

[Delphi/400]

FireDAC 実践プログラミングテクニック



略歴

1985年12月6日生まれ
2009年3月 甲南大学 経営学部卒業
2009年4月 株式会社ミガロ. 入社
2009年4月 システム事業部配属

現在の仕事内容:

Delphi/400 を利用したシステム開発や保守作業を担当。Delphi および Delphi/400 のスペシャリストを目指して精進している。

1. はじめに
2. FireDAC のデータ取得に関するテクニック
 - 2-1. データベースエンジン変更時のマッピングルール
 - 2-2. ソート順の指定方法
 - 2-3. フェッチによるパフォーマンス効果
3. FireDAC のデータ更新に関するテクニック
 - 3-1. 双方向更新機能の活用
 - 3-2. FireDAC のトランザクション制御
4. ファイルメンバーの制御
5. まとめ

1.はじめに

Delphi/400 10 Seattle のリリース以降、新規開発や他の接続方式からの移行において最新のデータベースエンジンである FireDAC が採用されることが多くなった。2016 年版『ミガロ. テクニカルレポート』の SE 論文でも、「新データベースエンジン FireDAC を使ってみよう!」と題して FireDAC の基本的な使用方法を紹介している。

本稿では、FireDAC をさらに使いこなすための実践的なテクニックを検証し、紹介していく。データベースエンジンの操作には大きく分けてデータ取得とデータ更新があり、各章に分けて技術トピックをまとめている。

なお、本稿では Delphi/400 の最新バージョンである Delphi/400 10.2 Tokyo の環境を使用している。

2.FireDACのデータ取得に関するテクニック

2-1.データベースエンジン変更時のマッピングルール

Delphi/400 と IBM i との間でデータをやり取りする場合、値の受け渡しのために、データベースエンジンではコンポーネントでデータ型を自動的に設定する。

このデータ型のルールはデータベースエンジンによって異なり、BDE、dbExpress、FireDAC では一致しないデータ型もある。たとえば整数や小数 4 桁以内の実数は BCD 型、小数 5 桁以上の実数は FMTBCD 型として扱われる。

【図1】【図2】

FireDAC で新規開発する場合はそのまま問題ないが、従来の BDE や dbExpress から FireDAC に移行するときは、データ型が異なる部分を変更しておく必要がある。たとえばデータの処

理時に FireDAC が受け取るデータ型が、既存のプログラムで設定されているデータ型と差異がある場合、そのまま処理すると、エラーが発生してしまう【図3】。

また、FireDAC で新規開発する場合でも、BCD 型のフィールドは内部的に数値を Currency として保持するため、「17.0 桁」のような小数 4 桁以内かつ Currency 型で扱えない巨大な桁数をセットしようとするとうエラーになってしまう。【図4】

これらの現象に対してマッピング (変換) のルールを設定することで、データベースエンジン間での違いを吸収し、差異があるデータ型を従来と同様のデータ型として扱うことが可能となる。以下に設定手順を紹介する。

まず、TFDConnection コンポーネントをダブルクリックすると、FireDAC 接続エディタが表示される。ここで設定した内容は、紐づく各 TFDQuery や TFDTable にも一括で適用される。個別の TFDQuery や TFDTable コンポー

図1 接続方式とDelphi/400のデータ型

データタイプ	型	BDE	dbExpress	FireDAC
A	半角のみの文字型	TStringField	TStringField	TStringField
O	全半角混合の文字型	TStringField	TStringField	TStringField
J	全角のみの文字型	TStringField	TStringField	TStringField
L	日付型	TDateField	TDateField	TDateField
T	時刻型	TTimeField	TTimeField	TTimeField
Z	タイムスタンプ型	TDateTimeField	TSQLTimeStampField	TSQLTimeStampField ★
P または S	1～9桁の整数型	TIntegerField	TIntegerField	TBCDField ★▼
	10～18桁の整数型	TFloatField	TFloatField	TBCDField ★▼
	有効桁数が18以内かつ 小数4桁以内の実数型	TFloatField	TFloatField	TBCDField ★▼
	上記以外の数値型	TFloatField	TFloatField	TFMTBCDField ★▼
B	2進数型	TIntegerField	使用不可	使用不可
H	16進数型	TStringField	使用不可	使用不可

★ = BDEから移行時にマッピングが必要

▼ = dbExpressから移行時にマッピングが必要

図2 フィールド型の違い

8. 0桁の整数フィールドだが、マッピング未設定の場合はBCD型として扱う (BDEやdbExpressではInteger型)

ネット単位でも同様の接続エディタを持っているため、特定の1つのコンポーネントのみ例外的に特殊処理が必要な場合は、個別に設定を行うことも可能である。

次にオプションタブを選択し、「継承したルールを無視」チェックボックスをオンにすると、データマッピングルールの明細が入力可能になる。エラーメッセージの内容が、たとえば先の【図3】のように「FDTable: フィールド'○○○○'の型が一致しません。Integer が必要ですが実際はBCDです。」と表示された場合、BCD型で受け取る数値をInteger型として認識させればエラーとならずに読み書きが行える。また、IBM iと通信を行う際に考慮が必要なフィールドの型は、【図1】のように数値以外にも存在するため、マッピングルール例としては【図5】のような設定も有効である。

なお、マッピングの設定を行っていないと、数値をBCD型(TBCDField)としてIBM iと値の受け渡しを行うことになるが、Delphi/400側ではTBCDFieldを内部的にCurrencyに一度変換するため、【図6】【図7】のように従来とおりのロジックでフィールド値の読み書きを行うことができる。

2-2. ソート順の指定方法

IBM iからデータを複数レコード取得する際、その並び順は大きく分けて「①EBCDIC順」「②ASCII順」「③到着順」の3種類のルールが存在する。EBCDIC順はIBM iと同じ「A～Z→0～9」の並び順、ASCII順はWindowsと同様の「0～9→A～Z」の並び順、そして到着順は対象ファイルのレコード登録順(物理レコード順)となっている。【図8】

以下に、【図9】のようなファイルが存在した場合に、FireDACにおいてそれぞれの並び順でデータを取得する方法を紹介する。

① EBCDIC 順

EBCDIC 順でデータの取得を行う場合、FireDAC では TFDQuery を使用する。TFDQuery の SQL 文内 で ORDER BY 句を使ってフィールドの並

