM titles;
    SELECT * FROM authors;
FORWARD TO;
```

リモートプロシージャコール (RPC)

SAP Sybase IQ ユーザは、この機能をサポートしているリモートサーバへのプロシージャコールを発行することができます。

SAP Sybase IQ、SQL Anywhere、Adaptive Server、Oracle、DB2 はこの機能をサポートしています。リモートプロシージャコールの発行は、ローカルプロシージャコールの使用と類似しています。

リモートプロシージャの作成

管理者は、Interactive SQL でリモートプロシージャを作成できます。

前提条件

MANAGE REPLICATION システム権限が必要です。

手順

リモートプロシージャが結果セットを返すことができる場合は、たとえすべてのケースで結果セットを返せるわけではなくても、ローカルプロシージャ定義には RESULT 句を含めてください。

1. ホストデータベースに接続します。
2. プロシージャを定義する文を実行します。例：

```
CREATE PROCEDURE RemoteWho()
AT 'bostonase.master.dbo.sp_who'
```

リモートプロシージャを呼び出すときにパラメータを指定する例を次に示します。

```
CREATE PROCEDURE RemoteUser ( IN username CHAR( 30 ) )
AT 'bostonase.master.dbo.sp_helpuser';
CALL RemoteUser( 'joe' );
```

リモートトランザクション

リモートサーバが関与するトランザクションの管理には、*2 フェーズコミットプロトコル*を使用します。

SAP Sybase IQ は、ほとんどの場合においてトランザクションの整合性を保証します。

リモートトランザクション管理

リモートサーバに関連するトランザクションを管理する方法として、2 フェーズコミットプロトコルが使用されます。SQL Anywhere は、ほとんどの場合においてトランザクションの整合性を保証します。しかし、1 つのトランザクションで 2 つ

以上のリモートサーバが呼び出されるときには、分散した作業単位が未定の状態で残る可能性があります。2 フェーズコミットプロトコルを使用する場合でも、リカバリ処理は含まれません。

ユーザのトランザクションを管理する通常の論理は、次のようになっています。

1. SQL Anywhere は、BEGIN TRANSACTION 通知でリモートサーバの作業を開始します。
2. トランザクションのコミットの準備が整うと、SQL Anywhere は、トランザクションの一部であったリモートサーバのそれぞれに PREPARE TRANSACTION 通知を送信します。これによって、リモートサーバがトランザクションをコミットする準備が整っていることを確実にします。
3. PREPARE TRANSACTION 要求が失敗すると、すべてのリモートサーバは現在のトランザクションをロールバックするよう指示されます。
   PREPARE TRANSACTION 要求がすべて成功すると、サーバはトランザクションに関わるリモートサーバのそれぞれに、COMMIT TRANSACTION 要求を送信します。

BEGIN TRANSACTION によって開始すればどのようば文でも、トランザクションを開始できます。BEGIN TRANSACTION を明示しない場合は、SQL 文はリモートサーバに送信されて、1 つのリモートの作業単位として実行されます。

リモートトランザクションの制限

リモートトランザクション管理では、セーブポイントおよびネストされた文に制限があります。

トランザクション管理には、次のような制限があります。

- セーブポイントは、リモートサーバには伝達されません。
- リモートサーバに関わるトランザクションで、**BEGIN TRANSACTION** 文と **COMMIT TRANSACTION** 文の組がネストされている場合は、最も外側の組のみが処理されます。矛盾する **BEGIN TRANSACTION** 文と **COMMIT TRANSACTION** 文の組は、リモートサーバには送信されません。

内部操作

この項では、リモートサーバ上で SAP Sybase IQ がクライアントアプリケーションに代わって実行している基本手順について説明します。

クエリの解析

文は、クライアントから受信されると、データベースサーバによって解析されます。有効な SQL Anywhere の SQL 文でないと、エラーが発生します。

クエリの正規化

クエリの正規化では、参照されているオブジェクトを検証し、データ型の互換性をチェックします。

たとえば、次のようなクエリがあるとします。

```
SELECT *
FROM t1
WHERE c1 = 10
```

このクエリの正規化では、c1 カラムを持つテーブル t1 がシステムテーブルに存在することを検証します。 また、c1 カラムのデータ型が値 10 と合っているかを確認します。 たとえば、このカラムのデータ型が DATETIME であった場合、この文は拒否されます。

クエリの前処理

クエリの前処理では、クエリの最適化の準備をします。ここで SQL 文の表現が変更されることもあるので、SQL Anywhere がリモートサーバに引き渡す実際の SQL 文は、セマンティック上は同じであっても構文が元のものと同じとはかぎりません。

前処理は、ビューによって参照されるテーブルでクエリが操作できるよう、ビューの拡張を行います。処理を効率化するため、式が並べ替えられ、サブクエリが変更される場合があります。たとえば、いくつかのサブクエリがジョインに変換される場合があります。

文の完全なパススルー

効率性を考慮して、SQL Anywhere では、文に含まれるできるだけ多くの要素をリモートサーバに渡します。多くの場合、これは元々 SQL Anywhere に指定された完全な文です。

SQL Anywhere は、次のような場合に完全な文を渡します。

- 文内のテーブルがどれも同じリモートサーバに存在している。
- リモートサーバが文内のすべての構文を処理できる。

まれに、リモートサーバが作業を行うよりも、SQL Anywhere がいくつかの作業を行った方が効率が良い場合があります。たとえば、SQL Anywhere のソートアルゴリズムの方が優れていることがあります。この場合は、ALTER SERVER 文を使用して、リモートサーバの機能の変更を検討します。

文の部分的なパススルー

ある文に複数のサーバへの参照が含まれている場合、またはリモートサーバではサポートされていない SQL 機能を文が使用している場合は、クエリは複数の単純な部分に分解されます。

SELECT

引き渡すことのできない部分が取り除かれながら、SELECT 文は分解されていきます。取り除かれた部分の処理は SQL Anywhere によって実行されます。たとえば、次の文内にある ATAN2 関数をリモートサーバが処理できないとします。

```
SELECT a,b,c
WHERE ATAN2( b, 10 ) > 3
AND c = 10;
```

リモートサーバに送信される文は、次のように変換されます。

```
SELECT a,b,c WHERE c = 10;
```

次に、SQL Anywhere がローカルで `WHERE ATAN2( b, 10 ) > 3` を中間結果セットに適用します。

ジョイン

文に複数の場所にあるテーブル間のジョインが含まれている場合、IQ は連結されたテーブルのジョインを、それらが配置されているサーバにプッシュしようと試みます。そのジョインの結果は、IQ によって他のリモートテーブルまたはローカルテーブルの結果とジョインされます。IQ は、常にできるだけ多くのジョイン操作をリモートサーバにプッシュしようとします。IQ がリモートテーブルをローカル IQ テーブルとジョインする場合、IQ はサポートしているジョインアルゴリズムを使用することができます。

アルゴリズムの選択はコストの推定に基づきます。これらのアルゴリズムには、ネストされたループ、ハッシュ、またはソートマージのジョインを含めることができます。

IQ とリモートテーブルの間でネストループのジョインが選択された場合、リモートテーブルはできるだけジョインの一番外側のテーブルにされます。これは、ネットワーク I/O のコストが高いことにより、通常はリモートテーブルに対する参照の方がローカルテーブルよりコストがかさむためです。

UPDATE と DELETE

条件に合うローが検出されたとき、SQL Anywhere が UPDATE 文または DELETE 文全体をリモートサーバに渡すことができない場合は、元の WHERE 句のできるだけ多くの部分を持つ SELECT 文に文を変更して、WHERE CURRENT OF *cursor-name* を指定する位置付け UPDATE 文または DELETE 文を続けます。

たとえば、リモートサーバが関数 ATAN2 をサポートしていないとします。

```
UPDATE t1
SET a = atan2( b, 10 )
WHERE b > 5;
```

この文は次のように変換されます。

```
SELECT a,b
FROM t1
WHERE  b > 5;
```

ローが検出されるたびに、SQL Anywhere は a の新しい値を計算して、次の文を実行します。

