WHITE PAPER

Borland Delphi とSQL Anywhere Studio を使用したアプリケーション開発

概要

Borland Delphiは、Windows用のRADツールです。

本書では、DelphiとSQL Anywhere Studioを使用したアプリケーション開発の概要について、また両環境に適切な使用方法と一般的問題解決に関して言及します。43ページの「付録D」には、本書で説明する問題特定と解決に使用する製品とそのバージョン番号表を収録しました。また、本書で説明するインタフェースのWebページも記載しています。

e-BUSINESS ANYWHERE

目次

概要 ..... 表紙

Delphi 環境 5

Delphi で ASA を使用するための設定 6

ODBC データ・ソースの作成 6

Titan のインストールおよび Titan のエイリアス作成 8

NativeDB for SQL Anywhere のインストール 10

Delphi コントロールの概要 11

コントロール・プロパティのバインド設定例 11

Delphi 経由で ASA データベースに接続する 12

BLOB の例 16

DBGrid の例 24

付録 A 35

Delphi で ODBCExpress インタフェースを使用するための設定 35

付録 B 38

ASA 7.0 のサンプルの修正 38

付録 C 41

プライマリ・キーの問題 41

付録 D 43

法的注意 44

Delphi 環境

デフォルトでは、Delphi は BDE (Borland Database Engine) という汎用の ODBC (Open DataBase Connectivity) インタフェースを使用します。プログラミングの柔軟性を向上させるために、Delphi では ODBCExpress、Titan SQLAnywhere Developer、NativeDB for SQL Anywhere などの他のデータベース・インタフェースも使用することができます。ODBCExpress は、Datasoft Ltd 製の汎用 ODBC インタフェースです。ODBCExpress を使用した場合は、完全に BDE に置き換わってアプリケーションにコンパイルされます。Titan は、Reggatta Systems 製のインタフェースで、Adaptive Server Anywhere (ASA) Embedded SQL インタフェースを使用して組み込みます。Titan SQL Anywhere Developer は、Sybase の SQL Anywhere データベースを使用する場合に高速性を実現するツールです。NativeDB は、Liodden Data A/S 製のインタフェースで、Embedded SQL インタフェースを使用します。NativeDB は、ASA のより高度な機能 (コールバック、サーバ実行のキャンセル、SP の再開など) を利用するように設計されています。ODBCExpress を Delphi で使用するように設定する場合の詳細については、35 ページの「付録 A」を参照してください。Titan を Delphi で使用するように設定する場合の詳細については、8 ページの「Titan のインストールおよび Titan のエイリアス作成」を参照してください。NativeDB を Delphi で使用するように設定する場合の詳細については、10 ページの「NativeDB for SQL Anywhere のインストール」を参照してください。

図 1：は、Delphi 環境を最初に開いたときの 表示を示しています。

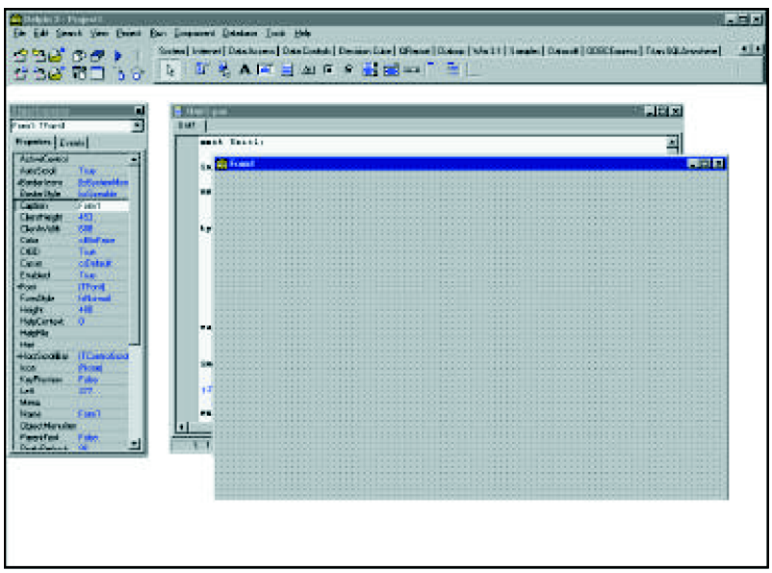

図 1：

Delphi で ASA を使用するための設定

ODBC データ・ソースの作成

Adaptive Server Anywhere (ASA) データベースを作成した後は、Delphi で使用できるように簡単に設定することができます。ASA データベースを作成せずに ASA 7.0 に付属のサンプル・データベースを使用する場合は、「付録 B」の手順を参照してください。BDE および ODBCExpress のいずれも ODBC を使用するため、以下の手順に従ってデータベースの ODBC データ・ソースを設定する必要があります。

1. Sybase Central 4.0 を起動します (デフォルトでは [スタート] → [プログラム] → [SQL Anywhere 7] → [Sybase Central 4.0] を使用します)。Sybase Central の左側の [Adaptive Server Anywhere 7] にある ' ユーティリティ ' フォルダをダブルクリックします。画面の右側に、ツールのリストが表示されます。

2. [ODBC アドミニストレータ] をダブルクリックします (または、[スタート] → [プログラム] → [Sybase SQL Anywhere 7] → [Adaptive Server Anywhere 7] → [ODBC アドミニストレータ] を選択します)。

3. そのマシンに自分がログインしたときにだけ表示されるソースを使用する場合は、[ユーザー DSN] タブにフォーカスされていることを確認し、[追加] をクリックします (図 2:)。マシン上の NT サービスを含むすべてのユーザに表示されるソースを使用する場合は、[システム DSN] タブにフォーカスされていることを確認し、[追加] をクリックします。

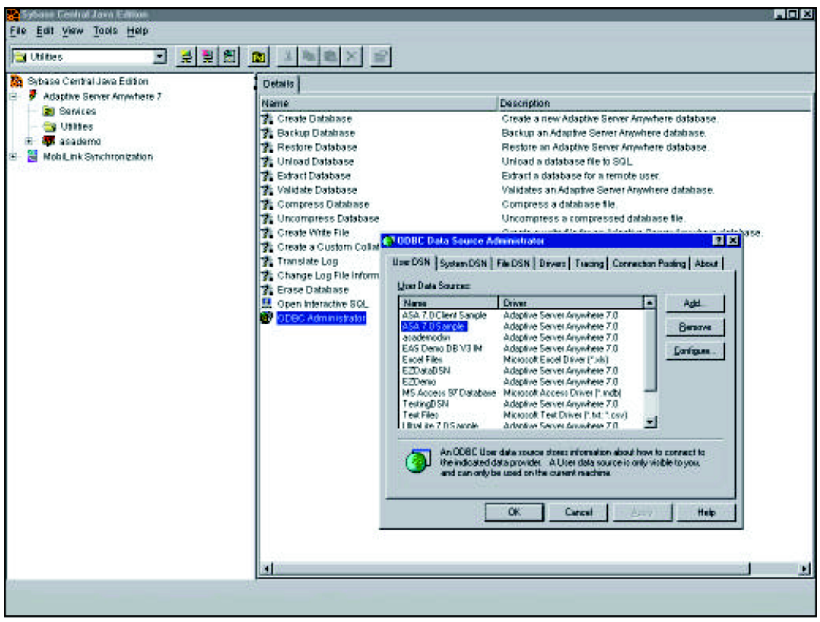

図 2 :

4. [Adaptive Server Anywhere 7.0] を選択し、[完了] をクリックします。タブが複数ある別のウィンドウが表示されます (図 3 :)。

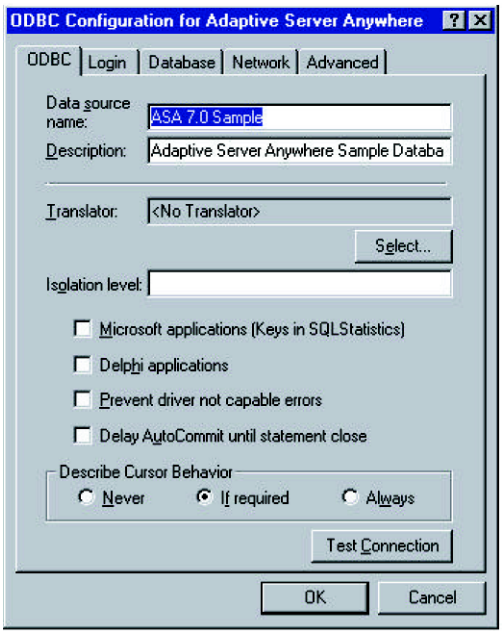

図 3 :

5. ODBC データ・ソースを作成するには、以下の情報を指定する必要があります。

   i. [ODBC] タブで、データベースに対応するデータ・ソース名を [データ・ソース名] に入力します。ODBC データ・ソースは、この名前を使用して指定します。

   ii. [ログイン] タブで、[ユーザ ＩＤ とパスワードの入力] ラジオ・ボタンをクリックし、該当するテキスト・ボックスにユーザ名およびパスワードを入力します。

   iii. [データベース] タブで、データベースの保存場所のパスを [データベース・ファイル] テキスト・ボックスに入力します。パスを入力するか [参照] ボタンをクリックし、データベースのあるディレクトリに移動して [開く] をクリックします。

   iv. [データベースが実行していなければ自動的に起動] および [最終切断後、自動的にデータベースをシャットダウン] のチェックボックスを選択します。

   v. すべての設定が正しいかどうかを確認するため、[ODBC] タブを選択し、[テスト接続] ボタンをクリックします。設定がすべて正しければ、接続が成功したことを示す [接続成功] というウィンドウが表示されます。

6. [OK] をクリックします。作成したデータソース名が、[ODBC データソース・アドミニストレータ] ウィンドウの [ユーザー DSN] タブまたは [システム DSN] タブに表示されます。

7. 再度 [OK] をクリックします。これで、Delphi で使用可能なデータベース用の ODBC データ・ソースが作成されました。

Titan SQL Anywhere for Delphi 3 バージョン 3.02p は、SQL Anywhere バージョン 5.x で使用するように設計されています。ASA 6.x または ASA 7 で作成されたデータベースがある場合は、Delphi アプリケーションを作成する前に追加手順が必要です。追加手順には、互換性ライブラリの設定や、レジストリへの新しい Titan エイリアスの登録を行います。新しいエイリアスの作成については、8 ページの「Titan のインストールおよび Titan のエイリアス作成」を参照してください。

互換性ライブラリは、指定された接続文字列を使用して Adaptive Server Anywhere バージョン 7 のインタフェース・ライブラリに接続することで機能します。この接続に失敗した場合は、互換性ライブラリは SQL Anywhere バージョン 5 のライブラリを使用して SQL Anywhere データベースに接続します。32 ビットの Windows (Win32) では、互換性ライブラリ (dbl50t.dll)、バージョン 5 のインタフェース・ライブラリ (dbl50to.dll)、バージョン 7 のインタフェース・ライブラリ (dblib7.dll) は、すべて同一ディレクトリにインストールされます。このディレクトリは通常、ASA のインストール場所にあります。たとえば、デフォルトでは C:\Program Files\Sybase\SQL Anywhere 7\win32 に保存されます。互換性ライブラリが動作するには、システム・パスでバージョン 7 のインストール・ディレクトリをバージョン 5 のインストール・ディレクトリよりも前になるようにインストールする必要があります。このようにインストールすることで、バージョン 5 のインタフェース・ライブラリよりも先に互換性ライブラリがアプリケーションに認識されます。互換性ライブラリを最初にインストールしていれば、各バージョンのディレクトリは正しい順序で自動的に保存されます。詳細については、SQL Anywhere マニュアルを開き ([スタート] → [プログラム] → [Sybase SQL Anywhere 7] → [SQL Anywhere マニュアル] で表示します)、' 互換性 ' を検索して、' 互換性ライブラリの使用 ' をダブルクリックします。

Titan のインストールおよび Titan のエイリアス作成

Titan SQLAnywhere Developer インタフェースを設定するには、以下の手順に従って Delphi にパッケージをインストールする必要があります。

1. ツールバーの [コンポーネント] を選択し、[パッケージのインストール] をクリックします。

2. [追加] ボタンをクリックし、Titan のダウンロード先ディレクトリから 'SQATITAN.DPL' というパッケージを選択します。

3. [開く] をクリックし、[OK] をクリックしてインストールします。

[Titan SQL Anywhere] という新しいタブが Delphi のコンポーネント・バーに表示されます。

Titan SQL Anywhere with ASA 7.0 を使用するために必要な新しい Titan エイリアスをレジストリに登録するには、以下の手順に従います。

1. [スタート] をクリックし、メニューから [ファイル名を指定して実行] を実行します。

2. 'regedit' と入力し、[OK] をクリックします。

3. [レジストリ・エディタ] ウィンドウで 'HKEY_LOCAL_MACHINE a SOFTWARE a Titan a SqlAnywhere a Aliases' を開きます (図 4 :)。

