中嶋 祥子

株式会社ミガロ.

RAD事業部 技術支援課

Delphi/400:ローカルキャッシュ活用術

ClientDataSet の利用を中心に
Delphi/400 でのローカルキャッシュの有効な活用方法を紹介する。

略歴
1968 年 2 月 23 日生まれ
1990 年奈良女子大学大学家政学部卒
2002 年株式会社ミガロ . 入社
2002 年 11 月 RAD 事業部配属

現在の仕事内容
お客様からの Delphi/400 に関する技術的な質問や問い合わせに対応している。また、メルマガ「Migaro News」やホームページの Tips など、開発に役立つ情報も担当。

1. はじめに

クライアントデータセットを使用すると、サーバー上のデータをローカル PC のキャッシュ上に保持できるが、そのメリットとは何だろうか。

Delphi/400 は、クライアントデータセットを使用しなくても、IBM i（AS/400）との通信パフォーマンスが高い。だが、キャッシュ上のデータにアクセスすることで、サーバー（IBM i）との通信回数を減らすことが可能になり、さらにパフォーマンスのよいアプリケーションが作成できる。それだけでなく、キャッシュ上にあるという特性を生かした機能が用意されているため、さまざまなオペレーションが簡単に実現できる。

2. 使用コンポーネント

ローカルキャッシュ上でサーバー上のデータを扱う場合、TClientDataSet を使用する。その時にサーバーへ接続するデータセット（TTable、TQuery 等）とこの TClientDataSet をつなぐために、TDataSetProvider もあわせて利用する。

TTable のデータを TDBGrid に表示するプログラムを例にしてみよう。

通常であれば、4 つのコンポーネントがプロパティで関連付けられている。これに、TDataSetProvider、TClientDataSet の 2 つのコンポーネントを加え、プロパティの関連付けを行うだけで、ローカルキャッシュ上でデータを扱える。

なお、TDBGrid に表示するデータセットは TTable から TClientDataSet になるため、TDataSource のプロパティを TClientDataSet に変更しておく。【図 1】

以上の処理で、TClientDataSet を操作すると、キャッシュ上のデータを操作することになる。また、TTable と基本的な操作は同じであるため、違和感なく利用できる。

3. データ操作

実際に、どのようなことが行えるのかを説明していこう。

レコード移動

First、Last、Next、Prior などの標準のデータセットメソッドが使用できる。

また、キャッシュ上にデータを保持する TClientDataSet 独自の方法として、RecNo プロパティでレコード番号を指定して位置付けを行える。【ソース 1】【図 2】

なお、この RecNo プロパティは、現在のレコード位置を取得することもできる。【ソース 2】【図 3】

また、リレーショナルデータベースでは、基本的に RecCount プロパティでレコード件数の参照は行えないが、TClientDataSet では取得可能である。【ソース 3】【図 4】

フィルタ処理

TTable はフィルタ処理でレコード抽

図1

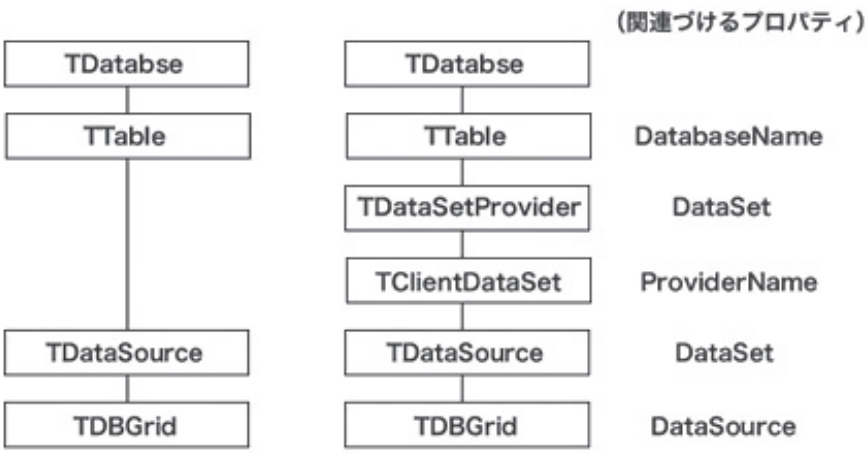

TDatabse
TTable
TDataSource
TDBGrid
TDatabse
TTable
TDataSetProvider
TClientDataSet
TDataSource
TDBGrid
（関連づけるプロパティ）
DatabaseName
DataSet
ProviderName
DataSet
DataSource

