FireDAC

【A2】Delphi/C++Builderテクニカルセッション

はじめてのFireDAC

2013年9月13日

株式会社三菱電機ビジネスシステム

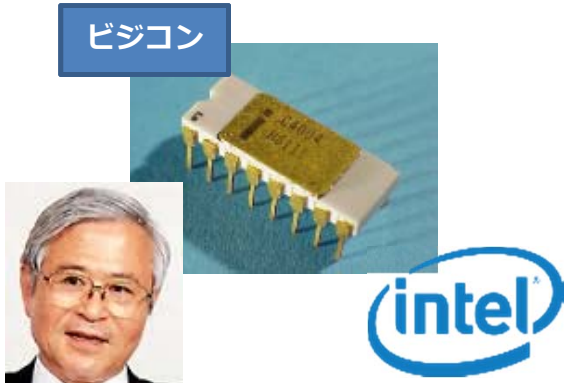
田中 芳起

Ver.1.0.6

自己紹介

会社：株式会社三菱電機ビジネスシステム

- 三菱電機100%出資(4億円)
- 1973年12月設立、創立40年
- 中堅・中小企業向けソフトウェアの開発・販売
- http://www.melb.co.jp/



嶋 正利

名前：田中 芳起（たなか よしき）

- 会社では、PMO (Project Management Office) を担当
- Delphiとは 1.0US版 からの付き合い

- ホームページ：http://www.avsoft.jp/
- ブログ　　：http://avsoft.typepad.jp/blog/
- Facebook　：https://www.facebook.com/yoshiki.tanaka.942/

プライベートの紹介

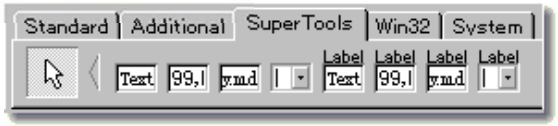
コンポーネントの開発・公開

・SuperEdit （入力系コンポーネント）
・SuperGrid （グリッド・コンポーネント）
・大手コンビニのシステムで使用されている

FamilyMart

ampm

SUPEREDIT



SUPERGRID

NO	カラム名	Null可?	データ型	長さ	表示	演算子	下限値	上限値	並び順序	昇順/降順
1	EMP_ID	×	NUMBER	4	☑	BETWEEN	100	150		
2	EMP_NAME	×	VARCHAR2	20	☑	=			1	ASC
3	EMP_SEX	○	VARCHAR2	2	☑	=				
4	EMP_DEP	○	VARCHAR2	6	☑	=				
5	EMP_JOB	○	VARCHAR2	6	☑	=				
6	EMP_ADD	○	VARCHAR2	8	☑	=				
7	EMP_OFFICE	○	NUMBER		☑	=				

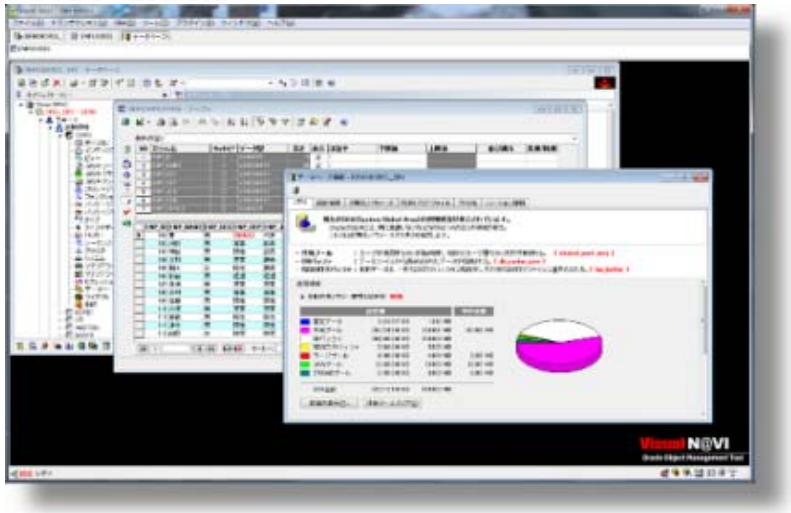
プライベートの紹介

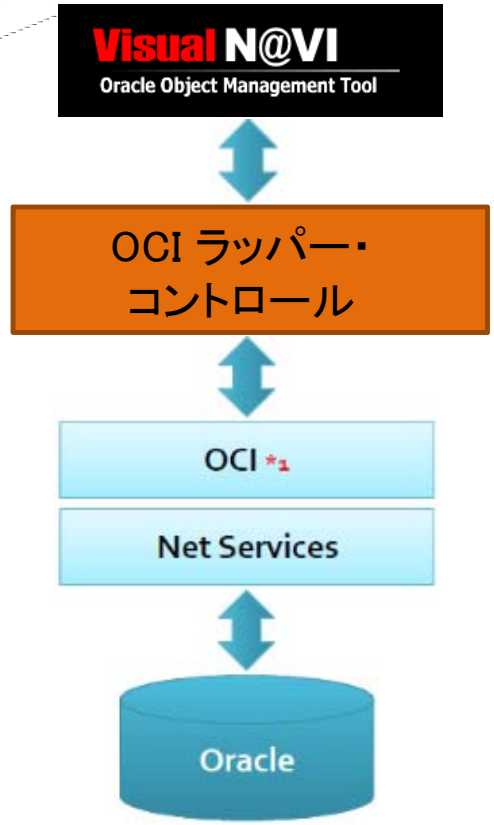
統合型開発支援ツールの開発・公開

free

- Oracleアプリケーション開発支援
- Guiによるデータベース管理機能
- 統合型開発支援ツール
- オラクル社のOCI*1を使用した高速接続を実現

VISUAL NAVI





※ Facebookページで情報発信しています

*1 **OCI**(Oracle Call Interface)は、Oracle社が提供するAPIです

アジェンダ

- **BDEのおさらい**
- BDEとは?
- BDE(Borland Database Engine)の構造
- データベースコンポーネント
- コンポーネント内ではBDEにどうアクセスしているか?
- BDE環境設定ユーティリティ(BDE Administrator)
- BDEの現状
- BDEの抱える課題

- **BDEからの移行**
- BDEから移行するための選択肢
- BDEから移行する「**新たな**」選択肢

- **FireDACについて**
- FireDACとは?
- FireDACを使用するメリットは‥
- FireDACアプリケーションの構造
- データベースドライバの対応表
- BDEとFireDAC 主要コンポーネントの比較
- データベースクラスの継承関係

アジェンダ

- **TADConnectionコンポーネント**
- プロパティの比較
- 接続の設定‥Oracleの場合
- トランザクション管理
- ログイン制御
- データ型のマッピングを変更する
- Connection Editor でSQLを実行する
- SQL文の実行
- DBMSのサーバ／クライアントのバージョン番号を求める

- **TADPhysXXXXDriverLinkコンポーネント**
- TADPhysXXXXDriverLinkとは?
- データベースとドライバの対応表

- **TADGUIxWaitCursorコンポーネント**
- TADGUIxWaitCursorとは?
- BDEアプリケーションからの移植
- FireMonkeyアプリケーションでの注意点

アジェンダ

- **TADTableコンポーネント**
- プロパティの比較
- カーソルを移動する
- データの更新
- レコードの検索
- Bookmark を利用する
- データコントロールへの描画抑制
- 行の選択
- 項目の参照

- **TFieldコンポーネント**
- TFieldコンポーネントとは?
- プロパティの比較
- TFieldのプロパティ／イベントを設定する
- 特定フィールドを非表示にする
- その他プロパティの設定を変更する
- 計算項目を追加する
- LOB(Large OBject)データの登録／保存

アジェンダ

- **TADQueryコンポーネント**
- プロパティの比較
- SQL文を設定する
- 静的問い合わせ
- 動的問い合わせ
- 大量のＤＭＬを高速に実行する
- 複数SQLのデータセットを利用する
- データベース例外を捕捉する
- マクロを使う
- データセットの状態を調べる
- 結果セットの変更

- **TADUpdateSQLコンポーネント**
- キャッシュアップデート
- プロパティの比較
- キャッシュアップデートの設定
- 更新情報のサーバーへの書き込み
- OnUpdateRecord イベント
- OnUpdateError イベント

アジェンダ

- **TADStoredProcコンポーネント**
- プロパティの比較
- PROCEDUREの実行
- FUNCTIONの実行
- PACKAGEの実行

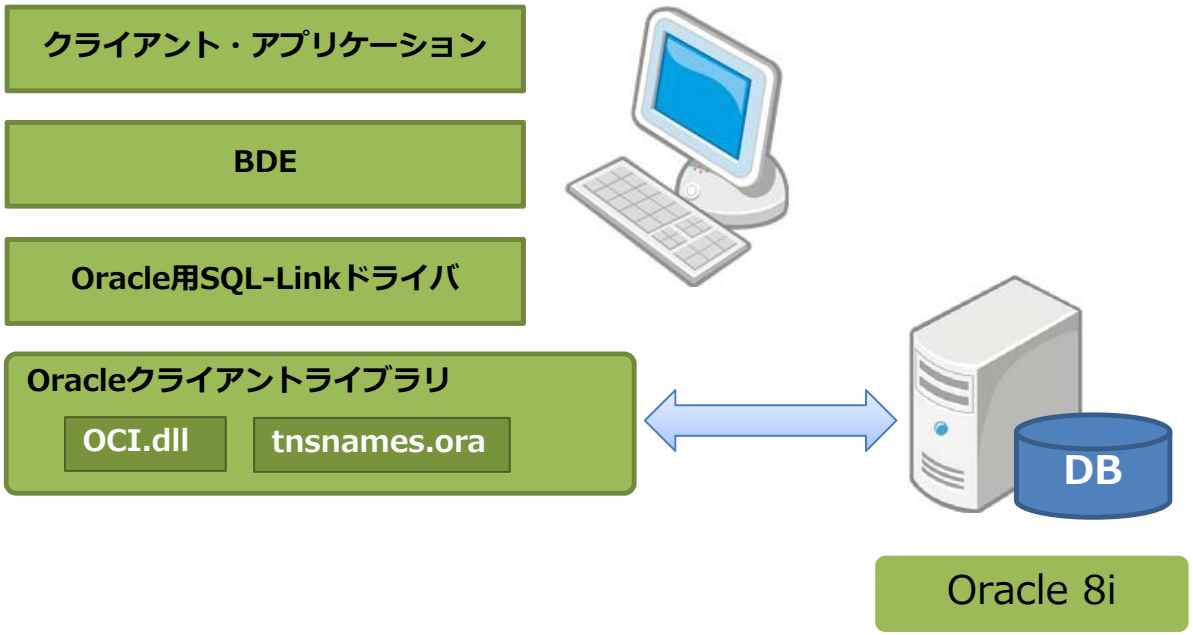
- **まとめ**
- まとめ
- 参考情報

はじめてのFireDAC

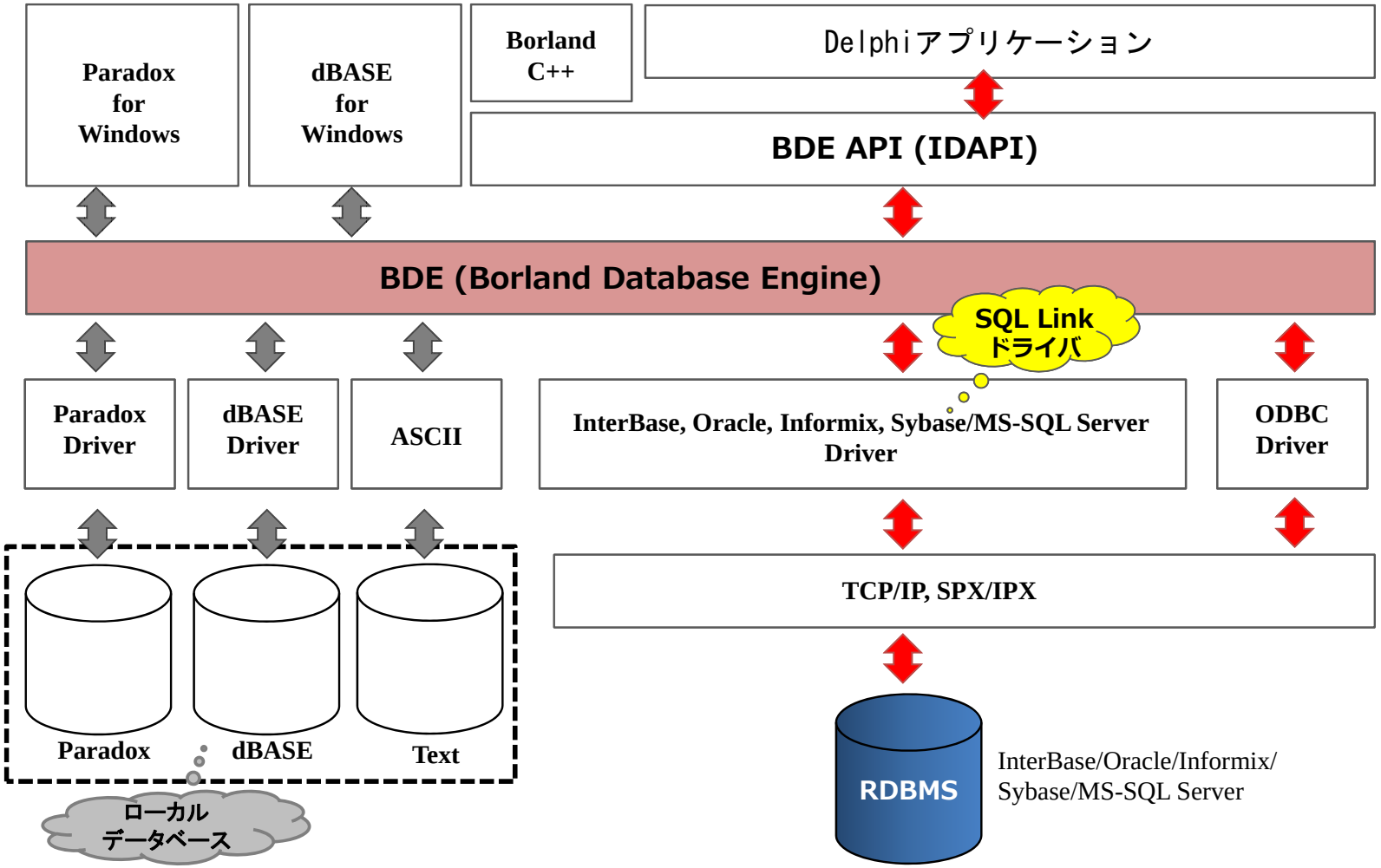
BDEのおさらい

BDEとは?