び順を指定すると、取得されるデータは ORDER BY で指定されたフィールドで EBCDIC 順に並べることができる。【ソース1】【図10】。

フィールドに降順を指定する際は、IBM i の STRSQL コマンド実行時と同様に ORDER BY 句の中で降順指定フィールドの後ろに「DESC」と記載する。ちなみに、BDE や dbExpress では TClientDataSet と紐付けを行い、TDataSetProvider の設定値によって EBCDIC 順になるよう指定する方法がある。

② ASCII 順

ASCII 順でデータのソートを行う場合、Delphi/400 のクライアント側で Index を設定する方法が最もシンプルである。TFDTable を使用して接続する場合、TFDTable 自身または紐付けた TClientDataSet に、並べたい順にセミコロンの区切りで IndexFieldNames プロパティを設定する。フィールドに降順を指定したい場合は、TFDTable の IndexFieldNames プロパティに「(フィールドID):D」と指定することで、そのフィールドだけ降順にすることができる。【ソース2】【図11】

なお、従来の BDE や dbExpress で降順を指定する際に必要であった、TClientDataSet 側の IndexDefs および IndexName プロパティで降順フィールドを別途指定する方法は FireDAC でも使用可能である。

また、TFDQuery を使用している場合、TClientDataSet に紐付けた上、前述の IndexDefs および IndexName プロパティを指定する。降順がない場合は IndexFieldNames プロパティでもよい。この Index 指定は ORDER BY 句よりも優先されるため、ASCII 順の並びでデータが表示される。

③ 到着順

では、特に並び順を指定せずに TFDTable や TFDQuery をオープンするとどうなるのか。答えは到着順になる場合と、EBCDIC 順になる場合の両方がある。並び順を指定していない場合は、並び順が保証されておらず、どちらの並び順になるかは、IBM i の STRSQL コマンドで SQL を実行すると確認が可能

である (STRSQL の結果と同じ並び順になる)。

意図的に到着順でレコードを表示させたい場合は、TFDQuery を使用し、SQL 文内で RRN (物理ファイルが内部保持している相対レコード番号) に対して ORDER BY 句を掛けることで、対象ファイルのレコードを到着順 (=レコードが追加された順) に並べることが可能となる。【ソース3】【図12】。

2-3. フェッチによるパフォーマンス効果

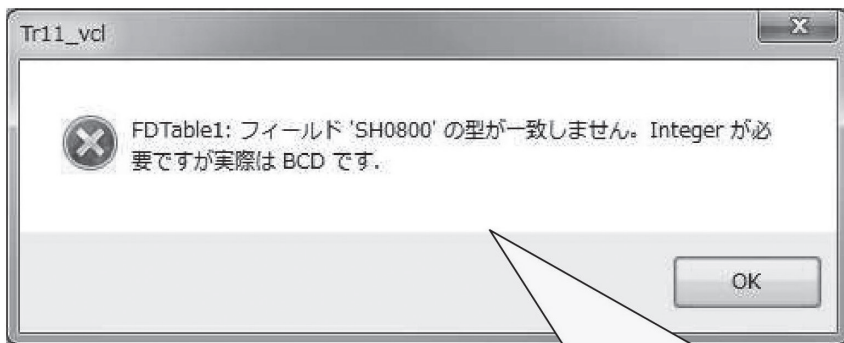
TFDConnection コンポーネントの FetchOptions プロパティによって、データをクライアント側へ転送する際の設定を変更できる。たとえば数十万件といった大量データをオープンしようとする、それだけで処理に時間がかかることが多い。しかし FireDAC の場合は、初期設定で 50 件ずつレコードがフェッチされるようになっており、データを 50 件ずつ取得・表示することによって、オープン命令からすぐに 50 件のレコードが表示できる。【図13】

この際、明細表示後にカレントレコードが最終レコード (50 件目) まで到達した状態で次のレコードを取得しようとすると、次の 50 件を取得する。この件数は、FetchOptions プロパティ内の RowsetSize サブプロパティで変更ができる。この機能は、従来から使われる TClientDataSet の PacketRecords プロパティと同じような使い方ができる。

逆にデータの件数があまり多くなく、かつデータを一括で全件表示させる必要がある場合は、初期設定で 50 件ずつになっているレコードのフェッチの設定を無制限になるように設定できる。設定は FetchOptions プロパティ内の Mode サブプロパティを初期設定の「fmOnDemand」から「fmAll」に変更する。【図14】

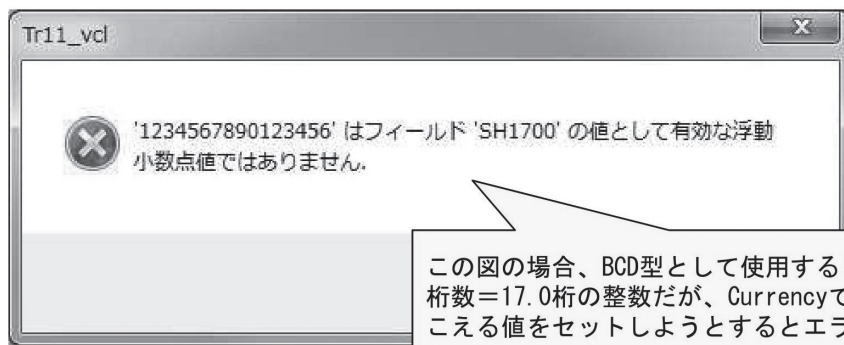
なお、TClientDataSet と紐付けている場合は RowsetSize の値よりも TClientDataSet の PacketRecords の値が優先される。この構成では実際にデータを持つコンポーネントが TClientDataSet となるからである。