```
UPDATE t1
SET a = 'new value'
WHERE CURRENT OF CURSOR;
```

new value と等しい値が a にある場合、位置付け UPDATE は必要なく、リモートに送信されません。

テーブルスキャンを必要とする UPDATE 文または DELETE 文を処理するには、位置付け UPDATE または DELETE (WHERE CURRENT OF *cursor-name*) を実行する機能をリモートのデータソースがサポートしていなければなりません。データソースによってはこの機能をサポートしていません。

注意： テンポラリテーブルは更新できない

中間テンポラリテーブルが必要な場合は、UPDATE または DELETE を実行できません。これは ORDER BY を持つクエリと、サブクエリを持つクエリのいくつかで発生します。

リモートデータアクセスのトラブルシューティング

ここでは、リモートサーバにアクセスするときのトラブルシューティングのヒントについて説明します。

リモートデータに対してサポートされない機能

次の SQL Anywhere 機能はリモートデータではサポートされていません。

- リモートテーブルでの ALTER TABLE 文
- プロキシテーブルに定義されたトリガ
- リモートテーブルを参照する外部キー
- READTEXT 関数、WRITETEXT 関数、TEXTPTR 関数
- 位置付け UPDATE 文と DELETE 文
- 中間テンポラリテーブルを必要とする UPDATE 文と DELETE 文

- リモートデータに対して開かれたカーソルの後方スクロール。フェッチ文は NEXT または RELATIVE 1 でなければなりません。
- プロキシテーブルを参照する式を含む関数の呼び出し
- リモートテーブルのカラムにリモートサーバに関するキーワードがある場合、そのカラムのデータにはアクセスできない。CREATE EXISTING TABLE 文を実行して定義をインポートすることはできますが、そのカラムを選択することはできません。

大文字と小文字の区別

SQL Anywhere データベースの大文字と小文字の区別の設定は、アクセスするリモートサーバが使用する設定に合わせてください。

デフォルトでは、SQL Anywhere データベースは大文字と小文字を区別しないで作成されます。この設定では、大文字と小文字を区別するデータベースから選択を行ったときに、予期しない結果が発生することがあります。ORDER BY または文字列比較がリモートサーバに送信されるか、それともローカルの SQL Anywhere によって評価されるかによって、発生する結果が異なります。

接続のテスト

次の手順で、リモートサーバに接続できることを確認します。

- Interactive SQL などのクライアントツールを使用してリモートサーバに接続できることを確認してから、SQL Anywhere を設定します。
- リモートサーバに対して簡単なパススルー文を実行して、接続とリモートログインの設定を確認します。次に例を示します。

  ```
  FORWARD TO RemoteSA {SELECT @@version};
  ```
- リモートサーバとの対話をトレースするために、リモートトレーシングを有効にします。次に例を示します。

  ```
  SET OPTION cis_option = 7;
  ```

  リモートトレーシングを有効にすると、データベースサーバのメッセージウィンドウにトレーシング情報が表示されます。この出力をファイルに記録するには、データベースサーバの起動時に -o サーバオプションを指定します。

ODBC を使用したリモートデータアクセスの接続

ODBC を介してリモートデータベースにアクセスする場合、リモートサーバへの接続には名前が付けられます。名前を使用して接続を削除し、リモート要求をキャンセルできます。

接続の名前は、ASACIS_*conn-name* のようになります。*conn-name* は、ローカル接続の接続 ID です。接続 ID は、sa_conn_info ストアドプロシージャから取得できます。

マルチプレックスサーバでのリモートデータアクセス

セカンダリサーバの新しいプロキシサーバにアクセスしたときに、タイミングによってはサーバが緊急シャットダウンする場合があります。

サーバの緊急シャットダウンが発生した場合は、新しいプロキシテーブルを使用する前に、サーバに再接続するか、しばらく待ってから新しいトランザクションを起動する必要があります。

付録： SQL リファレンス

このマニュアル内のタスクで使用される SQL 文のリファレンス資料。

ALTER SERVER 文

リモートサーバの属性を変更します。**ALTER SERVER** による変更は、次にリモートサーバに接続するまで有効になりません。

クイックリンク：

「パラメータ」 (769 ページ)

「例」 (771 ページ)

「使用法」 (771 ページ)

「標準」 (771 ページ)

「パーミッション」 (771 ページ)

構文

```
ALTER SERVER server-name
    [ CLASS 'server-class' ]
    [ USING 'connection-info' ]
    [ CAPABILITY 'cap-name' { ON | OFF } ]
    [ CONNECTION CLOSE [ CURRENT | ALL | connection-id ] ]

server-class – （構文に戻る）
    { ASAJDBC
        | ASEJDBC
        | SAODBC
        | ASEODBC
        | DB2ODBC
        | MSSODBC
        | ORAODBC
        | ODBC }

connection-info – （構文に戻る）
    { machine-name:port-number [ /dbname ] | data-source-name }
```

パラメータ

（先頭に戻る） (769 ページ)

- **cap-name –** サーバ機能の名前

- **CLASS –** サーバクラスを変更します。
- **USING –** JDBC ベースのサーバクラスを使用する場合、USING 句は *hostname:port-number [/dbname]* となります。
  - **hostname –** リモートサーバを実行するマシン。
  - **portnumber –** リモートサーバが受信する TCP/IP ポート番号。SAP Sybase IQ および SAP Sybase SQL Anywhere® のデフォルトのポート番号は 2638 です。
  - **dbname –** SQL Anywhere リモートサーバでは、*dbname* を指定しない場合、デフォルトのデータベースが使用されます。Adaptive Server の場合は、デフォルトが master データベースです。*dbname* を使用しない場合は、他の方法 (**FORWARD TO** 文など) を使って別のデータベースを指定します。

  ODBC ベースのサーバクラスを使用する場合、USING 句は *data-source-name* になります。これは ODBC データソース名です。
- **CAPABILITY –** サーバの機能の ON と OFF を切り替えます。サーバの機能は、システムテーブル SYSCAPABILITY に格納されています。サーバ機能の名前は、システムテーブル SYSCAPABILITYNAME に格納されています。サーバへの最初の接続が確立されるまで、SYSCAPABILITY テーブルには、リモートサーバのエントリは含まれません。最初の接続時に、SAP Sybase IQ はサーバに対して機能に関する問い合わせを行って、SYSCAPABILITY に結果を格納します。後続の接続では、このテーブルからサーバの機能が取得されます。

  通常、サーバの機能を変更する必要はありません。クラス ODBC の汎用サーバの機能を変更しなければならない場合はあります。
- **CONNECTION CLOSE –** ユーザがリモートサーバへの接続を確立した場合、ユーザがローカルデータベースから接続を切断しないかぎり、リモート接続は閉じられません。CONNECTION CLOSE 句を使用すると、リモートサーバへの接続を明示的に閉じることができます。この句は、リモート接続が非アクティブまたは不要になった場合に役立ちます。

  次の SQL 文は互いに同じであり、リモートサーバへの現在の接続を閉じるために使用されます。

```
ALTER SERVER server-name CONNECTION CLOSE
ALTER SERVER server-name CONNECTION CLOSE CURRENT
```

  この構文を使用すると、リモートサーバへの ODBC 接続と JDBC 接続の両方を閉じることができます。これらの文の実行に SERVER OPERATOR システム権限は不要です。

  また、接続 ID を指定して特定のリモート ODBC 接続を切断することも、ALL キーワードを指定してすべてのリモート ODBC 接続を切断することもできます。接続 ID または ALL キーワードを指定して JDBC 接続を閉じようとすると、

エラーが発生します。*connection-id* で指定された接続が現在のローカル接続ではない場合、接続を閉じるにはユーザに SERVER OPERATOR システム権限が必要です。

例

（先頭に戻る） (769 ページ)

- **例 1 –** 名前が `ase_prod` である Adaptive Server サーバのサーバクラスを変更して、SAP Sybase IQ への接続が ODBC ベースになるようにします。データソース名は `ase_prod` です。

```
ALTER SERVER ase_prod
CLASS 'ASEODBC'
USING 'ase_prod'
```

- **例 2 –** サーバ `infodc` の機能を変更します。

```
ALTER SERVER infodc
CAPABILITY 'insert select' OFF
```

- **例 3 –** リモートサーバ `rem_test` への接続をすべて閉じます。

```
ALTER SERVER rem_test
CONNECTION CLOSE ALL
```

- **例 4 –** 接続 ID 142536 を使用するリモートサーバ `rem_test` への接続を閉じます。

```
ALTER SERVER rem_test
CONNECTION CLOSE 142536
```

使用法

（先頭に戻る） (769 ページ)

関連する動作：

- オートコミット

標準

（先頭に戻る） (769 ページ)

- SQL - ISO/ANSI SQL 文法のベンダ拡張。
- SAP Sybase Database 製品 - Open Client/Open Server でサポートされています。

パーミッション

（先頭に戻る） (769 ページ)

SERVER OPERATOR システム権限が必要です。

CREATE EXISTING TABLE 文

リモートサーバ上の既存のテーブルを表す新しいプロキシテーブルを作成します。

クイックリンク：

「パラメータ」へ移動 (772 ページ)

「例」へ移動 (773 ページ)

「使用法」へ移動 (774 ページ)

「標準」へ移動 (774 ページ)

「パーミッション」へ移動 (775 ページ)

構文