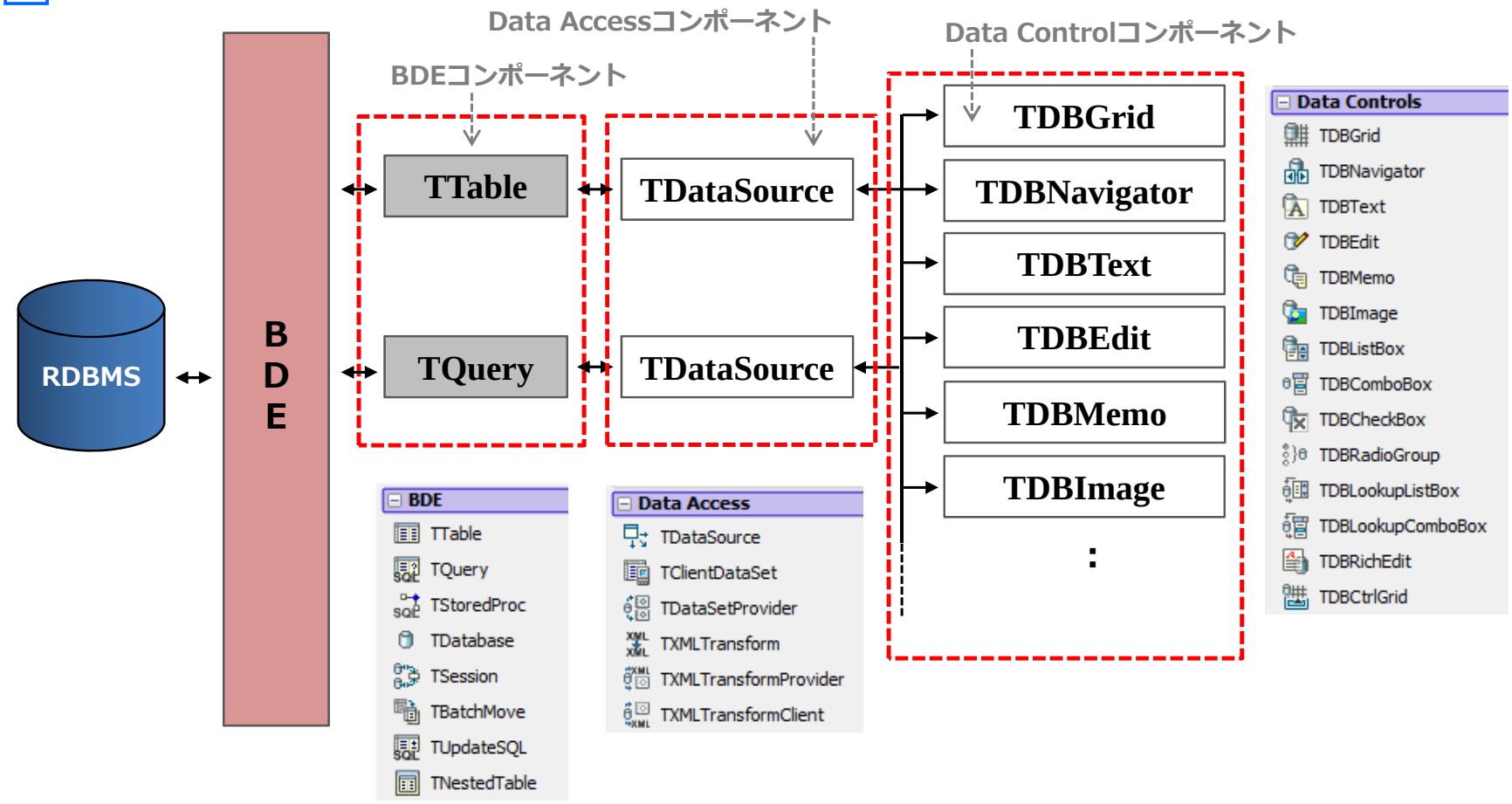
- BDEは、RDBMSとDelphi(C++Builder)アプリケーションをつなぐミドルウェア
- Paradox、dBASEのデータベースエンジンとして使われている
- ローカルデータベース以外にも、サーバーデータベースにも透過的にアクセスが可能
- BDE APIを使用して直接BDEを使用することが可能



BDE (Borland Database Engine) の構造



データベースコンポーネント



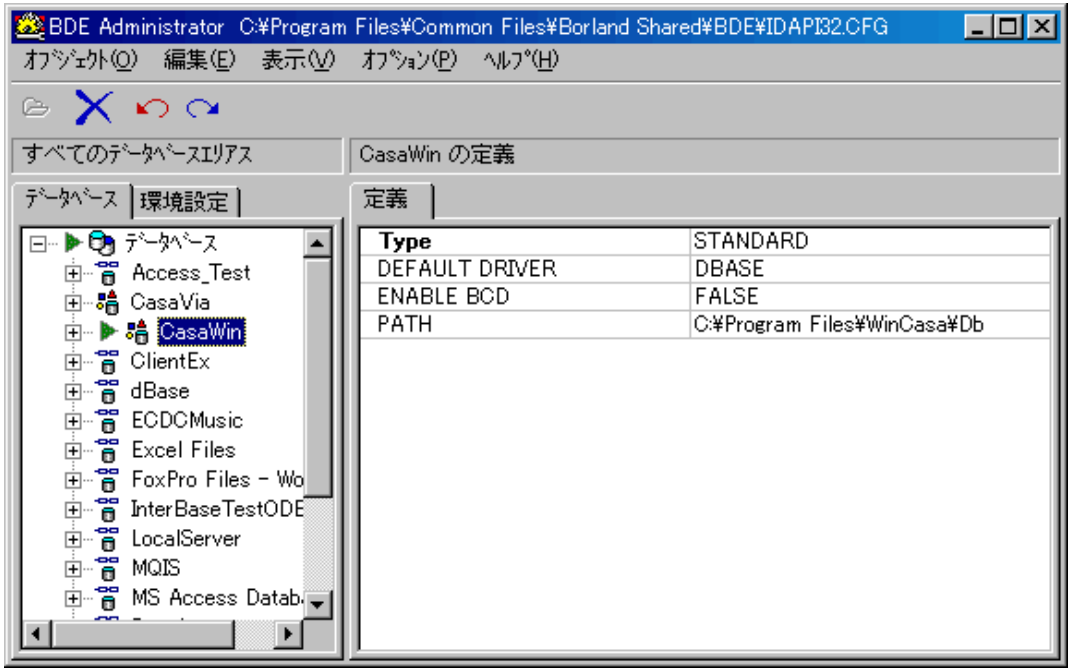
コンポーネント内ではBDEにどうアクセスしているか?

次のソースコードは、TTableコンポーネントの「DeleteTableメソッド」の内部処理

```
procedure TTable.DeleteTable;
begin
  CheckInactive;
  SetDBFlag(dbfTable, True);
  try
    Check(DbiDeleteTable(DBHandle, NativeTableName, GetTableTypeName));
  finally
    SetDBFlag(dbfTable, False);
  end;
end;
```

メソッド内部で**「DbiDeleteTable」**というIDAPI関数を呼び出して、テーブルの削除を実現している

BDE環境設定ユーティリティ(BDE Administrator)



データベースタブ
・BDEエリアス*1の設定・管理

環境設定
・ドライバ情報の管理

***1 BDEエリアス**
データベース接続に必要な情報をまとめて管理するBDEオブジェクト

BDEの現状

BDEは2002年に開発・保守が終了
・不具合があっても修正パッチの提供はされていない

BDEの最新バージョンは 5.2
・Delphi 7/C++Builder 6以降、BDEのバージョンは更新されていない
・現在のDelphi/C++Builder製品に付属するBDEは、あくまで過去の資産保守用

動作保証プラットフォームは Windows XPまで
・Windows XP以降にリリースされたOSバージョンは動作保証がされていない
（例えば、Windows Vista/7/8、Windows2003/2008 Serverなど）

BDEの抱える課題

近年の(PC)ハードウェア構成にマッチしない
・4G越えのHDDの空き容量が正しく認識されない
・マルチコア／プロセッサ搭載のPC上での動作が不安である

OSの新機能に対しては、Vista以降対応していない
・UAC(User Account Control)
・ASLR(Address Space Layout Randomization)によるアドレス衝突が発生
・BDEアプリを複数同時起動した場合、初期化エラーが発生することがある

対応しているデータベースのバージョンが古い
・Oracle(8.1.6)、MS-SQL Server(7.0)、DB2(5.0)‥
・詳細は以下のURLで確認
http://support.embarcadero.com/jp/article/36002

とにかく配布が面倒
・アプリケーションを配布する場合、BDEとSQL-Linkも一緒に配布する必要がある
・環境設定ユーティリティでBDEやエリアス設定が必要
・BDEのバージョンやエリアスの重複等、実行環境に依存する

最近のニーズに適合したアプリケーションが作成できない
・64ビット化、Unicode対応、クロスプラットフォーム、マルチデバイス‥

はじめてのFireDAC

BDEからの移行

BDEから移行するための選択肢

BDE以外で、従来のDelphi/C++Builderから
標準で利用可能なデータベースフレームワークの種類

- dbExpress
- dbGO(ADO/OLEDB)
- IBExpress (InterBase専用)

データアクセスアーキテクチャの違い
・カーソルの違い（サーバーサイドカーソル／クライアントサイドカーソル）
・フィルタの指定有無／条件

BDE DataSetとの非互換が多く、移植性が悪い
・dbExpressのTSQLTable(Query)とBDEコンポーネントのメソッドやプロパティが異なる

実行パフォーマンスが出ない
・BDEでは1秒で済んでいたものが、dbExpressやdbGOで21秒もかかった

BDEから移行する「新たな」選択肢

BDE以外で、Delphi/C++Builderから利用可能なデータベース・フレームワークの新定番

- dbExpress
- dbGO(ADO/OLEDB)
- IBExpress (InterBase専用)

はじめてのFireDAC

FireDACについて

FireDACとは?

- **AnyDACの「Embacadero版」**

- **ユニバーサルデータアクセスが可能な、一体化されたコンポーネント群**
  Oracle/MS-SQL Server/DB2/MySQL/PostgreSQL/InterBase/Firebird/SQLite/SQL Anywhere/Advantage DB/
  Access/Informix/DataSnapなどに高速なネイティブアクセスが可能
  対応ドライバは、次のURLを参照
  http://docs.embarcadero.com/products/rad_studio/firedac/frames.html?frmname=topic&frmfile=Databases.html

- **DelphiおよびC++Builder向け**
  RAD Studio/ Delphi/ C++Builder XE4、XE5
  詳しくは、次のURLを参照
  http://www.embarcadero.com/jp/products/rad-studio/firedac-faq

- **マルチデバイス**
  Win32, Win64, Mac OS X, iOS, Android

- **ハイパフォーマンス**
  BDEと同等、それ以上のデータアクセススピード

- **共通化されたAPI**
  マクロ機能を使ってSQLの方言や微妙な違いを吸収
  修正可能なデータマッピング機能によるデータ型の統一化

FireDACを使用するメリットは・・

BDEとの互換性が高い
・データアクセスアーキテクチャが類似している
・従来のBDE DataSetとの互換性を備えたDataSetクラスがある
・CachedUpdatesモードも利用可能

