데이터베이스 어플리케이션 제작의 팁들

데이터베이스 어플리케이션 들을 제작하다 보면 여러가지 작은 난관에 부딪힐 때가 많다. 이럴 때에는 아주 간단한 한 줄의 코딩이나 힌트도 커다란 힘이 된다는 것을 많이 경험하게 된다.
이번 장에서는 델파이로 데이터베이스 어플리케이션을 제작할 때 유용하게 사용할 수 있는 여러가지 팁들을 소개하고자 한다.

기본적인 참고사항

먼저 데이터베이스 어플리케이션을 제작할 때 흔히 만날 수 있는 일반적인 팁에 대해서 알아보도록 하자.

- 동적 앨리어스를 활용하자.

앨리어스는 동적으로 만들어 사용하는 것이 좋다. 다시 말해, BDE 를 이용하여 앨리어스를 등록하는 것보다는 TDatabase 컴포넌트를 사용하여 런타임-환경에서 앨리어스를 만드는 것이 좋다.

- UpdateMode 는 upWhereChanged 로 !

UpdatMode 는 보통 upWhereALL 을 사용하는데 upWhereChanged 를 사용하는 것이 데이터베이스 성능을 향상시키는데 좋다. 다만, 멀티-uer 인 경우에는 처음 레코드에 수정을 가한 사람이 종료하기 전에 데이터의 변경을 가할 경우에 문제가 발생할 수 있다. 이런 경우에는 반드시 upWhereALL 을 사용해야 한다.

- SQL 문장을 수행하기 전에는 꼭 Prepare 문을 수행한다.

Prepare 문장이 중요한 이유는, SQL 문장을 수행하기 위한 최적화 작업을 수행하기 때문이다.

- Join 보다는 View 를 많이 만들어라 !

Join 작업으로 동적으로 View 를 만드는 것보다는, 아예 고정시켜 놓고 사용하는 것이 훨씬 좋다는 것은 설명하지 않아도 쉽게 알 수 있을 것이다.

- CD 타이틀을 만들 경우에 읽기 전용의 파라독스 테이블 사용하기

CD 타이틀의 경우 데이터베이스 파일이 읽기 전용 저장 매체인 CD 에 담겨 있다. 이 경우 데이터베이스 파일을 하드 디스크로 복사한 다음 앨리어스를 생성해서 사용해도 되지만, 하드 디스크 공간을 낭비하는 문제가 있다.
BDE 는 Pdxusers.lck 와 Paradox.lck 파일을 네트워크 등의 멀티유저 환경에 로킹(locking) 정보 파일로 사용한다. 문제는 이 두 파일이 읽기 전용일 경우에는 에러가 발생하며, 오동작 한다는 점이다. 이중에서 PARADOX.LCK 파일은 DOS 용 파라독스 테이블을 지원하기 위한 것이므로 없어도 무방하다. 다만, 비어 있는 PDXUSRS.LCK 파일을 미리 하드디스크에 생성하여 CD 타이틀의 데이터베이스 위치에 넣으면 문제를 해결할 수 있다.
그리고, 다음과 같은 코드를 사용한다.

```
DbiAcqPersistTableLock(Database1.handle, 'PARADOX.DRO','PARADOX' ) );
```

이때 TDatabase 컴포넌트 파라미터의 패스 값에 해당 CD 타이틀의 테이블 위치 값이 입력되도록 해야 한다.

- 그 밖에 ...

1. 폼의 삭제에는 Free 보다 Release 를 사용하시는 것이 좋다.
2. RDBMS 의 성능을 제대로 사용하려면 저장 프로시저를 많이 사용하라 !
3. 캐쉬 업데이트를 적극적으로 사용한다.
4. 필드 에디터(Field Editor)에서 만들어진 고정된 필드 객체는 포인터로 연결되어 프로그램 수행상의 성능이 좋다.
5. 멀티-쓰레드를 적극적으로 활용하여, SQL 문장을 RDB 에 던져 놓고 멍청하게 노는 프로그램이 되지 않도록 한다.