```
CREATE EXISTING TABLE [owner.]table_name
    [ ( column-definition, … ) ]
    AT 'location-string'

column-definition - (back to Syntax)
    column-name data-type [ NOT NULL ]

location-string - (back to Syntax)
    remote-server-name.[db-name].[owner].object-name | remote-server-
name;[db-name];[owner];object-name
```

パラメータ

（先頭に戻る） (772 ページ)

- **column-definition –** カラム定義を指定しない場合、SAP Sybase IQ は、リモートテーブルから取得するメタデータからカラムリストを導出します。カラム定義を指定した場合、SAP Sybase IQ は、そのカラム定義を検証します。SAP Sybase IQ では、カラム名、データ型、長さ、NULL プロパティについて、次の点をチェックします。
  - カラム名が一致しなければなりません (大文字小文字は無視されます)。
  - **CREATE EXISTING TABLE** 文のデータ型は、リモートロケーションのカラムのデータ型と一致するか、またはそのデータ型に変換可能でなければなりません。たとえば、ローカルカラムのデータ型が NUMERIC として定義されているのに対して、リモートカラムのデータ型が MONEY である場合があります。データ型が一致しないテーブル、またはその他の不整合が存在するテーブルから選択した場合、エラーが発生する可能性があります。

- 各カラムの NULL プロパティがチェックされます。ローカルカラムの NULL プロパティがリモートカラムの NULL プロパティと同じでない場合、警告メッセージが出力されますが、文はアボートしません。
- 各カラムの長さがチェックされます。CHAR、VARCHAR、BINARY、DECIMAL、NUMERIC の各カラムの長さが一致しない場合は、警告メッセージが出力されますが、コマンドはアボートしません。**CREATE EXISTING** 文には、実際のリモートカラムリストのサブセットだけをインクルードできます。

- **AT –** リモートオブジェクトのロケーションを指定します。AT 句は、デリミタとしてセミコロン (;) をサポートします。セミコロンがロケーション文字列のどこかにある場合、そのセミコロンはフィールドデリミタです。セミコロンがない場合は、ピリオドがフィールドデリミタです。ピリオドを使用すると、データベースおよび所有者の各フィールドにファイル名と拡張子を使用できます。セミコロンのフィールドデリミタは、現在サポートされていないサーバクラスで主に使用されていますが、ピリオドもフィールドデリミタとして機能する状況ではセミコロンも使用できます。

  たとえば、次の文は、テーブル proxy_a1 をリモートサーバ myasa の SQL Anywhere データベース mydb にマッピングします。

```
CREATE EXISTING TABLE
proxy_a1
AT 'myasa;mydb;;a1'
```

例

（先頭に戻る） (772 ページ)

- **例 1 –** リモートサーバ server_a にある nation テーブルのプロキシテーブル nation を作成します。

```
CREATE EXISTING TABLE nation
( n_nationkey int,
  n_name char(25),
  n_regionkey int,
  n_comment char(152))
AT 'server_a.db1.joe.nation'
```

- **例 2 –** リモートサーバ server_a にある blurbs テーブルのプロキシテーブル blurbs を作成します。SAP Sybase IQ は、リモートテーブルから取得したメタデータからカラムリストを導出します。

```
CREATE EXISTING TABLE blurbs
AT 'server_a.db1.joe.blurbs'
```

- **例 3 –** SAP Sybase IQ リモートサーバ remote_iqdemo_srv にある Employees テーブルのプロキシテーブル rda_employee を作成します。

```
CREATE EXISTING TABLE rda_employee
AT 'remote_iqdemo_srv..dba.Employees'
```

使用法

（先頭に戻る） (772 ページ)

CREATE EXISTING TABLE は **CREATE TABLE** 文の変形です。EXISTING キーワードを **CREATE TABLE** で使用すると、テーブルがすでにリモートに存在していて、そのメタデータが SAP Sybase IQ にインポートされるように指定できます。これにより、リモートテーブルはユーザに対して可視なエンティティとして作成されます。SAP Sybase IQ はテーブルを作成する前に、テーブルが外部の場所に存在することを確認します。

プロキシテーブルとして使用するテーブルに、30 文字よりも長い名前を付けることはできません。

オブジェクト (ホストデータファイルまたはリモートサーバオブジェクトのどちらかとして) が存在しない場合、この文は拒否されてエラーメッセージが出力されます。

ホストデータファイルまたはリモートサーバテーブルのインデックス情報が抽出されて、システムテーブル sysindexes のローを作成するために使用されます。これにより、サーバ用語のインデックスとキーが定義されて、クエリオプティマイザがこのテーブルに存在する可能性のあるすべてのインデックスを考慮できるようになります。

参照整合性制約は、必要に応じてリモートロケーションに渡されます。

シンプレックス環境では、同じノード上でリモートテーブルを参照するプロキシテーブルを作成することはできません。マルチプレックス環境では、マルチプレックス内で定義されたリモートテーブルを参照するプロキシテーブルを作成することはできません。

たとえば、シンプレックス環境で、同じノード上で定義されたベーステーブル `Employees` を参照するプロキシテーブル `proxy_e` を作成しようとすると、**CREATE EXISTING TABLE** 文が拒否され、エラーメッセージが表示されます。マルチプレックス環境では、マルチプレックス内で定義されたリモートテーブル `Employees` を参照する任意のノード (コーディネータまたはセカンダリ) からプロキシテーブル `proxy_e` を作成する場合に、**CREATE EXISTING TABLE** 文が拒否されます。

標準

（先頭に戻る） (772 ページ)

- SQL - ISO/ANSI SQL 準拠。

- SAP Sybase Database 製品 - Open Client/Open Server でサポートされています。

パーミッション

（先頭に戻る） (772 ページ)

自分が所有するテーブルの場合、次のいずれかが必要です。

- CREATE ANY TABLE システム権限
- CREATE ANY OBJECT システム権限

任意のユーザが所有するテーブルの場合、CREATE ANY TABLE システム権限が必要です。

CREATE SERVER 文

サーバを ISYSSERVER テーブルに追加します。

クイックリンク：

「パラメータ」 (776 ページ)

「例」 (776 ページ)

「使用法」 (776 ページ)

「標準」 (777 ページ)

「パーミッション」 (777 ページ)

構文

```
CREATE SERVER server-name
    CLASS 'server-class'
    USING 'connection-info'
    [ READ ONLY ]

server-class – （構文に戻る）
    { ASAJDBC
        | ASEJDBC
        | SAODBC
        | ASEODBC
        | DB2ODBC
        | MSSODBC
        | ORAODBC
        | ODBC }

connection-info – （構文に戻る）
    { machine-name:port-number [ /dbname ] | data-source-name }
```

パラメータ

（先頭に戻る） (775 ページ)

- **USING –** JDBC ベースのサーバクラスを使用する場合、USING 句は *hostname:port-number [/dbname]* となります。
  - **hostname –** リモートサーバを実行するマシン。
  - **portnumber –** リモートサーバが受信する TCP/IP ポート番号。SAP Sybase IQ および SAP Sybase SQL Anywhere® のデフォルトのポート番号は 2638 です。
  - **dbname –** SQL Anywhere リモートサーバでは、*dbname* を指定しない場合、デフォルトのデータベースが使用されます。Adaptive Server の場合は、デフォルトが master データベースです。*dbname* を使用しない場合は、他の方法 (**FORWARD TO** 文など) を使って別のデータベースを指定します。

  ODBC ベースのサーバクラスを使用する場合、USING 句は *data-source-name* になります。これは ODBC データソース名です。
- **READ ONLY –** リモートサーバが読み込み専用データソースであることを指定します。更新の要求は SAP Sybase IQ によって拒否されます。

例

（先頭に戻る） (775 ページ)

- **例 1 –** JDBC ベースの Adaptive Server サーバ用のリモートサーバを `ase_prod` という名前で作成します。このマシン名は "banana" でポート番号は 3025 です。

```
CREATE SERVER ase_prod
CLASS 'asejdbc'
USING 'banana:3025'
```

- **例 2 –** "apple" というマシンに、名前が `testasa` で、ポート番号 2638 で受信する SQL Anywhere のリモートサーバを作成します。

```
CREATE SERVER testasa
CLASS 'asajdbc'
USING 'apple:2638'
```

- **例 3 –** Oracle サーバ用の `oracle723` という名前のリモートサーバを作成します。この ODBC データソース名は "oracle723" です。

```
CREATE SERVER oracle723
CLASS 'oraodbc'
USING 'oracle723'
```

使用法

（先頭に戻る） (775 ページ)

CREATE SERVER 文は、SAP Sybase IQ カタログからリモートサーバを定義します。

関連する動作

- オートコミット

標準

（先頭に戻る） (775 ページ)

- SQL - ISO/ANSI SQL 準拠。
- SAP Sybase Database 製品 - Open Client/Open Server でサポートされています。

パーミッション

（先頭に戻る） (775 ページ)

SERVER OPERATOR システム権限が必要です。

CREATE TABLE 文

データベースまたはリモートサーバに新しいテーブルを作成します。

クイックリンク：

「パラメータ」 (779 ページ)

「例」 (790 ページ)

「使用法」 (793 ページ)

「標準」 (794 ページ)

「パーミッション」 (795 ページ)

構文