4. 'Aliases' を右クリックし、[新規] を選択して、[キー] を選択します。

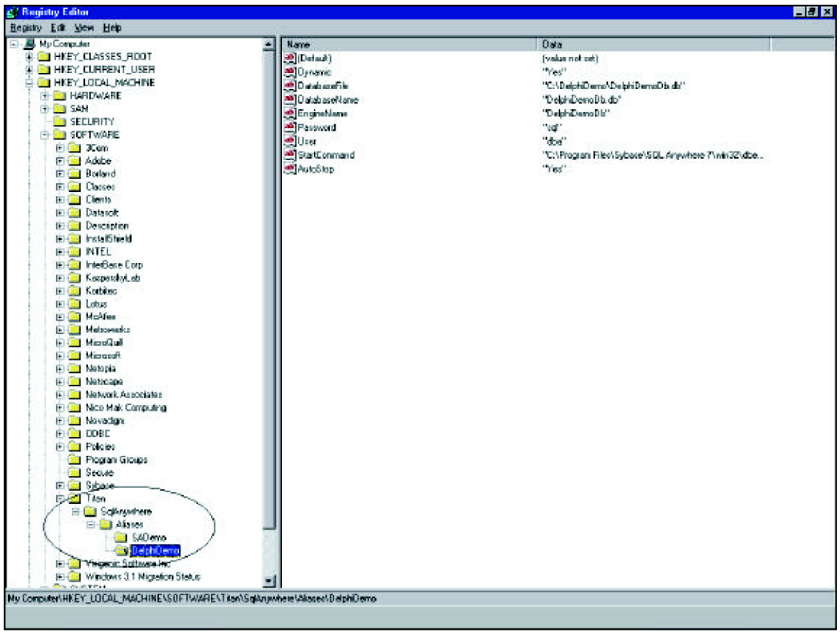

図 4 :

5. フォルダが表示されます。Alias という名前を、データベースを示す名前に変更することができます。このフォルダがデータベースに対応付けられます。

6. 作成したエイリアスの名前を右クリックし、[新規] を選択して、[文字列] を選択します。

7. 以下を実行します。

   i. 'Dynamic' と入力し、[Enter] を押します。

   ii. 'Dynamic' という名前をダブルクリックします。ボックスが表示されます。[値のデータ] 編集ボックスに 'Yes' と入力します。

8. 手順 6 および 7 を、以下の項目に対して繰り返します。

   i. 'DatabaseFile'(手順7のi. を実行)、データベースが保存されるディレクトリ ('c:\Program Files\Sybase\SQL Anywhere 7\asademo.db' など) (手順 7 の ii. を実行)

   ii. 'DatabaseName'、データベースの名前 ('asademo.db' など)

   iii. 'EngineName'、エンジン名 (ほとんどの場合は、データベース名と同一で、'asademo' などを指定します)

iv. 'Password'、データベースのパスワード ('sql' など)

v. 'User,、データベースのユーザ ID('dba' など)

vi. 'StartCommand'、実行可能ファイルdbeng6の場所('c:\ProgramFiles\Sybase\SQL Anywhere 7\win32\dbeng7.exe' など)

vii. 'AutoStop'、'Yes'

9. [レジストリ・エディタ] ウィンドウを閉じます。

NativeDB for SQL Anywhere のインストール

NativeDB は、レジストリ設定、ODBC ソース、BDE エイリアスを使用しないため、以下の手順を実行した後すぐに NativeDB を Delphi で使用することができます。

Titan インタフェースのインストールと同様に、Delphi に 2 つのパッケージをインストールする必要があります。これらのパッケージはインストール順序が重要であるため、注意して以下の手順に従ってください。

1. Delphi で、ツールバーの [コンポーネント] を選択し、[パッケージのインストール] をクリックします。

2. [追加] ボタンをクリックし、NativeDB\NativeDB\Delphi* のダウンロード先ディレクトリから 'NdbPack*.dpl' というパッケージを選択します ('*' は、使用する Delphi のバージョンに置き換えます)。

3. [開く] をクリックします。[設計時パッケージ] ウィンドウに新しいエントリが表示されます。

4. [追加] ボタンを再度クリックし、NdbSa*.dpl というパッケージを選択します ('*' は、使用する Delphi のバージョンに置き換えます)。

5. [開く] をクリック後、次に [OK] をクリックしてインストールします。

6. ツールバーの [ツール] を選択し、[環境オプション] をクリックします。

7. [ライブラリ] タブを選択します。

8. [ライブラリ パス] 編集ボックスで、Delphi* という名前のフォルダのパスを入力します ('*' は、使用する Delphi のバージョンに置き換えます)。パスは、'C:\NativeDB\Delphi*' のように入力します。

Delphi のコンポーネント・バーに、[NativeDB] という新しいタブが表示されます。

Delphi コントロールの概要

コントロール・プロパティのバインド設定例

Delphi は、実際のプログラミングなしで多数のタスクを実行できるように設計されています。特定のコンポーネントは、[オブジェクト インスペクタ] を使用してリンクまたはバインドし、相互に対話するように設定することができます。たとえば、BDE を使用し、'Table1' というテーブル・コンポーネントがある場合に [オブジェクト インスペクタ] で [データベース名] を選択し、リストからデータベースを選択して、そのテーブルを対応付けるデータベースを設定することができます。このリストには、マシン上のすべての ODBC データ・ソースまたはエイリアス が表示されます。これらのうちの 1 つを選択すると、指定したデータベースにテーブル・コンポーネントがバインドされます。

図 5 に、コンポーネントのバインド例を示します。'Form1' というフォームで、BDE インタフェースから 3 つのコンポーネントをバインドしています。これらの 3 つのコンポーネントをバインドするには、以下の手順に従います。

1. 'TTable'、'TDataSource'、'DBGrid' の各コンポーネントをフォームに配置します。

2. [オブジェクト インスペクタ] でドロップダウン・リストを使用して、'TTable' コンポーネントの 'DatabaseName' プロパティを ODBC データ・ソース 'ASA 7.0 Sample' に設定してバインドします。

3. 'DataSource' コンポーネントを選択し、[オブジェクト インスペクタ] でドロップダウン・リストを使用して 'DataSet' プロパティを 'Table1' に設定することで、'DataSource' コンポーネントを 'Table1' 経由でデータベースにバインドします。

4. 'DBGrid' コンポーネントは、'DataSource' 経由でデータベースにバインドします。バインドするには、'Form1' で 'DBGrid' コンポーネントを選択し、[オブジェクト インスペクタ] で 'DataSource' プロパティを 'DataSource1' に設定します。これで、'DBGrid' でデータベース内のテーブルを表示することができます。

データ・ソースの設定機能は非常に強力です。データ・ソースの設定により、追加のプログラミングなしで多数のコンポーネントがデータ・ソース経由でデータベースにアクセスすることができます。'DBGrid' は、'DataSource' プロパティ経由でデータベースにアクセスするコンポーネントの例です。

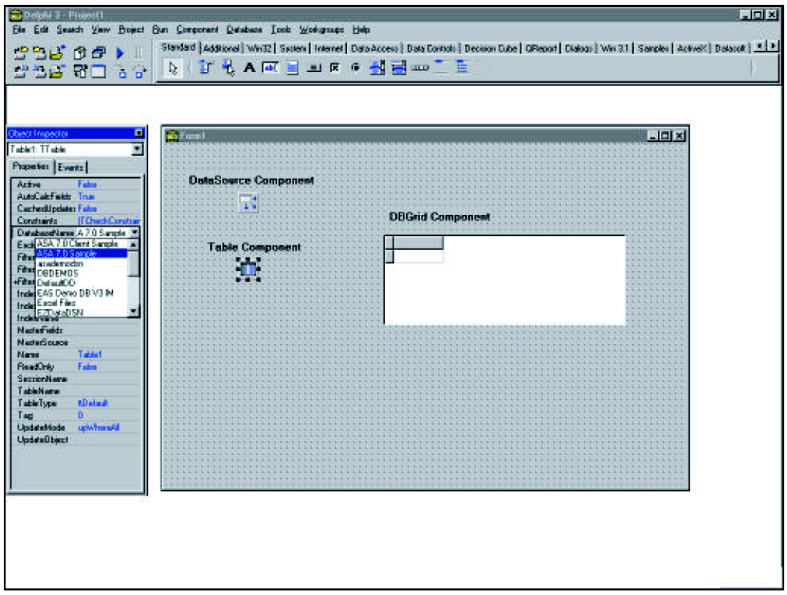

図 5 :

コンポーネントを相互にバインドする場合には、エラーが発生することがあります。たとえば、'TTable' コンポーネントを 'DBGrid' コンポーネントにバインドすると、'Access violation' エラーが発生します。これは、Delphi のバグです。このエラーの詳細については、24 ページの「DBGrid の例」を参照してください。

Delphi 経由で ASA データベースに接続する

BDE

前の項では、コントロールのバインドについて、'TTable' コンポーネントを ODBC データ・ソースにバインドする例を使用して説明しました。コンポーネントを ODBC データ・ソースにバインドするのは、そのソースに対応付けられたデータベース固有の情報に他のコンポーネントがアクセスできるように ASA データベース・エンジンを起動するためです。'TTable' コンポーネントの場合は、ASA データベース・エンジンを起動するには、以下の手順に従います。

1. 設計時または実行時に、'TableName' プロパティを設定します。

2. 'Active' プロパティを True に設定します。

Delphi で ASA データベース・エンジンを起動できる他の BDE コンポーネントとして 'TDatabase' を設定するには、以下の手順に従います。

1. コンポーネントを選択してフォーム上に配置します。

2. [オブジェクト インスペクタ] で、またはコードを使用して、'AliasName' プロパティを設定します。このプロパティは、ODBC データ・ソースです。

3. 'DatabaseName' プロパティを設定します。通常は 'AliasName' と同一です。

4. 'Connected' プロパティを 'True' に設定すると、データベース・エンジンが起動します。

ODBCExpress

ODBCExpress を使用した ASA データベース・エンジンへの接続は、BDE を使用した 'TDatabase' コンポーネントの使用と同様に行います。データベース・エンジンを起動するには、以下の手順に従います。

1. コンポーネント・パレットの [ODBCExpress] タブで、'THdbc' コンポーネントを選択し、フォーム上に配置します。

2. [オブジェクト インスペクタ] で、またはコードを使用して、'DataSource' プロパティを ODBC データ・ソースに設定します。

3. 'Connected' プロパティを 'True' に設定します。

Titan SQLAnywhere Developer

使用するデータベースのエイリアス名を作成したら (詳細については 8 ページの「Titan のインストールおよび Titan のエイリアス作成」を参照)、Titan コンポーネントを使用することができます。'TtsTable' または 'TtsDatabase' のいずれかのコンポーネントをフォームに配置します。'TtsTable' コンポーネントの場合は、以下の手順に従います。

1. 'DatabaseName' プロパティを、使用するデータベース用に作成したエイリアス名に設定します。

2. 'TableName' プロパティをデータベース内のテーブルに設定します。

3. 'Active' プロパティを 'True' に設定します。

'TtsDatabase' コンポーネントの場合は、以下の手順に従います。

1. 'AliasName' プロパティを、使用するデータベース用に作成したエイリアス名に設定します。

2. 'Connected' プロパティを 'True' に設定します。

Titan インタフェースを使用するときに ASA データベースを自動起動すると、問題が発生する場合があります。その場合は、BDE および ODBCExpress の場合と同じ方法でデータベース・エンジンを起動します。ASA エンジンを自動起動しようとすると、'Database Name required to start server' というエラーが発生することがあります。このエラーを回避するには、以下の手順に従います。

1. BDE または ODBCExpress のコンポーネントを使用して、必要なデータベース・エンジンを起動します。この方法で起動する場合は、Titan と BDE または ODBCExpress の間でエイリアス名が同一であることを確認します。

2. Titan コンポーネントを使用して、すでに起動されているエンジンに接続します。

NativeDB for SQL Anywhere

NativeDB コンポーネントを使用して ASA データベース・エンジンを 起動する手順は、BDE、ODBCExpress、Titan　の場合とは少し異なります。NativeDB　コンポーネントである 'TAsaSession' を使用して、Watcom SQL 4 から ASA　7 までの任意のバージョンの ASA に接続することができます。たとえば、ASA 7　のエンジンに接続するには、以下の手順に従います (図 6：を参照)。

1. 'TAsaSession' コンポーネントをフォームに配置します。

2. [オブジェクト インスペクタ] で、'LibraryFile' プロパティを 'dblib7.dll' に設定します。この dll は、使用する ASA のバージョンによって異なります。たとえば、ASA 6 の場合は 'dblib6.dll'、ASA 5.x の場合は 'dbl50t.dll' のようになります。

3. 'LoginDatabase'プロパティを、データベースが保存される場所のパスおよび名前 ('C:\Program Files\Sybase\SQL Anywhere 7\asademo.db' など) に設定します。実行中のエンジンに接続する場合は、'LoginDatabase'　プロパティにはデータベース名だけを設定すれば接続できます。