오류가 발생된 파라독스 테이블의 복구

델파이에서 기본적으로 지원하는 파라독스 구조의 테이블은 좀처럼 오류가 발생되지 않는 구조의 데이터베이스이다. 이를 활용하면 일반적인 업무용은 물론 LAN 공유 프로그램도 충분히 만들어 낼 수 있다. 이렇게 실무에 적용하다 보면 테이블에 오류가 발생되는 것은 기정사실로 보아야만 한다. 프로그래머는 이런 오류가 발생한 경우를 예측해야 하고. 이의 대처 방법을 준비해야 하는데 이번에는 이 오류를 수정하는 방법을 알아보자.
이렇게 오류를 수정하기 위해서는 DLL 파일이 필요하다. 이 DLL 은 파라독스를 사용하는

어떤 프로그램에서도 사용할 수 있다.
이 DLL 을 받아오려면, http://www.inprise.com/devsupport/bde/utilities.html에 접근하면 많은 유틸리티를 얻을 수 있다. 이 곳에서 TUtility v4.01 과, Paradox Table Repair 키트를 얻을 수 있다.
Paradox Table Repair 에는 TUtil32.pas 유닛이 있다. 이 유닛의 내용을 살펴보면. TUInit, TUVerifyTable, TURebuildTable, TUGetCRTblDescCount, TUFillCRTblDesc, TUFillCURProps , TUGetExtTblProps , TUExit , TUGetErrorString 등의 함수들을 지원한다. 이중에서 가장 중요한 함수가 TUverifyTable 과 TURebuildeTable 인데, 이 함수들은 테이블 검사와 테이블 재복구 기능을 수행한다.
그러면, DLL 파일에 포함된 기능을 사용할 클래스를 설계해 보자. 이 클래스는 테이블을 검사하고 복구할 때 progress bar 를 지원하기 위한 콜백 함수를 만드는 것이 주목적이다. 콜백 함수에 대한 더 자세한 내용은 42 장의 내용을 참고하기 바란다.

```
TBDEUtil = class
  CbInfo: TUVerifyCallback;
        //콜백 지원을 위한 레코드형 (작업량, 테이블이름, 실제 Process, 인덱스 정보)
  TUProps: CURProps;        //파라독스를 지원하기 위한 DBI 함수 중에 DB 커서에 대한 정보
  hDb: hDBIDb;              //DB 의 핸들 값을 저장
  vhTSes: hTUSes;           // Word 형(테이블 세션정보 저장)
  constructor Create;
  destructor Destroy; override;
  function GetTCursorProps(szTable: String): Boolean;
  procedure RegisterCallBack;
  procedure UnRegisterCallBack;
end;

// Progress 를 보여주기 위한 콜백 함수.
function GenProgressCallBack(ecbType: CBType; Data: LongInt; pcbInfo: Pointer):
  CBRType; stdcall;
var
  CBInfo: TUVerifyCallBack;
begin
  CBInfo := TUVerifyCallBack(pcbInfo^);
  if ecbType = cbGENPROGRESS then
    case CBInfo.Process of          //현재 검사하고 있는 Process 정보는?
                                    //CBInfo.percentdone: 현재 진행된 퍼센트값
```

```
      TUVerifyHeader:
        begin
          헤더에 해당하는 Progress.Position := CBInfo.percentdone;
        end;
      TUVerifyIndex:
        begin
          Index 에 해당하는 Progress.Position := CBInfo.percentdone;
        end;
      TUVerifyData:
        begin
          Data 에 해당하는 Progress.Position := CBInfo.percentdone;
        end;
      TURebuild:
        begin
          재구성에 해당하는 Progress.Position := CBInfo.percentdone;
        end;
      end;
  Result := cbrUSEDEF;        //cbrABORT, cbrCONTINUE 등의 리턴 값, cbrUSEDEF 는 사용자 정의
end;

constructor TBDEUtil.Create;
begin
  TUInit(vhtSes);             //지정 테이블을 사용할 수 있도록 DLL 을 초기화
end;

destructor TBDEUtil.Destroy;
begin
  Check(TUExit(vhtSes));      //사용한 테이블의 정보를 해제
  inherited Destroy;
end;

function TBDEUtil.GetTCursorProps(szTable: String): Boolean;
          //테이블의 정보를 읽어낸다. (세션정보, 테이블 명, 읽어올 정보 )
begin
  if TUFillCURProps(vHtSes, PChar(szTable), TUProps) = DBIERR_NONE then
    Result := True
```

```
  else Result := False;
end;

procedure TBDEUtil.RegisterCallback;
begin
  Check(DbiRegisterCallBack(nil, cbGENPROGRESS, 0,
    sizeof(TUVerifyCallBack), @CbInfo, GenProgressCallback));
          //GenProgressCallBack 프로시저를 cbGEBPROGRESS 에 콜백 시킴
          //세션정보, 콜백 타입, 콜백의 버퍼 크기, 버퍼 데이타, 보낼 콜백 함수
end;

procedure TBDEUtil.UnRegisterCallback;
begin
  Check(DbiRegisterCallBack(nil, cbGENPROGRESS, 0,
    sizeof(TUVerifyCallBack), @CbInfo, nil));
          //만들어 넣은 콜백 함수를 해제
end;
```