図3 マッピングエラー①



BDEやdbExpressでInteger型として使用していたフィールドからそのまま移行すると、BCD型ではないというエラーになる

図4 マッピングエラー②



この図の場合、BCD型として使用するフィールドで桁数=17.0桁の整数だが、Currencyで扱える範囲をこえる値をセットしようとするエラーになる

```

end;

function TBCDField.GetAsBCD: TBCd;
var
  C: System.Currency;
begin
  if GetValue(C) then CurrToBcd(C, Result) else Result := NullBcd;
end;

function TBCDField.GetAsCurrency: Currency;
begin
  if not GetValue(Result) then Result := 0;
end;

function TBCDField.GetAsFloat: Double;
begin
  Result := GetAsCurrency;
end;

function TBCDField.GetAsInteger: Longint;
begin
  Result := Longint(Round(GetAsCurrency));
end;

function TBCDField.GetAsLargeint: Longint;
begin
  Result := Largeint(Round(GetAsCur

```

※Delphi標準のData.DB.pas内のロジック
TBCDFieldはデータの演算誤差を防ぐため内部的にCurrencyで値を取り扱っている

3.FireDACのデータ更新に関するテクニック

3-1.双方向更新機能の活用

データベースエンジンには読み込んだレコードを直接更新できる「双方向データセット形式」と、読み込み専用で更新はできないがパフォーマンスが高い「単方向データセット形式」がある。

従来のBDE接続ではTTableが双方向データセット形式で、dbExpressでは単方向データセット形式になっている。

FireDACではBDEと同様に双方向データセット形式となっているため、適切な設定を行えば読み込んだレコードへ直接更新することができる。以下に手順を記載する。

まず、先述のマッピングの設定と同様、TFDConnection側でUpdateOptionsプロパティを【図15】のように設定する。

この設定が行われていないとTFDTableやTFDQueryをオープンした際に読み取り専用として開くため、データの更新処理は正しく行えず、エラー等が発生する可能性がある。

次に、読み込んだデータを編集して更新する場合の設定を行う。それぞれ目的にあわせてUpdateOptionsプロパティのUpdateModeサブプロパティを【図16】のように設定する。

TFDTableにおける各処理では内部的にSQL文を生成・実行する仕組みを持っており、レコード参照・追加・変更・削除時に、実際にはそれぞれSELECT・INSERT・UPDATE・DELETEのSQLがTFDTableによって発行されている。

この中でUPDATEまたはDELETEを行う場合には、UpdateModeサブプロパティの設定値によって、どのレコードを更新対象とするかが決定する。【図16】のとおり、UpdateModeで設定した値によって、どのフィールドを更新条件とするかが決定してWHERE句を生成している。【図17】

続いて、更新条件とするフィールド側にも設定を行う。オブジェクトインスタクタ上にフィールドがある場合は【図

18】のように指定する。また、設計画面上ではフィールドが存在せず、ソース内で設定を行う場合は【ソース4】のように指定する。なお、UpdateModeサブプロパティが「upWhereAll」の場合は全フィールドを更新条件とするため指定不要であるが、IBM iのデータにPC側で扱えないコード等が含まれていると条件が一致しない可能性があるので注意が必要である。

UpdateModeサブプロパティが「upWhereAll」以外の場合、この指定を行っておかないと、Postの際にどのフィールドを基準に更新(WHERE句を作成)するかプログラムが把握できないため、注意が必要である。「upWhereKeyOnly」の場合は更新条件となるフィールドがないため、プログラムが自動的にupWhereAllとして更新を行う。「upWhereChanged」の場合は値を変更したフィールドだけが更新条件になるため、同じ値の他レコードもすべて更新対象になってしまう。

3-2.FireDACのトランザクション制御

FireDACにおけるトランザクション制御は、従来のBDEと近い実装方法で行うことができる。設定方法を説明する。

まずTFDConnectionをダブルクリックしてFireDAC接続エディタを開き、ODBCAdvancedパラメータを【図19】のように設定する。このパラメータにはライブラリリストなどの他の指定もできるが、複数設定の指定を行う場合は、セミコロンで区切る必要がある。

各機能においてトランザクション処理を行う方法は、従来のBDEと同様である。【ソース5】

トランザクション制御において従来のBDEと異なり注意が必要となるポイントを少し補足する。それはジョブ終了時の制御である。

BDEでは、StartTransactionからCommitまでの間にTDatabaseとの接続を明示的に終了した場合、ジョブが終了するためトランザクションはロールバックされる。しかしFireDACでは、接続終了時のデフォルトの動作設定の初期値がCommitになっているため、トランザクションがコミットされる。これ

をBDEと同様に接続終了時のデフォルトの動作設定をRollbackにするために、【図20】のようにTxOptionsプロパティの設定を行う。なお、Delphi/400アプリケーションを強制終了した場合など、ジョブが異常終了する場合はBDEと同様ロールバックされる。

4.ファイルのメンバー制御

最後にデータの取得・更新に共通したテクニックとしてメンバーの制御方法を紹介する。IBM iでは1つのデータベース・ファイルに対して複数のメンバーを持つことができる。たとえば、ワークファイルを使用する際にユーザーごとにそれぞれメンバーを指定したい場合、従来のBDEではTTableのTableNameプロパティでメンバーを直接指定できるが、FireDACではSQLで動作するため直接指定できない。【図21】

このとき任意のメンバーに対して接続する方法は大きく2種類存在する。

1つ目は、ネイティブ接続と併用可能な場合にTAS400コンポーネントからOVRDBFコマンドを発行し、ファイル名を一時変更する方法である。【ソース6】のように、ロジック内でTFDTableをオープンする前にネイティブ接続側でOVRDBFコマンドを直接発行すれば、DLTOVRコマンドで解除するか、別のOVRDBFコマンドで上書きするまで、指定したメンバーを参照・更新できるようになる【図22】。必要に応じて、TFDTableをクローズした後に、DLTOVRコマンドで一時変更を削除すれば解除できる。

2つ目は、FireDAC単体でメンバーを制御する方法である。これはWebアプリケーションやDataSnapアプリケーションといったネイティブ接続とジョブが別れてしまう構成で有効な実装方法となる。

まずSQLで実行できる「CREATE ALIAS」「DROP ALIAS」でメンバーに対して別名を作成し、TableNameプロパティでそのエイリアスを指定する。【ソース7】【図23】