```
CREATE [ { GLOBAL | LOCAL } TEMPORARY ] TABLE
    [ IF NOT EXISTS ] [ owner. ]table-name
    ... ( column-definition [ column-constraint ] ...
    [ , column-definition [ column-constraint ] ...]
    [ , table-constraint ] ... )
    |{ ENABLE | DISABLE } RLV STORE

    ...[ IN dbspace-name ]
    ...[ ON COMMIT { DELETE | PRESERVE } ROWS ]
    [ AT location-string ]
    [PARTITION BY
       range-partitioning-scheme
       | hash-partitioning-scheme
       | composite-partitioning-scheme ]
```

```
column-definition – (back to Syntax)
    column-name data-type
      [ [ NOT ] NULL ]
      [ DEFAULT default-value | IDENTITY ]
      [ PARTITION | SUBPARTITION ( partition-name IN  dbspace-name
[ , ... ] ) ]

default-value – (back to column-definition)
    special-value
    | string
    | global variable
    | [ - ] number
    | ( constant-expression )
    | built-in-function( constant-expression )
    | AUTOINCREMENT
    | CURRENT DATABASE
    | CURRENT REMOTE USER
    | NULL
    | TIMESTAMP
    | LAST USER

special-value – (back to default value)
    CURRENT
    { DATE
    | TIME
    | TIMESTAMP
    | USER
    | PUBLISHER }
    | USER

column-constraint – (back to Syntax)
    [ CONSTRAINT constraint-name ] {
      { UNIQUE
          | PRIMARY KEY
          | REFERENCES table-name [ ( column-name ) ] [ action ]
        }
        [ IN dbspace-name ]
        | CHECK ( condition )
        | IQ UNIQUE ( integer )
     }

table-constraint – (back to Syntax)
    [ CONSTRAINT constraint-name ]
    {  { UNIQUE ( column-name [ , column-name ] … )
       | PRIMARY KEY ( column-name [ , column-name ] … )
       }
    [ IN dbspace-name ]
       | foreign-key-constraint
       | CHECK ( condition )
       | IQ UNIQUE ( integer )
    }

foreign-key-constraint – (back to table-constraint)
    FOREIGN KEY [ role-name ] [ ( column-name [ , column-name ] … ) ]
    …REFERENCES table-name [ ( column-name [ , column-name ] … ) ]
```

```
   …[ actions ] [ IN dbspace-name ]

actions – (back to foreign-key-constraint)
   [ ON { UPDATE | DELETE } RESTRICT ]

location-string – (back to Syntax) or (back to composite-partitioning-
scheme)
   { remote-server-name. [ db-name ].[ owner ].object-name
      | remote-server-name; [ db-name ]; [ owner ];object-name }

range-partitioning-scheme – (back to Syntax)
   RANGE ( partition-key ) ( range-partition-decl [,range-partition-decl ... ] )

partition-key – (back to range-partitioning-scheme) or (back to hash-
partitioning-scheme)
   column-name

range-partition-decl – (back to range-partitioning-scheme)
   VALUES <= ( {constant-expr
      | MAX } [ , { constant-expr
      | MAX }]... )
   [ IN dbspace-name ]

hash-partitioning-scheme – (back to Syntax) or (back to composite-
partitioning-scheme)
   HASH ( partition-key [ , partition-key, … ] )

composite-partitioning-scheme – (back to Syntax)
   hash-partitioning-scheme SUBPARTITION range-partitioning-scheme
```

パラメータ

（先頭に戻る） (777 ページ)

- **IN –** column-definition 句、column-constraint 句、table-constraint 句、foreign-key 句、および partition-decl 句で、オブジェクトが作成される DB 領域を指定するために使用します。IN 句を省略した場合、SAP Sybase IQ はテーブルが割り当てられている DB 領域にオブジェクトを作成します。

  この句で SYSTEM を指定し、永久テーブルまたはテンポラリテーブルをカタログストアに置くことができます。IQ_SYSTEM_TEMP を指定すると、テンポラリなユーザオブジェクト (テーブル、パーティション、またはテーブルインデックス) を IQ_SYSTEM_TEMP に格納できます。または、**TEMP_DATA_IN_SHARED_TEMP** オプションが 'ON' に設定されており、IQ_SHARED_TEMP DB 領域に RW ファイルが含まれている場合は、IQ_SHARED_TEMP に格納できます (IN 句を IQ_SHARED_TEMP とともに指定することはできません)。これ以外の IN 句の使用はすべて無視されます。デフォルトでは、すべての永久テーブルはメイン IQ ストアに、すべてのテンポラリテーブルはテンポラリ IQ ストアに配置されます。グローバルテンポラリテー

ブルとローカルテンポラリテーブルを IQ ストアに置くことは絶対にできません。

以下の構文はサポートされていません。

```
CREATE LOCAL TEMPORARY TABLE tab1(c1 int) IN IQ_SHARED_TEMP
```

BIT データ型のカラムは DB 領域に明示的に配置することはできません。以下は BIT データ型に対してサポートされていません。

```
CREATE TABLE t1(c1_bit bit IN iq_main);
```

- **ON COMMIT** – テンポラリテーブルに対してのみ使用できます。デフォルトで、テンポラリテーブルのローは COMMIT のときに削除されます。
- **AT** – location-string 句で指定されたリモートロケーションにマップするプロキシテーブルを作成します。プロキシテーブル名は最大で 30 文字までです。`AT` 句は、デリミタとしてセミコロン (;) をサポートします。セミコロンが location-string 句のどこかにある場合、そのセミコロンはフィールドデリミタです。セミコロンがない場合は、ピリオドがフィールドデリミタです。これにより、データベースフィールドと所有者フィールドでファイル名と拡張子を使用できます。

  セミコロンのフィールドデリミタは、現在サポートされていないサーバクラスで主に使用されていますが、ピリオドもフィールドデリミタとして機能する状況ではセミコロンも使用できます。たとえば、次の文は、テーブル `proxy_a` をリモートサーバ `myasa` の SQL Anywhere データベース `mydb` にマッピングします。

  ```
  CREATE TABLE proxy_a1
  AT 'myasa;mydb;;a1'
  ```

  外部キー定義は、リモートテーブルでは無視されます。リモートテーブルを参照するローカルテーブルの外部キー定義も無視されます。プライマリキー定義は、サーバがプライマリキーをサポートする場合、リモートサーバに送信されます。

  シンプレックス環境では、同じノード上でリモートテーブルを参照するプロキシテーブルを作成することはできません。マルチプレックス環境では、マルチプレックス内で定義されたリモートテーブルを参照するプロキシテーブルを作成することはできません。
- **IF NOT EXISTS** – 指定したオブジェクトがすでに存在する場合、変更は行われず、エラーは返されません。
- **{ ENABLE | DISABLE } RLV STORE** – このテーブルをインメモリリアルタイム更新の対象として RLV ストアに登録します。IQ テンポラリテーブルはサポートされていません。この値は、データベースオプション **BASE_TABLES_IN_RLV** の値よりも優先されます。この値を ENABLE に設定するには、CREATE

TABLE システム権限と、RLV ストアの DB 領域に対する CREATE パーミッションが必要です。

- **column-definition –** テーブルカラムを定義します。使用可能なデータ型については、『リファレンス：ビルディングブロック、テーブル、およびプロシージャ』の「SQL データ型」を参照してください。同じテーブル内の 2 つのカラムが同じ名前を持つことはできません。最大 45,000 のカラムを作成可能ですが、1 つのテーブルに 10,000 を超えるカラムを作成すると、パフォーマンスの低下を招くおそれがあります。
  - **[NOT] NULL] –** NULL 値を含めるか、除外するかを設定します。NOT NULL を指定した場合や、カラムが UNIQUE 制約または PRIMARY KEY 制約を受ける場合は、カラムに NULL 値を含めることができません。NULL を許可するカラム数のテーブルごとの制限は、約 8*(database-page-size - 30) です。
  - **DEFAULT default-value –** CREATE TABLE (および ALTER TABLE) 文の DEFAULT キーワードでカラムのデフォルト値を指定します。DEFAULT 値は、カラムの値を指定しない INSERT (または LOAD) 文でカラムの値として使用されます。
  - **DEFAULT AUTOINCREMENT –** DEFAULT AUTOINCREMENT カラムの値は、テーブル内の各ローをユニークに識別します。この種のカラムは、Adaptive Server との互換性を考慮して IDENTITY カラムとも呼ばれます。IDENTITY/DEFAULT AUTOINCREMENT カラムには、挿入や更新の際に自動的に生成される連続した数値が格納されます。IDENTITY/DEFAULT AUTOINCREMENT を使用する場合、カラムは整数データ型のいずれか、または真数型で、位取りを 0 にする必要があります。カラムの値は NULL でもかまいません。テーブル名は、所有者の名前で修飾して指定する必要があります。

    ON でテーブルへの挿入を行います。IDENTITY/DEFAULT AUTOINCREMENT カラムへの値が指定されなければ、カラム内のどの値よりも大きいユニークな値が生成されます。カラムへ格納する値を INSERT に指定すれば、その値が使用されます。指定した値がそのカラムの現在の最大値よりも小さい場合、指定した値は後続の挿入に対して生成する値の起点として使用されます。

    ローを削除しても IDENTITY/AUTOINCREMENT カウンタはデクリメントされません。ローの削除によって作成されたギャップは、挿入を行うときに明示的に割り当てることによってのみ埋めることができます。IDENTITY/AUTOINCREMENT カラムへの挿入を実行するには、データベースオプション IDENTITY_INSERT をテーブル名に設定する必要があります。

    以下に、IDENTITY カラムを持つテーブルを作成し、データを明示的に追加する例を示します。