먼저 간단한 예로 이 TUTIL32.dll 을 사용하여 테이블의 정보를 읽어 보자. 만드는 프로시저는 테이블의 정보를 읽어내는 TableInfo 루틴이다.

```
procedure TableInfo;
var
  Buffer, Table: String;
begin
  Table := '원하는 테이블명';
   if BDEUtil.GetTCursorProps(Table) then
          //실제 테이블의 정보를 읽어 낸다
     with BDEUtil.TUProps do
          //읽어온 정보는 TUProps 에 있다
     begin
       IntToStr(iFields);                //필드 갯수
       IntToStr(iRecBufSize);            //레코드 크기
       IntToStr(iIndexes);               //인덱스 수
       InttoStr(iValChecks);             //유효성 검사
       IntToStr(iRefIntChecks);          // 참조무결성 수
```

```
        IntToStr(iRestrVersion);          //재구성 버전
        IntToStr(iPasswords);             //패스워드
        IntToStr(iCodePage);              //코드 페이지
        IntToStr(iBlockSize);             //블럭 크기
        IntToStr(iTblLevel);              //테이블 레벨
      end;
end;
```

이제 간단한 구조를 보았으니 본격적으로 테이블의 오류를 검증하는 루틴을 살펴보자.

```
function Trepair_form.Tableverify(FileName: String): Boolean;
          // 테이블의 오류 검증하는 루틴
var
  Msg, L: Integer;
  Table: String;
  Ret: Boolean;
begin
  Screen.Cursor := crHourGlass;
  Ret := False;
  try
    Table := FileName;
    Check(TUExit(BDEUtil.vHtSes));
    Check(TUInit(BDEUtil.vHtSes));
          //DLL 을 사용하기 위해 종료/시작을 반복하여 초기화를 수행한다
          //이 DLL 은 작업을 수행 할 때마다, 재 초기화작업을 거쳐야 한다
    BDEUtil.RegisterCallBack;
          //Progress 를 출력하기 위해 콜백 함수를 설정 한다
    try
      if TUVerifyTable(BDEUtil.vHtSes, PChar(Table), szPARADOX, 'VERIFY.DB',
        nil, 0, Msg) = DBIERR_NONE then
          //실제 TUtil32.dll 의 TUverifyTable 함수를 호출하여 검증 작업을 실시한다
          //세션 핸들 값, 테이블 명, 드라이버 속성, 에러출력 DB, 패스워드, 옵션, 에러 레벨-오류
            가 발생하면 오류값을 읽어온다
      begin
          //오류가 발생하였을 경우에 메시지를 보여준다
        case Msg of
```

```
          0: eLabel.Caption := '테이블 확인 완료. 해당 테이블에 오류가 없습니다.';
          1: eLabel.Caption := '테이블 확인 완료. 해당 테이블에 비교된 테이블입니다.';
          2: eLabel.Caption := '테이블 확인 완료. 확인이 부정확하게 종료되었습니다.';
          3: eLabel.Caption := '테이블 확인 완료. 테이블을 재구성 하십시오.';
          4: eLabel.Caption := '테이블 확인 완료. 테이블을 재구성할 수 없습니다!.';
          else
            begin
              eLabel.Caption := '테이블 오류 확인 실패.';
                                //구분할 수 없는 에러가 발생했을 경우..
               Ret := True;
            end;
          end;
        end;
      finally
        BDEUtil.UnRegisterCallBack;                  //콜백 함수를 원위치 시킨다
      end;
    finally
      Screen.Cursor := crDefault;
    end;
    Tableverify := Ret;
end;
```