この場合はOVRDBFとは異なり、CREATE ALIASは別名で実体を作成するため、既に存在する名前エイリア

図5 マッピング設定

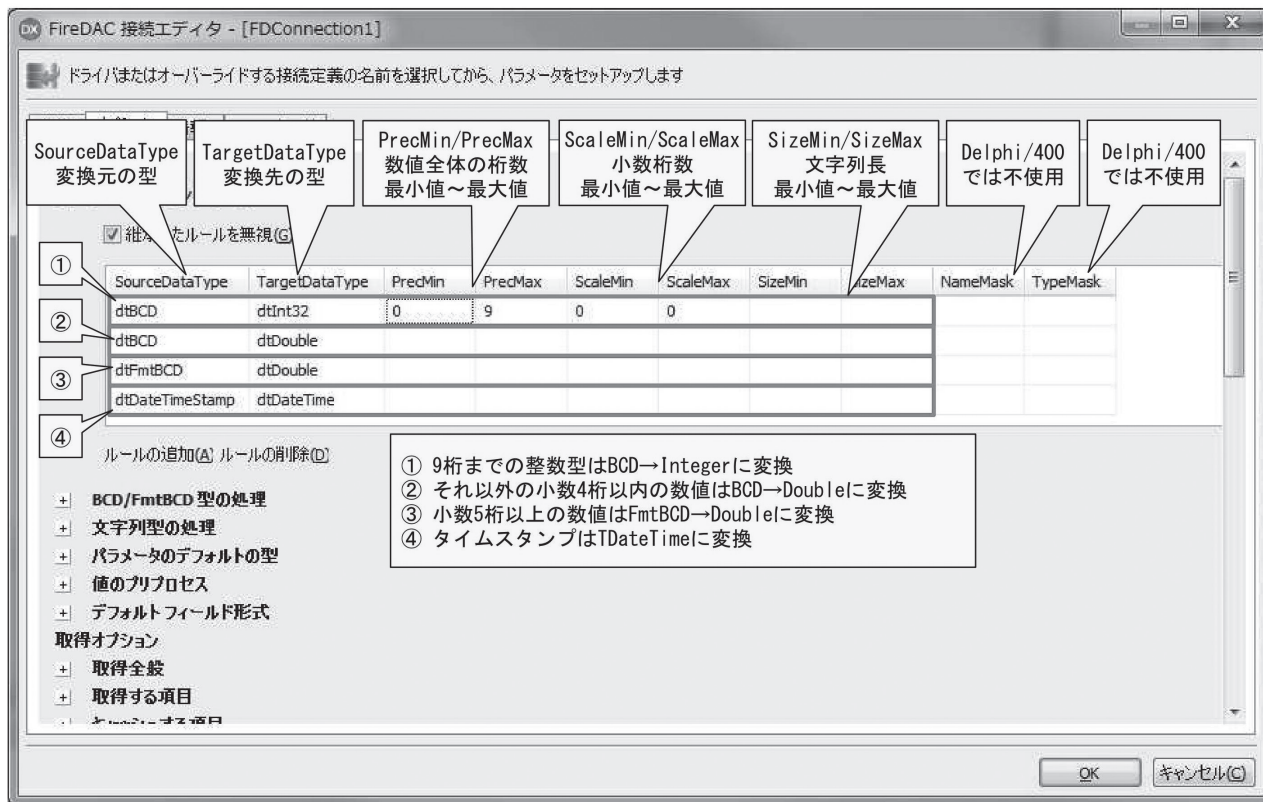


図6 FieldByNameのロジック

BCD型のフィールドに対しても、AsIntegerで値取得可能

```

procedure TForm2.Button8Click(Sender: TObject);
var
    iVal: Integer;
begin
    iVal := FDTTable1.FieldByName('SH0800').AsInteger;
    Edit2.Text := IntToStr(iVal);
    ShowMessage(IntToStr(iVal));
end;
    
```

【図2】のBCD型フィールド (値は100が入っている)

取得値をEdit2に転送

取得値をメッセージ表示

取得値をEdit2に転送

※AsCurrencyやAsFloatでも読み取れるが、Currencyで扱える範囲の値であることが条件

スを作成しようとするエラーになってしまう。名前の一部にジョブ番号を使用するなど、エイリアス名が重複しないよう留意する必要がある。

以上が FireDAC でのメンバーの制御テクニックである。

5.まとめ

本稿では、Delphi/400 の開発者目線で、検証済みの技術ポイントを中心に、実践的な FireDAC のテクニックを紹介した。

最近では Windows10 対応などを中心に Delphi/400 のバージョンアップを行う開発も多く、それに伴って BDE や dbExpress から FireDAC へ変更する機会が増えている。その中で、参照・更新や、メンバー処理にまつわるテクニックを把握しておけば、これまでのプログラムをスムーズに移行できる。

これまでのシステム開発経験を本稿で共有することで、Delphi/400 の新機能である FireDAC を広く活用していただければ幸いである。

M

図7 FieldByNameのロジック

BCD型のフィールドに対しても、AsIntegerで値更新可能

```

procedure TForm2.Button7Click(Sender: TObject);
var
    iVal: Integer;
begin
    iVal := StrToIntDef(Edit2.Text, 0);
    FTable1.Append;
    FTable1.FieldByName('SH0800').AsInteger := iVal;
    FTable1.Post;
end;
    
```

Tableにレコードを追加

【図2】のBCD型フィールドに、Integer値をセット

更新

セットした値が正しく更新されている

SHAA18	SH0018	SH0800
A000000001	TEST	100
		200

※AsCurrencyやAsFloatでも書き込めるが、Currencyで扱える範囲の値であることが条件

図8 データの並び順について

この「A型」フィールド(SHAA18)がキーのファイルで、上からこの順番に5件レコードを登録した場合

```

    行の位置指定 . . . . .
    行 . . . . . 1 . . . . . 2 . . . . .
    A型          O型
    000001 TEST_KEY                1 件目          0
    000002 カブシキイシャミカロ    2 件目          0
    000003 1234567890              3 件目          0
    000004 KEY_TEST                4 件目          0
    000005 9876543210             5 件目          0
    ***** 報告書の終わり *****
    
```