図2

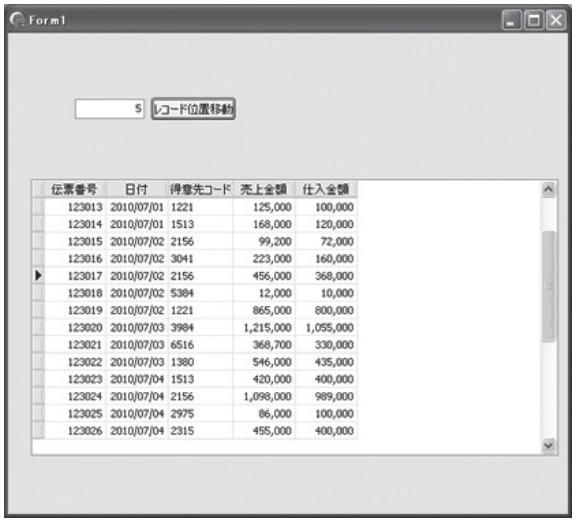

Form1
5
レコード位置移動

図3

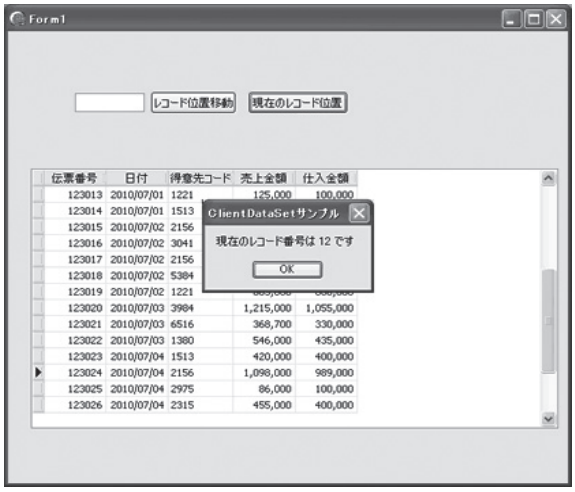

Form1
レコード位置移動
現在のレコード位置
ClientDataSetサンプル
現在のレコード番号は 12 です
OK

図4

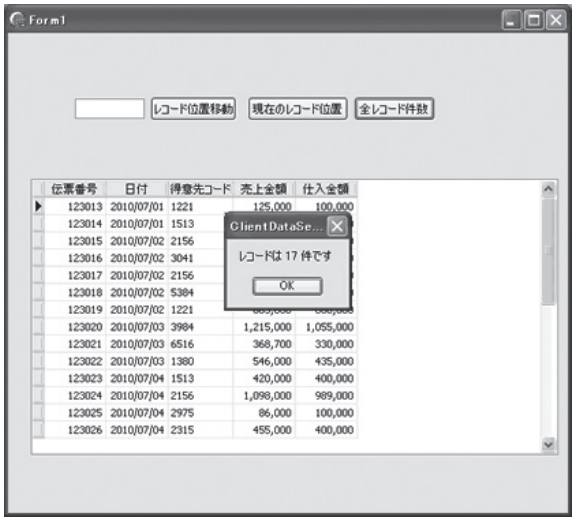
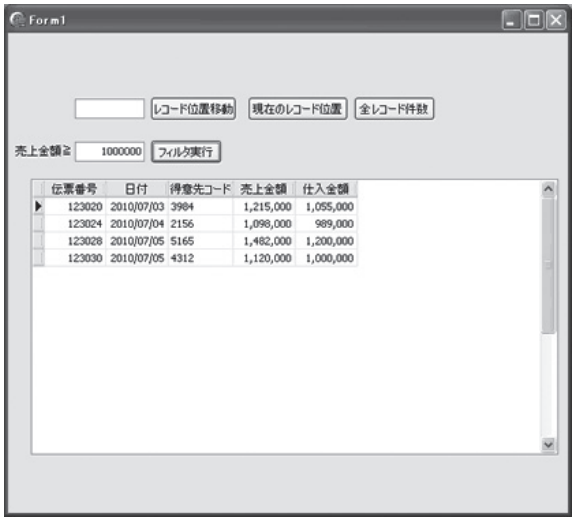