4. 'LoginEngineName'　プロパティに名前を指定します。この名前がエンジン名になります。通常は、データベース名と同じ名前を指定します。このプロパティを指定しない場合は、'LoginDatabase'　プロパティで指定したデータベースの名前がエンジン名として使用されます。

5. 'LoginUser' を正しいユーザ ID('dba' など) に設定します。

6. 'LoginPassword' を 'LoginUser' プロパティで指定したユーザに対応する正しいパスワード ('sql' など) に設定します。

7. [オブジェクト インスペクタ] で、'ServerParams' プロパティに dbeng7 の接続パラメータ ('start=dbeng7.exe' など) を設定します。

8. 'ServerType' プロパティを 'stServer' に設定します。

以上の設定が終了したら、'Connected' プロパティを 'True' に設定すると、データベース・エンジンが起動します。

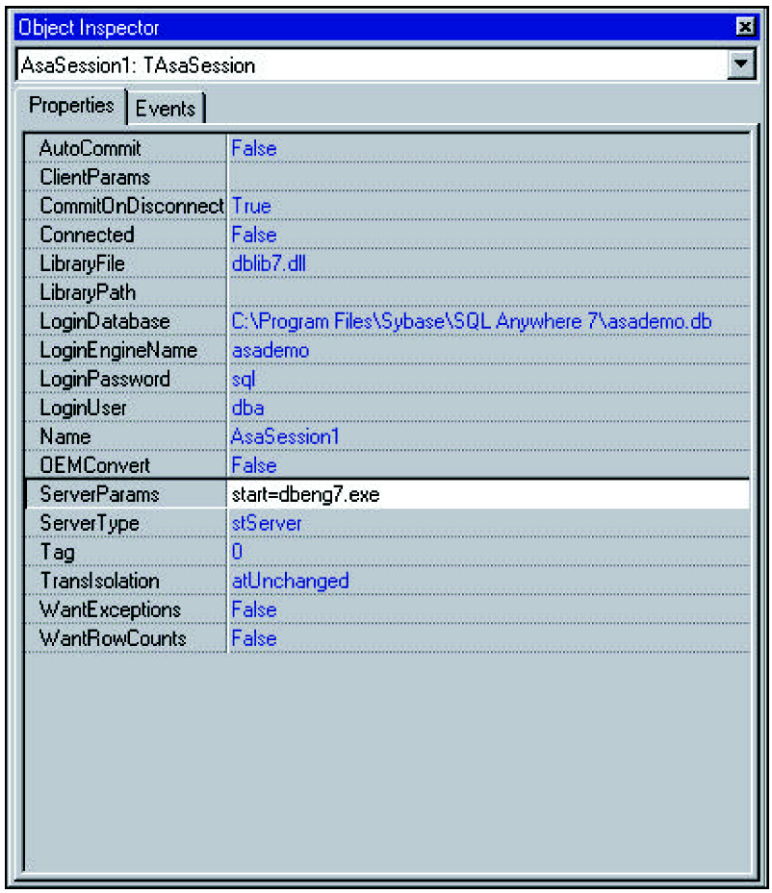

図 6 :

'TAsaSession' コンポーネントを設定したら、データベース・アクセス・コンポーネントを使用することができます。データベース・アクセス・コンポーネントを設定するには、以下の手順に従います。

1. 'TAsaDataset' コンポーネントをフォームに配置します。

2. ドロップダウン・メニューを使用して、'Session' プロパティを先に指定した TAsaSession コンポーネントの名前 ('AsaSession1' など) に設定します。

3. データベース内のテーブルにアクセスするように 'SQL' プロパティを設定します。このプロパティが設定されていない場合は、エラーが発生します。

4. 'Active' プロパティを 'True' に設定します。

'TAsaDataset'コンポーネントで'Active'プロパティをTrueに設定した場合は、先に'TAsaSession' コンポーネントの 'Connected' プロパティを 'True' に設定する必要はありません。'Active' プロパティを 'True' に設定した場合は、'TAsaSession' コンポーネントで設定したデータベース・エンジンが自動的に起動します。

BLOB の例

ASA データベースを Borland Delphi で使用する場合の最も一般的な問題の 1 つに、バイナリ大規模オブジェクト (BLOB) に関する問題があります。BLOB は、大きなデータ・セットで、そのサイズのために特別な方法で処理する必要があります。BLOB は、通常は画像ファイルやサウンド・ファイルです。ここで説明するすべての BLOB はビットマップですが、バイナリに変換された任意の種類の情報を BLOB として使用できます。

BDE

BLOB に関する問題は、デフォルトのエンジンである BDE for Delphi 3 を使用する場合に発生します。1.4MB を超える BLOB の操作は、Delphi ではサポートされていない可能性があります。ただし、このサイズ以下の BLOB は問題なく操作することができます。BDE を使用して BLOB をデータベースに挿入する例を以下に示します。

例 1：BDE を使用して BLOB をデータベースに挿入する

```
// このプロシージャは、BLOB をテーブルに格納します
procedure TForm1.LoadBlobClick(Sender: TObject);
var
  nextnum : Integer;
begin
// このコード部分は、テーブルを開きます。開くのに失敗した場合は、
// エラー・メッセージを表示します。
  try
    Table1.Open;
  except
    ShowMessage('Unable to Open Table');
    Table1.Close;
  end;
// テーブルがすでに開かれているかどうかを確認します。
  if Table1.Active = True then
  begin
// テーブルにローが含まれていない場合は、nextnum を 0 に設定します。
// それ以外の場合は、'nextnum' を 'keyfld' カラムの最後の数値に 1 加算した
// 値に設定します。
    if(Table1.RecordCount = 0) then
      nextnum := 1
    else
    begin
// ** このコードが、値を nextnum に代入するのに適した方法でない理由、および
// これよりも適した方法については 「付録 C：プライマリ・キーの問題」を
// 参照してください。**
      Table1.Last;
      nextnum := Table1.FieldByName('keyfld').asInteger + 1;
    end;
// この部分は、ローをテーブルに挿入し、'keyfld' カラムに 'nextnum' を入力し、
//'Edit1' ボックスを使用して指定したパスおよび名前を
//'imagefld' カラムに入力します。
    Table1.Insert;
    Table1.FieldByName('keyfld').Value := (nextnum);
    TBlobfield(Table1.FieldByName('imagefld')).LoadFromFile(Edit1.Text);
    Table1.Post;
```

```
      Table1.Close;
    // この部分は、BLOB を 'Image1' ボックスに表示します。
      Image1.Picture.LoadFromFile(Edit1.Text);
      StatusBar1.SimpleText := 'Image loaded into table';
    end;
  end;
```

以下に、いくつか注意すべき点を示します。

1. この例では、BLOB は 'TImage' ボックスを使用して表示されるビットマップ・ファイルです。

2. 'keyfld' および 'imagefld' は、テーブル内のカラムです。'keyfld' には整数値のみが使用でき、ASA ではカラムのデフォルトに ' autoincrement' が設定されます。また、'keyfld' はプライマリ・キーにも設定されています。つまり、'keyfld' カラムのすべての数値はユニークになっている必要があります。詳細については、41 ページの「付録 C : プライマリ・キーの問題」を参照してください。'imagefld' には、'long binary' 型だけを入力できます。'long binary' 型は、BLOB を格納するために使用します。

注 : ここで使用するテーブル 'blob' に対して実行する Create table 文は、以下のとおりです。
CREATE TABLE blob (keyfld int primary key default autoincrement, imagefld long binary)

3. ビットマップが 1.4MB を超える場合は、以下のエラー・メッセージが表示されます。'Invalid BLOB length'

図 7 : は、前述のコードで作成したサンプル・アプリケーションです。[Pick your Alias] の下のリスト・ボックスには、実行時のすべての ODBC データ・ソース名が表示されます。この図では、画像の格納先データベースおよびテーブルに接続する ODBC データ・ソースは 'DelphiDemo' になっています。最初の Edit ボックスに表示されるパスおよびファイル名は、[Load Blob into table] ボタンをクリックしたときに表示され、DelphiDemoDb というデータベースに追加される画像の場所と名前です。

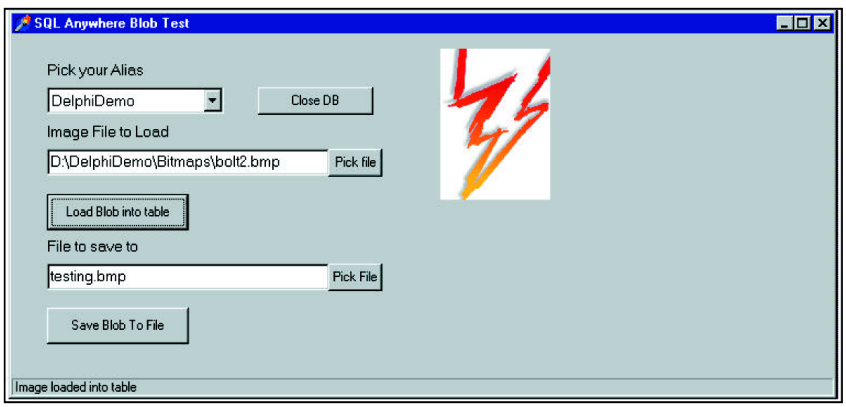

図 7 :

挿入の場合とは異なり、BDE を使用して BLOB をファイルに保存する場合には問題は発生しません。以下に、BLOB をファイルに保存するコード例を示します。'imagefld' は、テーブル内のカラムの名前です。

例 2：BDE を使用して BLOB をファイルに保存する

```
// このプロシージャは、テーブルから BLOB を取得し、ファイルに保存します。
procedure TForm1.SaveBlobToFileClick(Sender: TObject);
begin
// テーブルを開きます。開くのに失敗した場合は、エラー・メッセージを
// 表示します。
  try
    Table1.Open;
  except
    ShowMessage('Unable to Open Table');
    Table1.Close;
  end;
// テーブルが開いている場合は、最後の値にジャンプし、'Edit2' ボックスを使用
// してユーザが指定したパスおよびファイル名で画像を保存します。
  if Table1.Active = True then
  begin
    Table1.Last;
    TBlobField(Table1.FieldByName('imagefld')).SaveToFile(Edit2.Text);
    Table1.Close;
// 画像を 'Image2' ボックスに表示します
    Image2.Picture.LoadFromFile(Edit2.Text);
    StatusBar1.SimpleText := ('Image saved to ' + Edit2.text);
  end;
end;
```

ODBCExpress

ODBCExpress エンジンをインストールし、Delphi で使用することができます。インストール手順については、35 ページの「付録 A：Delphi で ODBCExpress インタフェースを使用するための設定」を参照してください。インストールすると、Delphi 専用の ODBCExpress コンポーネントが使用可能になります。BDE コンポーネントの代わりにこのインタフェースを使用する、または併用することで、Windows アプリケーション を作成することができます。ODBCExpress は、カーソルを完全にサポートしています。つまり、フロント・エンドでローをキャッシュする必要なく、結果セットで前方および後方に移動することができます。BLOB は必要に応じてデータベースから取得するため、フロント・エンドのキャッシュが原因でメモリ不足が発生することなく、結果セットで必要なだけ BLOB を使用することができます。Delphi 3 および ASA 6.0.3 データベースと ASA 7.0 データベースを使用してこれをテストすると、1.4MB を超えるものも含めてさまざまなサイズの BLOB を操作することができます。例 3 では、ODBCExpress エンジンを使用して BLOB を Delphi に挿入する方法を説明しています。この例の BLOB はビットマップです。また、'keyfld' および 'imagefld' はテーブル内のカラムで、データ型はそれぞれinteger型(およびデフォルトのautoincrement)、long binary 型です。

例 3：ODBCExpress エンジンを使用して BLOB をデータベースに挿入する

```
// このプロシージャは、BLOB をテーブルに格納し、画像を表示します。
procedure TForm1.LoadBlobClick(Sender: TObject);
var
  imagefld : TMemoryStream;
  nextnum : integer;
begin
// メモリ・ストリーム・オブジェクトを作成します。
   imagefld := TMemoryStream.Create;
```

```
// データセットを特定のテーブル (testblob1) に設定し、データセットを開き ,
// テーブル内のローの数を特定します。
    OEDataSet1.Table := 'blob';
    OEDataSet1.Open;
    OEDataSet1.Last;
// これは、テーブルの最終ローに 1 加算した値になります。
    nextnum := OEDataSet1.FieldValues['keyfld'] + 1;
    with Hstmt1 do
    begin
//BLOB カラムの imagefld を使用した insert 文を設定および準備します
      SQL := 'INSERT INTO blob (keyfld, imagefld) VALUES (?, ?)';
      Prepare;
// 値をパラメータに割り当てます。
      imagefld.LoadFromFile(Edit1.Text);
//BindBinary メソッドを使用して BLOB をバインドします。
      BindInteger(1, nextnum);
      BindBinary(2, imagefld);
// 文を実行し、データソースに BLOB を挿入します。
      Execute;
    end;
// 実行後は、メモリ・ストリームを破棄することができます。
    imagefld.Free;
// この部分は、画像を画面に表示します。
    Image1.Picture.LoadFromFile(Edit1.Text);
end;
```