ハイパフォーマンス
・BDEと同等、それ以上のデータアクセススピード

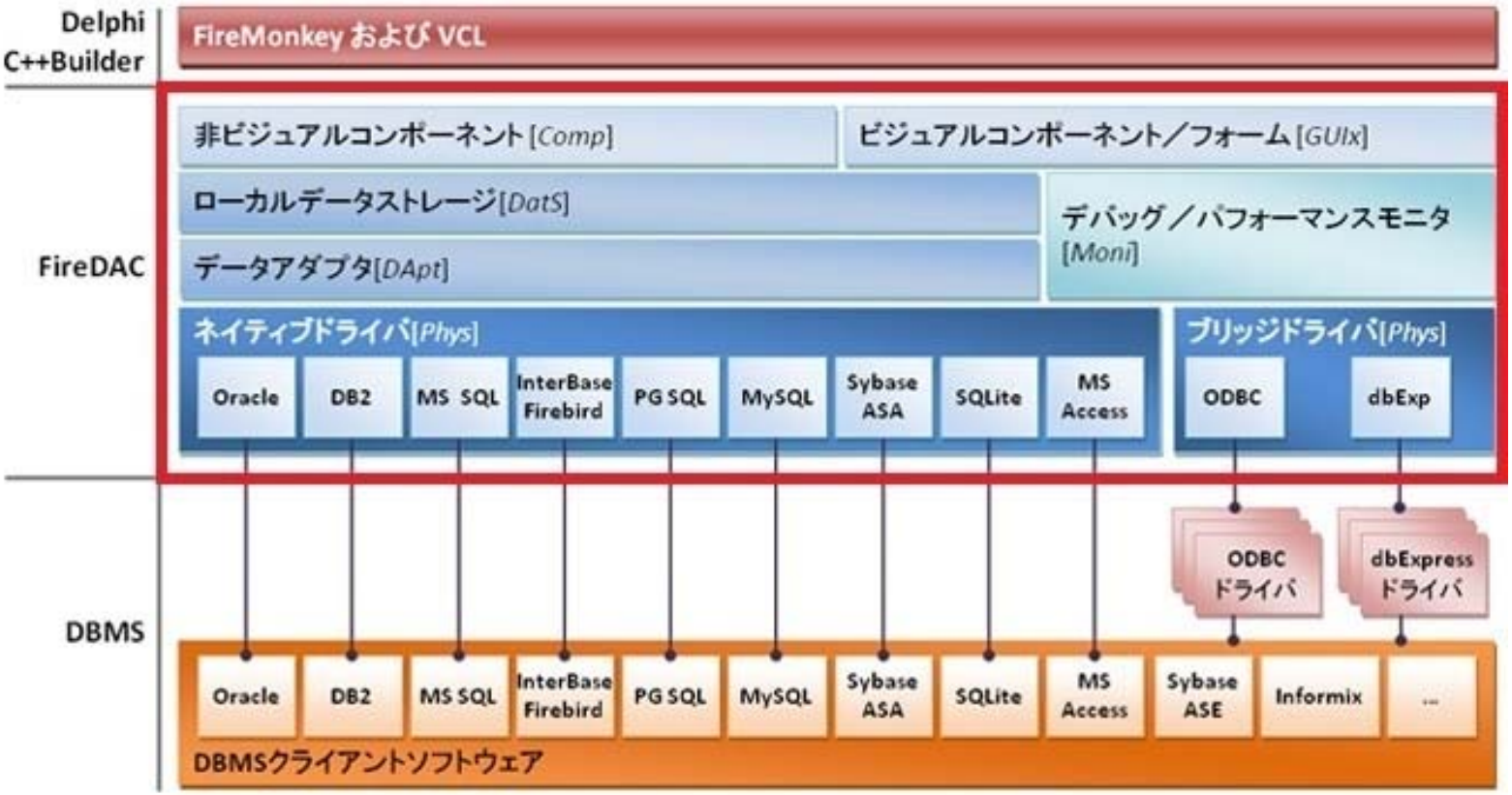
配布が簡単
・専用のデータベースドライバやインストーラが不要

Professionalエディションでもリモート接続が可能

FireDACアプリケーションの構造

- FireDACアプリケーションは、3層構造
- 赤枠がFireDACコンポーネント本体
- 開発には32ビットのクライアントライブラリが必須

注意！

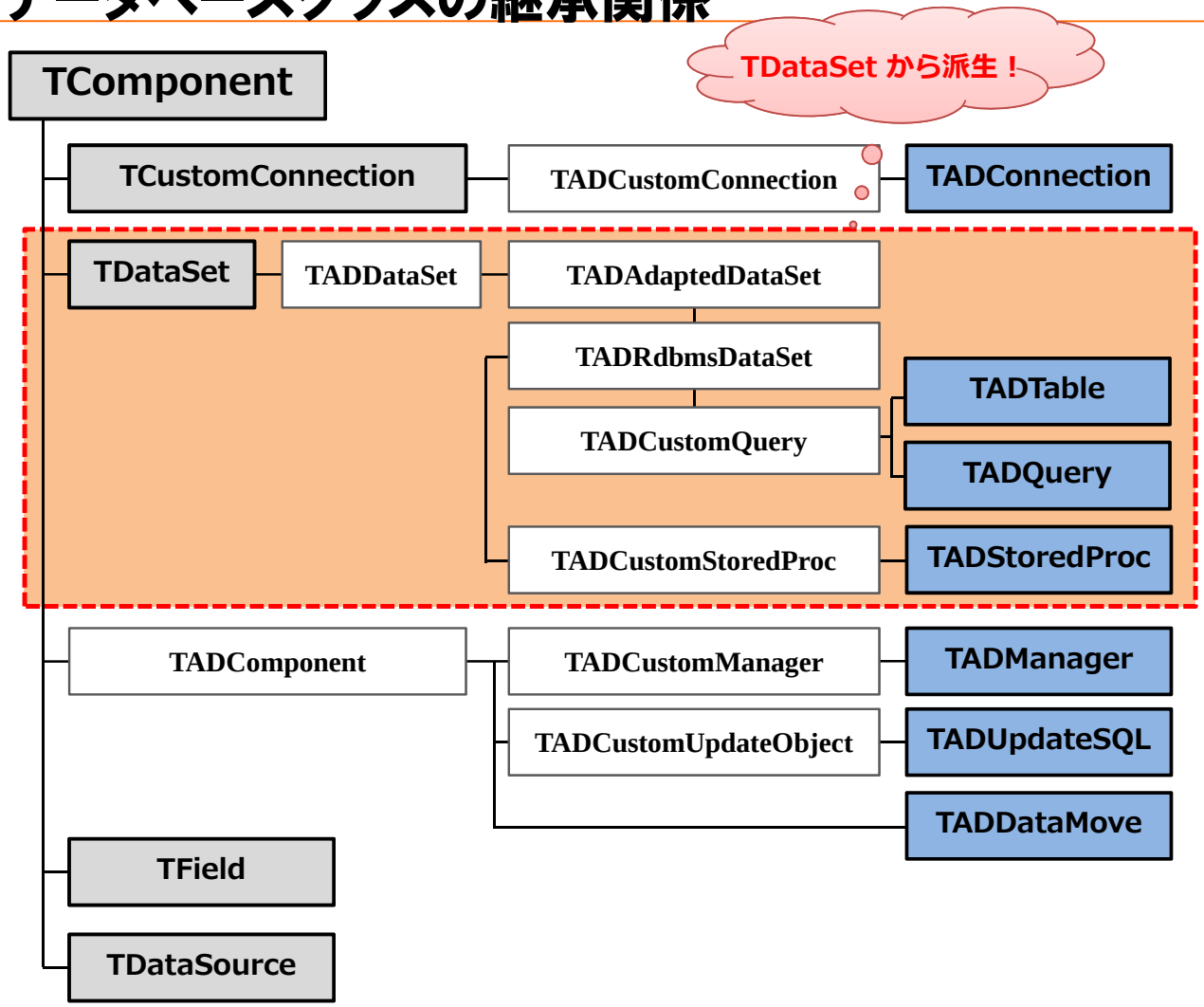


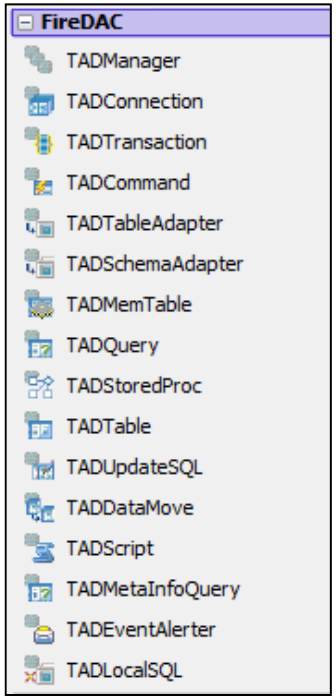
BDEとFireDAC 主要コンポーネントの比較

BDE	FireDAC	解説
TDatabase	**TADConnection**	○
TSession	**TADManager**	
TTable	**TADTable**	○
TQuery	**TADQuery**	○
TStoredProc	**TADStoredProc**	○
TUpdateSQL	**TADUpdateSQL**	○
TBatchMove	**TADDataMove**	
–	**TADPhysXXXXDriverLink**	○
–	**TADGUIxWaitCursor**	○

※ BDEのコンポーネント名に**AnyDAC**の「**AD**」を付加したネーミング
※ BDEの対応は、次のサイトを参照
http://docs.embarcadero.com/products/rad_studio/firedac/frames.html?frmname=topic&frmfile=index.html
[参照先] **Contents > Migrating BDE applications > BDE name counterparts**

データベースクラスの継承関係





XXX ：FireDAC

XXX ：Delphi標準

はじめてのFireDAC

TADConnectionコンポーネント

プロパティの比較

BDE (TDatabase)	FireDAC (TADConnection)	備　考
AliasName	**ConnectionDefName**	エリアスを指定
DatabaseName	**ConnectionName**	アプリケーション内で参照するときのエリアス名
Connect	**Connect**	Open/Closeメソッドと等価
Params	**Params**	接続パラメータを保持
DriverName	**DriverName**	DriverIDを指定 *1
KeepConnection	**KeepConnection**	True:接続を保持（既定値） ResourceOptionsに移動
–	**LoginDialog**	TADGUIxLoginDialogコンポーネントを指定
LoginPrompt	**LoginPrompt**	True:ログインダイアログボックスの表示
SessionName	**–**	TADManagerコンポーネントで指定
TransIsolation	**–**	アイソレーション(分離)レベルを指定 TxOptions.Isolationプロパティで指定

***1 DriverIDは、次のURLで確認**

http://docs.embarcadero.com/products/rad_studio/firedac/frames.html?frmname=topic&frmfile=Databases.html

接続の設定・・Oracleの場合（1/2）

Connection Editorで設定

オブジェクトインスペクタの下にある**「Connectin Editor ...」**をクリックする

Driver ID:
リストから接続するサーバーのIDを指定
選択すると既定値が、パラメータ欄に表示される

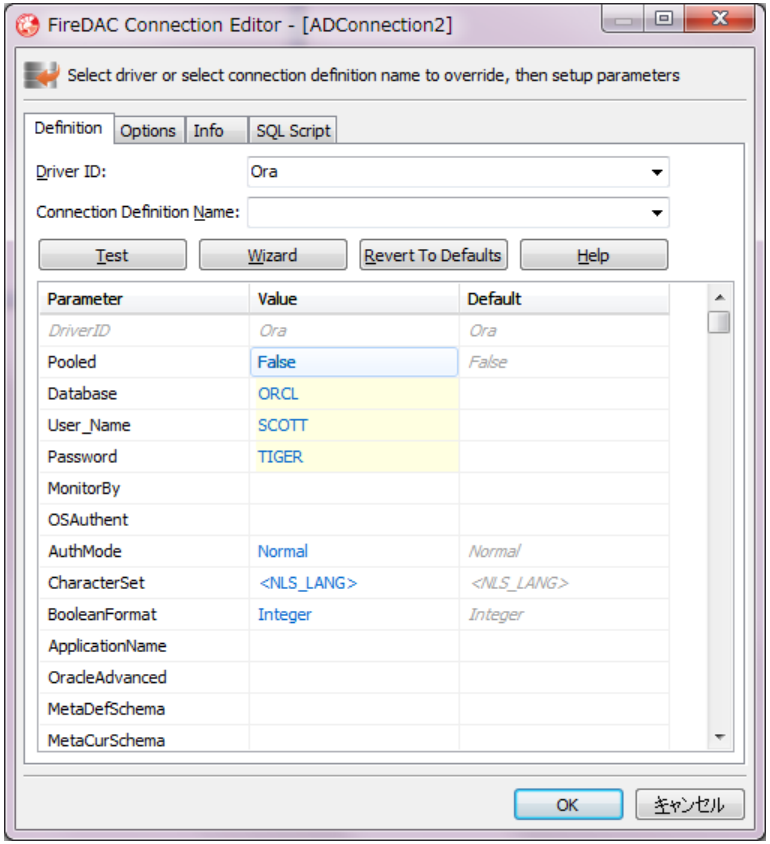
Connection Definition Name:（必須ではない）
エリアス名をリストから選択
エリアスは「FireDAC Explorer」で設定

Database:
「TNSNames.ora」に設定されている「ネットサービス名」を入力

User_Name/Password:
ユーザ名とパスワードを入力

AuthMode:
接続モードをリストから選択

※他のパラメータは、適宜設定して下さい
※入力後、**「Test」**ボタンを押して接続を確認



接続の設定・・Oracleの場合（2/2）

コードで設定

実行時に接続に必要なパラメータをコードで指定する**（赤字の部分）**

```
try
  with ADConnection1 do
  begin
    // 接続パラメータのセット
    Params.Add('DriverID=Ora');
    Params.Add('Database=ORCL');
    Params.Add('User_Name=SCOTT');
    Params.Add('Password=TIGER');
    Params.Add('AuthMode=Normal');
    // 接続
    Open;
  end;
except
  MessageDlg('データベースと接続できません！ ', mtError, [mbOK], 0);
end;
```

※セキュリティには十分な配慮をして下さい

トランザクション管理

Delphiでは「暗黙トランザクション」と「明示トランザクション」の2種類がサポートされている

ここでは明示トランザクションを説明します

プロパティ／メソッド	説　明
InTransaction	True: トランザクションが進行中
StartTransaction	指定されたアイソレーションレベル*1でトランザクションを開始
Commit	変更内容をすべてコミットし、トランザクションを終了
Rollback	変更内容を取り消し、トランザクションを終了

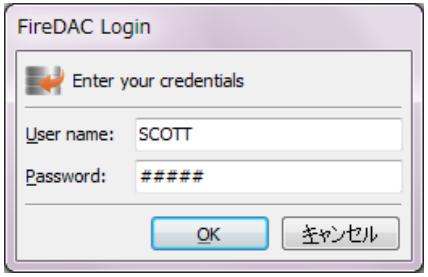
```
begin
  if not ADConnection1.InTransaction then
    ADConnection1.StartTransaction;
  try
    … トランザクション中のDML実行

    // トランザクションをコミット
    ADConnection1.Commit;
  except
    // トランザクションをロールバック
    ADConnection1.Rollback;
    raise;
  end;
end;
```

*1 アイソレーションレベルは**TxOptions.Isolationプロパティ**で指定します

ログイン制御

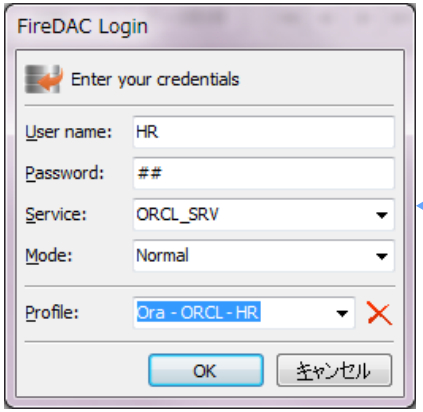
標準のログインダイアログボックス

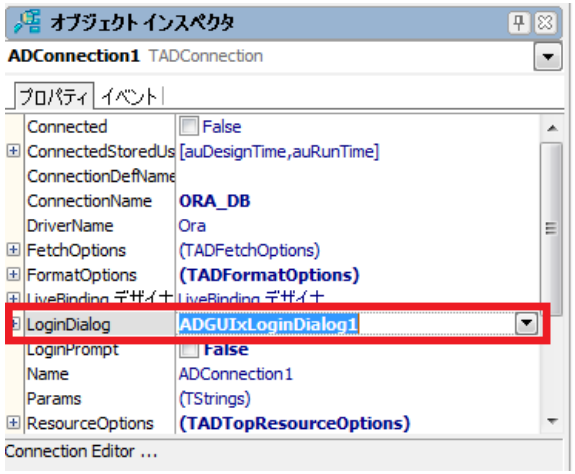
LoginPrompt が True (既定値)だと標準のログインダイアログボックスが表示される



ログインダイアログボックスのカスタマイズ

Formに**TADGUIxLoginDialogコンポーネント**を配置し、TADConnectionコンポーネントのLoginDialogプロパティを指定すると、ログインダイアログボックスを変更することができる

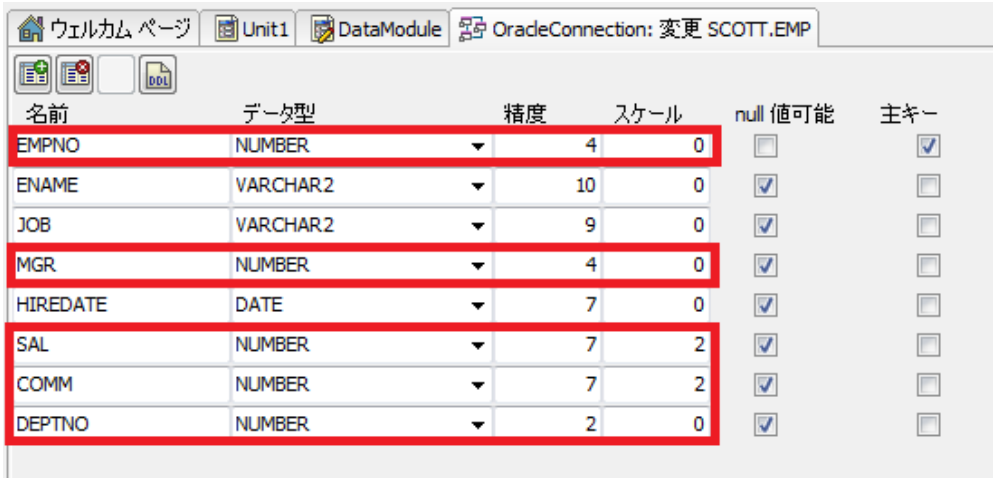




データ型のマッピングを変更する（1/3）

・FireDACのマッピングルールは dbExpress に類似している
・FireDACではマッピングルールの変更が可能（dbExpress では不可）
・マッピングルールは TADMapRuleクラス で定義・管理される
・マッピングは接続（セッション）単位、データセット単位で設定することができる
・マッピングルールを変更することにより、より効率的な型変換が可能となる

[EMPテーブル]



型変換

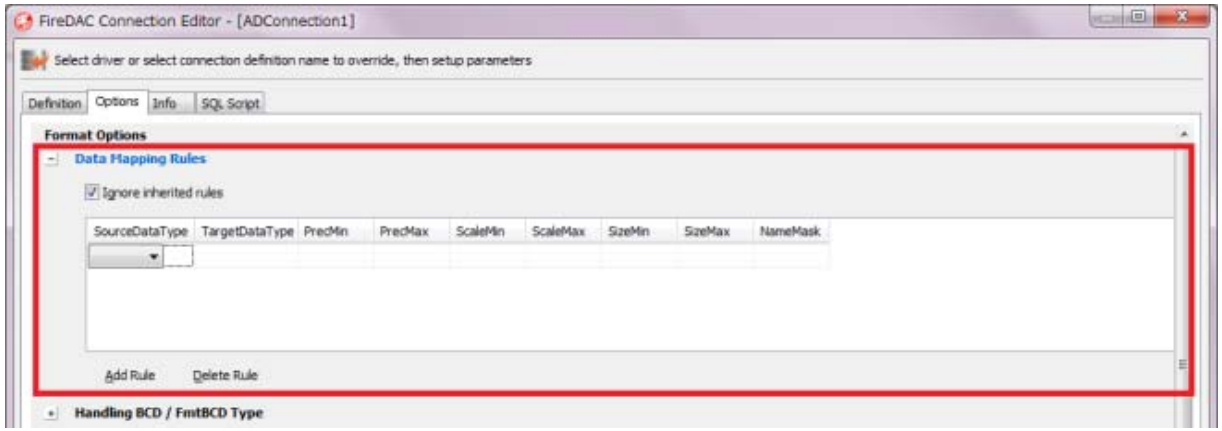
[TFieldコンポーネント]

TBCDField
TStringField
TStringField
TBCDField
TDateTimeField
TBCDField
TBCDField
TBCDField

※OracleのNUMBER型が TBCDField (2進化10進数項目) にマッピングされてしまう

データ型のマッピングを変更する（2/3）

・Connection Editorで設定する
・Options タブの「Data Mapping Rules」を開く
・「Ignore inherited rules」にチェックを付け、定義の設定を有効にする



プロパティ	説　明
SourceDataType	ソース・データの型（タイプ）を指定
TargetDataType	変換後のデータの型（タイプ）を指定
PrecMin/PrecMax	数字精度（precision）の範囲を指定
ScaleMin/ScaleMax	数字スケール（小数点以下の桁数）の範囲を指定
SizeMin/SizeMax	文字列長の範囲を指定
NameMask	カラム名を指定