Form1
レコード位置移動
現在のレコード位置
全レコード件数
ClientDataSe...
レコードは 17 件です
OK

図5


Form1
レコード位置移動
現在のレコード位置
全レコード件数
売上金額≧
1000000
フィルタ実行

出を行えるが、TClientDataSet でも可能である。【ソース 4】【図 5】

画面で条件を指定して、そのデータを抽出するケースを考えてみよう。

SQL を使用して条件抽出する場合、SQL は条件を変更する度に「サーバーへの要求」→「サーバー上での処理」→「サーバーからの送信」という処理が発生する。

一方、TClientDataSet は、キャッシュ上にデータがすでに取り込まれているため、それが不要になる。つまり処理が高速で行える。

インデックスの作成

TClientDataSet は自由にインデックスを変更できるので、データの並び替えも行うことができる。

方法には、次の 2 通りがある。

(1) IndexFieldName プロパティで指定

1 つのフィールドで昇順に並び替えを行う時、このプロパティにフィールド名を文字列で設定する。

例えば、DBGrid のタイトルクリック時にその列フィールドで並び替えを行うのであれば、ソースのように記述することで、インデックスの切り替えが可能になる。【ソース 5】

(2) IndexDefs プロパティで指定

こちらのプロパティを使用すれば、複数のフィールドで並び替えができる。インデックスを作成し、その Fields プロパティでインデックス項目を指定する。

フィールドの区切りはセミコロン「;」なので、例えば「日付（HIZUKE)」「売上金額（URIK)」の順であれば、Fields プロパティで次のように記述する。

・Fields プロパティ値：HIZUKE;URIK

設定後、TClientDataSet の Index Name プロパティで指定することで、このインデックスが反映される。【図 6】【図 7】

また、特定のフィールドを降順にすることもできる。前述の Fields プロパティのうち、降順にするフィールドを DescFields プロパティに記述するだけである。

例えば「売上金額（URIK)」だけ降順にするのであれば、DescFields プロパティに次のように記述するだけで、「日付 (昇順)」「売上金額 (降順)」の順にデータが並び変わる。【図 8】

・DescFields プロパティ値：URIK

上記のように、コード内で Fields プロパティと DescFields プロパティを書き換えることにより、降順 / 昇順を切り替えることができる。複数のフィールドを記述する時には、Fields プロパティと同様にセミコロン「;」を使用する。

最後に、並び替えの項目を変更するには、以下のようにする。

ソース例では、2 つのインデックスを用意し、切り替えを行っている。【ソース 6】

(1) IndexFieldName プロパティで指定

プロパティ値を変更する。

(2) IndexDefs プロパティで指定

IndexDefs プロパティで詳細なインデックス情報を複数設定し、IndexName プロパティで IndexDefs プロパティの設定名を変更する。

集計項目

TClientDataSet には集計項目と呼ばれるものがあり、合計値や平均値などが取得できる（※)。また図表に、集計演算子をまとめた。【図 9】

利用するには、データセットに項目を追加する方法を用いる。「項目の新規作成」で、「型」を Aggregate、「項目の種類」は集合体にすると項目が追加される。

例えば、合計値を求めたい時には、Expression プロパティに SUM 演算子を使用した式を記述する。また、Active プロパティを True にしなければならない。

表示は TDBEdit を使用すると、TDBGrid 上の明細データ値を変更した時にも、即座に反映される。その際、項目の表示形式を Alignment や DisplayFormat プロパティで設定できる。【図 10】【図 11】

※注意
集計対象となるのはキャッシュされているレコードだけなので、後述の PacketRecords プロパティを指定している場合、全レコードでなくキャッシュされているレコード内の集計や平均となる。

計算項目/内部計算項目

通常のデータセットでも計算項目は存在するが、TClientDataSet には計算項目とは別に、内部計算項目がある。計算項目はレコードが変更されたり項目編集が行われる度に発生するのに対し、内部計算項目はデータとして保存され、項目として扱うことが可能である。