例 1 と同様に、変数 'nextnum' を使用して 'keyfld' カラムの値を指定します。より適した値の指定方法の詳細については、41 ページの「付録 C : プライマリ・キーの問題」を参照してください。また、構文が BDE と少し異なる点に注意してください。また、例には次のような行があります。

```
OEDataSet1.Table := 'blob';
```

これは、プログラミングでテーブル名をデータ・ソースに代入する方法です。他の方法として、例 1 および 2 では、フォーム上で 'TTable' コンポーネントを選択し、アクセスするテーブル名を [オブジェクト インスペクタ] で 'Table Name' プロパティに入力しています。ODBCExpress では、任意のサイズの BLOB を操作できるだけでなく、処理速度が非常に高速です。

図 8 は、前述のコードで生成したアプリケーションです。ここで、'DelphiDemo' は、'blob' というテーブルのあるデータベースに接続する ODBC データ・ソース名です。最初の編集ボックスに書き込まれるパスおよびファイル名は、[Load Blob into Table] ボタンをクリックしたときに表示され、テーブルにロードされる画像の場所と名前です。ここで表示される画像は、サイズが 1.4MB を超えているため、BDE を使用して表示することができません。

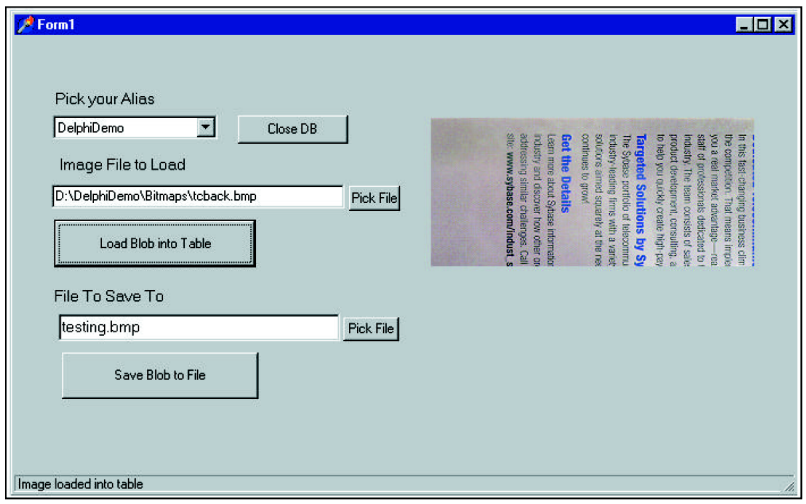

図 8：

ODBCExpress を使用した BLOB の保存は、BDE を使用した保存と似ています。保存を実行する方法を次の例で説明します。'blob' は、アクセスされるテーブル名です。'imagefld' は、'blob' 内のカラム名です。

例 4：ODBCExpress を使用して BLOB をファイルに保存する

```
procedure TForm1.SaveBlobClick(Sender: TObject);
begin
    OEDataSet1.Table := 'blob';
// テーブルを開きます。開くのに失敗した場合は、エラー・メッセージを
// 表示します
    try
      OEDataSet1.Open;
    except
      ShowMessage('Unable to Open Table');
      OEDataSet1.Close;
    end;
// テーブルが開いている場合は、テーブルの最終ローにジャンプし、
//'Edit2' ボックスを使用してユーザが指定したパスおよび名前を 'imagefld' カラム
// に入力します。
    if OEDataSet1.Active = True then
    begin
      OEDataSet1.Last;
      TBlobField(OEDataSet1.FieldByName('imagefld')).SaveToFile(Edit2.Text);
// この部分は、保存した画像を 'Image2' ボックスに表示します。
      Image2.Picture.LoadFromFile(Edit2.Text);
    end;
end;
```

Titan SQLAnywhere Developer

ODBCExpress と同様に、Titan SQLAnywhere Developer をインストールし、Delphi で使用することができます。この製品のインストール方法の詳細については、8 ページの「Titan のインストールおよび Titan のエイリアス作成」を参照してください。Titan SQLAnywhere

Developer をインストールすると、専用のコンポーネントが使用可能になり、標準の Delphi コンポーネントを使用することもできるように設計されています。Titan コンポーネントを使用する場合は、さまざまなサイズの BLOB を操作することができます。例 5 では、Titan SQLAnywhere Developer で BLOB をテーブルにロードする方法を説明しています。注意すべき点を以下に示します。

1. 'blob' は、BLOB(ここではビットマップ) が格納されるテーブルの名前です。

2. 'blob' には、'keyfld' と 'imagefld' の 2 つのカラムがあります。'keyfld' は、データ型が 'integer' で、'autoincrement' に設定されます。'imagefld' のデータ型は 'long binary' です。

3. この例の構文は、例 1 とほぼ同一です。異なる点は、以下のとおりです。

   i. 'tsTable' コンポーネントが 'TTable' コンポーネントの代わりに使用されるため、例 1 で 'Table1' となっている部分がこの例では 'tsTable1' になっています。

   ii. この例は、数値を 'keyfld' に挿入しません。例 1 では、変数 nextnum(テーブルの最終ローの 'keyfld' カラムの値に 1 を加算した値が代入されます) を使用して、数値を挿入しています。「付録 C：プライマリ・キーの問題」では、値を nextnum に代入する別の方法と、例 1 で使用する方法よりも例 5 で使用する方法を推奨する理由を説明しています。

   iii. この例には、'tsDatabase1.Commit' という行があります。Titan SQLAnywhere Developer を使用する場合には、変更をコミットするためのデフォルト値が BDE または ODBCExpress のデフォルトとは異なる値に設定されるため、この行が必要になります。コードにこの行を記述していない場合は、最初は画像がテーブルに格納されますが、テーブルを閉じると変更が無効になります。

例 5：Titan SQLAnywhere を使用して BLOB をファイルにロードする

```
procedure TForm1.LoadBlobButtonClick(Sender: TObject);
begin
  tsTable1.TableName := 'blob';
  try
    tsTable1.Open;
  except
    ShowMessage('Unable to Open Table');
    tsTable1.Close;
  end;
  if tsTable1.Active = True then
  begin
//BLOB をテーブル blob に挿入します。
tsTable1.Fields[0].Required := false; // この文は、autoincrement を有効にします。
    tsTable1.Insert;
    TBlobField(tsTable1.FieldByName('imagefld')).LoadFromFile(Edit1.Text);
    tsTable1.Post;
    tsDatabase1.Commit;
    tsTable1.Close;
//BLOB を Image1 に表示します。
    Image1.Picture.LoadFromFile(Edit1.Text);
  end;
end;
```

Titan SQLAnywhere Developer を使用して BLOB をファイルに保存するのは、単純なタスクです。'Table1' が 'tsTable1' になっている点を除き、コードは例 2 のものと同一です。

例 6：Titan SQLAnywhere を使用して BLOB をファイルに保存する

```
procedure TForm1.SaveBlobToFileButtonClick(Sender: TObject);
begin
// テーブルを開きます。開くのに失敗した場合は、エラー・メッセージを
// 表示します
  try
    tsTable1.Open;
  except
    ShowMessage('Unable to Open Table');
    tsTable1.Close;
  end;
// テーブルが開いている場合は、テーブルの最終ローにジャンプし、
//'Edit2' ボックスを使用してユーザが指定したパスおよび名前を
//'imagefld' カラムに入力します。
  if tsTable1.Active = True then
  begin
    tsTable1.Last;
    TBlobField(tsTable1.FieldByName('imagefld')).SaveToFile(Edit2.Text);
    tsTable1.Close;
    Image2.Picture.LoadFromFile(Edit2.Text);
    StatusBar1.SimpleText := ('Image saved to ' + Edit2.Text);
  end;
end;
```

NativeDB for SQL Anywhere

ODBCExpress および Titan SQLAnywhere Developer と同様に、NativeDB をインストールし、Delphi で使用することができます。NativeDB for SQL Anywhere のインストールの詳細については、10 ページの「NativeDB for SQL Anywhere のインストール」を参照してください。NativeDB コンポーネントは、Titan SQLAnywhere Developer の場合と同様に、標準の Delphi コンポーネントを利用できるように設計されています。さらに、NativeDB が提供するコンポーネントを使用して、データベースを操作します。

NativeDB コンポーネントを使用して、すべてのサイズの BLOB を操作することができます。例 7 では、NativeDB で BLOB をテーブルにロードする方法を説明しています。

例 7：NativeDB を使用して BLOB をデータベースにロードする

```
procedure TForm1. LoadBlobClick(Sender: TObject);
begin
// この部分は、テーブルを開きます。開くのに失敗した場合は、
// エラー・メッセージを表示します。
    try
      AsaDataset1.Open;
    except
      ShowMessage('Unable to Open Table');
      AsaDataset1.Close;
    end;
// テーブルがすでに開かれているかどうかを確認します。
// 開かれている場合は、選択された BLOB をデータベースにロードします。
    if AsaDataset1.Active = True then
```

```
    begin
// この部分は、ローをテーブルに挿入し、'imagefld' カラムに、
//'Edit1' ボックスを使用してユーザが指定したパスおよび
// ファイル名を入力します。
      AsaDataset1.Insert;
      TBlobField(AsaDataset1.FieldByName('imagefld')).LoadFromFile(Edit1.Text);
      AsaDataset1.Post;
      AsaSession1.Commit;
      AsaDataset1.Close;
// この部分は、BLOB を 'Image1' ボックスに表示します。
      Image1.Picture.LoadFromFile(Edit1.Text);
    end;
end;
```

注意すべき点を以下に示します。

1. この例では、どのテーブルがアクセスされるかが明示されていません。'TAsaDataset' コンポーネントの 'SQL' プロパティは、'Select * from blob' に設定されます。この文は、テーブル 'blob' がアクセスされることを示します。テーブル 'blob' には、'keyfld' と 'imagefld' の 2 つのカラムがあります。'keyfld' は、データ型が 'integer' で、デフォルトの 'autoincrement' に設定されます。'imagefld' のデータ型は 'long binary' です。

2. BDE、ODBCExpress、Titan とは異なり、NativeDB はデフォルト の 'autoincrement' を自動的に設定します。これは、AsaDataset.Fields[0].Required を最初に false に設定することで行います。詳細については、「付録 C：プライマリ・キーの問題」を参照してください。

3. また、やはり明示されていませんが、'TAsaDataset' コンポーネントの 'ReadOnly' プロパティが自動的に 'False' に設定されます。つまり、BLOB は実際にはテーブルにロードされません。このプロパティを 'True' に設定した場合は、BLOB がテーブルに挿入されます。

NativeDB を使用して BLOB をファイルに保存するコードは、BDE コンポーネントを使用してファイルを保存する場合とほぼ同一です。実際には、例 8 の NativeDB コンポーネントを使用する構文と、例 2 の BDE コンポーネントを使用する構文は、例 2 で 'Table1' となっている部分が例 8 では 'AsaDataset1' になる点を除いて同一です。また、NativeDB では、'AsaSession1' コンポーネントの 'AutoCommit' が 'False' に設定されている場合に限り、Titan の場合と同様に明示的にポスティングをコミットする必要があります。

例 8：NativeDB を使用して BLOB をファイルに保存する

```
procedure TForm1. SaveBlobToFileClick(Sender: TObject);
begin
// テーブルを開きます。開くのに失敗した場合は、エラー・メッセージを
// 表示します。
    try
      AsaDataset1.Open;
    except
      ShowMessage('Unable to Open Table');
      AsaDataset1.Close;
    end;
// テーブルが開いている場合は、最終値にジャンプし、
//'Edit2' ボックスを使用してユーザが指定したパスおよび名前で
// 画像を 'imagefld' に保存します。
    if AsaDataset1.Active = True then
```

```
    begin
      AsaDataset1.Last;
      TBlobField(AsaDataset1.FieldByName('imagefld')).SaveToFile(Edit2.Text);
      AsaDataset1.Close;
// 画像を 'Image2' ボックスに表示します。
      Image2.Picture.LoadFromFile(Edit2.Text);
    end;
end;
```

DBGrid の例

'DBGrid' は、Delphi のコンポーネント・パレットの [Data Control] タブに追加できるデータ・アウェア・コンポーネントです。データ・アウェア・コンポーネントは、データ・ソースにバインドされたデータベースに格納されたデータを認識するコンポーネントです。Delphi で ASA を使用する場合には、'DBGrid' コンポーネントを使用すると問題が発生することが判明しています。これらの問題を解決するため、[Delphi アプリケーション] チェックボックスが追加されました。このチェックボックスは、ODBC データ・ソースを設定するときに表示されます。このチェックボックスの表示される場所は、3 ページの図 3 で確認してください。'DBGrid' コンポーネントを使用する場合は、使用する ODBC データ・ソースの [Delphi アプリケーション] チェックボックスの選択を解除してください。注：'DBGrid' コンポーネントが ODBCExpress および BDE の各インタフェースでブックマークを使用することが判明しています。ブックマークの詳細については、18 ページの「ODBCExpress」を参照してください。