SHAA18	SH0018	SHAA18	SH0018	SHAA18	SH0018
カブシキイシャミカロ	2 件目	1234567890	3 件目	TEST_KEY	1 件目
KEY_TEST	4 件目	9876543210	5 件目	カブシキイシャミカロ	2 件目
TEST_KEY	1 件目	KEY_TEST	4 件目	1234567890	3 件目
1234567890	3 件目	TEST_KEY	1 件目	KEY_TEST	4 件目
9876543210	5 件目	カブシキイシャミカロ	2 件目	9876543210	5 件目

①EBCDIC順 半角カナ→英字→数字の順に並ぶ	②ASCII順 数字→英字→半角カナの順に並ぶ	③到着順 レコード登録順に並ぶ
-----------------------------	----------------------------	--------------------

図9 データの並び順について

上からこの順番にレコードを登録したファイル
(登録後に一部レコードを削除し、現在は16レコード)

行	SHAA10	SHAA11	SHAA12
000001	MIGARO_001	SANKEIBLD_1	REPORT_00001
000002	MIGARO_001	AAAAAAAAA_1	REPORT_00002
000003	MIGARO_001	TECHNICAL_2	REPORT_00005
000004	MIGARO_001	123456789_2	REPORT_00006
000005	356356_002	TECHNICAL_1	REPORT_00009
000006	MIGARO_002	OSAKACITY_1	REPORT_00010
000007	MIGARO_002	XXXXXXXXX_2	REPORT_00013
000008	MIGARO_002	TECHNICAL_2	REPORT_00014
000009	356356_001	TECHNICAL_1	REPORT_00017
000010	356356_001	333333333_1	REPORT_00018
000011	356356_001	NANIWA_KU_2	REPORT_00021
000012	356356_001	テクニカルポ- 2	REPORT_00022
000013	MIGARO_002	SYSTEMIKA_1	REPORT_00025
000014	356356_002	MIGARON_ 1	REPORT_00026
000015	356356_002	MINATOMAC_2	REPORT_00029
000016	356356_002	XYZXYZXYZ_2	REPORT_00030

***** ***** 報告書の終わり *****

フィールドについて
(SHAA10~12はいずれもAタイプ)

①SHAA10=キー、昇順
値は英字始まりのレコードと
数字始まりのレコードが混在

②SHAA11=非キー
レコードによって様々な値を登録

③SHAA12=非キー
"REPORT_" +レコード登録時の連番を登録
(※削除レコードの連番は欠番)

ソース1 EBCDIC順参照のソース記述例

```

procedure TForm1.Button4EClick(Sender: TObject);
begin
  FDQuery1.Close;
  FDQuery1.SQL.Clear;
  FDQuery1.SQL.Add(' SELECT * FROM TR11F05 ');
  FDQuery1.SQL.Add(' ORDER BY SHAA11, SHAA12 ');
  FDQuery1.Open;
end;
    
```

TFDQueryのSQLに、STRSQLで指定する際と
同じようにORDER BYを掛ける

図10 EBCDIC順参照の取得結果

SHAA10	SHAA11	SHAA12
356356_001	テクニカルポ- 2	REPORT_00022
MIGARO_001	AAAAAAAAA_1	REPORT_00002
356356_002	MIGARON_ 1	REPORT_00026
356356_002	MINATOMAC_2	REPORT_00029
356356_001	NANIWA_KU_2	REPORT_00021
MIGARO_002	OSAKACITY_1	REPORT_00010
MIGARO_001	SANKEIBLD_1	REPORT_00001
MIGARO_002	SYSTEMIKA_1	REPORT_00025
356356_002	TECHNICAL_1	REPORT_00009
356356_001	TECHNICAL_1	REPORT_00017
MIGARO_001	TECHNICAL_2	REPORT_00005
MIGARO_002	TECHNICAL_2	REPORT_00014
MIGARO_002	XXXXXXXXX_2	REPORT_00013
356356_002	XYZXYZXYZ_2	REPORT_00030
MIGARO_001	123456789_2	REPORT_00006
356356_001	333333333_1	REPORT_00018

SQLで指定した通りに、
EBCDIC順(半角カナ→英字→数字の順)で
SHAA11の昇順>SHAA12の昇順に並ぶ

ソース2 ASCII順参照のソース記述例

```

procedure TForm1.Button4WClick(Sender: TObject);
begin
  FTable1.Close;
  FTable1.TableName := 'TR11F05';
  FTable1.IndexFieldNames := 'SHAA11;SHAA12:D';
  FTable1.Open;
end;

```

TFTableにIndexFieldNamesを指定
 ・複数フィールドはセミコロンで区切る
 ・降順にしたいフィールドは後ろに「:D」を付ける

図11 ASCII順参照の取得結果

SHAA10	SHAA11	SHAA12
MIGARO_001	123456789_2	REPORT_00006
356356_001	333333333_1	REPORT_00018
MIGARO_001	AAAAAAAAA_1	REPORT_00002
356356_002	MIGARON_1	REPORT_00026
356356_002	MINATOMAC_2	REPORT_00029
356356_001	NANIWA_KU_2	REPORT_00021
MIGARO_002	OSAKACITY_1	REPORT_00010
MIGARO_001	SANKEIBLD_1	REPORT_00001
MIGARO_002	SYSTEMIKA_1	REPORT_00025
356356_001	TECHNICAL_1	REPORT_00017
356356_002	TECHNICAL_1	REPORT_00009
MIGARO_002	TECHNICAL_2	REPORT_00014
MIGARO_001	TECHNICAL_2	REPORT_00005
MIGARO_002	XXXXXXXXX_2	REPORT_00013
356356_002	XYZXYZXYZ_2	REPORT_00030
356356_001	テクニカルホ-2	REPORT_00022

IndexFieldNamesで指定した通りに、
 ASCII順(数字→英字→半角カナの順)で
 SHAA11の昇順>SHAA12の降順に並ぶ

ソース3 到着順参照のソース記述例

```

procedure TForm1.Button4TClick(Sender: TObject);
begin
  FDQuery1.Close;
  FDQuery1.SQL.Clear;
  FDQuery1.SQL.Add(' SELECT TR11F05.*, RRN(TR11F05) AS RRN1 FROM TR11F05 ');
  FDQuery1.SQL.Add(' ORDER BY RRN1 ');
  FDQuery1.Open;
end;