```
CREATE TABLE mytable(c1 INT IDENTITY);
SET TEMPORARY OPTION IDENTITY_INSERT = "DBA".mytable;
INSERT INTO mytable VALUES(5);
```

最大値よりも小さいロー番号を明示的に挿入すると、後続のローで明示的に割り当てなくても、その最大値より 1 大きい値に自動的にインクリメントされます。

カラムに直前に挿入された値は、グローバル変数 @@identity を調べることによって確認できます。

- **IDENTITY –** AUTOINCREMENT デフォルトを使用する代わりの Transact-SQL® 互換の代替手段です。SAP Sybase IQ では、IDENTITY 句または DEFAULT AUTOINCREMENT 句のどちらかを使用して IDENTITY カラムを作成できます。

- **table-constraint –** データベース内のデータの整合性を保証します。整合性制約には次の 4 つのタイプがあります。

  - **UNIQUE –** 1 つまたは複数のカラムによってテーブル内の各ローがユニークに識別されるように指定します。 テーブル内の 2 つのローは、指定されたすべてのカラム中に同じ値を持つことはできません。1 つのテーブルに複数の一意性制約が存在することがあります。
  - **PRIMARY KEY –** UNIQUE 制約と同じですが、プライマリキー制約はテーブルに 1 つしか作成できない点が違います。プライマリキー制約と一意性制約を同じカラムに指定することはできません。プライマリキーは通常、特定のローにとって最適な識別子を識別します。たとえば、顧客番号は顧客テーブルのプライマリキーです。
  - **FOREIGN KEY –** 一方のカラムセットに対する値を制限し、他方のテーブルのプライマリキーまたは一意性制約の値と一致させます。たとえば、外部キー制約を使用して、請求書テーブルの顧客番号が顧客テーブルの顧客番号と確実に一致するようにできます。

    ローカルテンポラリテーブルに対しては外部キー制約を作成できません。グローバルテンポラリテーブルは ON COMMIT PRESERVE ROWS で作成してください。
  - **CHECK –** 任意の条件を検証できます。たとえば、検査制約を使用して `Gender` カラムに Male と Female の値しか含まれないようにすることができます。テーブル内のどのローも、制約に違反することは許されません。**INSERT** 文または **UPDATE** 文によってローが制約に違反する場合、操作は許可されず、この文の結果は取り消されます。

    カラムの検査制約に記述され、先頭に '@' の記号が付くカラム識別子は、実際のカラム名のプレースホルダです。したがって、次のように記述された文は、

    ```
    CREATE TABLE t1(c1 INTEGER CHECK (@foo < 5))
    ```

次の文とまったく同じになります。

```
CREATE TABLE t1(c1 INTEGER CHECK (c1 < 5))
```

テーブルの検査制約に記述され、先頭に '@' の記号が付くカラム識別子はプレースホルダではありません。

ある文によって整合性制約に違反するデータベースへの変更が生じた場合、その文は事実上実行されず、エラーがレポートされます「事実上」とは、エラーが検出されるより前にこの文が行った変更がすべて取り消されることを示します。

SAP Sybase IQ はそのカラムの HG インデックスを作成することにより、シングルカラム UNIQUE 制約を強制します。

注意： BIT データ型のカラムに、UNIQUE 制約または PRIMARY KEY 制約を定義することはできません。また、BIT データ型のカラムのデフォルトでは NULL 値を使用できませんが、カラムを明示的に定義して、NULL 値を使用できるように変更できます。

- **column-constraint –** カラムに格納可能な値を制限します。カラム制約とテーブル制約によってデータベース内のデータの整合性が保証されます。整合性制約の違反を起こす文は、実行が完了しません。このような文がエラー検出の前に行った変更は取り消され、エラーがレポートされます。カラム制約は、対応するテーブル制約の省略形です。たとえば、次の文は同じです。