BDE

BDE を使用して 'DBGrid' を操作する場合は、最初にいくつかの設定が必要です。

1. 'Table'、'DataSouce'、'DBGrid' の各プロパティは、相互にバインドされています (コンポーネントのバインドについては、11 ページの「コントロール・プロパティのバインド設定例」を参照してください)。

2. 'DBGrid' でデータベースの情報を表示するには、'Active' プロパティを設定する必要があります。'Table' コンポーネントをフォームで選択すると、[オブジェクト インスペクタ] の [Properties] タブに 'Active' プロパティが表示されます。これを使用して、このプロパティを設定することができます。'Active' プロパティは、'True' に設定する必要があります。'True' に設定していない場合は、アプリケーションを実行しても 'DBGrid' にデータが表示されません。'Active' プロパティは、'Table Name' を指定した後にだけ 'True' に設定することができます。

図 9 : に、'DBGrid' アプリケーションを示します。この図では、'TTable' コンポーネントの 'Active' プロパティが 'True' に設定されています。

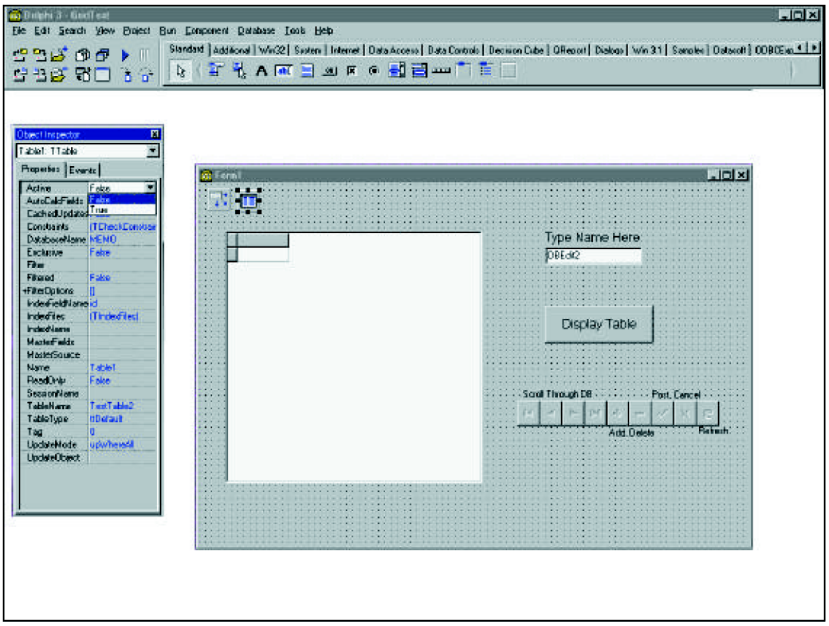

図 9 :

'Active' プロパティを 'True' に設定すると、Delphi の終了時に、'Access violation at address 1F4ADCD4. Read of address 1F4ADCD4' というエラーが発生します。このエラーは重大なエラーではなく、アプリケーションには影響はありません。このエラーは、ASA エンジンとアプリケーションの間の接続が切断されたと Delphi が認識し、そのときにデータベースからの情報を表示しようとしているために発生するものです。

アプリケーションをコンパイルすると、プロジェクトと同名の実行可能ファイルが作成されます。この実行可能ファイルは、プロジェクトと同じ場所に保存されます。このファイルを (Windows のエクスプローラなどから) 実行すると、作成したアプリケーションが実行されます。この実行可能ファイルを閉じても前述のエラーは発生しないため、アプリケーションへの影響はありません。

前述のエラーを回避するには、'Active' プロパティを 'true' に設定し、終了時にこのプロパティを false に戻すイベントを記述します。たとえば、テーブルのデータを表示するボタンを作成し、フォームを閉じるときに 'Active' プロパティを 'false' に設定する 'OnClose' イベントを記述することができます。以下の 2 つのプロシージャで、これらのタスクを実行することができます。これらのプロシージャを使用するには、[オブジェクト インスペクタ] で 'Active' プロパティを 'false' に、'Table Name' プロパティをブランクに設定する必要があります。また、クリックされるボタン (図 9 : のフォームでは 'Display Table' というボタン) の 'Name' を DisplayButton に設定し、[オブジェクト インスペクタ] の [イベント] タブでイベント 'OnClick' を 'DisplayButtonClick' プロシージャに設定します。最後に、フォームの [イベント] タブで、イベント 'OnClose' を 'FormClose' に設定します。

```
procedure TForm1.DisplayButtonClick(Sender: TObject);
begin
// テーブル・コンポーネントをテーブル 'Grid' にバインドします
    Table1.TableName := 'Grid';
//'Active' プロパティを True に設定します。
```

```
    Table1.Active := True;
end;
// ユーザがフォームを閉じるときに 'Active' プロパティを False に設定します。
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    Table1.Active := False;
end;
```

Delphi 5 を使用した場合には、イベントを使用して 'Active' プロパティを 'True' および 'False' に設定する必要がないことがわかりました。この場合、Delphi を閉じてもエラーは発生しません。ただし、'Active' プロパティを 'True' のままにしないようにイベントを記述することを推奨します。

'DBGrid' を使用してローの追加および削除、前後のレコードへの移動、テーブルの編集を行うには、'DBNavigator' コンポーネントを使用するのが簡単です。その場合は、'DBNavigator' コンポーネントを選択し、[オブジェクト インスペクタ] で 'DataSource' プロパティを設定して、このコンポーネントを 'DataSource' にバインドする必要があります。'DBNavigator' を使用する場合に、ローを挿入すると問題が発生することがあります。[Navigator] バーにある [+] ボタンを実行時にクリックすると、値を入力する箇所にブランクのローが挿入されます。入力された値を通知するには、[Navigator] バーにある [3] ボタンをクリックします。この時点では、テーブルを更新するまで、追加したローは表示されません。更新するには、[?]([Refresh]) ボタンをクリックします。ただし、クリックした後に、'Table does not support this operation because it is not uniquely indexed' というエラーが表示されることがあります。この問題を修正するには、'Table' コンポーネントをフォームで選択し、[オブジェクト インスペクタ] で 'IndexFieldName' プロパティをテーブル内のいずれかのカラム名に設定します。これにより、テーブル内のローの識別が可能になり、テーブルを更新して新しいローを表示できるようになります。また、この手順により、テーブルのローがこのカラムを基準にしてソートされます。たとえば、'IndexFieldName' を integer 型のフィールドである 'id' に設定すると、このカラムの数値を基準にしてローが昇順でソートされます。

コード中で、[オブジェクト インスペクタ] を使用して 'Active' プロパティを 'true' に設定すると、テーブル内のローをフェッチする SQL 文が実行されます。'IndexFieldName' プロパティがブランクのままの場合は、'SELECT id, name FROM Grid' という SQL 文 が実行されます。ここで、id と name はテーブル Grid 内のすべてのカラムの名前です。'IndexFieldName' がテーブル内のカラム (ここでは id) に設定されると、その代わりに SELECT id, name FROM Grid ORDER BY id という文が実行されます。ORDER BY 句は、ローを 'IndexFieldName' プロパティで指定されたカラムを基準にしてソートします。ローを特定の順序でフェッチした後は、そのテーブルにユニークなインデックスがあるためテーブルを更新できると Delphi が認識します。選択されたカラムがプライマリ・キーでない場合は、インデックスをカラムに設定することを推奨します。これにより、パフォーマンスが向上します。インデックスの詳細については、"SQL Anywhere マニュアル " を参照してください ([スタート] → [プログラム] → [Sybase SQL Anywhere 7] → [SQL Anywhere マニュアル] で表示します)。

図 10 : は、'IndexFieldName' プロパティをカラム 'id' に設定したものです。'DBNavigator' コンポーネントを、円で囲んで示しています。

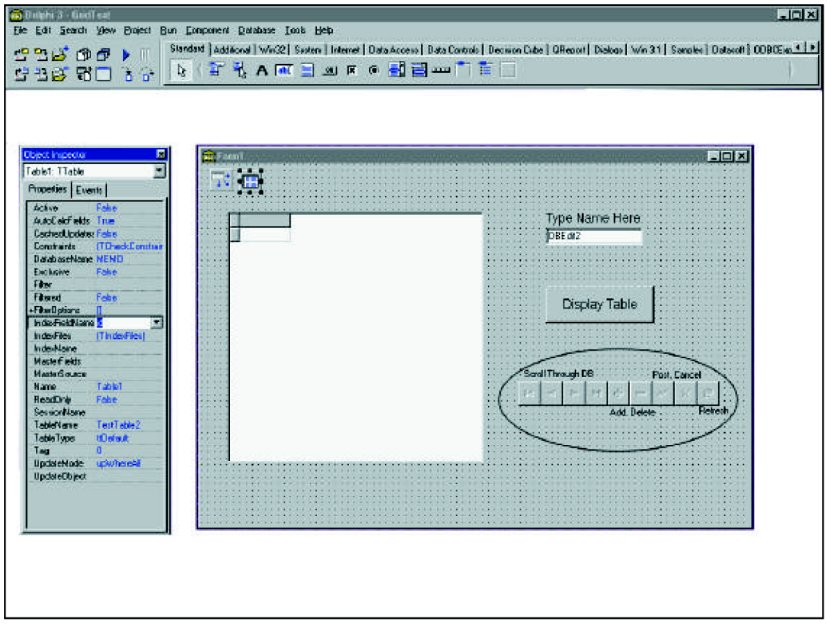

図 10 :

データ型が 'Integer 型 ' で、デフォルトの 'autoincrement' に設定されているカラムがある ASA テーブルを使用する際にも、問題が発生することがあります。これは、'DBNavigator' を使用して 'DBGrid' にローを挿入した場合に発生します。Delphi では、このカラムに値が指定されていない場合に、自動的に値の設定がされることを認識しません。ローを挿入するときに、他のカラムに値を入力し、'DBNavigator' コンポーネントで [3] をクリックして新しいローを通知するとします。このとき、本来であれば autoincrement によってそのローの数値が加算されます。しかし実際には、'Field ID must have a value' というエラーが発生します。ここでの 'ID' とは、値を指定しなくても自動的に数値が加算されるように default autoincrement を設定したカラムの名前です。このエラーの原因は、Delphi の TField.Required プロパティに 'True' が自動的に設定されるためです。この問題を修正し、'DBGrid' で autoincrement が使用できるようにするには、以下の手順に従います。

1. 'Form' を選択し、フォーム上のコンポーネントが強調表示されていない状態にします。[オブジェクト インスペクタ] の上部に、Form1: TForm または (ユーザ指定フォーム名) : TForm と表示されていることを確認します。

2. [オブジェクト インスペクタ] の [イベント] タブを選択し、行 'OnShow' イベントを選択します。'OnShow' イベントの右側の空白部分をクリックします。コードを記述するための 'FormShow' というプロシージャ・ヘッダが作成されます (注 : 'OnShow' イベント内にこのプロシージャを作成することで、アプリケーションを実行するとすぐに autoincrement 機能が動作するようになります)。

3. プロシージャ 'FormShow' に以下のようなコードを記述します。

```
procedure TForm1.FormShow(Sender: TObject);
begin
    Table1.Fields[0].Required := False;
end;
```

Delphi では、テーブル内の各カラムに、0 から始まる番号が対応付けられています。上記のコードの 0 は、ASA でデフォルの ' autoincrement' が設定されているカラムを示します。

上記のプロシージャは、'Active' プロパティが 'True' に設定されている場合にだけ機能します。それ以外の場合は、アプリケーションを実行すると、'List index out of bounds (0)' というエラーが発生します。これは、'Active' プロパティが 'False' の場合には、フォームで 'DBGrid' にカラムが表示されないためです。これを解決するには、以下のコードを追加します。

```
Table1.Fields[0].Required := False;
```

このコードは、前述のボタンクリックイベントなどの他のイベントに追加します。

'DBGrid' を ASA データベースで使用する場合に、テーブル内に表示されるカラムのデータ型が 'long varchar' 型のときに、'DBGrid' でデータベース内の実際のカラム値の代わりに '(MEMO)' と表示されるという問題が発生します。'DBGrid' 内のカラムでは、long varchar 型のサイズの文字列を表示できないからです。このサイズのデータ型を表示可能なデータアウェアコンポーネントは、'DBMemo' コンポーネントだけです。特定のカラム値を表示する 'DBMemo' コンポーネントを設定するには、以下の手順に従います。

1. フォーム上の 'DBMemo' コンポーネントを選択します。

2. 'DataSource' プロパティを正しく設定します。

3. 'DataField' プロパティを、表示するカラムの名前に設定します。

前述のすべての例および問題では、'TTable' コンポーネントを使用しています。データベース内のテーブルにアクセスするために使用可能な他のコンポーネントに、'TQuery' コンポーネントがあります。このコンポーネントを使用すると、'DBGrid' コンポーネントを使用した時より柔軟にテーブルを表示することができます。'TTable' を使用した場合は、'DBGrid' でテーブル内のすべてのカラムおよびローが表示されます。'TQuery' を使用すると、必要なカラムおよびローだけを表示する SQL 文を作成することができます。SQL 文の実行後に表示されるレコードを、クエリの結果セットと呼びます。