이제 테이블의 오류를 체크하여 낼 수 있다. 이렇게 체크하여 오류가 발생한 테이블은 다음의 복구 루틴으로 테이블을 재복구할 수 있다.

```
procedure Trepair_form.TableRebuild(FileName: String);       //테이블을 재구성한다
var
  iFld, iIdx, iSec, iVal, iRI, iOptP, iOptD: Word;
  szTable, Backup: String;
  Rslt: DBIResult;
  Msg: Integer;
  TblDesc: CRTBlDesc;
begin
  Screen.Cursor := crHourGlass;
  try
    Check(TUExit(BDEUtil.vHtSes));
```

```
Check(TUInit(BDEUtil.vHtSes));                    //TUtil32.dll 을 재구성한다
szTable := FileName;
BDEUtil.RegisterCallBack;                         //콜백을 지정한다
try
  Check(TUVerifyTable(BDEUtil.vHtSes, PChar(szTable), szPARADOX, 'VERIFY.DB',
    nil, 0, Msg));                                //Rebuild 하기 전에 테이블을 한번 검사한다
  Rslt := TUGetCRTblDescCount(BDEUtil.vhTSes, PChar(szTable), iFld,
    iIdx, iSec, iVal, iRI, iOptP, iOptD);         //재구성하기 위한 정보를 읽어 온다
                //(세션 값, 테이블 명, 필드 수, 인덱스 수, 레코드 수, 유효성 수, 참조 무결성
                   수, 파라미터 옵션, 옵션 데이터 크기)
  if Rslt = DBIERR_NONE then      //에러가 없으면, 테이블의 구성 정보를 읽어 낼 수 있으면..
  begin                           // 정보를 제대로 읽어오면.. 재구성이 가능하다
    FillChar(TblDesc, SizeOf(CRTBlDesc), 0);
    StrPCopy(TblDesc.szTblName, szTable);
    TblDesc.szTblType := szParadox;
    TblDesc.szErrTblName := 'Rebuild.DB';
                //재구성하는 테이블은 파라독스이며 에러가 발생하면 Rebuild.db 에 저장한다
    TblDesc.iFldCount := iFld;      //필드 수 만큼..
    GetMem(TblDesc.pFldDesc, (iFld * SizeOf(FldDesc)));
                                    //필드의 수를 읽어서 필요한 만큼의 메모리 할당
    TblDesc.iIdxCount := iIdx;
    GetMem(TblDesc.pIdxDesc, (iIdx * SizeOf(IdxDesc)));
                                    //인덱스 수를 읽어서 필요한 만큼의 메모리 할당
    TblDesc.iSecRecCount := iSec;
    GetMem(TblDesc.pSecDesc, (iSec * SizeOf(SecDesc)));
                                    //테이블 정보의 크기만큼의 메모리 할당.
    TblDesc.iValChkCount := iVal;
    GetMem(TblDesc.pvchkDesc, (iVal * SizeOf(VCHKDesc)));
                                    //유효성 정보를 읽어 들임.
    TblDesc.iRintCount := iRI;
    GetMem(TblDesc.printDesc, (iRI * SizeOf(RINTDesc)));
                                    //참조 무결성 정보를 읽어들임.
    TblDesc.iOptParams := iOptP;
    GetMem(TblDesc.pfldOptParams, (iOptP * sizeOf(FLDDesc)));
                                    //파라미터 정보를 읽어들임
    GetMem(TblDesc.pOptData, (iOptD * DBIMAXSCFLDLEN));
```

```
        try
          Rslt := TUFillCRTblDesc(BDEUtil.vhTSes, @TblDesc, PChar(szTable), nil);
                                  //테이블 정보 읽어 들이기
          if Rslt = DBIERR_NONE then
          begin
            //에러없이 읽어 들이면..
            Backup := 'Backup.Db';
            if TURebuildTable(BDEUtil.vhTSes, PChar(szTable), szPARADOX,
              PChar(Backup), 'KEYVIOL.DB', 'PROBLEM.DB', @TblDesc)
                                  //테이블의 재복구를 시도한다.
                                  //(세션 값, 테이블 명, 드라이버 타입, 백업 Db 명, 에러 Db 명,
                                     문제저장 Db 명, 테이블 구조 정보 값)
              = DBIERR_NONE then runOK.Caption := '재복구 성공 !'
            else runOK.Caption := '재복구 실패 !';
          end
          else
            MessageDlg('테이블의 구조나 내용에 이상이 있음: ', mtError, [mbok], 0);
        finally
          FreeMem(TblDesc.pFldDesc, (iFld * SizeOf(FldDesc)));
          FreeMem(TblDesc.pIdxDesc, (iIdx * SizeOf(IdxDesc)));
          FreeMem(TblDesc.pSecDesc, (iSec * SizeOf(SecDesc)));
          FreeMem(TblDesc.pvchkDesc, (iVal * SizeOf(VCHKDesc)));
          FreeMem(TblDesc.printDesc, (iRI * SizeOf(RINTDesc)));
          FreeMem(TblDesc.pfldOptParams, (iOptP * sizeOf(FLDDesc)));
          FreeMem(TblDesc.pOptData, (iOptD * DBIMAXSCFLDLEN));
                                  //할당된 메모리를 해제
        end;
      end;
    finally
        BDEUtil.UnRegisterCallBack;          //콜백 함수를 돌려 놓는다.
    end;
  finally
      Screen.Cursor := crDefault;
  end;
end;
```