データ型のマッピングを変更する（3/3）

マッピングルールは、コードでも設定できる

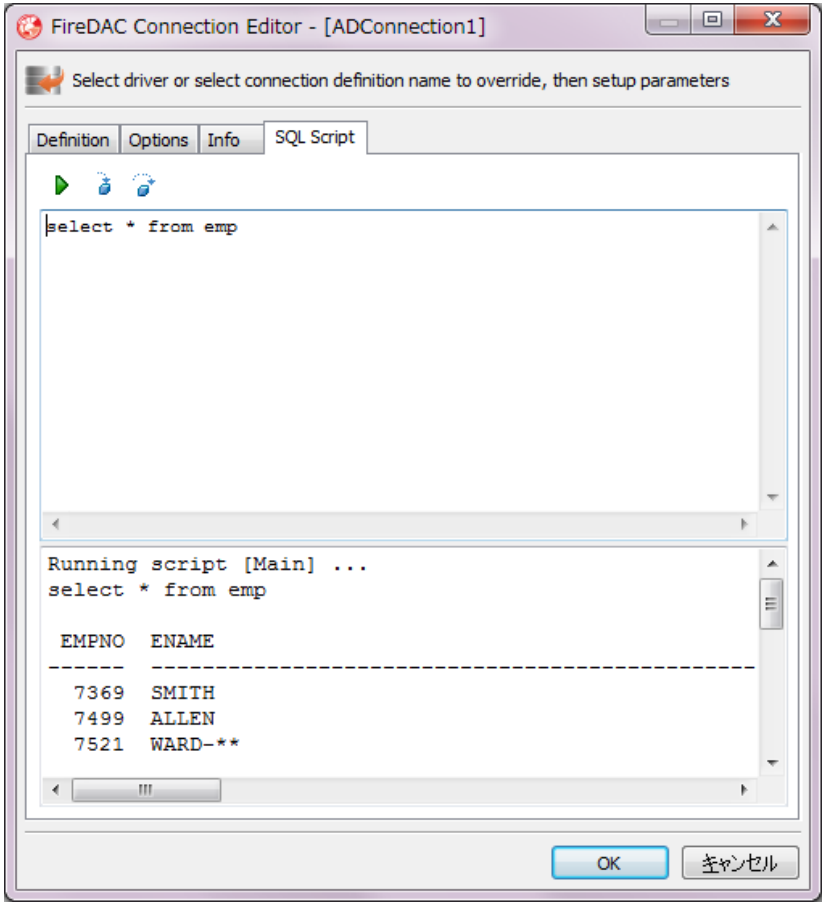
```
with ADConnection1.FormatOptions do
begin
  //自分自身のマッピング・ルールを使う
  OwnMapRules := True;
  //ルールの定義 ①
  with MapRules.Add do
  begin
    PrecMin := 10;
    PrecMax := 10;
    ScaleMin := 0;
    ScaleMax := 0;
    SourceDataType := dtFmtBCD;
    TargetDataType := dtInt32;
  end;
  //ルールの定義 ②
  with MapRules.Add do
  begin
    NameMask := '%ID';
    TargetDataType := dtInt32;
  end;
end;
```

［ルールの解説］

①数字フィールドの精度が10桁の数値項目は、BCD(2進化10進数)から「32ビットの整数型」で処理する
②フィールド名に「ID」の文字が含まれている数値項目は、「32ビットの整数型」で処理する

Connection Editor でSQLを実行する

SQL Script タブで、一時的なSQL文の実行が可能



SQL文の実行

ExecSQLメソッドを使用するとTADConnectionコンポーネントでもSQL文の実行が可能

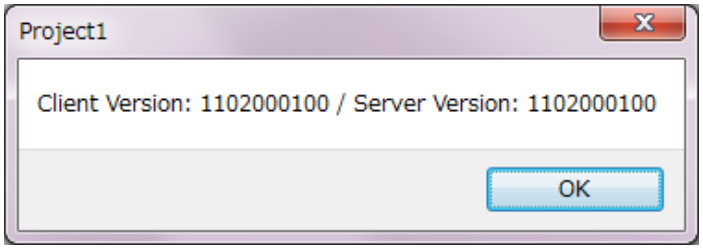
```
const
  strSQL = 'CREATE TABLE TABLE1(NO NUMBER(4) NOT NULL, NAME VARCHAR2(10), KINGAKU
NUMBER(7,2))';
begin
  try
    // SQL文の実行
    ADConnection1.ExecSQL(strSQL);
  except
    MessageDlg('SQLの実行に失敗しました！ ', mtError, [mbOK], 0);
  end;
end;
```

DBMSのサーバ／クライアントのバージョン番号を求める

TADConnectionコンポーネントの ConnectionMetaDataIntf プロパティ で取得する

```
function TDataModule1.GetDBMSversion: String;
const
  StrVer = 'Client Version: %.10d / Server Version: %.10d';
var
  oMetaIntf: IADPhysConnectionMetadata;
begin
  oMetaIntf := ADConnection1.ConnectionMetaDataIntf;
  try
    Result := Format(StrVer, [oMetaIntf.ClientVersion, oMetaIntf.ServerVersion]);
  finally
    oMetaIntf:= nil;
  end;
end;
```

[実行結果]



はじめてのFireDAC

TADPhysXXXXDriverLinkコンポーネント

TADPhysXXXXDriverLinkとは?

- FireDACで新たに追加されたコンポーネント
- 接続するデータベースに対応したドライバ(TADPhysXXXXDriverLink)が必要
- ドライバとなっているが中身はPascalで書かれたインターフェイス
- ドライバは実行モジュールに組み込まれる
- FireDACアプリケーションでは、1つは配置することが必要

※配置しないで実行すると次のエラーが表示される

データベースとドライバの対応表

データベース	FireDAC
Oracle	TADPhysOracleDriverLink
DB2	TADPhysDB2DriverLink
MS SQL Server	TADPhysMSSQLDriverLink
InterBase／FireBird	TADPhysIBDriverLink
PostgreSQL	TADPhysPgDriverLink
MySQL	TADPhysMySQLDriverLink
Sybase ASE	TADPhysODBCDriverLink
SQLite	TADPhysSQLiteDriverLink
MS Access	TADPhysMSAccessDriverLink
ODBCブリッジドライバ	TADPhysODBCDriverLink
dbExpressブリッジドライバ	TADPhysDBXDriverLink
DataSnap	TADPhysDataSnapDriverLink

※ 対応ドライバは、次のURLを参照
http://docs.embarcadero.com/products/rad_studio/firedac/frames.html?frmname=topic&frmfile=Databases.html

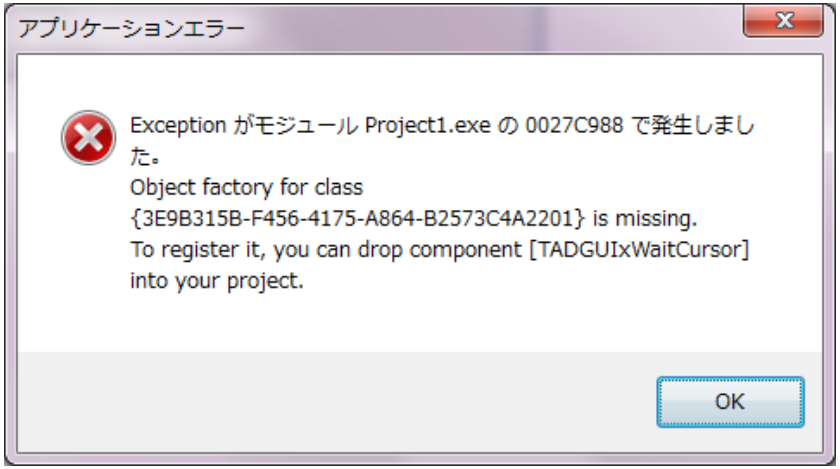
はじめてのFireDAC

TADGUIxWaitCursorコンポーネント

TADGUIxWaitCursorとは?

- FireDACで新たに追加されたコンポーネント
- IADGUIxWaitCursorインタフェースを実現するためのコンポーネント
- Forms、FMX、および Consol プログラムで使用することができる
- FireDACアプリケーションでは、1つは配置することが必要

※配置しないで実行すると次のエラーが表示される



BDEアプリケーションからの移植

- BDEで時間の掛かる処理は、次の様なカーソルの制御が一般的

```
  Screen.Cursor := crSQLWait;
  try
    ......
  finally
    Screen.Cursor := crDefault;
  end;
```

- FireDACでは次の様なコードに置き換えることもできる（必須ではない！）

```
uses
  uADStanFactory, uADGUIxIntf;
  ......
var
  oWait: IADGUIxWaitCursor;
begin
  ......
  ADCreateInterface(IADGUIxWaitCursor, oWait);
  oWait.StartWait;
  try
    ......
  finally
    oWait.StopWait;
  end;
end;
```

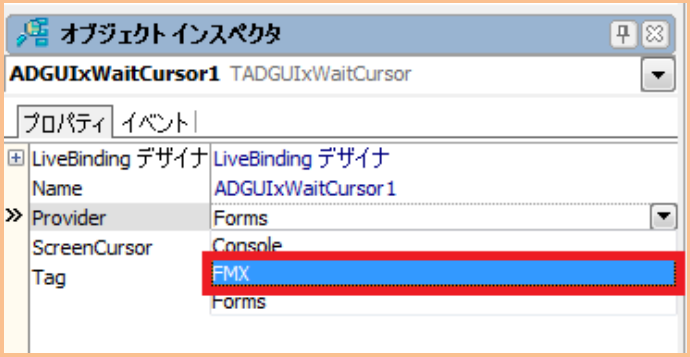
FireMonkeyアプリケーションでの注意点

・**TADGUIxWaitCursorのProviderプロパティが「Form」に設定されているとエラーが発生する**

↓

・**Providerプロパティ**は次のように設定

- FireMonkeyアプリケーション：**FMX**
- VCLアプリケーション　　：**Form**
- コンソールアプリケーション　：**Console**



はじめてのFireDAC

TADTableコンポーネント

プロパティの比較

BDE (TTable)	FireDAC (TADTable)	備　考
Active	**Active**	Open/Closeメソッドと等価
AutoCalcFields	**AutoCalcFields**	True:OnCalcFields イベントの発生。 計算項目の計算を意図的に止めたいときはFalseにする
Bof	**Bof**	True:カーソルがデータセットの先頭にある
CachedUpdates	**CachedUpdates**	True:キャッシュアップデートが使用可能
DatabaseName	**ConnectionName**	アプリケーション内で参照するときのエリアス名
Eof	**Eof**	True:カーソルがデータセットの最後にある
Exclusive	**Exclusive**	True:排他的にテーブルをアクセスする
IndexFieldNames	**IndexFieldNames**	テーブルのインデックスフィールドを表示
IndexName	**IndexName**	テーブルの二次インデックスをリストから選択
MasterFields	**MasterFields**	リンクする項目をセミコロンで区切って指定
MasterSource	**MasterSource**	テーブル同士をリンクする場合、リンクするテーブルのTDataSourcを指定
TableName	**TableName**	テーブル名を指定
ReadOnly	**ReadOnly**	True:読み取り専用 UpdateOptionsオプションに移動
UpdateMode	**UpdateMode**	更新レコードの検索方法をリストから選択 UpdateOptionsオプションに移動

カーソルを移動する

データセット上でレコード間の移動を行う場合、次のメソッドを使う

メソッド	機　能
First	先頭行へ移動する
Last	最終行へ移動する
Next	次の行へ移動する
Prior	１つ前の行へ移動する
MoveBy(n)	n行分、行を移動する 移動は、現在の行を基準にして相対的に行われる

[MoveByの使用例]

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ADTable1.MoveBy(-10); // 行を10行戻す
  ADTable1.MoveBy(20);  // 行を20行進める
end;
```

データの更新

データセットのレコードを操作するメソッドには次のものがある

メソッド	機　能
Edit	データセットを編集状態にする
Append	最終行にレコードを追加して挿入状態にする
Insert	現在の行にレコードを追加して挿入状態にする
Post	データセットに変更を登録しデータセットを参照状態にする
Cancel	変更を取り消しデータセットを参照状態にする
ClearFields	現在のレコードのすべての項目の内容を消去する
Delete	現在の行を削除して、参照状態にする

[行の挿入]

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FldName: array[0..2] of TField;
begin
  with ADTable1 do
  begin
    // 各フィールドのTFieldを取得
    FldName[0] := FindField('NO');      // コード
    FldName[1] := FindField('NAME');    // 商品名
    FldName[2] := FindField('KINGAKU'); // 金　額
    // 空の行を挿入
    Insert;
    // 各フィールドに値をセット
    FldName[0].AsInteger := 001;
    FldName[1].AsString  := 'Delphi XE4';
    FldName[2].AsCurrency := 50000;
    // データの登録
    Post;
  end;
end;
```

[次の様に書き換えることもできる]

```
with ADTable1 do
begin
  // 各フィールドの値を指定して行を挿入する
  InsertRecord([001, 'Delphi XE4', 50000]);
end;
```

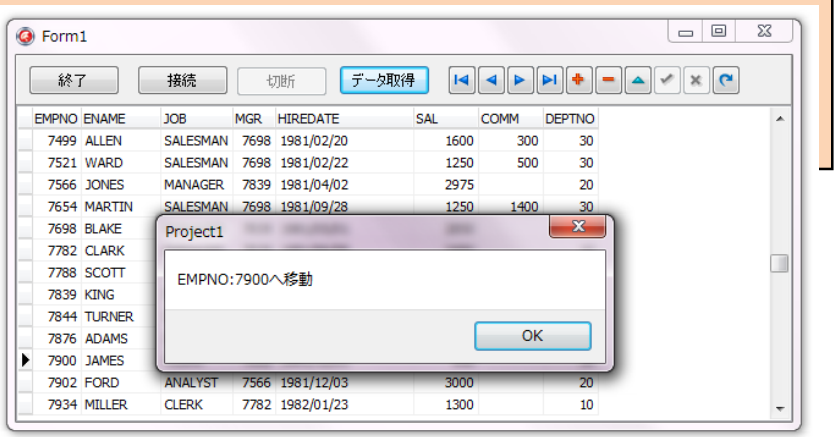
レコードの検索

レコードの検索には、次のメソッドを使う

メソッド	機　能
GotoKey	キーバッファに指定された値の位置へテーブルカーソルを移動する
GotoNearest	キーバッファに指定された値以上の値を持つレコードにテーブルカーソルを移動する
FindKey	引数に配列で指定された値に一致するレコードを検索し、テーブルカーソルを移動する
FindNearest	引数で指定された検索値以上の値を持つレコードにテーブルカーソルを移動する

[FindKeyの使用例]

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (ADTable1.FindKey([7900]) = True) then
    ShowMessage('EMPNO:7900へ移動')
  else
    ShowMessage('EMPNO:7900のレコードが見つかりません。');
end;
```



Bookmark を利用する

Bookmarkと呼ばれる目印を付けて、データセットの特定の位置に移動することができる

メソッド	機　能
BookmarkValid	ブックマークが正しくレコードを指しているかどうかを判定する
CompareBookmark	２つのブックマークが等しいかどうかを判定する
GetBookmark	カレントレコードにブックマークを付け、それを返す
GotoBookmark	ブックマークで指定したレコードへカーソルを移動する
FreeBookmark	不要になったブックマークを解放する

```
procedure TDataModule1.DoSomething(const Tbl: TADQuery);
var
  Bookmark: TBookmark;
begin
  Bookmark := Tbl.GetBookmark;
  Tbl.DisableControls;
  try
    Tbl.First;
    while not Tbl.Eof do
    begin
      // 処理記述
      Tbl.Next;
    end;
  finally
    Tbl.GotoBookmark(Bookmark);
    Tbl.EnableControls;
    Tbl.FreeBookmark(Bookmark);
  end;
end;
```

データコントロールへの描画抑制

カーソル移動を頻繁に行うと再描画処理が頻繁に発生し、レスポンスが低下する
それを抑制するには、次のメソッドを使用

メソッド	機　能
DisableControls	TDataSourceコンポーネントとのリンクを解除する
EnableControls	TDataSourceコンポーネントと、再度リンクする

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with ADQuery1 do
  begin
    DisableControls;
    try
      First;
      while not Eof do
      begin
        … データセット操作
        Next;
      end;
    finally
      EnableControls;
    end;
  end;
end;
```

行の選択

行の選択を行うためにはデータセットのフィルタ制御を使用する

プロパティ／イベント	機　能
Filter	フィルタ文字列を指定する
Filtered	True: フィルタ有効
OnFilterRecord	フィルタ有効時(True)に、データセット内でアクティブなレコードが変わるたびに発生

[プロパティで設定]

```
with ADTable1 do
begin
  Filter := 'JOB=''SALESMAN''';
  Filtered := True;
  Open;
end;
```

[イベントで設定]

```
ADTable1.Filtered := True;
      :
procedure TForm1.ADTable1FilterRecord(DataSet: TDataSet; var Accept: Boolean);
begin
  Accept := DataSet['JOB'] = 'SALESMAN';
end;
```

項目の参照

データセットは、列情報を項目コンポーネント(TField)として管理される
参照は、項目コンポーネントの**Fields**プロパティや**FieldByName**メソッドを使用する