図 11：は、'DBGrid' コンポーネントに表示されたクエリの結果セットです (Query コンポーネントは円で囲んで示しています)。「'Type ORDER BY or WHERE clause Here'」という見出しの 'Edit1' ボックスが空白のときに、[Display Table] ボタンをクリックすると、結果セットはテーブル全体になります。'Where' 句を入力して [Display Table] ボタンをクリックすると、選択したローおよびカラムのグループが表示されます。

'Where' 句を使用した SQL 文の例を以下に示します。

```
SELECT id FROM Grid WHERE name = 'your name'
```

この文で、'id' は、'Grid' テーブル内の表示対象のカラムを示します。ただし、'name' カラムの値が 'your name' のローだけが表示されます。

'Order By' 句を入力して [Display Table] ボタンをクリックすると、テーブル全体が特定のソート順で表示されます。'Order By' 句を使用した SQL 文の例を以下に示します。

```
SELECT * FROM Grid ORDER BY id
```

この文は、'Grid' テーブルのすべての行およびカラムを選択し、'id' カラムの値を基準にして行をソートします。

両方の句を入力して [Display Table] ボタンをクリックすると、選択した行およびカラムが特定の順序で表示されます。両方の句を使用した SQL 文の例を以下に示します。

```
SELECT * FROM Grid WHERE name = 'your name' ORDER BY id
```

この文は、'Grid' テーブルで 'name' カラムの値が 'your name' の行をすべて選択し、'id' カラムの値を基準にしてソートして表示します。

SQL 文を使用して 'DBGrid' に表示するデータを選択するため、前述の 'IndexFieldNames' プロパティは存在せず、必要ありません。表示されるテーブルをソートするには、'Order By' 句を SQL 文で指定します。修正後にテーブルを更新するには、SQL 文を再実行します。

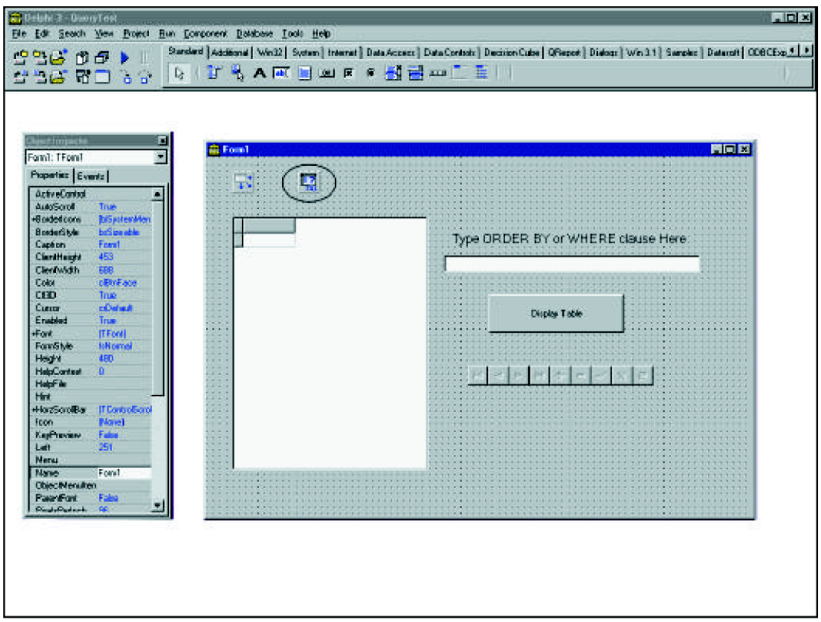

図 11 :

'TQuery' コンポーネントを 'DBGrid' コンポーネントと併用すると、さらに詳細な処理が可能です。データベース内の 1 つのテーブルに対するクエリの結果セットを表示する以外に、SQL 文を変更することで、データベース内の複数のテーブルに対するクエリの結果セットを表示することもできます。複数のテーブルを対象とした SQL 文の例を以下に示します。

```
SELECT * FROM Grid, AnotherGrid WHERE Grid.Name = 'your name'
```

'Grid. Name' は、'Grid' テーブル内のカラムです。この文は、Grid および AnotherGrid の両方の全カラムを選択し、'Grid' テーブルで Name が 'your name' の行だけを表示します。

クエリに 2 つの異なるテーブルが関係するため、結果セットを編集することはできません。つまり、このインタフェースを使用した通常の方法では、行の挿入、行の削除、既存の行の編集を実行することはできません。

ODBCExpress

'DBGrid' コンポーネントを ODBCExpress で使用する方法は、BDE で使用する場合とは少し異なります。ODBCExpress には、'Table' コンポーネントがなく、その代わりに 'OEDataSet' コンポーネントを使用します。'DBGrid' を 'OEDataSet' にバインドする方法が必要であるため、BDE の 'DataSource' コンポーネントをフォームに追加します。これで、Grid がデータベースと通信することができます。'OEDataSet' には、'SQL' プロパティがあります。このプロパティで SQL 文をデータベース内のテーブルに対して実行し、情報をさまざまな形式で表示することができます。これにより、'OEDataSet' コンポーネントで BDE の 'Query' コンポーネントと同様の機能が実現されます。'DBGrid' コンポーネントを BDE で使用する場合と同様に、'DBGrid' でテーブルを表示するには、'OEDataSet' の 'Active' プロパティを True に設定する必要があります。このコンポーネントが設計時に設定されている場合は、BDE の場合と同様に、Delphi を閉じるとアクセス違反エラーが発生します。この問題およびその推奨解決方法については、25 ページで説明しています。BDE と異なる他の点として、ODBCExpress はスクロール可能カーソルを使用します。Delphi 5 を使用した場合には、この問題は発生しないことが判明しました。これらの問題の発生は ASA 6.0.3 で判明したものですが、ASA 7.0 でも発生します。問題が解決された ASA のバージョンについては、「付録 D」を参照してください。

ODBCExpress を 'DBGrid' コンポーネントとともに使用する場合に、いくつか問題が発生することがあります。その 1 つとして、ローをテーブルに挿入できない場合があります。明示されていませんが、'OEDataSet' コンポーネントでは、SQL 文の結果セットに対してデフォルト値が ReadOnly になっています。したがって、ローの挿入、削除、更新を行うと、'Option value out of range' というエラーが発生します。このエラーを回避するには、以下の手順に従います。

1. フォーム上の 'OEDataSet' コンポーネントを選択します。

2. [オブジェクト インスペクタ] で、'hStmt' プロパティの左にある [+] をダブルクリックします。

3. 'Concurrency Type' というプロパティを選択し、ドロップダウン・リストから 'Values' または 'Row Versions' を選択します。

4. 'OEDataSet' には、自動的に 'False' に設定される 'Editable' プロパティもあります。データを実行時に編集する場合には、このプロパティを 'True' に設定する必要があります。

このドキュメントで説明する 3 つのインタフェースのうち、使用するカーソルの種類を変更できるのは ODBCExpress だけです。カーソルの種類を変更するには、以下の手順に従います。

1. フォーム上の 'OEDataSet' コンポーネントを選択します。

2. [オブジェクト インスペクタ] で、'hStmt' プロパティの左にある [+] をクリックします。

3. 展開されたリストに 'CursorType' プロパティが表示されます。[Forward Only]、[Dynamic]、[Keyset Driven]、[Static] のいずれかをドロップダウン・リストから選択します。

これらのカーソルの詳細については、Datasoft が提供する ODBCExpress ダウンロードに含まれる、OEWPaper.pdf というホワイトペーパーの「Cursor Types」を参照してください。

[Static] および [Keyset Driven] のカーソルで結果セットを構成するローは静的なままであるため、これらのカーソルのブックマーク値を取得することができます。ブックマークは、カーソル内のローを特定するために使用する値で、通常は結果セットでの位置を基準にしています。ブックマークは、結果セットでのみ有効です。[Forward only] および [Dynamic] のカーソルは、通常はローのブックマーク値を戻すことはできません 。したがって、[Dynamic] を選択した場合は、data-aware Controls で表示される多数のローを含む結果セットをスクロールしようとすると、問題が発生します。結果セットをスクロールしようとすると、多くの場合は 'Fetch type out of range' というエラーが発生します。結果セットでローの挿入、削除、更新を行った場合も、同じエラーが発生します。[Forward Only] カーソルを選択した場合は、結果セットを後方にスクロールすると同じエラーが発生します。この場合も、結果セットでローの挿入、削除、更新を行うと、同じエラーが発生します。したがって、ブックマークが必要な 'OEDataSet' コンポーネントを使用する場合は、[Keyset Driven] または [Static] のカーソルを使用することをお勧めします。Delphi 5 には、[Forward only] および [Dynamic] のカーソルを使用してもエラーが発生しないようにするオプションがあります。これらのオプションを設定するには、以下の手順に従います。

1. フォーム上の 'OEDataSet' コンポーネントを選択します。

2. [オブジェクト インスペクタ] で、'Cached' プロパティを 'True' に設定します。

3. 'hStmt' プロパティの左にある [+] をクリックします。

4. 'CursorType' プロパティを [Dynamic] または [Forward Only] に設定します。

このように設定した後は、結果セットでの前方と後方の両方向のスクロール、ローの挿入、削除、更新が可能になります。

結果セットが静的 (つまり [Static] カーソルまたは [Keyset riven] カーソルを使用) で、'OEDataSet' コンポーネントを使用する場合は、結果セットでローの挿入、削除、編集を行うと予期しない結果が発生することがあります。[Static] カーソルを ASA バージョン 6.0.3.2747 で使用した場合は、ローを削除するとエラーが発生します。[OK] をクリックすると、テーブル内で 1 つ以外の全ローが表示されなくなります。この問題は、ASA 6.0.3 .2934 EBF では解決されました。エラーの代わりに、'Invalid cursor state' というメッセージが表示されます。

[Keyset Driven] カーソルを使用して、ローを削除することができます。ただし、結果セットを閉じて再度開くと、削除されたローが再表示されます。これは、'Commit' を明示的に記述した場合にも発生します。ASA 6.0.3 .2934 EBF では、ローが再表示される問題は修正されていますが、別のエラーが発生します。以下の条件で、'Invalid cursor state' というエラーが発生することがあります。これは結果セットが静的であるため、操作対象のローは物理的には挿入、削除、更新されません。つまり、表示上は、挿入されたローは表示されなくなり、削除されたローは再表示され、更新されたローはそのまま残ります。しかし、'TOEDataSet' コンポーネントは 'TDataSet' コンポーネントから派生したコンポーネントで、挿入したローは表示され、削除されたローは表示されなくなり、変更されたローは更新さ

れることが要求されるため、ODBCExpress は通常の動作を変更し、'TDataSet' コンポーネントと同様に動作します。'TDataSet'コンポーネントと同様の動作を実現するため、'TOEDataSet' は削除および変更されたローを記録しておき、表示はそれに応じて調整されます。したがって、結果を再度開くと、削除されたローが再表示され、編集されたローが元の値に復元されます。挿入したローについて、ローが表示されない問題を回避するには、挿入したローのプライマリ・キーを記録しておき、結果セットを閉じて再度開き、挿入したローを新しい結果セットの一部として配置するのが唯一の方法です。

ebf とバージョン番号が同一の dbodbc6.dll ファイル (詳細については 43 ページの「付録 D」を参照) を、ASA のすべての dll があるディレクトリにインストールすると、'Invalid cursor state' エラーが解決されます。これにより、[Static] および [Keyset Driven] のカーソルを使用したローの削除、挿入、編集を問題なく実行できるようになります。

ASA 7.0 で上記の操作を行った場合は、結果が同じになる場合と異なる場合があります。[Forward only] および [Dynamic] のカーソルを使用した場合は、ASA 6.0 で発生する 'Fetch type out of range' エラーが ASA 7.0 の場合でも発生します。[Static] および [Keyset Driven] カーソルを ASA 7.0.x で使用した場合は、データベース内のローを削除または更新すると、'Syntax error or access violation. Update operation attempted on non-updatable query.' というエラーが発生します。また、ローをデータベースに挿入する場合は、エラーは発生しませんが、テーブルを更新すると、追加したローが表示されなくなります。これらのエラーは、ASA 6.0.3 と ASA 7.0.x で動作が異なるために発生します。ASA 6.0.3 の動作に戻すには、以下の手順に従います。

1. [スタート] メニューの [ファイル名を指定して実行] をクリックします。

2. 'dbisql -c "uid=dba;pwd=sql;dsn=the name of your ODBC source' と入力し、[OK] をクリックします。たとえば、ODBC ソース名が DelphiDemo の場合は、次のように入力します。

   dbisql -c "uid=dba;pwd=sql;dsn=DelphiDemo"

3. [ISQL] ウィンドウで、次のように入力します。

   set option public.ansi_update_constraints='off'