分かりやすい例として、計算項目でソートする場合を見てみよう。

まず、計算項目として粗利を作成する。【図 12】【ソース 7】

実行してみると、TDBGrid への表示は問題なく行われる。【図 13】

次に、計算項目である「粗利」でソートを行うために、ボタンで記述し実行してみると、エラーが発生してしまう。【ソース 8】【図 14】

これを内部計算項目に変更してみよう。変更する場合には、計算項目フィールドの FieldKind プロパティを fkCalculated から fkInternalCalc に変えることで行える。この状態で再びソートを行うと、正常に並び替えが行われている。【図 15】

計算項目では内部計算項目と異なり、データが保存されずその度に計算されるので、ソートを行う場合などには利用できない。その際には、内部的に保存される内部計算項目を使用する。

4. 更新処理

キャッシュ上での更新

更新処理を行うメソッドは通常のデータセットと同じで、Insert、Edit、Delete、Post を使用する。また、TDBGrid 上で直接操作したり、TDBNavigator から行うことも同じである。さらに、CancelUpdates メソッドを呼び出すと、すべての変更を取り消すことができる。

サーバーへの更新適用

キャッシュ上のデータ変更をサーバー側へ反映させる処理が必要であるが、ApplyUpdates メソッドを呼び出すだけである。

ただし、キャッシュ上のデータ変更をサーバー側へ一括で反映させるという機能上、トランザクション処理が必要である。BDE 接続の場合、以下の 2 点を行う。

図6

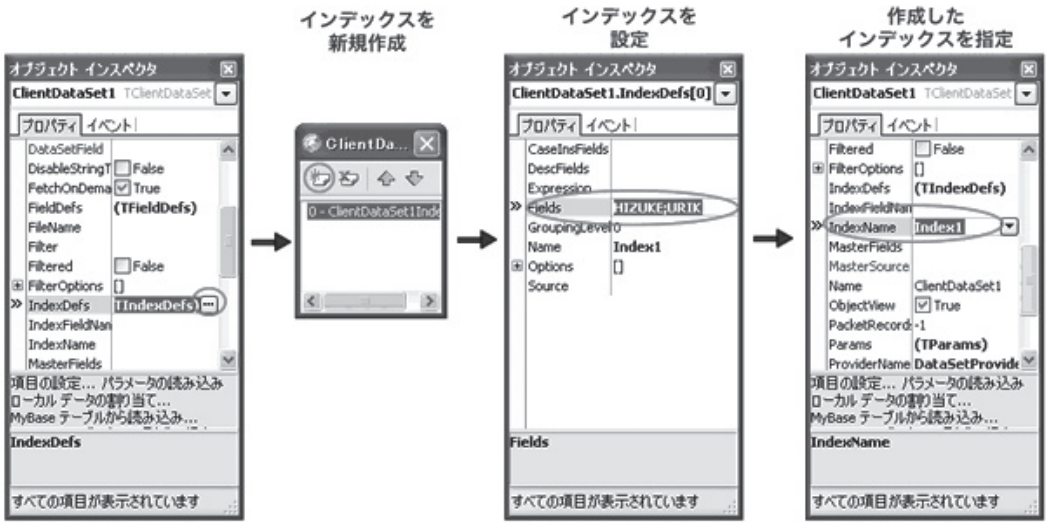

インデックスを
新規作成
インデックスを
設定
作成した
インデックスを指定
オブジェクト インスペクタ
ClientDataSet1 TClientDataSet
プロパティ イベント
IndexDefs
すべての項目が表示されています
ClientDa...
0 - ClientDataSet1Inde
ClientDataSet1.IndexDefs[0]
Fields
IndexName