プロパティ／メソッド	機　能
Fields	指定された項目順のTFieldコンポーネントを返す
FieldByName	指定された項目のTFieldコンポーネントを返す

[Fieldsプロパティを使用]

```
with ADTable1 do
begin
  S1 := Fields[0].AsString;
  S2 := Fields[1].AsString;
end;
```

[FieldByNameメソッドで設定]

```
with ADTable1 do
begin
  S1 := FieldByName('EMPNO').AsString;
  S2 := FieldByName(‘ENAME’).AsString;
end;
```

はじめてのFireDAC

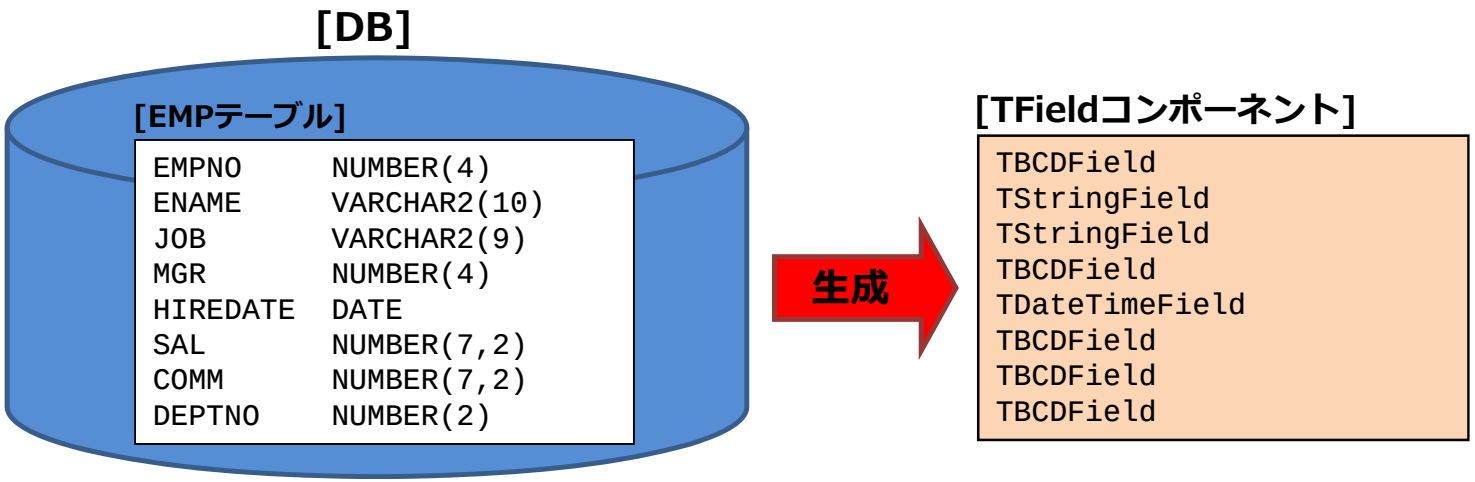
TFieldコンポーネント

TFieldコンポーネントとは？

・レコード内のフィールドにアクセスするためのコンポーネント

・TADTable/TADQuery がアクティブになったとき自動で生成される（動的生成）

・設計時に作成することも可能（静的生成）・・・「永続化フィールド」ともいう



主要プロパティ／メソッド

FireDACでは、プロパティ・イベント・メソッドが大幅に追加されている

プロパティ	説　明
Alignment	データの表示方法をリストから選択
DisplayFormat	数値フィールドの表示形式を指定 (#,##0)
DisplayLabel	TDBGridの列ヘッダーに表示するテキストを指定
DisplayWidth	コントロールが使用する文字数を指定
EditFormat *1	数値フィールドの入力形式を指定 (#,##0)
FieldName	物理的な列の名前が表示されている
Index	TFieldコンポーネントの順番を指定
MaxValue	BCD *2 型項目の最大値の上限を指定
MinValue	BCD 型項目の最小値の下限を指定
Visible	TDBGridに表示されるかどうかを指定
OnGetText	フィールドのDisplayTextまたはTextプロパティが参照されたときに発生
OnSetText	Textプロパティに値がセットされたときに発生
OnValidate	データがレコード バッファに書き込まれる直前に発生

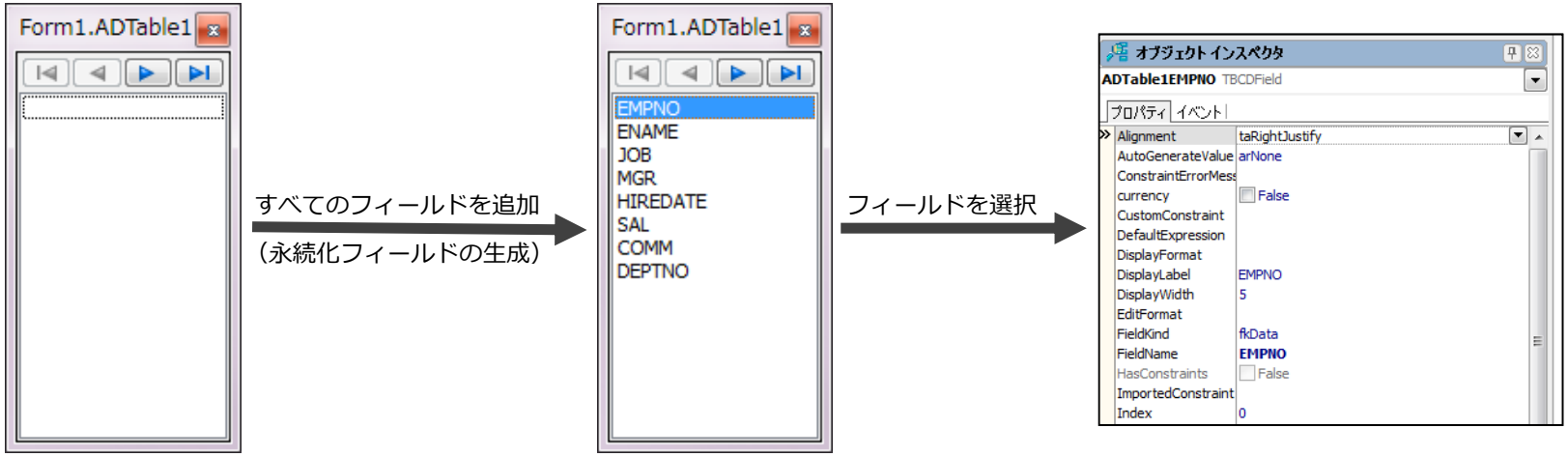
*1 **EditFormat**：設定されていない場合は、DisplayFormatが使用される

TFieldのプロパティ／イベントを設定する

- **TADTable/TADQuery コンポーネント上で右クリックし「フィールド エディタ…」を選択**
- **表示されたフィールドエディタ内で右クリックし「すべてのフィールドを追加」をクリック**
- **フィールドを選択し、「オブジェクト インスペクタ」でプロパティ／イベントを設定する**

※フィールドエディタに表示されていない項目は、アプリケーションで使用できない

[フィールド エディタ]



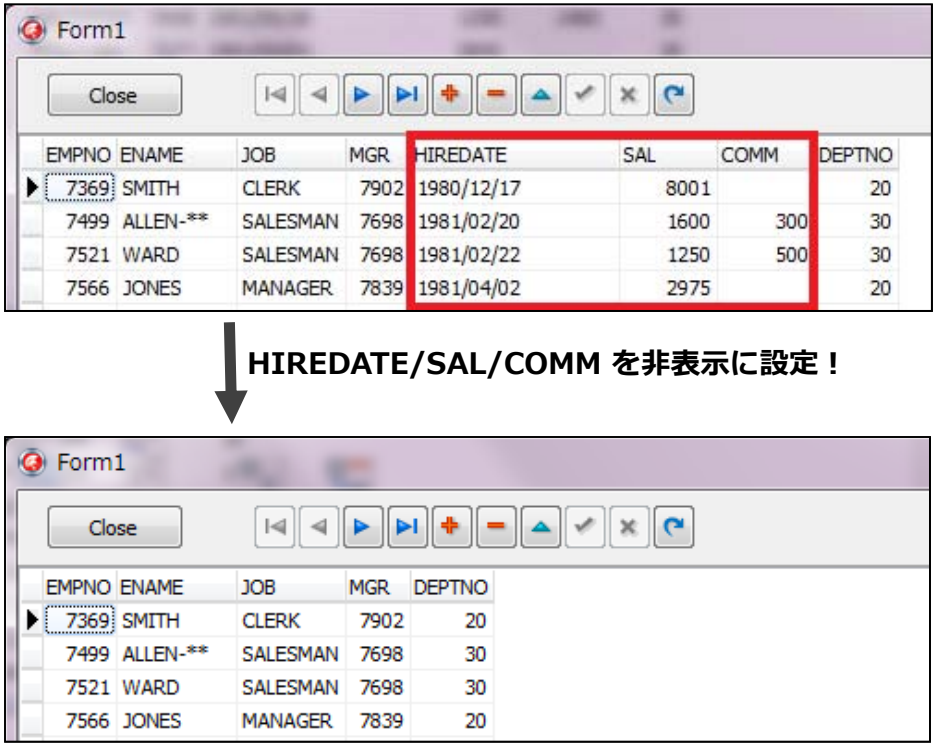
特定フィールドを非表示にする

TDBGridで特定フィールドを非表示にする場合は、次のいずれかの方法を使います

・フィールドエディタから該当項目を削除する（永続化フィールドから削除する）

or

・フィールドのVisibleプロパティをFalseにする



OnGetText イベント

フィールドの値によって表示内容を変更する

・請求区分が「0」なら「未」、以外なら「済」と表示するようにコードを記述

```
procedure TForm1.ADQuery1SEIKYUKBNGetText(Sender: TField; var Text: string;
  DisplayText: Boolean);
begin
  if Sender.AsInteger = 0 then
    Text := '未' else Text := '済';
end;
```

その他プロパティの設定を変更する

- **項目の表示順を変更する**
  フィールドエディタでドラッグ＆ドロップで順番を変更する

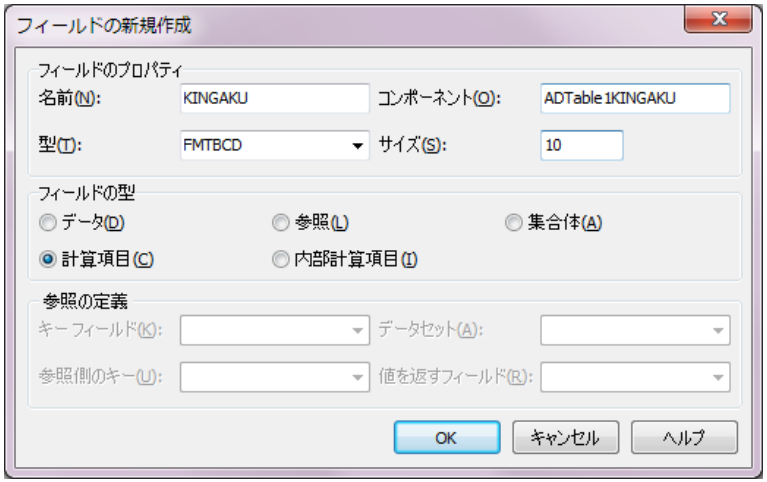
- **TDBGridの見出しを変更したい**
  項目コンポーネントの**DisplayLabel**で変更する

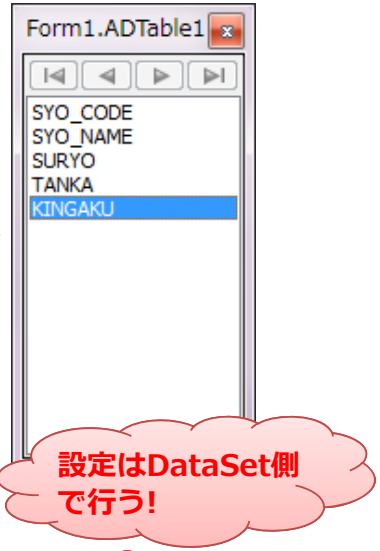
- **TDBGridの列の表示幅を変更したい**
  項目コンポーネントの**DisplayWidth**で変更する

計算項目を追加する

・フィールドエディタで右クリックし「フィールド新規作成...」を選択

・各項目を設定し［OK］を押下





・計算式はOnCalcFieldsイベントで定義

```
procedure TForm1.ADTable1CalcFields(DataSet: TDataSet);
begin
  ADTable1KINGAKU.Value := ADTable1SURYO.Value + ADTable1TANKA.Value;
end;
```

LOB (Large OBject) データの登録／保存 (1/2)

BLOB*1 データのテーブルへの登録とファイルへの保存方法は次の様になる

・BLOB格納用のテーブルを生成する

```
CREATE TABLE LOBTABLE
(
    BlobName     VARCHAR2(20),
    BlobDateTime DATE,
    BlobEntity   BLOB
)
```

・BLOBデータをテーブルへ登録する

BLOBフィールドを**TBlobFieldでキャストし、LoadFromFile メソッドで登録する**

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with ADQuery1 do
  begin
    Open;
    Append;
    Fields[0].AsString := 'Data1';
    Fields[1].AsDateTime := Now;
    TBlobField(ADQuery1.Fields[2]).LoadFromFile('C:¥Delphi.exe¥Test.bmp');
    Post;
  end;
end;
```

*1 **BLOB**(Binary Large Object): バイナリデータを格納する場合のデータ型。
キャラクタ型を格納する場合のデータ型を**CLOB**(Character Large Object) という

LOB (Large OBject) データの登録／保存 (2/2)

・**BLOBデータをファイルへ保存する**

BLOBフィールドを**TBlobFieldでキャストし、 SaveToFile メソッドでファイルへ出力する**

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  with ADQuery2 do
  begin
    Open;
    TBlobField(ADQuery2.FieldByName('BlobEntity')).SaveToFile('C:¥Delphi.exe¥Test.bmp');
  end;
end;
```

[実行結果]

はじめてのFireDAC

TADQueryコンポーネント

プロパティの比較

BDE (TQuery)	FireDAC (TADQuery)	備　考
Active	**Active**	Open/Closeメソッドと等価
AutoCalcFields	**AutoCalcFields**	True:OnCalcFields イベントの発生。 計算項目の計算を意図的に止めたいときはFalseにする
Bof	**Bof**	True:カーソルがデータセットの先頭にある
CachedUpdates	**CachedUpdates**	True:キャッシュアップデートが使用可能
DatabaseName	**ConnectionName**	アプリケーション内で参照するときのエリアス名
Eof	**Eof**	True:カーソルがデータセットの最後にある
IndexFieldNames	**IndexFieldNames**	テーブルのインデックスフィールドを表示
IndexName	**IndexName**	テーブルの二次インデックスをリストから選択
MasterFields	**MasterFields**	リンクする項目をセミコロンで区切って指定
MasterSource	**MasterSource**	テーブル同士をリンクする場合、リンクするテーブルの TDataSourcを指定
Params	**Params**	動的SQL文のパラメータを指定
ReadOnly	**ReadOnly**	True:読み取り専用 UpdateOptionsオプションに移動
SQL	**SQL**	SQL文を設定
UpdateMode	**UpdateMode**	更新レコードの検索方法をリストから選択 UpdateOptionsオプションに移動