```

TFDQueryのSQLに、フィールドではなく
 RRN(相対レコード番号)に対してORDER BYを掛ける

図12 到着順参照の取得結果

SHAA10	SHAA11	SHAA12	RRN1
MIGARO_001	SANKEIBLD_1	REPORT_00001	1
MIGARO_001	AAAAAAAAA_1	REPORT_00002	2
MIGARO_001	TECHNICAL_2	REPORT_00005	5
MIGARO_001	123456789_2	REPORT_00006	6
356356_002	TECHNICAL_1	REPORT_00009	9
MIGARO_002	OSAKACITY_1	REPORT_00010	10
MIGARO_002	XXXXXXXXX_2	REPORT_00013	13
MIGARO_002	TECHNICAL_2	REPORT_00014	14
356356_001	TECHNICAL_1	REPORT_00017	17
356356_001	333333333_1	REPORT_00018	18
356356_001	NANIWA_KU_2	REPORT_00021	21
356356_001	テクニカルホ-2	REPORT_00022	22
MIGARO_002	SYSTEMIKA_1	REPORT_00025	25
356356_002	MIGARON_1	REPORT_00026	26
356356_002	MINATOMAC_2	REPORT_00029	29
356356_002	XYZXYZXYZ_2	REPORT_00030	30

SQLで指定の通りに、RRNの昇順でソートされるため
 実質的に到着順(レコード登録順)にレコードが並ぶ
 ※ORDER BY句に「DESC」を付ければ、
 到着順の逆順で表示させることも可能

図13 フェッチのレスポンス

SHAA18 A型	SHOO18 O型	SHDATE 日付型	SHTIME 時刻型	SHJJ18 J型	SH0800	SH0900	SH1000	SH1100	SH1200	SH1300	SH1400
A000000001		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000002		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000003		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000004		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000005		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000006		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000007		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000008		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000009		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000010		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000011		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000012		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000013		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000014		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000015		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000016		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000017		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000018		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000019		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000020		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000021		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000022		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000023		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000024		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000025		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000026		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000027		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000028		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000029		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234
A000000030		2018/08/17	18:50:39		12345678	123456789	1234567890	12345678901	123456789012	1234567890123	12345678901234

(中略)

50,000レコードのデータをオープンしてから表示までの所要時間：
 ・フェッチ実施時（初期設定50レコード単位）= 0.239秒
 ・フェッチ未実施時 = 10.586秒

図14 フェッチ設定の変更

オブジェクト インспекタ

FDTable1 TFDTable

検索

プロパティ | イベント

- Constraints (TCheckConstraints)
- ConstraintsEnab False
- DetailFields
- Exclusive False
- FetchOptions (TFDFetchOptions)
 - AssignedValue: [evItems]
 - AutoClose True
 - AutoFetchAll afAll
 - Cache [fiBlobs, fiDetails, fiMeta]
 - CursorKind ckAutomatic
 - DetailCascade False
 - DetailDelay 0
 - DetailOptimize True
 - DetailServerCo False
 - Items [fiBlobs, fiDetails]
 - LiveWindowFa False
 - LiveWindowPa True
 - Mode **fmOnDemand**
 - fmAll
 - fmExactRecsMax
 - fmManual
 - fmOnDemand
 - RecordCountM fmAll
 - RecsMax fmExactRecsMax
 - RecsSkip fmManual
 - RowsetSize 30
 - Unidirectional False
- FieldOptions (TFieldOptions)
- Filter
- FilterChanges [rtModified, rtInserted, rtUnmodified]
- Filtered False
- FilterOptions []
- FormatOptions (TFDFormatOptions)
 - フィールド エディタ... クイック編集... クイック コピー名
 - ビジュアルにバインド... バインド ソースの追加

すべての項目が表示されています

FetchOptions内の「Mode」プロパティを「fmAll」に変更することで、全件一括取得に変更できる

図15 UpdateOptionsの設定

FDConnection1 TFDCConnection

検索

プロパティ イベント

StopOptions	[xoIfCmdsInactive,xoIfA
UpdateOptions	(TFDUpdateOptions)
AssignedValues	[uvUpdateMode,uv]
AutoCommitUpdates	<input type="checkbox"/> False
CheckReadOnly	<input type="checkbox"/> False
CheckRequired	<input type="checkbox"/> False
CheckUpdatable	<input type="checkbox"/> False
CountUpdatedRecords	<input checked="" type="checkbox"/> True
EnableDelete	<input checked="" type="checkbox"/> True
EnableInsert	<input checked="" type="checkbox"/> True
EnableUpdate	<input checked="" type="checkbox"/> True
FastUpdates	<input type="checkbox"/> False
FetchGeneratorsPoint	gpDeferred
GeneratorName	
LockMode	lmNone
LockPoint	lpDeferred
LockWait	<input type="checkbox"/> False
ReadOnly	<input type="checkbox"/> False
RefreshDelete	<input checked="" type="checkbox"/> True
RefreshMode	rmManual
RequestLive	<input checked="" type="checkbox"/> True
UpdateChangedFields	<input checked="" type="checkbox"/> True
UpdateMode	upWhereChanged
UpdateNonBaseFields	<input checked="" type="checkbox"/> True
UpdateTransaction	

接続エディタ... クイック編集... クイックコピー名

すべての項目が表示されています

FDConnection1の場合、UpdateOptions プロパティを次のとおり設定する

CheckReadOnly : False
(Trueだと編集できない)

CheckRequired : False
(Trueだと更新時に空のフィールドがあるとエラーになる)

CheckUpdatable : False
(更新時のTableの状態監視を止める)

RefreshMode : rmManual
(更新後、レコードを自動リフレッシュしない。リフレッシュ時のエラー防止)

UpdateOptions : upWhereChanged
(詳細は後述)

UpdateNonBaseFields : True
(更新時に正しくテーブルが指定されていないという内部エラー防止)