```
CREATE TABLE Products (
    product_num integer UNIQUE
)
CREATE TABLE Products (
    product_num integer,
    UNIQUE ( product_num )
)
```

通常、カラム制約を使用するのは、制約がテーブル内で複数のカラムを参照しない場合です。複数のカラムを参照する場合は、テーブル制約を使用する必要があります。

  - **IQ UNIQUE –** カラムの予期されるカーディナリティを定義し、カラムをフラット FP または NBit FP のどちらとしてロードするかを決定します。IQ UNIQUE(n) の値を明示的に 0 に設定すると、カラムはフラット FP としてロードされます。IQ UNIQUE 制約のないカラムは、FP_NBIT_AUTOSIZE_LIMIT、FP_NBIT_LOOKUP_MB、FP_NBIT_ROLLOVER_MAX_MB の各オプションで定義された上限まで暗黙的に NBit としてロードされます。

    - FP_NBIT_AUTOSIZE_LIMIT は、NBit としてロードする、重複しない値の数を制限します。

- FP_NBIT_LOOKUP_MB は、NBit ディクショナリの合計サイズのスレッショルドを設定します。
- FP_NBIT_ROLLOVER_MAX_MB は、NBit からフラット FP への暗黙的な NBit ロールオーバーで使用するディクショナリサイズを設定します。
- FP_NBIT_ENFORCE_LIMITS は、NBit ディクショナリのサイズ制限を強制します。このオプションはデフォルトで OFF になっています。

IQ UNIQUE を FP_NBIT_AUTOSIZE_LIMIT 未満の n 値とともに使用する必要はありません。自動サイズ機能によって、カーディナリティが低いか中程度のカラムはすべて NBit としてサイズ決定されます。カラムをフラット FP としてロードする場合や、重複しない値の数が FP_NBIT_AUTOSIZE_LIMIT を超えるときにカラムを NBit としてロードする場合は、IQ UNIQUE を使用します。

注意：

- 高い IQ UNIQUE 値を指定する際はメモリ使用率を考慮します。マシンリソースに制限がある場合、FP_NBIT_ENFORCE_LIMITS='OFF' (デフォルト) でロードしないでください。
  SAP Sybase IQ 16.0 以前は、16777216 を超える IQ UNIQUE *n* 値はフラット FP にロールオーバーされていました。16.0 では、より大きい IQ UNIQUE 値がトークン化でサポートされていますが、カーディナリティとカラム幅によっては、大量のメモリリソースが必要になる場合があります。
- BIT、BLOB、および CLOB の各データ型は NBit ディクショナリ圧縮をサポートしません。FP_NBIT_IQ15_COMPATIBILITY='OFF' である場合、これらのデータ型を含む CREATE TABLE 文または ALTER TABLE 文でゼロ以外の IQ UNIQUE カラムを指定すると、エラーが返されます。

- **column-constraint 句と table-constraint 句 –** カラム制約とテーブル制約によってデータベース内のデータの整合性が保証されます。
  - **PRIMARY KEY または PRIMARY KEY (column-name, …) –** テーブルのプライマリキーは、リストしたカラムで構成されます。プライマリキーに指定されたどのカラムにも NULL 値を格納することはできません。SAP Sybase IQ では、テーブル内の各ローが必ず、ユニークなプライマリキー値を持ちます。1 つのテーブルが持てる PRIMARY KEY は 1 つだけです。

    2 番目の形式 (PRIMARY KEY の後にカラムリストが続く形式) を使用する場合、カラムのリスト順ではなく定義順にカラムを含めるプライマリキーが作成されます。

    カラムを PRIMARY KEY、FOREIGN KEY、または UNIQUE として指定すると、SAP Sybase IQ によってそのカラムに自動的に High_Group インデックスが作成されます。マルチカラムプライマリキーの場合、このインデックス

は個々のカラムではなく、プライマリキーに対して作成されます。パフォーマンスを高めるため、各カラムに HG または LF インデックスを個別に作成してください。

- **REFERENCES primary-table-name [(primary-column-name)] –** カラムを、プライマリテーブルのプライマリキーまたは一意性制約に対する外部キーとして定義します。通常、外部キーは一意性制約のためのものではなく、プライマリキーのためのものです プライマリカラム名を指定する場合、この名前は一意性制約またはプライマリキーの制約を受けるプライマリテーブルのカラム名と一致する必要があります。また、この制約は、その 1 カラムだけで構成される必要があります。それ以外の場合、外部キーは第 2 のテーブルのプライマリキーを参照します。プライマリキーと外部キーは、データ型と精度、位取り、符号が同じであることが必要です。シングルカラムの外部キーには、ユニークでないシングルカラムの HG インデックスだけが作成されます。マルチカラム外部キーに対しては、SAP Sybase IQ はユニークでない複合 HG インデックスを作成します。ユニークまたはユニークでない HG インデックスのマルチカラム複合キーの最大幅は 1KB です。

  テンポラリテーブルは、ベーステーブルを参照する外部キーを持つことはできません。また、ベーステーブルも、テンポラリテーブルを参照する外部キーを持つことはできません。ローカルテンポラリテーブルは、外部キーを持つことも、外部キーによって参照されることもできません。
- **FOREIGN KEY [role-name] [(...)] REFERENCES primary-table-name [(...)] –** 別のテーブルのプライマリキーまたは一意性制約を参照する外部キーを定義します。通常、外部キーは一意性制約のためのものではなく、プライマリキーのためのものです (この説明では、この他方のテーブルをプライマリテーブルと呼びます)。

  プライマリテーブルのカラム名が指定されていない場合、プライマリテーブルのカラムは、テーブルのプライマリキーの中のカラムになります。外部キーカラム名が指定されていない場合、外部キーカラムは、プライマリテーブルの中のカラムと同じ名前になります。外部キーカラム名が指定されている場合は、プライマリキーのカラム名を指定する必要があります。これらのカラム名は、リスト内の位置に応じて一対になります。

  プライマリテーブルと外部キーテーブルが同じでない場合、参照されるキーには一意性制約またはプライマリキー制約を定義する必要があります。参照されるキーと外部キーは、カラムの数、データ型、符号、精度、位取りが同じでなくてはなりません。

  ローの外部キーの値は、外部キー内の 1 つまたは複数のカラムで、null を許可する外部キーカラムに null が格納されている場合を除き、プライマリテーブルのいずれかのローに格納された候補キー値として出現する必要があります。

明示的に定義されない外部キーのカラムは、プライマリテーブルの対応するカラムと同じデータ型で自動的に作成されます。これらの自動的に作成されたカラムは外部テーブルのプライマリキーの一部にはなりません。したがって、プライマリキーと外部キーの両方の中で使われるカラムは、明示的に作成する必要があります。

role-name は外部キーの名前です。*role-name* の主な機能は、同じテーブルに対する 2 つの外部キーを区別することです。*role-name* が指定されていない場合、role-name は次のように割り当てられます。

1. テーブル名と同じ *role-name* を含む外部キーが存在しない場合、テーブル名が *role-name* として割り当てられます。
2. テーブル名がすでに使用されている場合、*role-name* は、0 が埋め込まれた、テーブルに固有の 3 桁の数字と結合されたテーブル名になります。

参照整合性アクションは、データベース内で外部キー関係を維持するために取られるアクションを定義します。データベーステーブルからプライマリキー値が変更されたり削除されると、それに対応して何らかの修正が必要となる外部キー値が他のテーブルに存在する可能性があります。ON DELETE 句の後に続けて RESTRICT 句を指定できます。

- **RESTRICT –** データベースのどこかに対応する外部キーがあるにもかかわらず、プライマリキー値を更新または削除しようとすると、エラーになります。外部キーを更新して、候補キーに一致しない新しい値を作成しようとすると、エラーになります。この動作はデフォルトです。ただし、オプションで参照整合性に違反するローを拒否するように LOAD を指定した場合を除きます。これにより、文レベルでの参照整合性が確保されます。

  アクションを何も指定しないで CHECK ON COMMIT を使用すると、RESTRICT は DELETE のアクションに対して適用されます。SAP Sybase IQ は CHECK ON COMMIT をサポートしません。

  グローバルテンポラリテーブルは、ベーステーブルを参照する外部キーを持つことはできません。また、ベーステーブルも、グローバルテンポラリテーブルを参照する外部キーを持つことはできません。ローカルテンポラリテーブルは、外部キーを持つことも、外部キーによって参照されることもできません。
- **CHECK (条件) –** 条件に合わないローは許可されません。INSERT 文によってローがこの条件を満たさなくなる場合、操作は許可されず、この文の効果は取り消されます。

  変更は、条件が FALSE の場合にのみ拒否されます。条件が UNKNOWN の場合、変更は許可されます。CHECK 条件は SAP Sybase IQ では強制されません。

注意： 孤立した外部キーがないと確実にわかっている場合を除き、できる限り、SAP Sybase IQ に参照整合性 (すなわち、外部キーとプライマリキーの関係) を定義しないでください。

- **リモートテーブル –** 外部キー定義は、リモートテーブルでは無視されます。リモートテーブルを参照するローカルテーブルの外部キー定義も無視されます。プライマリキー定義は、サーバがプライマリキーをサポートする場合、リモートサーバに送信されます。
- **PARTITION BY –** 大きなテーブルを、より小さく管理しやすいストレージオブジェクトに分割します。各パーティションは親テーブルと同じ論理属性を共有しますが、別々の DB 領域に配置して個別に管理できます。SAP Sybase IQ は、次のような複数のテーブル分割スキームをサポートしています。
  - ハッシュパーティション
  - 範囲パーティション
  - 複合パーティション

  partition-key は、テーブル分割キーが格納されている 1 つまたは複数のカラムです。分割キーには、NULL 値 および DEFAULT 値を含めることができますが、次のカラムを含めることはできません。
  - LOB (BLOB または CLOB) カラム
  - BINARY または VARBINARY カラム
  - 長さが 255 バイトを超える CHAR または VARCHAR カラム
  - BIT カラム
  - FLOAT/DOUBLE/REAL カラム
- **PARTITION BY RANGE –** 分割カラム内の値の範囲によってローを分割します。範囲分割は、単一の分割キーカラムおよび最大 1024 パーティションまでに制限されています。range-partitioning-scheme 内の partition-key は、テーブル分割キーが格納されているカラムです。

```
range-partition-decl:
  partition-name VALUES <= ( {constant-expr |  MAX } [ ,
{ constant-expr |  MAX }]... )
    [ IN dbspace-name ]
```

  partition-name は、テーブルローが格納される新しいパーティションの名前です。パーティション名は、テーブル上にあるパーティションセット内でユニークである必要があります。パーティション名は必須です。
  - **VALUE –** 各パーティションの包括的な上限を (昇順に) 指定します。ユーザは、各ローが 1 つのパーティションのみに分配されるように、各範囲分割の分割基準を指定する必要があります。NULL は分割カラムに使用でき、

NULL を分割キー値に含んだローは最初のテーブル分割に属します。ただし、NULL をバインド値に指定することはできません。

最初のパーティションには、下限 (MIN 値) は設定されていません。分割キーの最初のカラムにある NULL セルのローは、最初のパーティションに移動します。最後のパーティションでは、包括的な上限または MAX を指定できます。最後のパーティションの上限値が MAX でない場合は、最後のパーティションの上限値よりも大きい分割キーの値を含んだローをロードまたは挿入すると、エラーが生成されます。

- **MAX** – 無制限の上限を示し、最後のパーティションに対してのみ指定できます。
- **IN** – partition-decl でパーティションのローが存在する DB 領域を指定します。

次の制限を設定すると、範囲分割されたテーブルの分割キーとバインド値がその影響を受けます。

- パーティションバインドは定数式でなく、定数として指定する必要があります。
- パーティションバインドは、パーティションの作成順に応じて、昇順で指定する必要があります。つまり、2 番目のパーティションの上限は最初のパーティションよりも高く指定する必要がある、というようになります。さらに、パーティションバインドの値は、対応する分割キーカラムのデータ型と互換性がなければなりません。たとえば、VARCHAR は CHAR と互換性があります。
- バインド値に対応する分割キーのカラムとは異なるデータ型が指定されていると、SAP Sybase IQ はバインド値を分割キーのカラムのデータ型に変換します。ただし、次の場合は例外となります。
- 明示的な変換は使用できません。この例では、INT から VARCHAR に明示的に変換しようとしてエラーが生成されます。

```
CREATE TABLE Employees(emp_name VARCHAR(20))
PARTITION BY RANGE(emp_name)
(p1 VALUES <=(CAST (1 AS VARCHAR(20))),
p2 VALUES <= (CAST (10 AS VARCHAR(20)))
```

- データロスにつながる暗黙的な変換は使用できません。この例では、パーティションバインドは分割キー型と互換性がありません。丸めを前提で処理を行うとデータロスにつながる可能性があり、エラーが生成されます。

```
CREATE TABLE emp_id (id INT) PARTITION BY RANGE(id) (p1 VALUES
<= (10.5), p2 VALUES <= (100.5))
```

- この例では、パーティションバインドと分割キーのデータ型の間には互換性があります。バインド値は FLOAT 値に直接変換されます。丸め処理は必要なく、変換はサポートされています。