이렇게 만들어진 두개의 함수를 사용하면 파라독스로 만들어진 테이블의 오류를 검사하고 복구할 수 있다. 이렇게 만들어진 루틴은 델파이로 만드는 프로그램 내부에 삽입하여 Idle time 에 검사를 수행하거나 오류가 발생되었을 경우에 복구할 수 있는 루틴으로 사용하면 좋을 것이다.

데이터베이스 테이블 생성 방법

파라독스 파일을 사용하는 경우 기본적인 필드를 가지는 테이블 파일을 생성할 수 있다. 이제 다음의 파일 구조를 가지는 파라독스 테이블을 프로그램에서 동적으로 생성하여 보자.

NameDB
Name Char 20 Primary
Addrtess Char 40

NikNameDB
Name Char 20 Primary
Nikname Char 20 Primary

TelDB
Name Char 20 Primary
Tel1 Char 20 Primary
Tel2 Char 20 Primary

그리고 데이터베이스 파일을 생성하면서 NikNameDB 에서 NameDB 에 Name 필드로 Table Lookup 을 연결하고, TelDB 에서 NameDB 에 Referential Integrity 로 Name 필드를 사용해서 연결하도록 할 것이다.
먼저 다음은 테이블을 생성하는데 사용되는 변수들에 대한 설명이다.

변 수	데이터 형	설 명
szDirectory	DBIPATH	만들고자 하는 Table 의 위치를 지정하는 변수
TableDesc	CRTblDesc	테이블 정보를 저장하는 변수
FieldsDesc	array[0..n] of FLDDesc	필드 정보를 저장하는 변수
IndexesOP	array[0..n] of CROpType	인덱스의 처리 방법에 대해 지정되는 변수
IndexesDesc	array[0..n] of IDEXDesc	인덱스 정보를 저장하는 변수
RefIntegOp	array[0..n] of CROpType	참조 무결성 처리 방법에 대해 지정되는 변수
RefInteg	array[0..n] of RINTDes	참조 무결성의 정보를 저장하는 변수

ValCheckOP	array[0..n] of CROpType	테이블 룩업(Lookup)처리 방법에 대해 지정하는 변수
ValCheckDesc	array[0..n] of VCHKDesc	테이블 룩업의 정보를 저장하는 변수

사용할 프로시저를 차례로 알아보자.