SQL文を設定する

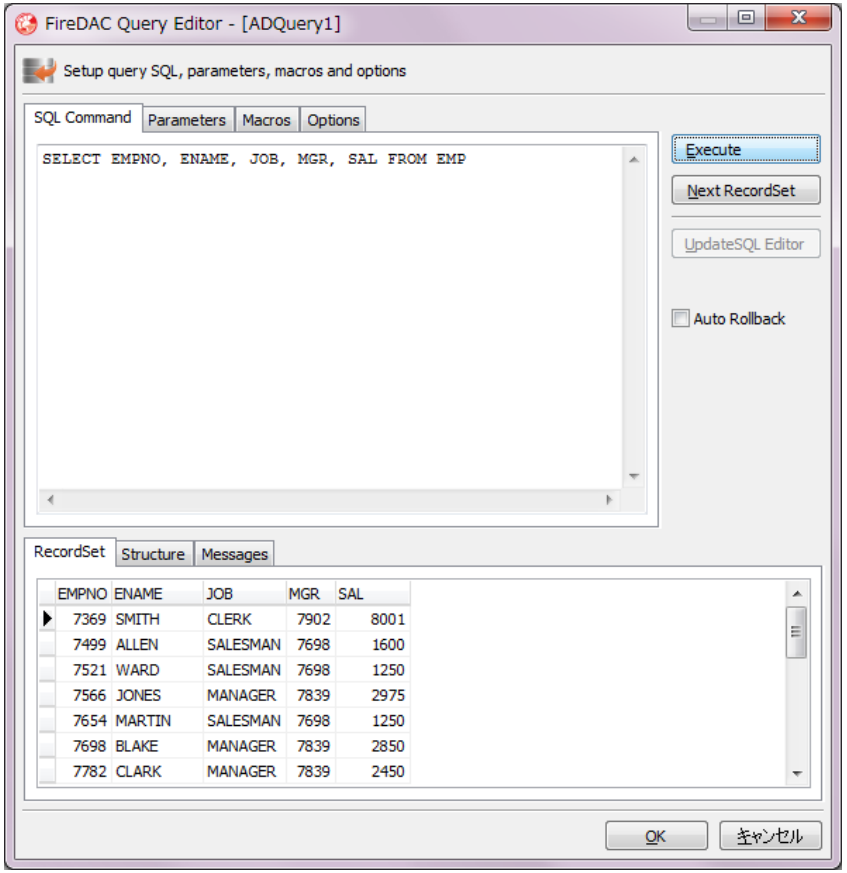
・付属のツールを使用すると、SQLの作成と確認を行うことができる
・TADQuery コンポーネント上で右クリックし「Query Editor ...」を選択

SQL Commandタブ:
実行するSQL文を指定
「Execute」ボタンを押すとSQLを実行

Parametersタブ:
SQL文にSQLパラメータが指定されている場合、
パラメータを指定

Macros タブ:
SQL文にマクロパラメータが指定されている場合、
パラメータを指定

Optionsタブ:
Format Optionsを指定



静的問い合わせ

SQL文を発行するメソッドには２種類ある

結果セットを返す場合と、返さない場合とでは使用するメソッドが異なる

メソッド	機　能
Open	SELECT文を使ってデータセットに問い合わせを行い、その結果を受け取る
ExecSQL	INSERT/UPDATE/DELETE文などのデータセットに対して操作を行い、 その結果を返さないSQL文を実行する

```
// すべてのユーザー一覧を返す(Oracle)
procedure TForm1.GetAllUsers(UserList: Tstrings);
begin
  UserList.Clear;
  with ADQuery1 do
  begin
    try
      Close;
      Open(‘select distinct username from sys.all_users order by username');
      while not Eof do
      begin
        UserList.Add(FieldByName('USERNAME').AsString);
        Next;
      end;
    except
      on E: EDatabaseError do
        MessageDlg(E.Message, mtError, [mbOK],0);
    end;
  end;
end;
```

動的問い合わせ

実行時にパラメータを指定してSQLを実行する

SQL文にSQLパラメータ*1 を埋め込むことで、変数のように利用できる

プロパティ／メソッド	機　能
Params[n]	変数配列n番目に対して値をセットします
ParamByName(Value)	引数Valueに指定された変数名に対して値をセットします

```
procedure TForm1.Button1Click(Sender: TObject);
const
  strSQL = 'SELECT * FROM EMP WHERE (EMPNO >= :startcode) AND (EMPNO < :endcode)';
begin
  with ADQuery1 do
  begin
    // SQL文をセット
    SQL.Clear;
    SQL.Add(strSQL);
    // 変数に値をセットする
    ParamByName('startcode').AsInteger := 7500;
    ParamByName('endcode').AsInteger := 7900;
    // リソースの割り当てと最適化を実行する
    Prepare;
    // SQLの実行
    Open;
  end;
end;
```

[次の様に書き換えることもできる]

```
Params[0].AsInteger := 7500;
Params[1].AsInteger := 7900;
```

*1 **SQLパラメータ**
バインド変数・ホスト変数とも呼ばれる。パラメータ名の前にコロン(**:**)を付けて記述します

大量のDMLを高速に実行する

- FireDACのDML配列機能を使う
- 複数のDML(INSERT/UPDATE/DELETE)文をパラメータ付きで一度に実行できる
- 高速実行の実現
  - 通常のExecSQL → 5.5秒
  - DML配列 → **0.03秒**

[INSERTの例]

メモリサイズに注意!

```
procedure TDataModule1.ExecuteDML;
var
  i: Integer;
begin
  // 実行するDML(INSERT)SQLをセット
  ADQuery1.SQL.Text:= 'INSERT INTO DML_TEST (TINT, TSTRING) VALUES(:f1, :f2)';
  // 配列の大きさをセット
  ADQuery1.Params.ArraySize := 1000;
  // 配列に値をセット
  for i:=0 to ADQuery1.Params.ArraySize do
  begin
    ADQuery1.ParamByName('f1').Values[i-1] := i;
    ADQuery1.ParamByName('f2').Values[i-1] := 'Str' + IntToStr(i);
  end;
  // SQLの実行
  ADQuery1.Execute(ADQuery1.Params.ArraySize, 0);
end;
```

複数SQLのデータセットを利用する

・TADMemTableを使う
・以下のコードは、２つのSQLで得られた結果を複数のTADMemTableに割り付けてセットする

```
procedure TDataModule1.ExecuteSQL;
begin
  ADMemTable1.Close;
  ADMemTable2.Close;
  // SQL実行
  ADQuery1.SQL.Clear;
  ADQuery1.SQL.Add('SELECT * FROM EMP');
  ADQuery1.Open;
  // すべてのレコードを取得
  ADQuery1.FetchAll;
  // ADQuery1のデータをADMemTable1に代入
  ADMemTable1.Data := ADQuery1.Data;
  // SQL実行
  ADQuery1.SQL.Clear;
  ADQuery1.SQL.Add('SELECT * FROM DEPT');
  ADQuery1.Open;
  // すべてのレコードを取得
  ADQuery1.FetchAll;
  // ADQuery1のデータをADMemTable2に代入
  ADMemTable2.Data := ADQuery1.Data;
  DataSource1.DataSet := ADMemTable1;
  ADQuery1.Close;
end;
```

データベース例外を捕捉する（1/2）

・BDEでは「EDBEngineError」を捕捉
・FireDACでは「EADDBEngineException」を捕捉
・BDEからの移行の場合は、コードの修正が必要

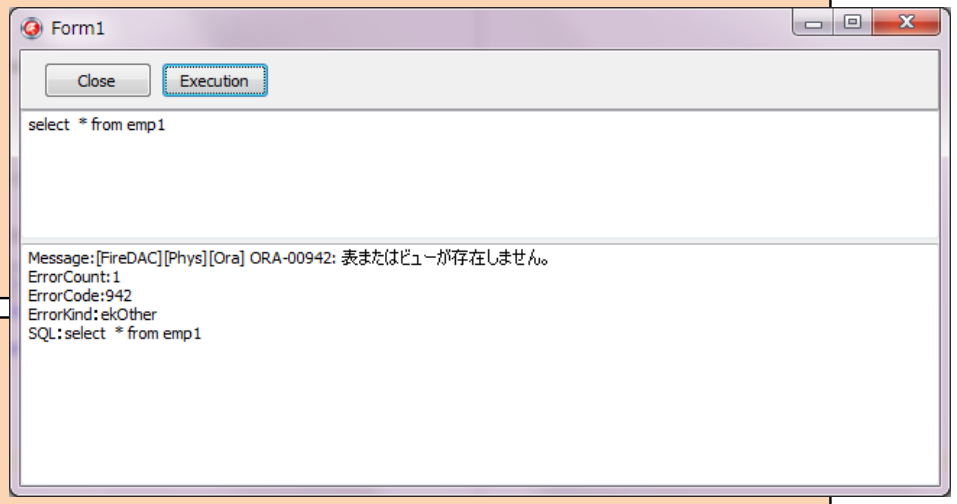
プロパティ	説　明
ErrorCode	DBMSのエラーコードを返す
ErrorCount	エラーの総数を示す
Errors[n]	AnyDAC/DBMSのエラー・警告、またはメッセージのリストを返す
Kind	FireDACで定義されているエラーの種類を返す
Params	実行に失敗したパラメータの値を返す
SQL	実行に失敗したSQL文を返す

データベース例外を捕捉する（2/2）

[例外の捕捉例]

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  try
    ADQuery1.SQL.Clear;
    ADQuery1.SQL.Add(Me
    ADQuery1.Open();
  except
    on E: EADDBEngineException do
    ShowError(E);
  end;
end;
```

エラーをキャッチする




```
procedure TForm1.ShowError(E: EADDBEngineException);
var
  i: Integer;
begin
  Memo2.Lines.Clear;
  Memo2.Lines.Add('Message:‘ + E.Message);
  Memo2.Lines.Add('ErrorCount:‘ + IntToStr(E.ErrorCount));
  for i := 0 to E.ErrorCount - 1 do
  begin
    Memo2.Lines.Add('ErrorCode:‘ + IntToStr(E.Errors[i].ErrorCode));
    case E.Kind of
      ekUserPwdInvalid: Memo2.Lines.Add(‘ErrorKind: ekUserPwdInvalid');
      ekUserPwdExpired: Memo2.Lines.Add(‘ErrorKind: ekUserPwdExpired');
      ekServerGone: Memo2.Lines.Add(‘ErrorKind: ekServerGone');
    else
      Memo2.Lines.Add(‘ErrorKind: ekOther');
    end;
    Memo2.Lines.Add('SQL:‘ + E.SQL);
  end;
end;
```

マクロを使う

TADQueryのSQL文にはマクロパラメータを書くことができる
マクロ機能を使うことにより、RDBMSによって異なるSQL方言を抽象化すること可能

例えば「**文字列を大文字や小文字に変換する**」関数もRDBMSで以下の様に異なる

RDBMS	大文字に変換	小文字に変換
MS Access	UCASE(文字列)	LCASE(文字列)
MS SQLServer	UPPER(文字列)	LOWER(文字列)
Oracle	UPPER(文字列)	LOWER(文字列)
MySQL	UCASE(文字列)	LCASE(文字列)
PostgreSQL	UPPER(文字列)	LOWER(文字列)

マクロを使うと次の様にSQL文を書くことができる（Oracle／MySQL）

```
{IF Oracle} SELECT UPPER('ABCabc') FROM DUAL {fi}
{IF MYSQL} SELECT UCASE('ABCabc') FROM DUAL {fi}
```

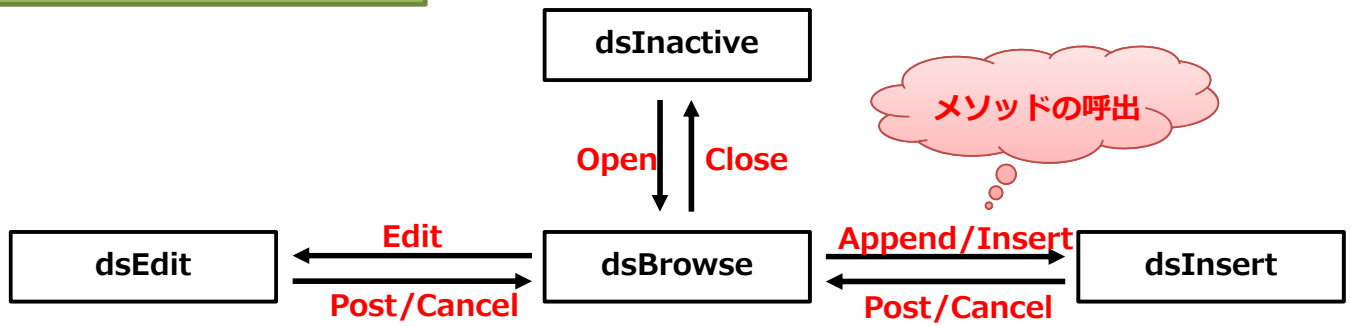
※ 詳しくは **FireDAC** のヘルプを参照

データセットの状態を調べる

TADQueryのStateプロパティを調べるとデータセットの状態を知ることができる

Stateプロパティ	状　態	説　明
dsInactive	非アクティブ状態	データセットが開かれていない
dsBrowse	参照状態	データセットは参照モードにある
dsEdit	編集状態	データセットは修正モードにある
dsInsert	挿入状態	データセットは挿入モードにある

Stateプロパティの変化



結果セットの変更

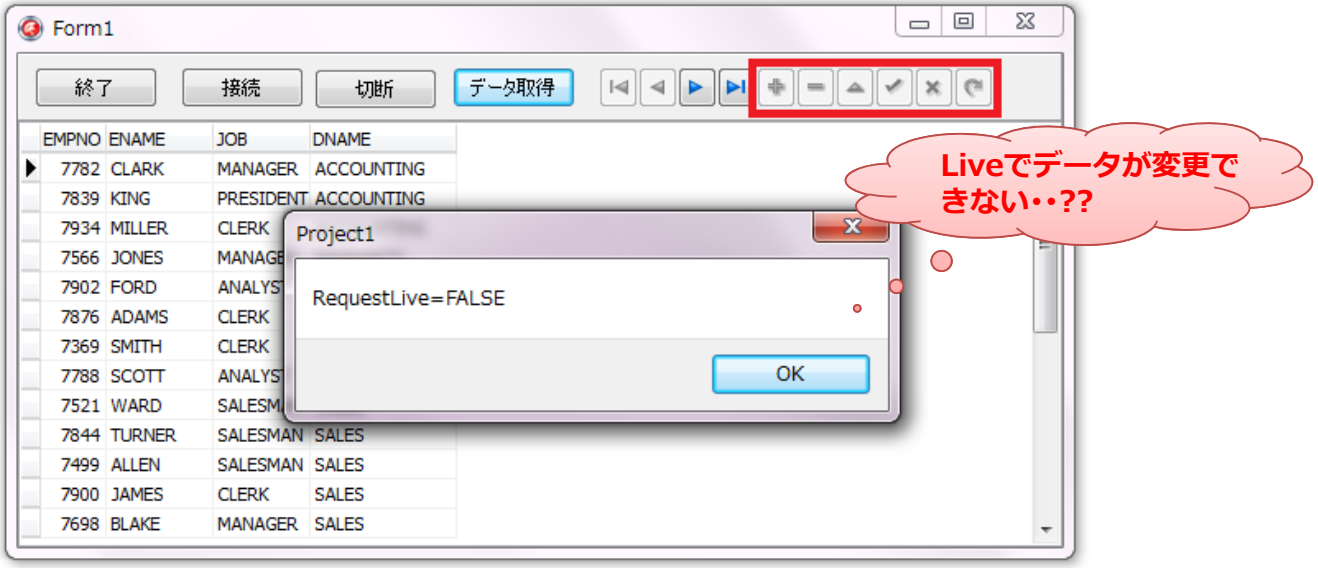
結果セットを変更可能とするか、読み取り専用とするかはRequestLiveプロパティで決定される

しかし、複数テーブルの結合や副問い合わせ、集合関数(SUM/AVG)が使われている場合は変更できない

プロパティ	機　能
RequestLive	True:データベースに更新可能な問い合わせの結果セットを要求する

次のSQL文を実行 (表結合)