図7

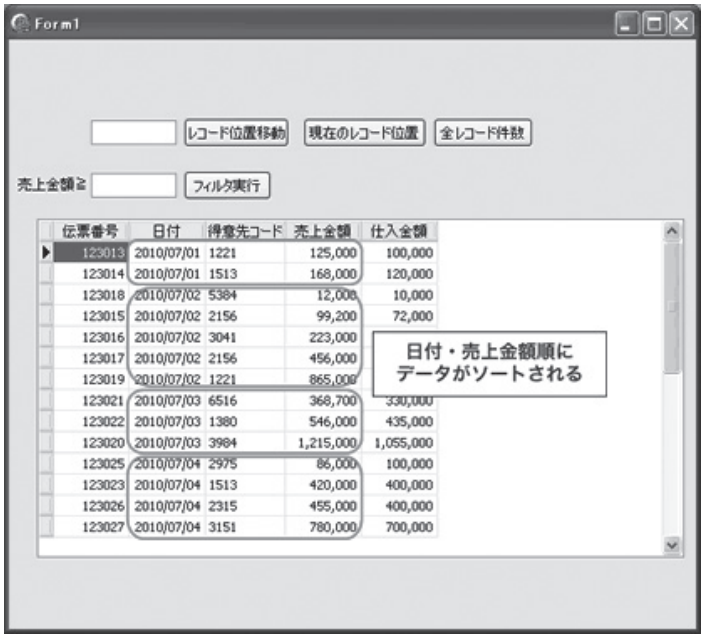
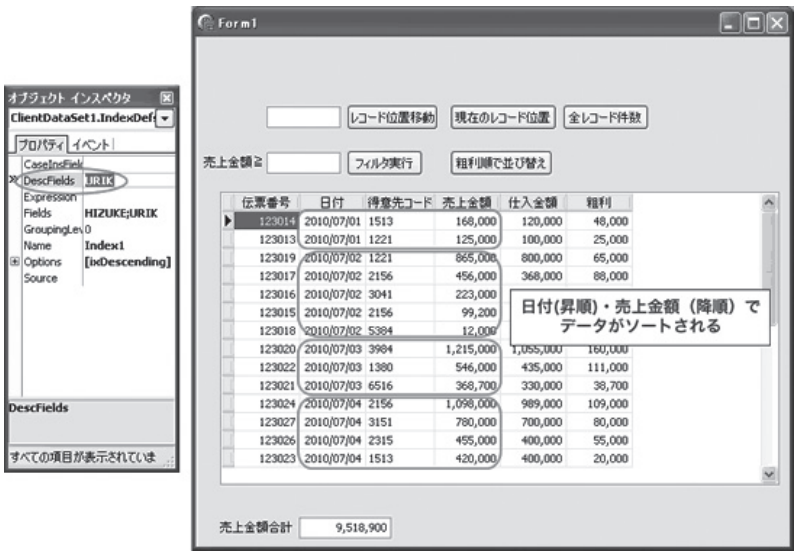

Form1
レコード位置移動
現在のレコード位置
全レコード件数
売上金額≧
フィルタ実行
伝票番号 日付 得意先コード 売上金額 仕入金額
日付・売上金額順に
データがソートされる

図8


Form1
レコード位置移動
現在のレコード位置
全レコード件数
売上金額≧
フィルタ実行
粗利順で並び替え
伝票番号 日付 得意先コード 売上金額 仕入金額 粗利
日付(昇順)・売上金額（降順）で
データがソートされる
売上金額合計 9,518,900
オブジェクト インスペクタ
DescFields

● IDCO400 ドライバのパラメータの TRAN_ISOLATION が、＊NONE 以外の値が設定されていること。（値は＊CHG、＊CS、＊ALL で、IBM i 上のコミットメント制御に同じ）。
● IBM i（AS/400）上で、対象ファイルのジャーナルが開始されていること。

エラーの取得

ApplyUpdates メソッドの引数は、その数のエラーが更新時に発生した場合に処理を停止する。

ApplyUpdates（-1）と引数を「-1」にすると、エラー数に制限がないことになるが、「0」にするとエラーが発生した時点で更新処理が停止する。

なお、戻り値は発生したエラー数を返すので、エラー発生時にメッセージを表示したり、トランザクション時にロールバックしたりといった制御を行うこともできる。【ソース 9】

また、TDataSetProvider の OnUpdateError で、ApplyUpdates 時のエラーを捕らえることができる。【ソース 10】

5. 注意点

データ量

TClientDataSet はとても便利な機能であるが、一旦すべてのデータをキャッシュ上に取り込むため、データ量が膨大だと時間が非常にかかることがある。その場合には、PacketRecords プロパティを利用する方法が考えられる。

例えば、TDBGrid にデータを表示する場合、データ量があまりにも多いと全件を読み込むには時間がかかるため、なかなかデータが表示されない。しかし、PacketRecords プロパティの値を 1 画面の表示データ件数分とすると、データの表示が早くなる。