```
CREATE TABLE id_emp (id FLOAT)
PARTITION BY RANGE(id) (p1 VALUES <= (10),
p2 VALUES <= (100))
```

- 非バイナリデータ型からバイナリデータ型に変換することはできません。たとえば、次の変換は実行できずに、エラーが返されます。

```
CREATE TABLE newemp (name BINARY)
PARTITION BY RANGE(name)
(p1 VALUES <= ("Maarten"),
p2 VALUES <= ("Zymmerman")
```

- NULL を範囲分割テーブルで境界として使用することはできません。
- 分割キーの最初のカラムのセル値が NULL と評価された場合、ローは最初のパーティションに挿入されます。SAP Sybase IQ は、1 つのカラムの分割キーのみをサポートしているため、分割キー内に NULL が含まれていると、ローは最初のパーティションに分配されます。

- **PARTITION BY HASH –** 内部ハッシュ関数によって処理された分割キーの値に基づいて、データをパーティションにマップします。ハッシュ分割キーは最大 8 カラムで、組み合わせた宣言カラム幅が 5300 バイト以下に制限されています。ハッシュパーティションの場合、テーブル作成者は分割キーカラムのみを決定します。パーティションの数と位置は内部的に決定されます。

  hash-partitioning 宣言内の partition-key は、1 つのカラムまたはカラムのグループです。その複合値によってデータの各ローが格納されるパーティションが決まります。

```
hash-partitioning-scheme:
  HASH  ( partition-key [ , partition-key, … ] )
```

  - **制限事項 –**
    - ハッシュ分割できるのはベーステーブルのみです。グローバルテンポラリテーブルやローカルテンポラリテーブルを分割しようとすると、エラーが発生します。
    - ハッシュパーティションの追加、削除、マージ、分割はできません。
    - カラムをハッシュ分割キーから追加または削除することはできません。
- **PARTITION BY HASH RANGE –** ハッシュ分割されたテーブルを範囲によってさらに分割します。hash-range-partitioning-scheme 宣言内の SUBPARTITION BY RANGE 句は、新しい範囲サブパーティションを既存のハッシュ範囲分割テーブルに追加します。

```
hash-range-partitioning-scheme:
PARTITION BY HASH  ( partition-key [ , partition-key, … ] )
    [ SUBPARTITION BY RANGE  ( range-partition-decl [ , range-
partition-decl ... ] ) ]
```

  ハッシュパーティションはデータの論理的な配分および配置方法を指定するのに対して、範囲サブパーティションはデータの物理的な配置方法を指定します。新しい範囲サブパーティションは、既存のハッシュ範囲分割テーブルと同じハッシュ分割キーを持つハッシュによって論理的に分割されます。範囲サブパーティションは 1 つのカラムに制限されています。

- **制限事項 –**
  - ハッシュ分割できるのはベーステーブルのみです。グローバルテンポラリテーブルやローカルテンポラリテーブルを分割しようとすると、エラーが発生します。
  - ハッシュパーティションの追加、削除、マージ、分割はできません。
  - カラムをハッシュ分割キーから追加または削除することはできません。

  注意： 範囲パーティションと複合分割スキームは、ハッシュ範囲パーティションと同様、個別にライセンスが必要な VLDB Management オプションを必要とします。

例

（先頭に戻る） (777 ページ)

- **例 1 –** 5 つのカラムを持つ SalesOrders2 という名前のテーブルを作成します。カラム FinancialCode、OrderDate、ID のデータページは DB 領域 Dsp3 にあります。整数カラム CustomerID のデータページは DB 領域 Dsp1 にあります。CLOB カラム History のデータページは DB 領域 Dsp2 にあります。プライマリキー (ID の HG) のデータページは DB 領域 Dsp4 にあります。

```
CREATE TABLE SalesOrders2 (
FinancialCode CHAR(2),
CustomerID int IN Dsp1,
History CLOB IN Dsp2,
OrderDate TIMESTAMP,
ID BIGINT,
PRIMARY KEY(ID) IN Dsp4
) IN Dsp3
```

- **例 2 –** 4 つのカラムを持つ fin_code2 というテーブルを作成します。カラム code、type、id のデータページはデフォルトの DB 領域にあり、データベースオプション DEFAULT_DBSPACE の値によって決まります。CLOB カラム description のデータページは DB 領域 Dsp2 にあります。外部キー fk1 (c1 の HG) のデータページは DB 領域 Dsp4 にあります。

```
CREATE TABLE fin_code2 (
code INT,
type CHAR(10),
description CLOB IN Dsp2,
id BIGINT,
FOREIGN KEY fk1(id) REFERENCES SalesOrders(ID) IN Dsp4
)
```

- **例 3 –** テーブル t1 を作成します。ここで、パーティション p1 はパーティション p2 に、パーティション p2 はパーティション p3 に隣接します。

```
CREATE TABLE t1 (c1 INT, c2 INT)
PARTITION BY RANGE(c1)
(p1 VALUES <= (0), p2 VALUES <= (10), p3 VALUES <= (100))
```

- **例 4 –** 6 つのカラムと 3 つのパーティションを持つ、RANGE 分割されたテーブル `bar` を作成し、データを日付に基づいたパーティションにマッピングします。

```
CREATE TABLE bar (
        c1 INT IQ UNIQUE(65500),
        c2 VARCHAR(20),
        c3 CLOB PARTITION (P1 IN Dsp11, P2 IN Dsp12,
           P3 IN Dsp13),
        c4 DATE,
        c5 BIGINT,
        c6 VARCHAR(500) PARTITION (P1 IN Dsp21,
           P2 IN Dsp22),
        PRIMARY KEY (c5) IN Dsp2) IN Dsp1
        PARTITION BY RANGE (c4)
        (P1 VALUES <= ('2006/03/31') IN Dsp31,
         P2 VALUES <= ('2006/06/30') IN Dsp32,
         P3 VALUES <= ('2006/09/30') IN Dsp33
        ) ;
```

各パーティションのデータページ割り付け：

パーティション	DB 領域	カラム
P1	Dsp31	c1、c2、c4、c5
P1	Dsp11	c3
P1	Dsp21	c6
P2	Dsp32	c1、c2、c4、c5
P2	Dsp12	c3
P2	Dsp22	c6
P3	Dsp33	c1、c2、c4、c5、c6
P3	Dsp13	c3
P1、P2、P3	Dsp1	c1 および他の共有データのルックアップストア
P1、P2、P3	Dsp2	プライマリキー (c5 の HG)

- **例 5 –** `HASH` 分割された (`table tbl42`) を作成します。このテーブルは、`PRIMARY KEY` (カラム c1) と `HASH PARTITION KEY` (カラム c4 および c3) を含みます。

```
CREATE TABLE tbl42 (
    c1 BIGINT NOT NULL,
    c2 CHAR(2) IQ UNIQUE(50),
    c3 DATE IQ UNIQUE(36524),
    c4 VARCHAR(200),
  PRIMARY KEY (c1)
    )
  PARTITION BY HASH ( c4, c3 )
```

- **例 6 –** PRIMARY KEY (カラム c1)、ハッシュ分割キー (カラム c4 および c2)、および範囲サブ分割キー (カラム c3) を指定して、ハッシュ範囲分割されたテーブルを作成します。

```
CREATE TABLE tbl42 (
    c1 BIGINT NOT NULL,
    c2 CHAR(2) IQ UNIQUE(50),
    c3 DATE,
    c4 VARCHAR(200),
  PRIMARY KEY (c1)) IN Dsp1

  PARTITION BY HASH ( c4, c2 )
  SUBPARTITION BY RANGE ( c3 )
  ( P1 VALUES <= (2011/03/31) IN Dsp31,
    P2 VALUES <= (2011/06/30) IN Dsp32,
    P3 VALUES <= (2011/09/30) IN Dsp33) ;
```

- **例 7 –** 図書データベース用にテーブルを作成し、貸出された図書の情報を保持します。

```
CREATE TABLE borrowed_book (
date_borrowed DATE NOT NULL,
date_returned DATE,
book       CHAR(20)
           REFERENCES library_books (isbn),
CHECK( date_returned >= date_borrowed )
)
```

- **例 8 –** リモートサーバ SERVER_A にテーブル t1 を作成し、リモートテーブルにマッピングされるプロキシテーブル t1 を作成します。

```
CREATE TABLE t1
( a  INT,
  b  CHAR(10))
AT 'SERVER_A.db1.joe.t1'
```

- **例 9 –** カラム c1 に特殊定数 LAST USER のデフォルト値を含むテーブル tab1 を作成します。

```
CREATE TABLE tab1(c1 CHAR(20) DEFAULT LAST USER)
```

- **例 10 –** カラム c1 を持つローカルテンポラリテーブル tab1 を作成します。