図16 UpdateOptionsの設定

Lockwait False

ReadOnly False

RefreshDelete True

RefreshMode **rmManual**

RequestLive True

UpdateChangedFields True

UpdateMode **upWhereKeyOnly**

UpdateNonBaseFields upWhereAll

UpdateTransaction upWhereChanged

upWhereKeyOnly

接続エディタ... クイック編集... クイックコピー名

すべての項目が表示されています

目的にあわせて、UpdateMode サブプロパティを次のとおり設定する

Edit~PostまたはDelete時に、更新条件とするフィールドを判定

upWhereAll
→全てのフィールドを更新条件とする
(全角フィールドが正しく判定されない場合がある)

upWhereChanged
→別途指定した更新条件フィールド
および、今変更したフィールド(Edit時)を更新条件とする

upWhereKeyOnly (初期値)
→別途指定した更新条件フィールドのみを更新条件とする

※AppendやInsertの場合はWHERE句を使用しないため影響なし

図17 更新条件とWHERE句

特定のレコードの値を変更して
Postしようとしたタイミング

SHAA18	SH0018	SH0800
A000000006	TEST	20
A000000007	TEST	20
A000000008	TEST	20

更新

SHAA18	SH0018	SH0800
A000000006	TEST	15
A000000007	TEST	20
A000000008	TEST	20

内部的には以下のようなSQLを生成している

```
UPDATE YSADA."YSADALIB/TR11F02"
SET SH0800 = :NEW_SH0800
WHERE SHAA18 = :OLD_SHAA18
AND SH0018 = :OLD_SH0018
AND SH0800 = :OLD_SH0800
```

どのフィールドをWHERE句の更新条件に指定するかを【図16】のUpdateModeによって決定する
(本図はupWhereAllに設定した場合のSQLで、3フィールド全てを更新条件としている)

図18 更新条件フィールドの設定

更新条件としたいフィールドの
ProviderFlagsプロパティで、
pfInKeyサブプロパティをTrueに設定
する

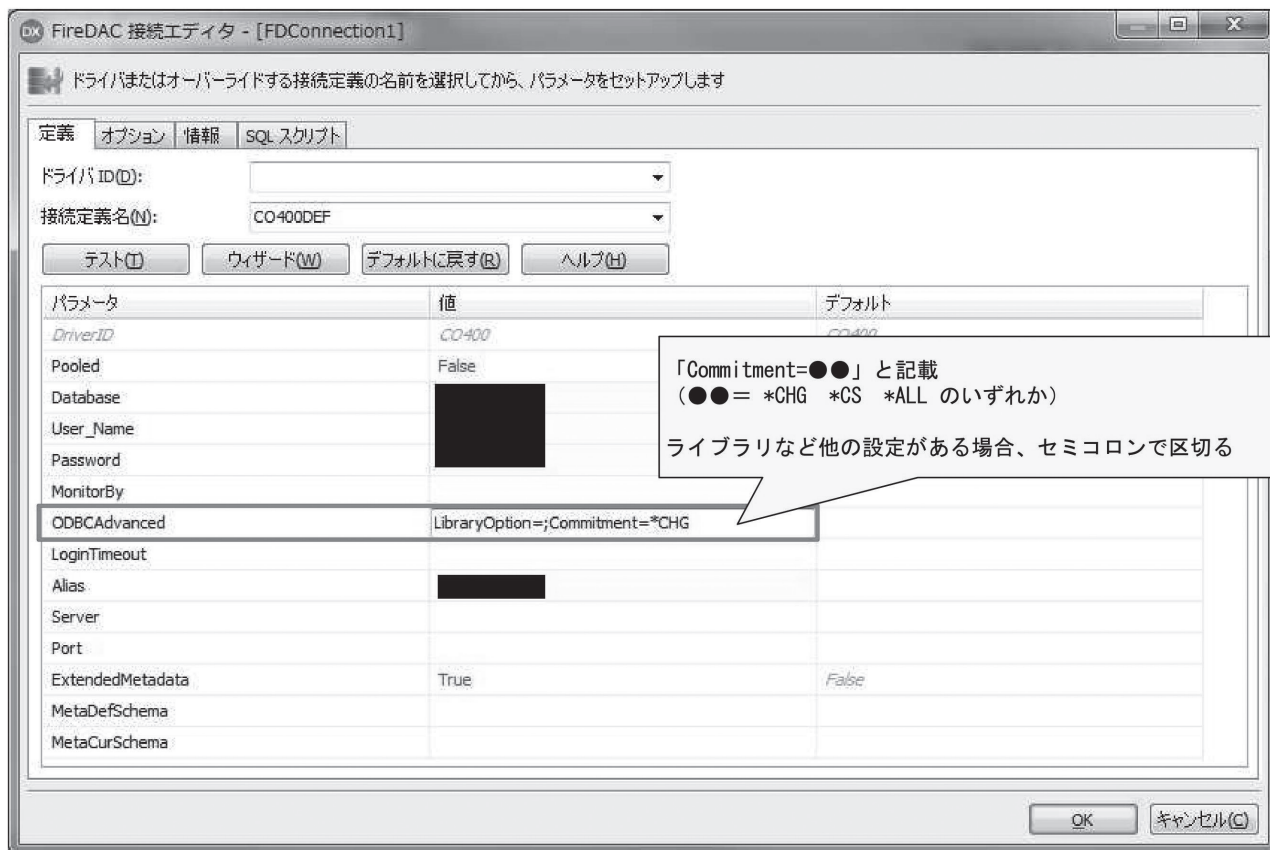
ソース4 更新キーフィールド設定のソース記述例

```
// 画面のデータをワークにセット
with tblUpdate do
begin
  TableName := 'TR11F02';
  Open;
  // 一意になるように更新キーフィールドを指定(複数指定可能)
  FieldByName('SHAA18').ProviderFlags := [pfInKey]; // キーフィールド
  First;

  for i := 1 to stgList.RowCount - 1 do
  begin
    Edit;
    // (各フィールド値セット)
    Post;
    Next;
  end;
end;
end;
```

フィールドのProviderFlagsプロパティを
直接ソースで指定する

図19 トランザクションに必要な設定



ソース5 トランザクションのソース記述

```

procedure TForm2.SpeedButton4Click(Sender: TObject);
begin
  if (not FDConnection1.InTransaction) then
  begin
    FDConnection1.StartTransaction;
  end;
end;

```

トランザクションの開始

```

procedure TForm2.SpeedButton5Click(Sender: TObject);
begin
  if (FDConnection1.InTransaction) then
  begin
    FDConnection1.Commit;
  end;
end;