スクロールダウンする時には次のデータの読み込みが行われるが、反対にスクロールアップする時には、以前のデータがキャッシュ上に残されているので、メモリにアクセスし表示は速い。

しかし、スクロールの途中で読み込みが完了していない状態では、キャッシュ上にはすべてのデータが存在しておらず、集合体やソート処理などが行えないため、せっかくの機能が十分活用できない。

このため、膨大なデータの時には全件を取り込むのではなく、ある程度サーバー上でデータを絞り適切な件数になるようにした方がよい。それによって本来の TClientDataSet の機能を十分活用しつつ、速度も得られるようになる。

レコードの順序

文字コードの関係で、AS/400 では並び順が「アルファベット→数字順」になるが、PC では「数字→アルファベット順」となる。

このためローカルに取り込んだ時、インデックスが英数混在の項目の場合、TClientDataSet では PC 上の並び順になる。もし IBM i（AS/400）での並び順とするのであれば、TDataSetProvider の Options プロパティにある poRetainServerOrder を True に設定する。これだけでレコードの順序が IBM i（AS/400）と TClientDataSet で一致する。

なお、別端末からサーバー上のデータが更新される場合、一旦キャッシュ上に取り込まれたデータと差異が生じることがある。このため変更されやすいデータについては、ローカルキャッシュの活用は適していない。

midas.dllの配布

TClientDataSet を使用したアプリケーションの配布時についてだが、midas.dll も同時に配布する必要がある。しかし、Delphi7 以降であればアプリケーションの uses 節に midaslib を追加する方法で、この dll の配布が不要になる。

6. まとめ

よく利用されるであろう機能を簡単に説明したが、容易に利用できることやそのメリットを実感していただけただろうか。

すべてを紹介しきれなかったが、興味を持たれたのであれば、Delphi のヘルプにも記載されているので、ぜひ一度目を通していただきたい。

なお、パフォーマンスについてはサーバー上での処理速度も当然ながら大きく影響するため、その点の考慮も必要であることはいうまでもない。とはいえ、通信回数を減らすという面では、クライアントデータセットは非常に有益である。

RPG や CL の実行をはじめとするサーバー側の機能のみならず、ローカルキャッシュも十分活用していただき、Delphi/400 の性能を十分に引き出してもらいたい。

M

図9

演算子	機能
Sum	式または数値型の項目の値の合計
Avg	式，数値型または日付時刻型の項目の平均値
Count	空白ではない値を持つ式または項目の数を返す
Min	式あるいは文字列型，数値型，または日付時刻型の項目の最小値を示す
Max	式あるいは文字列型，数値型，または日付時刻型の項目の最大値を示す