```
CREATE LOCAL TEMPORARY TABLE tab1(c1 int) IN IQ_SYSTEM_TEMP
```

この例では、次の場合、tab1 が IQ_SYSTEM_TEMP DB 領域に作成されます。

- DQP_ENABLED 論理サーバポリシーオプションが ON に設定されているが、読み取り／書き込み可能なファイルが IQ_SHARED_TEMP に存在しない場合
- DQP_ENABLED オプションが OFF、TEMP_DATA_IN_SHARED_TEMP 論理サーバポリシーオプションが ON に設定されているが、読み取り／書き込み可能なファイルが IQ_SHARED_TEMP に存在しない場合
- DQP_ENABLED オプションと TEMP_DATA_IN_SHARED_TEMP オプションが両方とも OFF に設定されている場合

この例では、次の場合、同じテーブル tab1 が IQ_SHARED_TEMP DB 領域に作成されます。

- DQP_ENABLED が ON で、読み取り／書き込み可能なファイルが IQ_SHARED_TEMP に存在する場合
- DQP_ENABLED が OFF、TEMP_DATA_IN_SHARED_TEMP が ON で、読み取り／書き込み可能なファイルが IQ_SHARED_TEMP に存在する場合

- **例 11 –** インメモリ RLV ストアでローレベルのバージョン管理とリアルタイムストレージを使用できるようしたテーブル tab1 を作成します。

```
CREATE TABLE tab1 ( c1 INT, c2 CHAR(25) ) ENABLE RLV STORE
```

使用法

（先頭に戻る） (777 ページ)

所有者名を指定することにより、別のユーザが使用するテーブルを作成できます。GLOBAL TEMPORARY または LOCAL TEMPORARY が指定されていない場合、テーブルはベーステーブルと呼ばれます。指定すると、テーブルはテンポラリテーブルとなります。

作成されたグローバルテンポラリテーブルは、ベーステーブルと同様にデータベース内に存在し、DROP TABLE 文によって明示的に削除されるまでデータベース内に残ります。テンポラリテーブル内のローは、ローを挿入した接続だけが参照できます。同じ、または異なるアプリケーションからの複数の接続が、同じテンポラリテーブルを同時に使用することもできます。このとき、それぞれの接続で参照できるのは自身のローだけです。この接続は、最初にグローバルテンポラリテーブルを参照し、存在していれば、そのテーブルのスキーマを継承します。接続が終了すると、テンポラリテーブルのローは削除されます。

ローカルテンポラリテーブルを作成するときは、所有者を指定しないでください。たとえば、テンポラリテーブルの作成時に、CREATE TABLE dbo.#temp(col1 int) のように所有者を指定すると、ベーステーブルが間違って作成されます。

その接続に同じ名前のローカルテンポラリテーブルがある場合、ベーステーブルまたはグローバルテンポラリテーブルを作成しようとしても失敗します。これは、新しいテーブルを owner.table がユニークに識別できないからです。

ただし、既存のベーステーブルまたはグローバルテンポラリテーブルとしてなら、ローカルテンポラリテーブルを同じ名前で作成できます。テーブル名への参照は、ローカルテンポラリテーブルにアクセスします。ローカルテンポラリテーブルが最初に解決されるからです。

次のシーケンス例を見てみましょう。

```
CREATE TABLE t1 (c1 int);
INSERT t1 VALUES (9);

CREATE LOCAL TEMPORARY TABLE t1 (c1 int);
INSERT t1 VALUES (8);

SELECT * FROM t1;
```

返される結果は 8 です。ローカルテンポラリテーブルが接続により削除されるまで、t1 に対する参照はいずれも、ローカルテンポラリテーブル t1 を参照します。

プロシージャの完了後も保持されるテーブルを作成する場合、DECLARE LOCAL TEMPORARY TABLE 文ではなく CREATE LOCAL TEMPORARY TABLE 文をプロシージャに使用します。CREATE LOCAL TEMPORARY TABLE 文を使用して作成されたローカルテンポラリテーブルは、明示的に削除するか接続が終了するまで保持されます。

CREATE LOCAL TEMPORARY TABLE を使用して IF 文で作成されたローカルテンポラリテーブルは、IF 文が完了した後も保持されます。

SAP Sybase IQ では、SAP Sybase IQ テーブルのテーブルレベルの暗号化の CREATE TABLE ENCRYPTED 句はサポートされていません。ただし、CREATE TABLE ENCRYPTED 句は、SAP Sybase IQ データベースの SQL Anywhere テーブルではサポートされています。

関連する動作：

- オートコミット

標準

（先頭に戻る） (777 ページ)

- SQL — ISO/ANSI SQL 文法のベンダ拡張。
  次にベンダ拡張を示します。
  - { **IN** | **ON** } *dbspace-name* 句
  - **ON COMMIT** 句

    - いくつかのデフォルト値
- SAP Sybase Database 製品 - Adaptive Server でサポートされていますが、いくつか相違があります。
    - **テンポラリテーブル** – ポンド記号 (#) を持つ **CREATE TABLE** 文でテーブル名に先行して、テンポラリテーブルを作成することができます。このようなテンポラリテーブルは、SAP Sybase IQ が宣言するテンポラリテーブルであり、現在の接続でしか使用できません。宣言されたテンポラリテーブルの詳細については、「DECLARE LOCAL TEMPORARY TABLE 文」を参照してください。
    - **物理的配置** – テーブルの物理的配置は SAP Sybase IQ と Adaptive Server では方法が異なります。Adaptive Server によってサポートされている **ON** *segment-name* 句は SAP Sybase IQ でもサポートされていますが、*segment-name* は IQ の DB 領域を参照します。
    - **制約** – SAP Sybase IQ は名前付き制約と名前付きデフォルトをサポートしませんが、制約とデフォルトの定義をデータ型定義にカプセル化できるユーザ定義データ型はサポートします。また、**CREATE TABLE** 文の明示的なデフォルトと CHECK 条件もサポートします。
    - **NULL** – (デフォルト) デフォルトで、Adaptive Server のカラムは、`NOT NULL` にデフォルト設定されていますが、SAP Sybase IQ でのデフォルト設定は `NULL` であり、`NULL` 値が許可されています。この設定は、`ALLOW_NULLS_BY_DEFAULT` オプションを使用して制御できます。「ALLOW_NULLS_BY_DEFAULT オプション [TSQL]」を参照してください。データ定義文を転送可能にするには、`NULL` または `NOT NULL` を明示的に指定します。

パーミッション

（先頭に戻る） (777 ページ)

テーブルタイプ	必要な権限
IQ メインストア内のベーステーブル	自分が所有するテーブル – テーブルが作成される DB 領域に対する CREATE 権限が必要。次のいずれかも必要。 • CREATE TABLE システム権限 • CREATE ANY OBJECT システム権限 任意のユーザが所有するテーブル – テーブルが作成される DB 領域に対する CREATE 権限が必要。次のいずれかも必要。 • CREATE ANY TABLE システム権限 • CREATE ANY OBJECT システム権限
グローバルテンポラリテーブル	自分が所有するテーブル – 次のいずれかが必要。 • CREATE TABLE システム権限 • CREATE ANY OBJECT システム権限 任意のユーザが所有するテーブル – 次のいずれかが必要。 • CREATE ANY TABLE システム権限 • CREATE ANY OBJECT システム権限
プロキシテーブル	自分が所有するテーブル – 次のいずれかが必要。 • CREATE PROXY TABLE システム権限 • CREATE ANY TABLE システム権限 • CREATE ANY OBJECT システム権限 任意のユーザが所有するテーブル – 次のいずれかが必要。 • CREATE ANY TABLE システム権限 • CREATE ANY OBJECT システム権限

DROP SERVER 文

SAP Sybase IQ システムテーブルからリモートサーバを削除します。

クイックリンク：

「例」へ移動 (797 ページ)

「使用法」へ移動 (797 ページ)

「標準」へ移動 (797 ページ)

「パーミッション」へ移動 (797 ページ)

構文

DROP SERVER *server-name*

例

（先頭に戻る） (796 ページ)

- **例 1 –** 次の例では、サーバ `IQ_prod` を削除します。

```
DROP SERVER iq_prod
```

使用法

（先頭に戻る） (796 ページ)

DROP SERVER を正常に実行するには、リモートサーバに定義されたプロキシテーブルをすべて削除しておく必要があります。

関連する動作

- オートコミット。

標準

（先頭に戻る） (796 ページ)

- SQL - ISO/ANSI SQL 準拠。
- SAP Sybase Database 製品 - Open Client/Open Server でサポートされています。

パーミッション

（先頭に戻る） (796 ページ)

SERVER OPERATOR システム権限が必要です。

索引

記号

数字

A

B

C

D

E

F

G

H

L

M

N

O

P

Q

R

S

T

U

V

W

X

あ

い

う

え

お

か

き

く

け

こ

さ

と

な

に

ね

は

ひ

ふ

ら

り

れ

ろ