```

トランザクションのコミット

```

procedure TForm2.SpeedButton6Click(Sender: TObject);
begin
  if (FDConnection1.InTransaction) then
  begin
    FDConnection1.Rollback;
  end;
end;

```

トランザクションのロールバック

図20 明示切断時設定の変更点

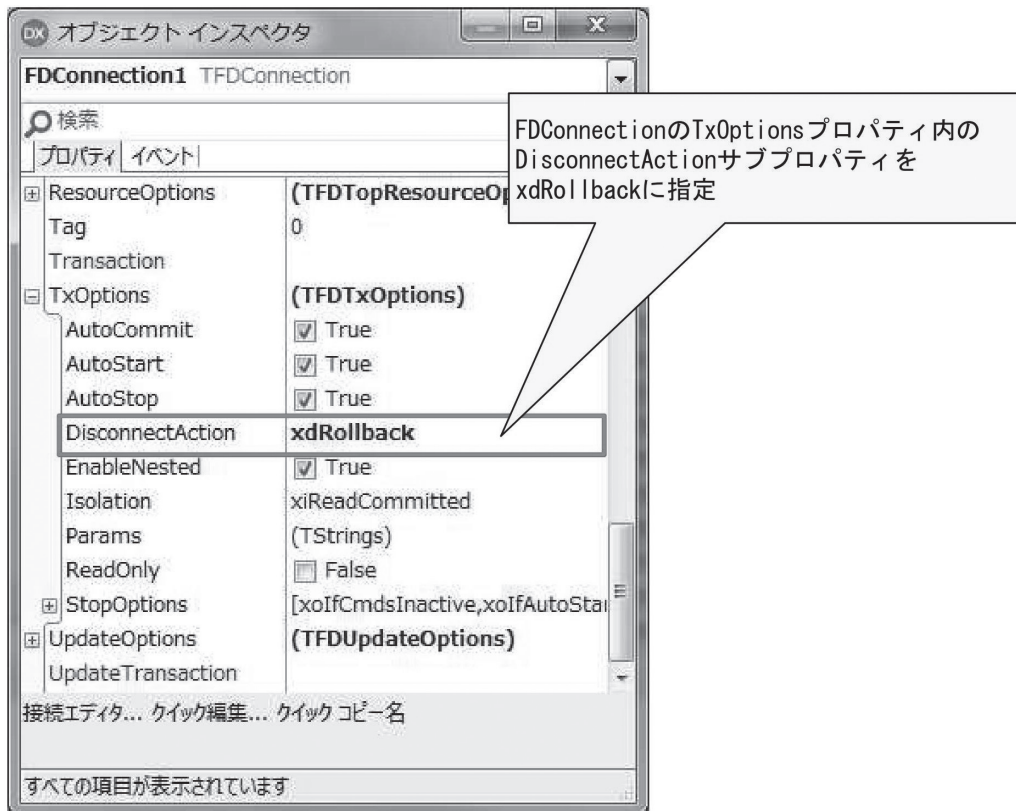


図21 TableNameのメンバー指定

OPT	メンバー	日付	テキスト
—	TR11F05	18/08/20	テストのソート順検証用
—	X_MBR1	18/08/20	別メンバー参照の検証用

※同一ファイルのメンバー一覧画面
 TR11F05 : 【図9】のレコードが入っている
 X_MBR1 : 今回参照用の別レコードが入っている

BDEのように、TableName プロパティに括弧書きでメンバーを指定する事はできない

[FireDAC][Phys][ODBC][SystemObjects][ClientObjects/400][SQLPrepareCur]カラム修飾子またはテーブルAが未定義である。

ソース6 OVRDBFによるメンバー参照

```

procedure TForm1.Button4Click(Sender: TObject);
begin
    FTable1.Close;
    FTable1.TableName := 'TR11F05';
    AS4001.RemoteCmd(' OVRDBF FILE (TR11F05) TOFILE (TR11F05) MBR (X_MBR1) OVRSCOPE (*JOB) ');
    FTable1.Open;
end;

```

RemoteCmdでネイティブ側からOVRDBFコマンドを発行し、メンバー指定を行う

テーブルのClose後、必要に応じてDLTOVRコマンドを発行し、メンバー指定を解除する
 AS4001.RemoteCmd(' DLTOVR FILE (TR11F05) OVRSCOPE (*JOB) ');

図22 OVRDBFのメンバー参照結果

SHAA10	SHAA11	SHAA12
MEMBER	X_MBR1	TEST01
MEMBER	X_MBR1	TEST02
MEMBER	X_MBR1	TEST03
MEMBER	X_MBR1	TEST04

【図9】と同じファイルだが、指定したメンバーのレコードが表示される

ソース7 ALIASによるメンバー参照

```
// ①テーブルオープンを行う処理
procedure TForm1.Button5Click(Sender: TObject);
begin
    FDTable1.Close;
    FDTable1.TableName := sAliasName;
    FDTable1.Open; // ここで②が走る
end;
```

TableNameにエイリアスの名前を指定
※他のジョブと重複しない名前を
予め変数にセットしておく
(「OO+ジョブ番号」形式を推奨)

```
// ②テーブルオープン時イベント
procedure TForm1.FDTable1BeforeOpen(DataSet: TDataSet);
begin
    FDCConnection1.ExecSQL('CREATE ALIAS ' + sAliasName + ' FOR TR11F05(X_MBR1)');
end;
```

メンバーを指定してエイリアスを生成

```
// ③画面クローズ時イベント
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    FDTable1.Close; // ここで④が走る
end;
```

※FormDestroyでは④が走らない場合がある

```
// ④テーブルクローズ時イベント
procedure TForm1.FDTable1AfterClose(DataSet: TDataSet);
begin
    FDCConnection1.ExecSQL('DROP ALIAS ' + sAliasName);
end;
```

エイリアスの削除

図23 ALIASのメンバー参照結果

SHAA10	SHAA11	SHAA12
MEMBER	X_MBR1	TEST01
MEMBER	X_MBR1	TEST02
MEMBER	X_MBR1	TEST03
MEMBER	X_MBR1	TEST04

【図22】と同様、指定したメンバーのレコードが表示される