図12

図10

図11

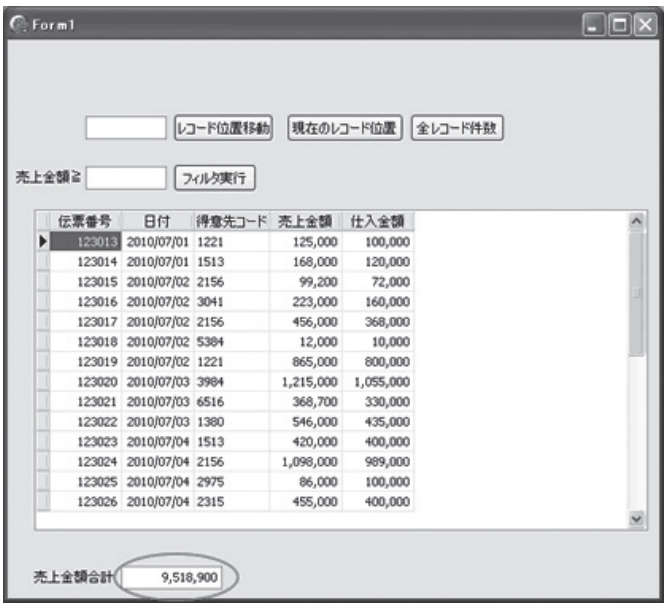


図13


Form1
レコード位置移動
現在のレコード位置
全レコード件数
売上金額≧
フィルタ実行
伝票番号 日付 得意先コード 売上金額 仕入金額 粗利
123013 2010/07/01 1221 125,000 100,000 25,000
123014 2010/07/01 1513 168,000 120,000 48,000
123015 2010/07/02 2156 99,200 72,000 27,200
123016 2010/07/02 3041 223,000 160,000 63,000
123017 2010/07/02 2156 456,000 368,000 88,000
123018 2010/07/02 5384 12,000 10,000 2,000
123019 2010/07/02 1221 865,000 800,000 65,000
123020 2010/07/03 3984 1,215,000 1,055,000 160,000
123021 2010/07/03 6516 368,700 330,000 38,700
123022 2010/07/03 1380 546,000 435,000 111,000
123023 2010/07/04 1513 420,000 400,000 20,000
123024 2010/07/04 2156 1,098,000 989,000 109,000
123025 2010/07/04 2975 86,000 100,000 -14,000
123026 2010/07/04 2315 455,000 400,000 55,000
売上金額合計 9,518,900

図14

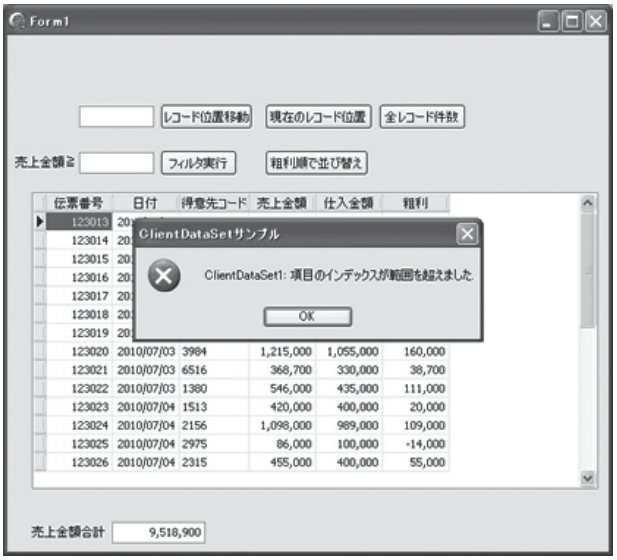
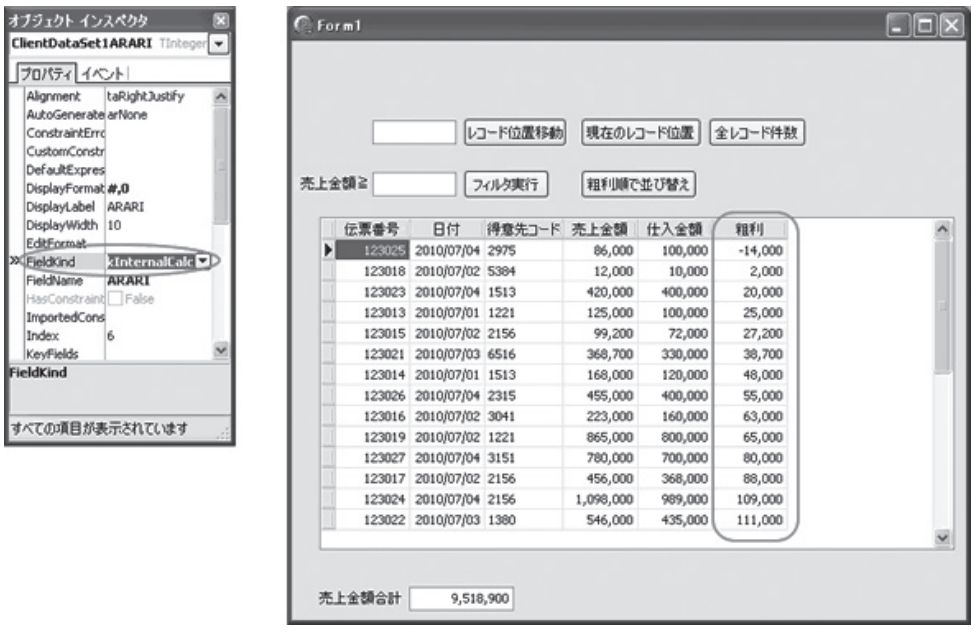