```
SELECT a.EMPNO, a.ENAME, a.JOB, b.DNAME
FROM EMP a, DEPT b
WHERE (a.DEPTNO = b.DEPTNO)
```



はじめてのFireDAC

TADUpdateSQLコンポーネント

キャッシュアップデート

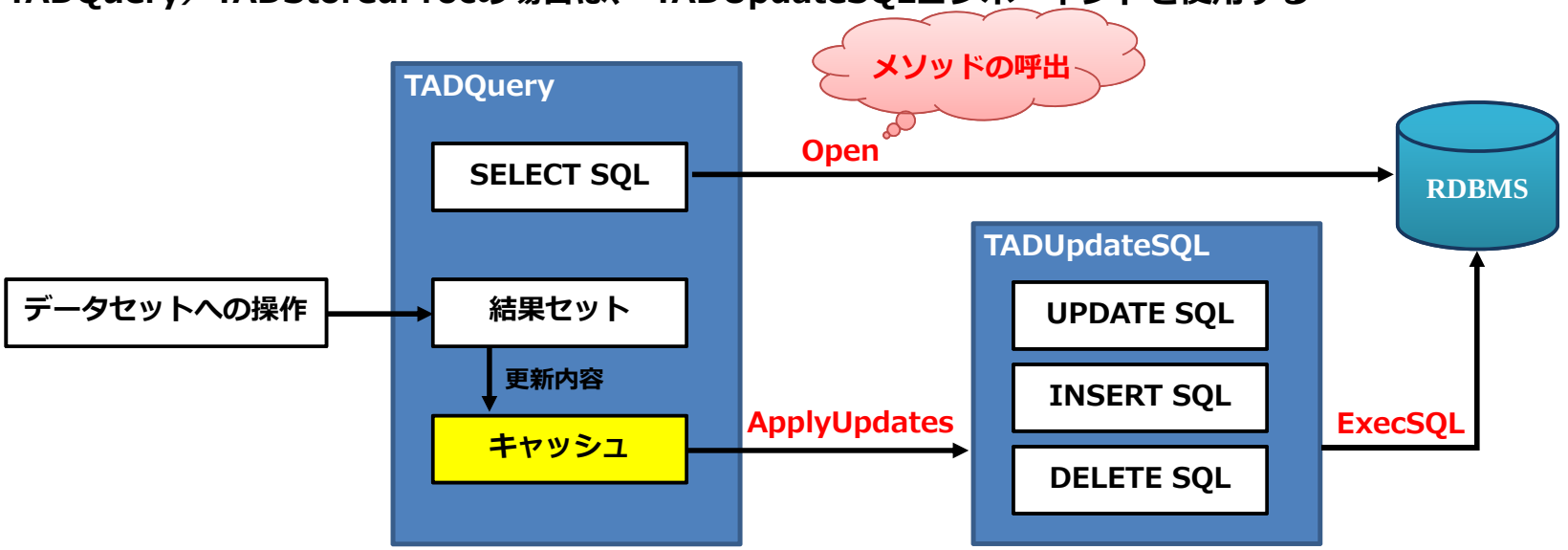
・キャッシュアップデートは、データベースへの遅延更新を実現する技術で、
ネットワーク負荷の軽減と、読み取り専用データセットの更新をサポート

・データセットの更新情報はすべてローカルキャッシュに保存され、
メソッドの呼び出しで追加／更新／削除・・をデータベースに反映される

TADQueryでも利用可

・TADTableの場合は、CachedUpdatesプロパティをTrueにすることでキャッシュアップデートを
利用することができる

・TADQuery／TADStoredProcの場合は、 TADUpdateSQLコンポーネントを使用する



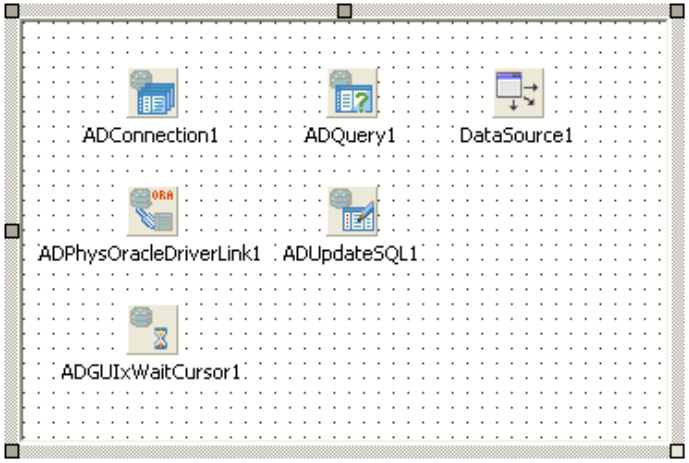
プロパティの比較

BDE (TUpdateSQL)	FireDAC (TADUpdateSQL)	備　考
–	**Connection**	TADConnectionコンポーネントを指定
–	**ConnectionName**	アプリケーション内で参照するときのエリアス名
DataSet	**DataSet** *1	TDataSetオブジェクト参照
DeleteSQL	**DeleteSQL**	結果セットで削除されたレコードをデータセットから削除するSQL文を設定
InsertSQL	**InsertSQL**	結果セットに挿入された新規レコードをデータセットにセットするSQL文を設定
ModifySQL	**ModifySQL**	結果セットで値が変更されたレコードをデータセットにセットするSQL文を設定

*1 **DataSetプロパティ：**public属性でないため設計時には使用できない

キャッシュアップデートの設定（1/2）

Form(または、データモジュール)にTADQuery(TADTable)、TADUpdateSQLを配置



TADQueryコンポーネントのプロパティを設定

TADQuery	設定内容
CachedUpdates	Trueに設定。
UpdateObject	TADUpdateSQLコンポーネントを指定

キャッシュアップデートの設定（2/2）

更新用のSQL文を「Update SQL Editor」で設定する

・TADUpdateSQLコンポーネントを選択し、オブジェクトインスペクタの下にある**「Update SQL Editor ...」**をクリックする

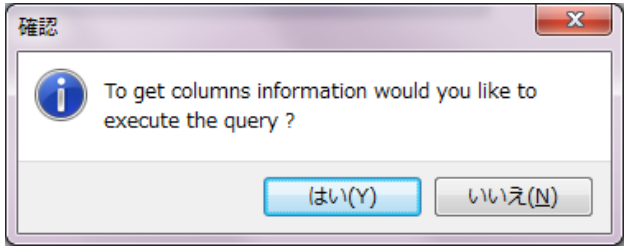
※SQL文は「更新する表単位」に必要
※項目を変更した場合は、**「Generate SQL」**ボタンを必ず押して下さい

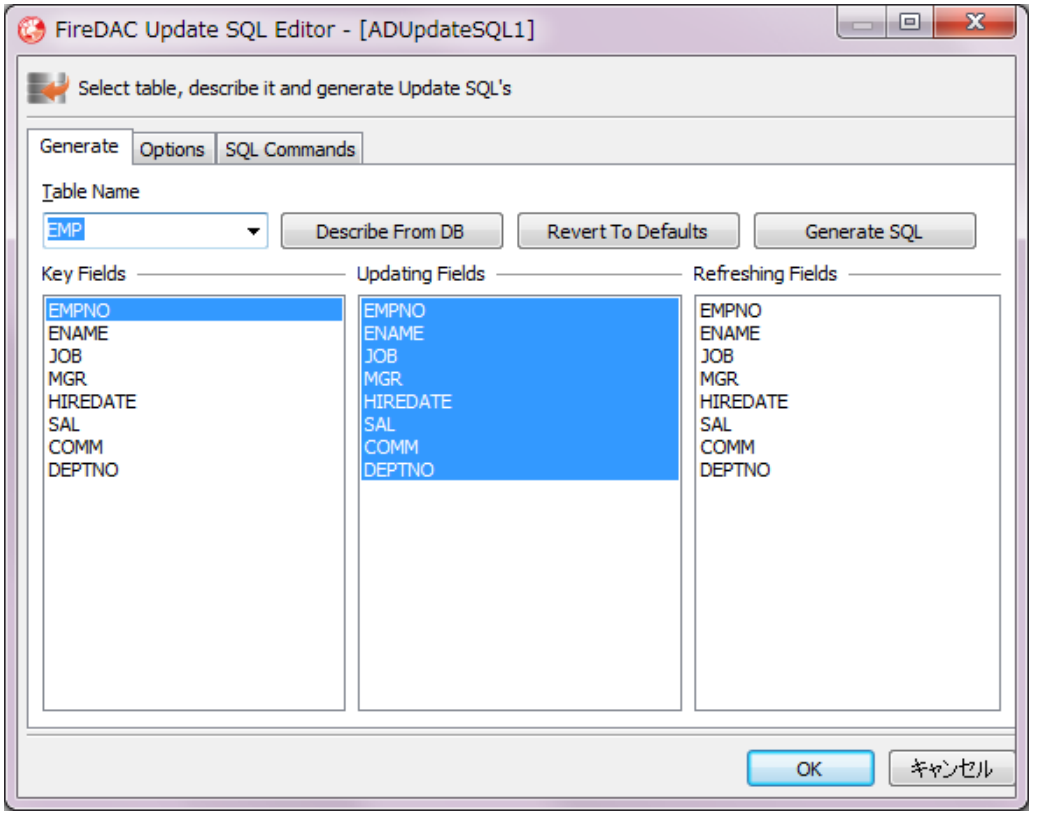
Generateタブ:
更新対象のテーブル名、キー項目、
更新する項目を設定

Optionタブ:
生成するSQL文の種類、形式等を設定

SQL Commandsタブ:
生成されたSQL文を確認することができる

※データセットが閉じていると次の警告が出る





更新情報のサーバーへの書き込み

サーバーへの書き込みは、ApplyUpdatesメソッドを呼び出す

メソッド	機　能
ApplyUpdates	キャッシュの更新情報をデータベースサーバーに書き込む
CancelUpdates	キャッシュに保留している更新をすべてクリアする
CommitUpdates	ローカルキャッシュの内容をクリアする

```
procedure TDataModule1.ExecutUpdate;
begin
  // トランザクションの開始
  if not ADConnection1.InTransaction then ADConnection1.StartTransaction;
  try
    // 変更をデータベースサーバーに書き込む
    if ADQuery1.UpdatesPending then ADQuery1.ApplyUpdates;
    // トランザクションのコミット
    ADConnection1.Commit;
  except
    // トランザクションのロールバック
    ADConnection1.Rollback;
    raise;
  end;
  // キャッシュアップデートバッファのクリア
  ADQuery1.CommitUpdates;
  ADQuery1.Refresh;
end;
```

OnUpdateRecord イベント

・キャッシュアップデートによる更新は、自動的に更新SQLを生成し実行される
※OnUpdateRecordイベントが割り当てられていると更新は行われません！

・逆な言い方をすれば、複雑な更新制御を施すことができる
・下のコードは、OnUpdateRecordイベントが割り当てられていない場合と等価

```
procedure TDataModule1.ADQuery1UpdateRecord(ASender: TDataSet;
  ARequest: TADUpdateRequest; var AAction: TADErrorAction;
  AOptions: TADUpdateRowOptions);
begin
  ADUpdateSQL1.Apply(ARequest, AAction, AOptions);
  AAction := eaApplied;
end;
```

クラス	内　容
TADUpdateRequest	arInsert, arUpdate, arDelete,　arLock, arUnlock, arFetchRow, arUpdateHBlobs, arDeleteAll, arFetchGenerators
TADErrorAction	eaFail, eaSkip, eaRetry, eaApplied, eaDefault,　eaExitSuccess, eaExitFailure
TADUpdateRowOption	uoCancelUnlock, uoImmediateUpd, uoDeferredLock, uoOneMomLock, uoNoSrvRecord, uoDeferredGenGet

OnUpdateError イベント

・次のコードでは、アプリケーション例外が実行される（サイレント例外）
・exceptの例外処理は実行されない

```
procedure TDataModule1.ExecutUpdate;
begin
  try
    ADQuery1.ApplyUpdates;
    ADQuery1.CommitUpdates;
  except
    // 例外処理を記述
    raise;
  end;
end;
```

例外が発生しない‥!?

・キャッシュアップデートの例外処理は OnUpdateErrorイベントに記述する

```
procedure TDataModule1.ADQuery1UpdateError(ASender: TDataSet;
  AException: EADException; ARow: TADDatSRow; ARequest: TADUpdateRequest;
  var AAction: TADErrorAction);
begin
  ADConnection1.Rollback;
end;
```

はじめてのFireDAC

TADStoredProcコンポーネント

プロパティの比較

FireDACでは、プロパティ・イベント・メソッドが大幅に追加されている

BDE (TStoredProc)	FireDAC (TADStoredProc)	備　考
–	**Connection**	TADConnectionコンポーネントを指定
DatabaseName	**ConnectionName**	アプリケーション内で参照するときのエリアス名
ExecProc	**ExecProc**	ストアドプロシージャの実行
–	**PackageName**	パッケージ名を指定
Params	**Params**	パラメータの名前、値、およびデータ型を設定
ParamBindMode	**ParamBindMode** *1	パラメータをストアドプロシージャの変数名か、変数順番かを指定
Prepare	**Prepare**	ストアドプロシージャをメモリーにロード **※記述位置に注意**
SchemaName	**SchemaName**	スキーマ名を指定
StoredProcName	**StoredProcName**	ストアドプロシージャ名を指定

[*1 ParamBindModeプロパティ]

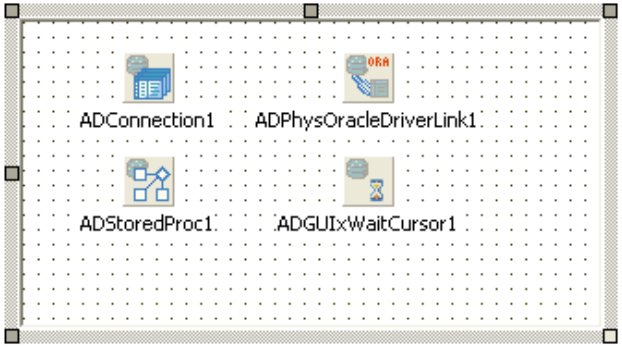
値	順序
pbByName	パラメータをストアドプロシージャの変数名で指定（デフォルト値）
pbByNumber	パラメータをストアドプロシージャの変数順番で指定

PROCEDUREの実行（1/4）

・以下のSQL文を実行し、Oracleサーバーにプロシジャ「ProcA」を登録する

```
CREATE OR REPLACE PROCEDURE ProcA (
  i1 IN NUMBER,
  i2 IN NUMBER,
  o1 OUT NUMBER) AS
BEGIN
  o1 := i1 + i2;
  RETURN;
END;
```

・データモジュールにコンポーネントを配置



PROCEDUREの実行（2/4）

・ConnectionName(またはConnection)プロパティを設定し、
TADConnectionコンポーネントと結び付ける

・データモジュールに、次のコードを記述（呼ばれる側）

```
procedure TDataModule1.ExcProcA(const i1, i2: Integer; var o1: Integer);
begin
  with ADStoredProc1 do
  begin
    // プロシジャ名の設定
    StoredProcName := 'ProcA';
    // プロシジャの読み込み
    Prepare;
    // パラメータの設定
    ParamByName('i1').AsInteger := i1;
    ParamByName('i2').AsInteger := i2;
    // プロシジャの実行
    ExecProc;
    // 実行結果の受け取り
    o1 := ParamByName('o1').AsInteger;
  end;
end;
```

[次の様に書き換えることもできる]

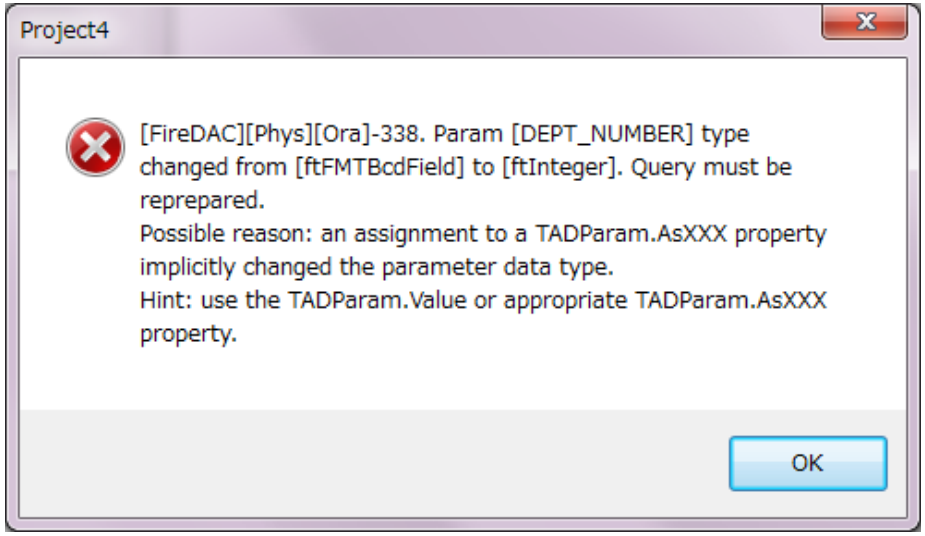
```
with ADStoredProc1 do
begin
  StoredProcName := 'ProcA';
  Prepare;
  Params[0].AsInteger := i1;
  Params[1].AsInteger := i2;
  ExecProc;
  o1 := Params[2].AsInteger;
end;
```

PROCEDUREの実行（3/4）

・以下のコードを記述（呼出側）