入力したら、[実行] ボタンをクリックします。

4. dbisql の実行が終了すると、先ほど入力した文が青色で強調表示されます。dbisql を終了します。

5. Delphi アプリケーションを終了して再起動します。データベース・エンジンが終了して再起動したことを確認します。

6. これで、[Static] または [Keyset Driven] カーソルを使用して、ローの挿入、削除、更新を問題なく行うことができます。

'OEQuery' コンポーネントを 'OEDataSet' コンポーネントの代わりに使用した場合、ごくわずかですが違いがあります。しかし、この二つのコンポーネントは実質的には同一です。'OEQuery' コンポーネントを 'DBGrid' コンポーネントとともに使用すると、'OEDataSet' コンポーネントを使用した場合と同じエラーが発生します。複数のテーブルを含むクエリを実行する場合は、ODBCExpress は複数のテーブルで名前が同一の 2 つのカラムを選択することができません。名前が同一の 2 つのカラムを選択する場合は、カラム名のエイリアスを

SQL の Select 文で指定する必要があります。また、複数のテーブルが関係する結果セットでのローの挿入、削除、編集については、このホワイトペーパーでは説明していません。

Titan SQLAnywhere Developer

'DBGrid' コンポーネントを Titan SQLAnywhere Developer で使用するには、BDE の 'DataSource' コンポーネントが必要です。また、このコンポーネントの 'DataSet' プロパティを、'tsTable' コンポーネントまたは 'tsQuery' コンポーネントにバインドする必要があります。これにより、'DBGrid' コンポーネントがデータベースと通信することができるようになります。BLOB の例と同様に、'DBGrid' および Titan を使用する構文は、'DBGrid' および BDE を使用する場合とほぼ同一です。BDE および ODBCExpress とは異なり、Titan SQLAnywhere Developer では、'Active' プロパティを 'True' に設定して Delphi を閉じても 'Access violation at address 1F4ADCD4. Read of address 1F4ADCD4' というエラーは発生しません。

Titan SQLAnywhere Developer および 'DBGrid' コンポーネントを使用した場合にだけ発生する問題の 1 つに、データを特定の順序で表示する際の問題があります。BDE では、'TTable' コンポーネントの 'IndexFieldNames' プロパティをテーブル内のカラムに設定すると、ORDER BY 句が fetch の最後に指定され、そのカラムを基準にして結果セットのローがソートされます。Titan の 'tsTable' コンポーネントを使用してこの操作を行うと、'tsTable1 has no index for fields id' というエラー ('id' は 'IndexFieldNames' に設定されたカラム名) が発生します。'tsTable' コンポーネントを使用して、特定の順序で結果セットを表示する方法はありません。

'DBGrid' では、データをソートする必要がある場合は、'tsQuery' コンポーネントを 'tsTable' コンポーネントの代わりに使用することを推奨します。'tsQuery' では、多くの異なる SQL 文をデータベースに対して実行できるように 'SQL' プロパティを設定し、データをより柔軟に表示することができます。このコンポーネントを使用すれば、結果セットのローの編集に関する問題は発生しません。クエリの結果セットの取得は高速で、BDE と比較すると、短時間で結果セットの最終ローにジャンプすることができます。複数のテーブルが関係するクエリを実行する場合は、結果セットのローの挿入、削除、更新を通常の方法で行うことはできません。

ここでも、Titan と同様に、テーブルを変更する場合は、Commit を明示的に記述する必要があります。これは、Titan で変更をコミットするためのデフォルト値が BDE または ODBCExpress とは異なる値に設定されるためです。

NativeDB for SQL Anywhere

ODBCExpress の場合と同様に、NativeDB には 'Table' コンポーネントがなく、その代わりに 'AsaDataset' コンポーネントを使用します。'DBGrid' を 'AsaDataset'(データベースのアクセスに使用します) にバインドするには、BDE の 'DataSource' コンポーネントをフォームに追加します。次に、'DataSource' を 'AsaDataset' コンポーネントに、'DBGrid' を 'DataSource' コンポーネントに設定します。'AsaDataset' コンポーネントは、'SQL' プロパティを使用してテーブルにアクセスします。したがって、1 テーブルからすべてのカラムを選択する以外に、1 つまたは複数のテーブルの任意のカラムを同時に選択するように SQL 文を記述することができます。'AsaDataset' コンポーネントの 'SQL' プロパティは、ODBCExpress の 'OEDataSet' コンポーネント、BDE の 'Query' コンポーネント、Titan の 'tsQuery' コンポーネントと同様の機能を実現します。BDE の場合と同様に、'DBGrid' でテーブルを表示するには、'AsaDataset' の 'Active' プロパティを 'True' に設定する必要があります。BDE および ODBCExpress とは異なり、NativeDB では前述の 'Access violation at address 1F4ADCD4' エラーは発生しません。

また、結果セットのローの挿入、削除、更新を行うには、'ReadOnly' プロパティを 'False' に設定する必要があります。

NativeDB には、'DBGrid' の垂直スクロール・バーを現在のローに配置する機能があります。BDE、ODBCExpress、Titan は、スクロール・スライドが最上部、最下部、中間のいずれかにある、3 状態のスクロール・バーをサポートしています。

付録 A

Delphi で ODBCExpress インタフェースを使用するための設定

BDE を使用してアプリケーションを作成している場合、データベースに対応する ODBC データ・ソースの設定後は、他の設定は不要です。ただし、ODBCExpress エンジンまたは Titan SQL Anywhere を使用する場合は、Delphi を起動する前にいくつかの手順を実行する必要があります。

ODBCExpress の場合は、最初に Delphi を起動し、以前のバージョンの ODBCExpress をアンインストールします。アンインストールの手順は、以下のとおりです。

1. ツールバーの [コンポーネント] を選択し、[パッケージのインストール] をクリックします。

2. [設計時のパッケージ] リスト・ボックスで ODBCExpress のパッケージを選択し、[削除] ボタンをクリックします。

3. [OK] をクリックしてパッケージを削除します。

4. ツールバーの [ツール] を選択し、[環境オプション] をクリックします。

5. [ライブラリ] タブを選択し、ODBCExpress コンポーネントが含まれるディレクトリの参照先を [ライブラリ パス] 編集ボックスから削除します。

ODBCExpress をインストールするには、以下の手順に従います。

1. ツールバーの [コンポーネント] を選択し、[パッケージのインストール] をクリックします (図 12：)。

2. [追加] ボタンをクリックし、ODBCExpress の解凍先ディレクトリにある 'Package' フォルダ内の OE.bpl ファイルを選択します。

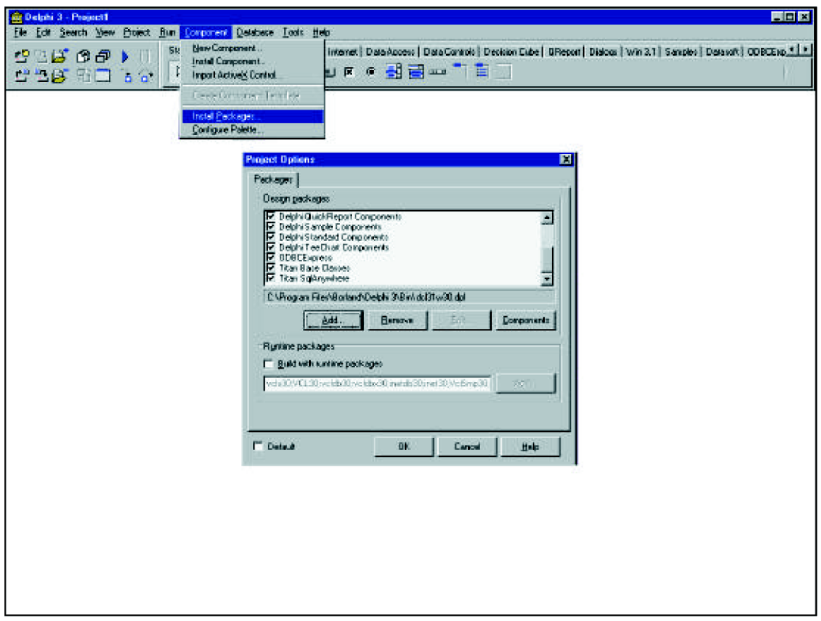

図 12 :

3. [OK] をクリックしてパッケージをインストールします。

4. ツールバーの [ツール] を選択し、[環境オプション] をクリックします。

5. [ライブラリ] タブを選択します。

6. [ライブラリ パス] 編集ボックスで、'Lib' フォルダおよび 'Package' フォルダ (ODBCExpress の解凍先) のパスを入力します。

ヘルプ・ファイルをインストールするには、以下の手順に従います。

1. ヘルプ・ファイルである 'OE32.HLP' および 'OE32.CNT' を、Delphi のメイン・ディレクトリにある 'Help' フォルダにコピーします。

2. Delphi の Help ディレクトリにある DELPHI3.CNT ファイルを開きます。

   :Index ODBCExpress Reference=oe32.hlp,

この行を、ファイルの最初の 'Index' セクションに追加します。

新しいバージョンでは、この作業は自動的に実行されます。

以上の手順が終了したら、Delphi を起動します。[コンポーネント] パレット (画面上部に表示されるウィンドウ) に、[ODBCExpress] という新しいタブが表示されます。以下に示す図 13 : を参照してください。提供される BDE コンポーネントの代わりにこれらのコンポーネントを使用して、ODBCExpress の機能を利用したアプリケーションを作成します。ODBCExpress のデータ型やさまざまな関数およびプロシージャのヘルプを表示するには、

ツールバーの [ヘルプ] を選択します。このヘルプでは、必要な情報を検索することができます。

図 13 :

これらの手順は、ODBCExpress のダウンロードに含まれる Readme ファイルでも説明されています。

付録 B

ASA 7.0 のサンプルの修正

ASA 7.0 に付属のサンプル・データベースは、asademoj.db という名前で、ASA のインストール先に保存されています。サンプル・データベースのデフォルトのパスは、C:\Program Files\Sybase\SQL Anywhere 7\asademoj.db です。最初に、asademoj.db のコピーを作成し、他のディレクトリにコピーして名前を変更します。この例では、asademoTest という名前に変更しています。データベースの名前を変更するのは、何か不具合が発生した場合に、元のデータベースからいつでもコピーを作成し直すことができるようにするためです。また、ログ・ファイルの名前も変更します。ログ・ファイルの名前を変更するには、[スタート] → [ファイル名を指定して実行] をクリックし、'dblog -t C:\asademoTest.log C:\asademoTest.db' と入力するのが最も簡単です。ここで、C:\asademoTest.log は、ログ・ファイルの保存場所のパスおよび保存ファイル名です。C:\asademoTest.db は、名前を変更したデータベースの場所のパスおよびファイル名です。

データベースを変更するには、以下の手順が最も簡単です。

1. [スタート] → [プログラム] → [Sybase SQL Anywhere 7] → [Sybase Central 4.0] をクリックして、Sybase Central 4.0 を起動します。表示されたウィンドウの左側の [Adaptive Server Anywhere 7] には、asademoj と表示されています。また、画面下部にある [スタート] バーの右端に、データベース・エンジンのアイコンが表示されます。

2. メニューバーの [ツール] を選択し、[接続] を選択します。[新しい接続] ウィンドウが表示されます。ドロップダウン・リストから [Adaptive Server Anywhere 7] を選択し、[OK] をクリックします。

3. [ID] タブで、[ユーザー] に 'dba'、[パスワード] に 'sql' と入力します。

4. [データベース] タブで、[データベース・ファイル] の横にある [参照] ボタンをクリックします。asademoj.dbのコピーを保存した場所を指定し、データベース (C:\asademoTest.db など) を選択します。

5. [OK] をクリックします。変化がないように見えますが、asademoj の左にある [+] をクリックすると、データベース名 (ここでは asademoTest) が表示されます。また、asademo が異なるアイコンで表示されます。

6. asademoTest の左にある [+] をクリックします。フォルダのリストが表示されます。リストの最初は、' テーブル ' になっています。

7. ' テーブル ' フォルダの左にある [+] をクリックすると、データベース内のすべてのテーブルの名前が表示されます。次のページの図 14：に、この時点での画面を示します。

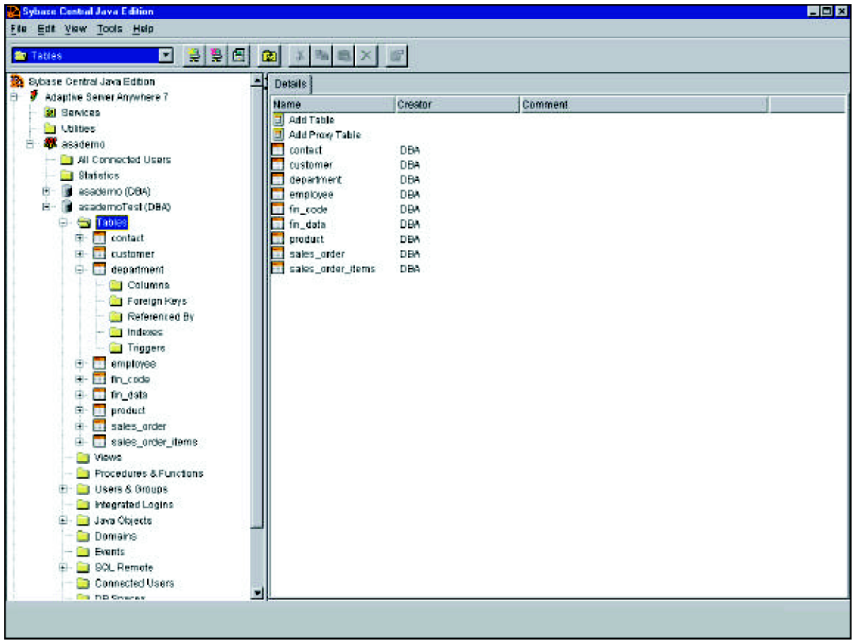

図 14 :