Form1
レコード位置移動
現在のレコード位置
全レコード件数
売上金額≧
フィルタ実行
粗利順で並び替え
伝票番号 日付 得意先コード 売上金額 仕入金額 粗利
ClientDataSetサンプル
ClientDataSet1: 項目のインデックスが範囲を超えました
OK
123020 2010/07/03 3984 1,215,000 1,055,000 160,000
123021 2010/07/03 6516 368,700 330,000 38,700
123022 2010/07/03 1380 546,000 435,000 111,000
123023 2010/07/04 1513 420,000 400,000 20,000
123024 2010/07/04 2156 1,098,000 989,000 109,000
123025 2010/07/04 2975 86,000 100,000 -14,000
123026 2010/07/04 2315 455,000 400,000 55,000
売上金額合計 9,518,900

図15


オブジェクト インスペクタ
ClientDataSet1.ARARI TInteger
プロパティ イベント
Alignment taRightJustify
AutoGenerate erNone
ConstraintErr
CustomConstr
DefaultExpres
DisplayFormat #,0
DisplayLabel ARARI
DisplayWidth 10
EditFormat
FieldKind kInternalCalc
FieldName ARARI
HasConstraint False
ImportedCons
Index 6
KeyFields
FieldKind
すべての項目が表示されています
Form1
レコード位置移動
現在のレコード位置
全レコード件数
売上金額≧
フィルタ実行
粗利順で並び替え
伝票番号 日付 得意先コード 売上金額 仕入金額 粗利
123025 2010/07/04 2975 86,000 100,000 -14,000
123018 2010/07/02 5384 12,000 10,000 2,000
123023 2010/07/04 1513 420,000 400,000 20,000
123013 2010/07/01 1221 125,000 100,000 25,000
123015 2010/07/02 2156 99,200 72,000 27,200
123021 2010/07/03 6516 368,700 330,000 38,700
123014 2010/07/01 1513 168,000 120,000 48,000
123026 2010/07/04 2315 455,000 400,000 55,000
123016 2010/07/02 3041 223,000 160,000 63,000
123019 2010/07/02 1221 865,000 800,000 65,000
123027 2010/07/04 3151 780,000 700,000 80,000
123017 2010/07/02 2156 456,000 368,000 88,000
123024 2010/07/04 2156 1,098,000 989,000 109,000
123022 2010/07/03 1380 546,000 435,000 111,000
売上金額合計 9,518,900

ソース1

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ClientDataSet1.RecNo := StrToInt(Edit1.Text);
end;
```

ソース2

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  ShowMessage('現在のレコード番号は '
              + IntToStr(ClientDataSet1.RecNo)
              + ' です');
end;
```

ソース3

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  ShowMessage('レコードは '
              + IntToStr(ClientDataSet1.RecordCount)
              + ' 件です');

end;
```

ソース4

```
procedure TForm1.Button4Click(Sender: TObject);
begin
  with  ClientDataSet1  do
  begin
    Filter := 'URIK >= ' + Edit2.Text;
    Filtered := True;
  end;
end;
```

ソース5

```
procedure TForm1.DBGrid1TitleClick(Column: TColumn);
begin
  ClientDataSet1.IndexFieldNames := Column.FieldName;
end;
```

ソース6

```
procedure TForm1.Button5Click(Sender: TObject);
begin
  if  ClientDataSet1.IndexName = 'Index1' then
    ClientDataSet1.IndexName := 'Index2'
  else
    ClientDataSet1.IndexName := 'Index1';
end;
```

ソース7

```
procedure TForm1.ClientDataSet1CalcFields(DataSet: TDataSet);
begin
  with  DataSet  do
  begin
    FieldByName('ARARI').AsInteger := FieldByName('URIK').AsInteger
                                     -FieldByName('SHIK').AsInteger;

  end;
end;
```

ソース8

```
procedure TForm1.Button5Click(Sender: TObject);
begin
  ClientDataSet1.IndexFieldNames := 'ARARI';
end;
```

ソース9

```
procedure TForm1.Button7Click(Sender: TObject);
begin
  if  ClientDataSet1.ApplyUpdates(-1) > 0  then
    ShowMessage('エラーが発生しました');
end;
```

ソース10

```
procedure TForm1.DataSetProvider1UpdateError(Sender: TObject;
  DataSet: TCustomClientDataSet; E: EUpdateError; UpdateKind: TUpdateKin
  var Response: TResolverResponse);
begin
  raise Exception.Create(E.Message);
end;
```