```
procedure TForm1.Button2Click(Sender: TObject);
var
  o1: Integer;
begin
  DataModule1.ExcProcA(1000, 250, o1);
  Edit1.Text := IntToStr(o1);
end;
```

・実行エラーが発生 !?



PROCEDUREの実行（4/4）

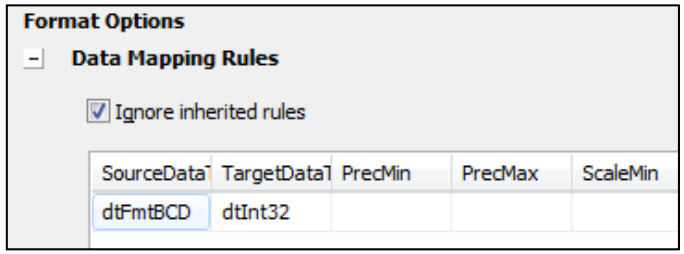
・**エラー回避策として次の2通りがある**

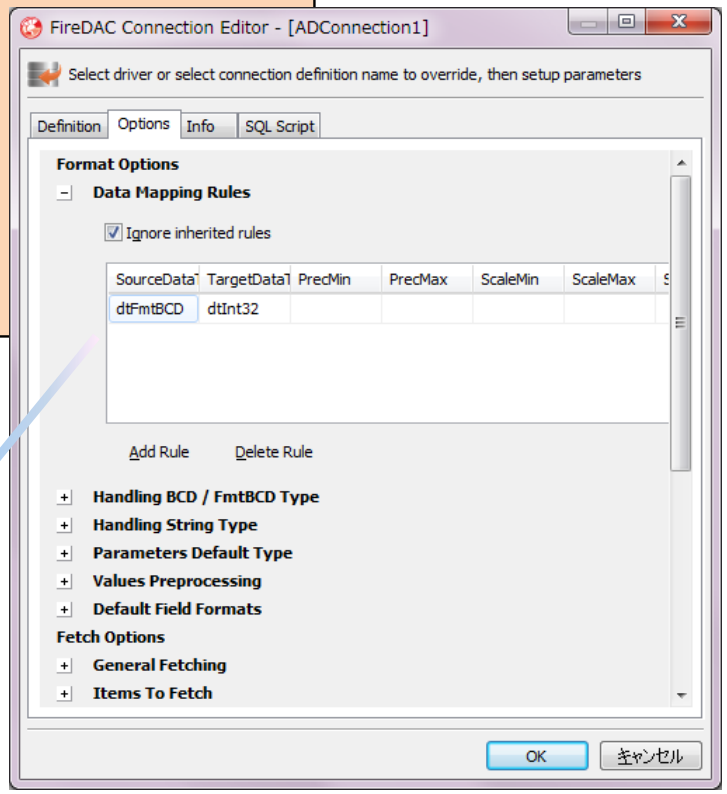
(1). Valueパラメータ(バリアント型)を使用する

```
procedure TDataModule1.ExcProcA(const i1, i2: Integer; var o1: Integer);
begin
  with ADStoredProc1 do
  begin
    StoredProcName := 'ProcA';
    Prepare;
    ParamByName('i1').Value := i1;
    ParamByName('i2').Value := i2;
    ExecProc;
    o1 := ParamByName('o1').Value;
  end;
end;
```

(2).データ型のマッピング・ルールを指定する

・**Connection Editor** の「Options」で設定する
・この例では、「dtFmtBCD」を「dtInt32」にマッピングする



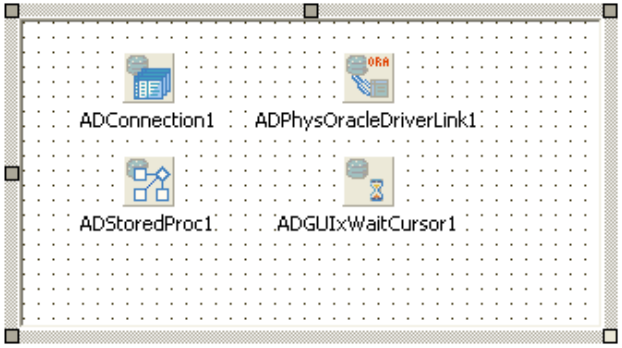


FUNCTIONの実行（1/3）

・以下のSQL文を実行し、Oracleサーバーにプロシジャ「FuncA」を登録する

```
CREATE OR REPLACE FUNCTION FuncA (
  i1 IN NUMBER,
  i2 IN NUMBER) RETURN NUMBER AS
BEGIN
  RETURN i1 + i2;
END;
```

・データモジュールにコンポーネントを配置



FUNCTIONの実行（2/3）

・ConnectionName(またはConnection)プロパティを設定し、
TADConnectionコンポーネントと結び付ける

・データモジュールに、次のコードを記述（呼ばれる側）

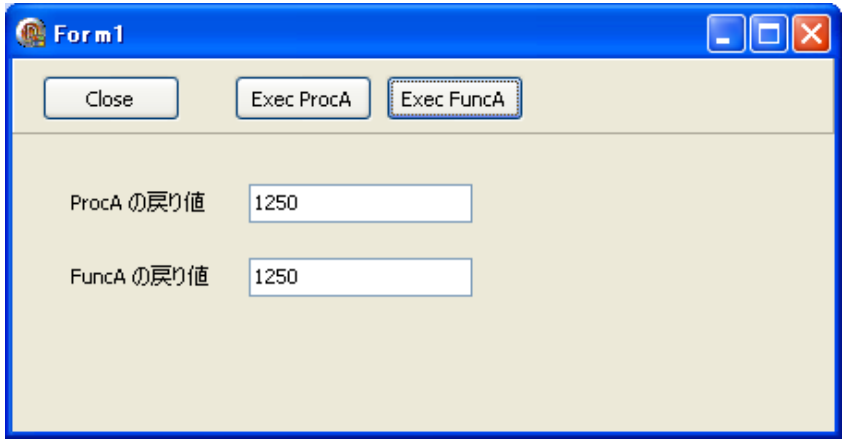
```
function TDataModule1.ExcFuncA(const i1, i2: Integer): Integer;
begin
  with ADStoredProc1 do
  begin
    // ファンクション名の設定
    StoredProcName := 'FuncA';
    // ファンクションの読み込み
    Prepare;
    // パラメータの設定
    ParamByName('i1').Value := i1;
    ParamByName('i2').Value := i2;
    // ファンクションの実行
    ExecProc;
    // 出力パラメータを返す
    Result := ParamByName('RESULT').Value;
  end;
end;
```

FUNCTIONの実行（3/3）

・以下のコードを記述（呼出側）

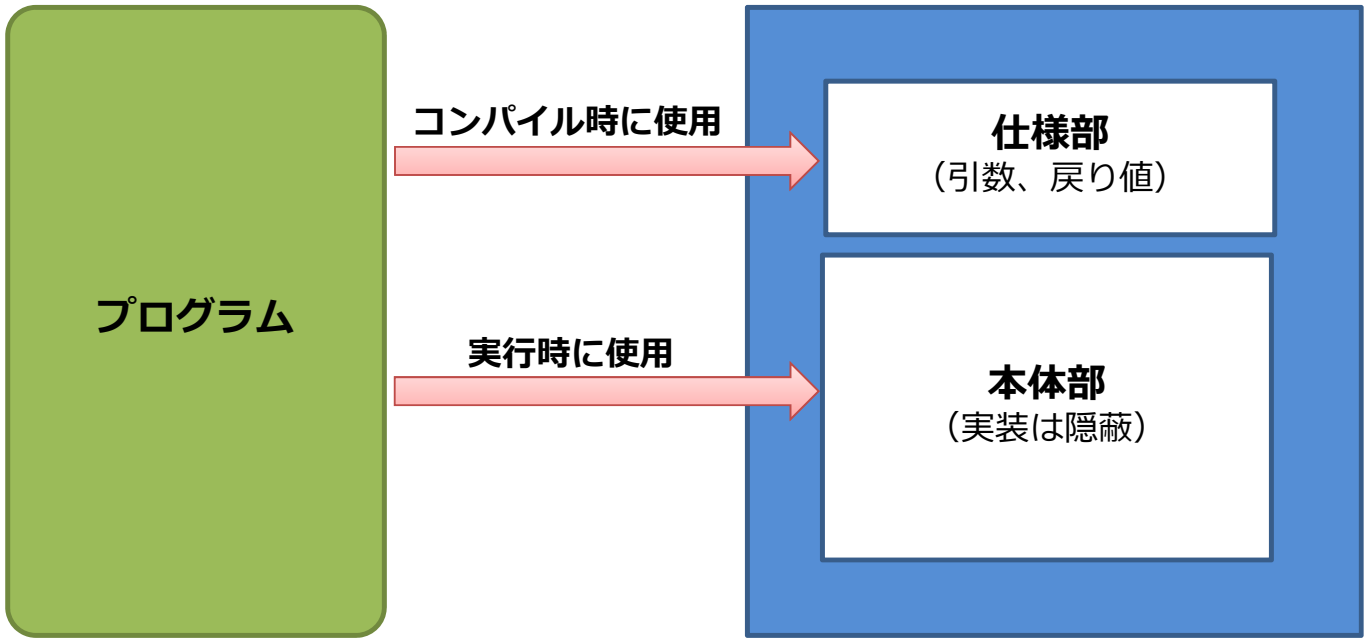
```
procedure TForm1.Button3Click(Sender: TObject);
var
  o1: Integer;
begin
  o1 := DataModule1.ExcFuncA(1000, 250);
  Edit2.Text := IntToStr(o1);
end;
```

[実行結果]



PACKAGEの実行（1/5）

・パッケージはORACLE固有のデータベースオブジェクト
・PROCEDUREやFUNCTION を１つにまとめることが可能
・パッケージは仕様部と本体部の２つのオブジェクトから構成



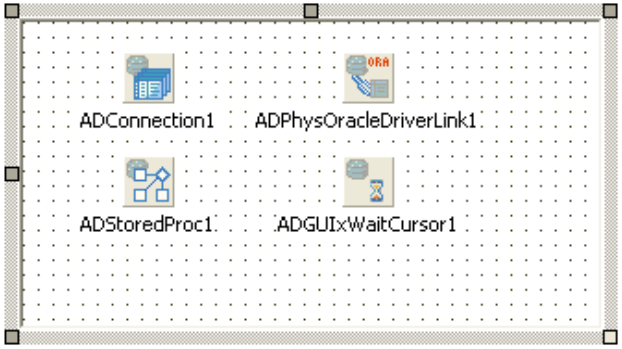
PACKAGEの実行（2/5）

・以下のSQL文を実行し、Oracleサーバーにプロシジャ「 PackA 」を登録する

```
/* 仕様部 */
CREATE OR REPLACE PACKAGE PackA AS
  PROCEDURE ProcA(i1 IN NUMBER, i2 IN NUMBER, o1 OUT NUMBER);
  FUNCTION FuncA(i1 IN NUMBER, i2 IN NUMBER) RETURN NUMBER;
END PackA;
/
/* 本体部 */
CREATE OR REPLACE PACKAGE BODY PACKA AS
  PROCEDURE ProcA (
    i1 IN NUMBER,
    i2 IN NUMBER,
    o1 OUT NUMBER) IS
  BEGIN
    o1 := i1 + i2;
    RETURN;
  END;

  FUNCTION FuncA (
    i1 IN NUMBER,
    i2 IN NUMBER) RETURN NUMBER IS
  BEGIN
    RETURN i1 + i2;
  END;
END PackA;
```

PACKAGEの実行（3/5）

・データモジュールにコンポーネントを配置



・ConnectionName(またはConnection)プロパティを設定し、
TADConnectionコンポーネントと結び付ける

・データモジュールに次のコードを記述

```
procedure TDataModule1.DataModuleCreate(Sender: TObject);
begin
  ADConnection1.Connected := True;
  ADStoredProc1.PackageName := 'PackA';
end;
```

PACKAGEの実行（4/5）

・データモジュールに、ProcA、FuncAを実行するコードを記述

[ProcA]

```
procedure TDataModule1.ExcProcA(const i1, i2: Integer; var o1: Integer);
begin
  with ADStoredProc1 do
  begin
    StoredProcName := 'ProcA';
    Prepare;
    ParamByName('i1').Value := i1;
    ParamByName('i2').Value := i2;
    ExecProc;
    o1 := Params[2].Value;
  end;
end;
```

[FuncA]

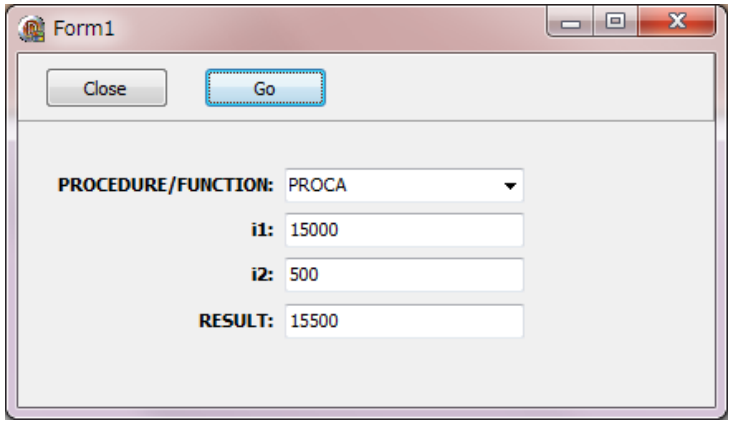
```
function TDataModule1.ExcFuncA(const i1, i2: Integer): Integer;
begin
  with ADStoredProc1 do
  begin
    StoredProcName := 'FuncA';
    Prepare;
    ParamByName('i1').Value := i1;
    ParamByName('i2').Value := i2;
    ExecProc;
    Result := ParamByName('RESULT').Value;
  end;
end;
```

PACKAGEの実行（5/5）

・以下のコードを記述（呼出側）

```
procedure TForm1.Button2Click(Sender: TObject);
var
  o1: Integer;
begin
  case ComboBox1.ItemIndex of
    0: // FuncA
      o1 := DataModule1.ExcFuncA(StrToInt(Edit1.Text), StrToInt(Edit2.Text));
    1: // ProcA
      DataModule1.ExcProcA(StrToInt(Edit1.Text), StrToInt(Edit2.Text), o1);
  end;
  Edit3.Text := IntToStr(o1);
end;
```

[実行結果]



はじめてのFireDAC

まとめ

まとめ

・BDEとの高い互換性により、移行工数を最低限に抑えることが可能

・いろいろなデータベースへの接続を可能とするコンポーネント群

・ハイパフォーマンス

・エンタープライズデータベースに簡単に接続することが可能

・統一的なデータアクセスの提供と、
高度なマクロ機能によりデータベース固有の処理も可能

・最新のOS・DBMSバージョンに対応

・約60個を超える豊富なオプション

参考情報

✓**FireDAC ヘルプ**
http://docs.embarcadero.com/products/rad_studio/firedac/frames.html?frmname=topic&frmfile=index.html

✓**RAD Studioマイグレーションセンター**
http://www.embarcadero.com/jp/rad-in-action/migration-upgrade-center

✓**FireDAC Q&A**
http://www.embarcadero.com/jp/products/rad-studio/firedac-faq

✓**BDE から FireDAC への移行 - Paradox から InterBase の場合**
http://edn.embarcadero.com/jp/article/42974

✓**AnyDACのデモ (Dmitry Arefiev)**
http://www.youtube.com/watch?v=vCM-PcZFdWA

ご清聴
ありがとうございました

「 Delphi talks 」メンバー募集中！

Facebookを利用して、Delphiのこと、C++Builderこと、いろいろなことを語り合いませんか?
時々、勉強会やオフ会もやっています。皆さまの参加をお待ちしています。

https://www.facebook.com/groups/delphitalks/