- 필드 정보값 넣기

```
procedure DefField (const sName: String; const iAFldType, iASubType, iAFldNum,
  iAUnits1, iAUnits2: Integer);
begin
  with FieldsDesc[iAFldNum] do
  begin
    iFldNum   := iAFldNum;
    StrPCopy(szName,sName);
    iFldType := iAFldType;
    iSubType := iASubType;
    iUnits1   := iAUnits1;
    iUnits2 := iAUnits2;
  end;
end;
```

실제 FLDDesc 의 값은 다음과 같다.

이 름	데이터 형	내 용
iFldNum	Word	필드의 번호로 파라독스의 경우 변하지 않는 수
szName	DBINAME	필드의 이름
iFldType	Word	필드의 타입, 변환모드에서 xltNONE 인 경우 해당 드라이버의 물리적인 타입이며 아닌 경우는 BDE 의 논리적인 타입이다
iSubType	Word	필드의 서브타입으로 변환모드의 설정 값에 따라 논리적인 서브 타입이나 드라이버의 물리적인 서브 타입이다.
iUnits1	Integer	문자, 숫자, 기타 등의 크기를 지정, 수치 타입인 경우 iUnit2 는 배율입니다.
iUnits2	Integer	소수점 자리 수, 기타 등의 정보를 지정한다.
iOffset	Word	레코드 버퍼에서 해당 오프셋을 보고 점프한다. (테이블 생성시에는 참고하지 않는다)
iLen	Word	바이트로 나타낸 필드의 계산된 길이(테이블 생성 시에는 참고 되지 않

		는다)
iNullOffset	Word	필드의 NULL 을 기록한다. (테이블 생성시에는 참고되지 않는다)
efldvVchk	FLDVchk	유효성 검사의 타입을 검사한다. (테이블 생성시에는 참고되지 않는다)
efldrRights	FLDRights	사용자를 위한 계산된 필드 수준 권한을 체크할 경우에 사용하는 필드이다. (테이블 생성 시에는 참고되지 않는다)

이상의 필드에서 필요한 필드만 선택해서 해당 필드의 정보를 채웁니다.

- 테이블의 인덱스 정보 만들기

```
procedure DefIndex (const sName,sTagName,sFormat,sKeyExp,sKeyCond: string;
                    const aFields: array of integer;
                    const iIndexPos,iAIndexID,iAFldsInKey,iAKeyLen,
                          iAKeyExptype,iABlockSize,iARestrNum: integer;
                    const bAPrimary,bAUnique,bADescending,bAMaintained,bASubSet,
                          bAExpIDX,bAOutOfDate,bACaseInsensitive: boolean);
var
  i: byte;
begin
  IndexesOp[iIndexPos] := crAdd;
  with IndexesDesc[iIndexPos] do begin
    StrPCopy(szName,sName); iIndexId := iAIndexId;
    StrPCopy(szFormat,sFormat); StrPCopy(szTagName,sTagName);
    StrPCopy(szKeyExp,sKeyExp); StrPCopy(szKeyCond,sKeyCond);
    iFldsInkey := iAFldsInkey; iKeyLen := iAKeyLen; iKeyExpType := iAKeyExpType;
    iBlocksize := iABlocksize; iRestrNum := iARestrNum;
    bPrimary := bAPrimary;  bUnique := bAUnique;  bDescending := bADescending;
    bMaintained := bAMaintained;  bSubset := bASubset; bExpIdx := bAExpIdx;
    bOutofDate := bAOutofDate; bCaseInsensitive := bACaseInsensitive;
    FillChar(aiKeyFld,SizeOf(aiKeyFld),#0);
    for i := Low(aFields) to High(aFields) do
      aiKeyFld[i] := aFields[i];
  end;
end;
```

해당소스를 살펴보기 전에 인덱스 정보(IdeDesc)에 들어가는 구조를 알아보자.