8. テーブルを追加する場合は、' テーブル ' フォルダをクリックし、右側にある [テーブルの追加] をダブルクリックします。フォームが表示されます。ここで、テーブル名の変更やカラムの追加を行うことができます。これは、テーブルに情報を入力するためのものではありません。単にカラム名を設定するだけです。

9. 既存のテーブル内のデータを変更する場合は、変更するテーブルの名前を右クリックし、[データの表示] を選択します。[ISQL] というウィンドウが表示されます。このウィンドウは、3 つのセクションで構成されています。最初の [SQL 文] というセクションには、'SELECT * FROM "DBA"."department"' というコマンド・ラインが含まれています。ここで、department は選択したテーブル名になっています。2 番目の [メッセージ] というセクションはブランクになっています。最後の [結果] というセクションには、カラムの見出しおよびそのカラムに対応する情報のリストが表示されます。

次の図 15: は、'department' テーブルで [データの表示] オプションを選択した場合の [ISQL] を示しています。

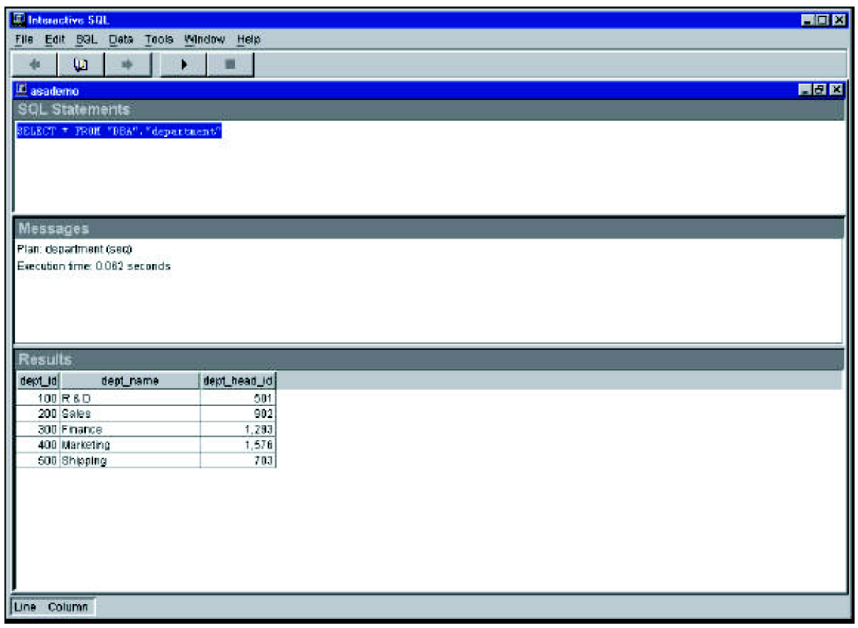

図 15：

新しいローを挿入するには、INSERT INTO department VALUES ('600', 'HR', '501'); というコマンドを入力し、[SQL 文の実行] ボタンをクリックします。ここで、'department' は、ローを挿入するテーブルの名前です。また、かっこで囲んだ値は、テーブル内のカラムに対応している必要があります。テーブル内の各カラムに値が設定されている必要があります。設定されていない場合は、エラーが発生します。また、入力した値は、そのカラムのデータ型に対応している必要があります。テーブルからローを削除するには、コマンドのセクションに DELETE FROM department WHERE dept_id=600; と入力し、[SQL 文の実行] ボタンをクリックします。このコマンドは、dept_id (department テーブル内のカラム) の値が 600 であるローのすべてのカラムを削除するだけでなく、テーブル内で dept_id が 600 のローもすべて削除します。

10. テーブルの変更が終了したら、[ISQL] ウィンドウを閉じます。Sybase Central が表示され、他のテーブルを変更することができます。asademoTest の変更を終了する場合は、メニューバーの [ツール] をクリックし、[切断] を選択します。ウィンドウが表示されます。ここで、asademoTest を選択し、[切断] をクリックします。Sybase Central が終了します。

付録 C

プライマリ・キーの問題

Adaptive Server Anywhere は、SQL リレーショナル・データベースです。このテクノロジを使用する場合は、第三正規形に従って設計します。第三正規形では、プライマリ・キー定義を利用します。プライマリ・キー (またはユニーク・インデックス) は、テーブル内の各ローをユニークに特定するために使用できるカラムまたはカラムの組み合わせです。つまり、1 つのテーブル内では、2 つのローでプライマリ・キーのカラム値が同一になることはありません。このため、通常はデータ型が 'integer' で、デフォルトが 'autoincrement' のカラムをプライマリ・キーに使用します。

16 ページの「BLOB の例」では、'blob' テーブルの 'keyfld' カラムがプライマリ・キーで、データ型が 'integer' およびデフォルトの 'autoincrement' になっています。ISQL を使用して blob を挿入する場合は、デフォルト値が設定されているため、'keyfld' カラムの値を省略することができます。autoincrement は、新しく挿入したローでユニークな値を設定します。Delphi ではこの機能が有効でないため、'keyfld' カラムに値を指定する必要があります。この値の設定方法は複数ありますが、4 番目に説明する方法以外では問題が発生する場合があります。

'keyfld' カラムに値を指定する最初の方法は、例 1 および例 3 で使用している方法です。nextnum という変数を作成し、以下の 2 行のコードで値を代入します。例 1 でのコードは、以下のとおりです。

```
Table1.Last;
nextnum := Table1.FieldByName('keyfld').asInteger + 1;
```

例 3 でのコードは、以下のとおりです。

```
OEDataSet1.Last;
nextnum := OEDataSet1.FieldValues['keyfld'] + 1;
```

このコードは、テーブルの最終ローにジャンプし、'keyfld' カラムの現在値に 1 加算した値を nextnum に代入します。nextnum は、新しいローの挿入時に、'keyfld' カラムの値として使用されます。この方法は、'keyfld' カラムの値を指定するのに適しているように見えますが、実際には問題が発生することがあります。テーブルに対して Insert 文を実行した場合は、このローがテーブルの最後に追加されない場合があります。したがって、テーブルの最終ローに 'keyfld' カラム用の値が格納されていない可能性があり、その結果、テーブルにすでに存在する値が nextnum に代入されることがあります。その場合は、Insert 文の実行時に 'Primary key for blob is not unique' というエラーが発生します。

2 番目の方法は、最初の方法で発生するエラーを回避する方法です。Insert 文を使用して新しいローをテーブルに追加する代わりに、Append メソッドを使用します。Append メソッドを呼び出した場合は、必ずテーブルの最後に新しいローが挿入されます。これで、最初の方法で説明した 2 行のコードを使用して、'keyfld' カラムにユニークな値を設定することができます。この方法は、テーブルに開始行がない、あるいはテーブル内のローがすでにソートされている場合には有効です。複数のユーザがテーブルに同時にローを追加しようとした場合には、この方法でも問題が発生します。

3 番目の方法は、'RecordCount' プロパティを使用します。このプロパティは、正しく使用した場合は、結果セット内のレコード数を返します。結果セットがテーブル全体の場合は、この方法を使用して 'keyfld' カラムの値を取得することができます。この値を正しく取得するには、テーブル内の最終レコードにジャンプする必要があります。BDE では、ジャンプするためのコードは以下のようになります。

```
Table1.Last  // To jump to the last row in the table
nextnum := Table1.RecordCount + 1;
```

この方法の問題は、'keyfld' カラムの値が 1 から開始し、最終値まで 1 ずつ増加する必要があることです。この理由は、デフォルトの 'autoincrement' が設定されたカラムを持つローを ASA で削除する場合の処理にあります。簡単な例で、デフォルトの 'autoincrement' が設定されたカラムを持つローを ASA で削除する場合の処理を説明します。テーブル内に、1、2、3、4、5 という 5 つのローがある場合に、ロー 4 およびロー 5 を削除すると、ロー 1、2、3 が残ります。新しいローを追加すると、'autoincrement' により 6 という値が設定されます。したがって、'keyfld' カラムの値で欠落している値がある、あるいは 1 から開始されていない場合には、テーブル内のローの数 (RecordCount が示す値) は 'keyfld' カラムでの次のロー値になりません。このため、プライマリ・キーで重複する値が発生する可能性があります。

'keyfld' カラムの値を設定するには、デフォルトが 'autoincrement' である ASA 側では不要な値を Delphi に認識させる方法を推奨します。BDE でこれを実現するコードを次に示します。

```
Table1.Fields[0].Required := False;
```

ODBCExpress では、コードは次のようになります。

```
OEDataSet1.Fields[0].Required := False;
```

Titan SQLAnywhere Developer では、コードは次のようになります。

```
tsTable1.Fields[0].Required := False;
```

Delphi では、テーブル内の各カラムを、0 から始まる番号で示します。上記のコードでは、カラム 0 がデフォルトとして 'autoincrement' を設定されたカラムであることを前提としています。このコードを実装すべき場所については、'autoincrement' について説明している 24 ページの「DBGrid の例」を参照してください。

付録 D

次の表は、このホワイトペーパーで説明している問題およびその解決方法を特定する際に使用した製品およびそのバージョンです。24 ページの「DBGrid の例」で説明している ebf for ASA のビルド番号は 2934 です。

	ASA	BDE	ODBCExpress	Titan	NativeDB
使用した Delphi 3 Client/Server Suite のバージョン	6.0.3.2747 7.0.0.313	4.0	4.53	3.02p	1.84
使用した Delphi 5 Enterprise のバージョン	6.0.3.2747 7.0.0.313	5.1.0.4	5.05	5	1.84

このホワイトペーパーで説明したインタフェースの詳細については、以下の Web ページを参照してください。

BDE: http://www.borland.com/delphi
ODBCExpress: http://www.odbcexpress.com
Titan SQL Anywhere Developer: http://www.reggatta.com/sqadev.html
NativeDB: http://www.nativedb.com

※本書は、米国 iAnywhere Solutions 社が作成およびテストしたものを日本語に翻訳したものです。

法的注意

Copyright(C) 2003 iAnywhere Solutions,Inc. All rights reserved.

Adaptive Server、iAnywhere、iAnywhere Solutions、SQL Anywhere、SQL Anywhere、Sybaseは、米国法人iAnywhere Solutions, Inc.または米国法人Sybase,Inc.とその系列会社の米国または日本における登録商標または商標です。その他の商標はすべて各社に帰属します。

Mobile Linkの技術には、Certicom,Inc.より供給を受けたコンポーネントが含まれています。これらのコンポーネントは特許によって保護されています。

本書に記載された情報、助言、推奨、ソフトウェア、文書、データ、サービス、ロゴ、商標、図版、テキスト、写真、およびその他の資料(これらすべてを"資料"と総称する)は、iAnywhere Solutions, Inc.とその供給元に帰属し、著作権や商標の法律および国際条約によって保護されています。また、これらの資料はいずれも、iAnywhere Solutions, Inc./Sybとその供給元の知的所有権の対象となるものであり、iAnywhere Solutions, Inc./Sybase,Inc.とその供給元がこれらの権利のすべてを保有するものとします。

資料のいかなる部分も、iAnywhere Solutionsの知的所有権のライセンスを付与したり、既存のライセンス契約に修正を加えることを認めるものではないものとします。

資料は無保証で提供されるものであり、いかなる保証も行われません。iAnywhere Solutionsは、資料に関するすべての陳述と保証を明示的に拒否します。これには、商業性、特定の目的への整合性、非侵害性の黙示的な保証を無制限に含みます。

iAnywhere Solutionsは、資料自体の、または資料が依拠していると思われる内容、結果、正確性、適時性、完全性に関して、いかなる理由であろうと保証や陳述を行いません。iAnywhere Solutionsは、資料が途切れていないこと、誤りがないこと、いかなる欠陥も修正されていることに関して保証や陳述を行いません。ここでは、「iAnywhere Solutions」とは、iAnywhere Solutions,Inc.とその部門、子会社、継承者、および親会社と、その従業員、パートナー、社長、代理人、および代表者と、さらに資料を提供した第三者の情報元や提供者を表します。

＊　本書は、米国iAnywhere Solutions, Inc.が作成・テストしたものを日本語に翻訳したものです。

アイエニウェア・ソリューションズ株式会社
http://www.ianywhere.jp

1010193