이 름	데이터 형	내 용
szName	DBITBLNAME	인덱스 이름
iIndexId	Word	인덱스 번호
szTagName	DBINAME	태그 명(dBase 전용)
szFormat	DBINAME	선택형 포맷 (BTree, HASH, 기타)
bPrimary	Bool	True 이면 기본 인덱스
bUnique	Bool	True 이면 인덱스는 유일한 키를 포함
bDescending	Bool	True 이면 내림차순
bMaintained	Bool	True 이면 유지 관리되는 인덱스
bSubset	Bool	True 이면 부분집합 인덱스 (dBase 전용)
bExpIdx	Bool	True 이면 표현식 인덱스이다 (dBase 전용)
iCost	Word	사용하지 않음..
iFldsInKey	Word	복합 인덱스에서 키 필드의 수를 지정
iKeyLen	Word	인덱스 생성 동안은 지정되지 않고, 바이트로 나타낸 키의 물리적인 길이를 가리킨다.
bOutofDate	Bool	True 이면 인덱스의 날짜가 지난 것이다.
iKeyExpType	Word	키 표현식의 타입을 지정(dBase 전용)
aiKeyFld	DBIKEY	키 안의 키 숫자 배열
szKeyExp	DBIKEYEXP	표현식 인덱스를 위한 키 표현식 지정(dBase 전용)
szKeyCond	DBIKEYEXP	부분집합 조건을 정의한 식을 지정, dBase 식으로 기술
bCaseInsensitive	Bool	True 이면 인덱스는 대소문자를 구별하지 않는다.
iBlockSize	Word	바이트로 나타낸 인덱스의 블록크기
iRestrNum	Word	인덱스 생성시에는 나타나지 않으나, 내부 재구성 정보를 가진다.
iUnUsed	Array[0..15] of word	사용하지 않음..

- 참조 무결성 처리

```
procedure DefRefInt (const iRintPos,iARintNum,iAFldCount: integer;
                     const aiAThisTabFld,aAiOthTabFld: array of integer;
                     const sRintName,sTblName: string; eAType: RINTType;
                     const eAModOP,eADelOP: RINTQual);
var
  i: byte;
```

```
begin
  RefIntegOp[iRintPos] := crAdd;
  with RefInteg[iRintPos] do begin
    iRintNum := iARintNum; StrPCopy(szRintName,sRintName);
    eType := eAType; StrPCopy(sztblName,StrPas(szDirectory)+stblName);
    eModOp := eAModOp; eDelOp := eADelOp; iFldCount := iAFldCount;
    FillChar(aiThisTabFld,SizeOf(aiThisTabFld),#0);
    for i := Low(aiAThisTabFld) to High(aiAThisTabFld) do
      aiThisTabFld[i] := aiAThisTabFld[i];
    FillChar(aiOthTabFld,SizeOf(aiOthTabFld),#0);
    for i := Low(aAiOthTabFld) to High(aAiOthTabFld) do
      aiOthTabFld[i] := aAiOthTabFld[i];
  end;
end;
```

RINTDesc 는 참조무결성의 구조를 입력하기 위한 데이터 구조이다.

이 름	데이터 형	내 용
iRintNum	Word	참조 무결성 번호
szRintName	DBINAME	태그 이름
eType	RINTType	타입(rintMASTER 또는 rintDEPENDANT)
szTblName	DBIPATH	연결되는 테이블 이름
eModOp	RINTQual	수정 한정사 (rintRESTRICT 또는 rintCASCADE)
eDelOp	RINTQual	삭제 한정사 (rintRESTRICT 또는 rintCASCADE)
iFldCount	Word	링크 키의 필드 수
aiThisTabFld	DBIKEY	이 테이블에서 참조 무결성을 구성하는 필드 값
aiOthTabFld	DBIKEY	다른 테이블의 필드 수

- 유효성 검사

```
procedure DefValCheck (const iValPos,iAFldNum: integer;
                       const aAMinVal,aAMaxVal,aADefVal: array of Byte;
                       const bARequired,bAHasMinVal,bAHasMaxVal,bAHasDefVal: boolean;
                       const sPict,sLkupTblName: string;
                       const eALKUPType: LKUPType);
var
```

```
  i: byte;
begin
  ValCheckOp[IValPos] := crAdd;
  with ValCheckDesc[IValPos] do begin
    iFldNum := iAFldNum; StrPCopy(szPict,sPict);
    bRequired := bARequired; bHasMinVal := bAHasMinVal;
    bHasMaxVal := bAHasMaxVal; bHasDefVal := bAHasDefVal;
    eLKUPType := eALKUPType; StrPCopy(szLkupTblName,sLkupTblName);
    FillChar(aMinVal,SizeOf(aMinVal),#0);
    for i := Low(aAMinVal) to High(aAMinVal) do
      aMinVal[i] := aAMinVal[i];
    FillChar(aMaxVal,SizeOf(aMaxVal),#0);
    for i := Low(aAMaxVal) to High(aAMaxVal) do
      aMaxVal[i] := aAMaxVal[i];
    FillChar(aDefVal,SizeOf(aDefVal),#0);
    for i := Low(aADefVal) to High(aADefVal) do
      aDefVal[i] := aADefVal[i];
  end;
end;
```

VCHKDesk 유효성 검사에 사용되는 구조는 다음과 같다.

이 름	데이터 형	내 용
iFldNum	Word	필드 수 (1-n)
bRequired	Bool	True 이면 필드 값이 필요하다
bHasMinVal	Bool	True 이며 최소 값을 가진다
bHasMaxVal	Bool	True 이면 최대 값을 가는다.
bHasDefVal	Bool	True 이면 디폴트 값을 가진다.
aMinVal	DBIVCHK	최소 값
aMaxVal	DBIVCHK	최대 값
aDefVal	DBIVCHK	디폴트 값
szPict	DBIPICT	그림 문자열
elkupType	LKUPType	탐색/채움 타입(Paradox 전용)
szLkupTblName	DBIPATH	탐색 테이블 이름(읽는 정보로만 사용)

유효성 LKUPType 의 값은 다음과 같다.

이 름	내 용
lkupNONE	테이블에 탐색기능이 없다
lkupPRIVATE	현재 필드만 + 전용
lkupALLOCORRESP	모든 해당 필드 + 도움말 없음
lkupHELP	현재 필드만 + 도움말
lkupALLOCORREPSHELP	모든 해당필드 + 도움말

- 테이블 생성에 필요한 정보 입력

다음 DefTable 프로시저는 원하는 테이블을 생성한다.

```
procedure DefTable (const sName,sType,sPassword: string;
  const iAFldCount,iAIDXCount,iAValChkCount,iARintCount: integer);
begin
  FillChar(TableDesc,SizeOf(CRTblDesc),#0);      //TableDesc 에 초기값 채우기..
  with TableDesc do
  begin
    StrPCopy(szTblName,sName); StrPCopy(szTblType,sType);
                    // 테이블명과 테이블의 타입을 ㅈ정하고
    bProtected := (sPassword <> '');              //패스워드가 있으면 세팅하고..
    if bProtected then
    begin
      StrPCopy(szPassword,sPassword);
      Session.AddPassword(sPassword);             //패스워드가 있으면 세션에 패스워드를 넣는다.
    end;
    bPack := True;
    iFldCount := iAFldCount; pFldDesc := @FieldsDesc;
    iValChkCount := iAValChkCount;
    pecrValChkOp := @ValCheckOp; pvchkDesc := @ValCheckDesc;
    iIDXCount := iAIDXCount;
    pecrIDXOp := @IndexesOp; pIDXDesc := @IndexesDesc;
  end;
end;
```

여기서 사용할 CRTblDesc 의 내용은 다음과 같다.

이 름	데이터 형	내 용
szTblName	DBITBLNAME	패스와 확장자를 포함한 테이블 명
szTblType	DBINAME	드라이버 타입
szErrTblName	DBIPATH	에러 테이블 명(선택 값)
szUserName	DBINAME	사용자 명 (있을 경우에..)
szPassword	DBINAME	패스워드(bProtected 가 True 인 경우) 파라독스 전용
szProtected	Bool	True 이면 암호화
szPack	Bool	테이블이 packing 되어야 하는 경우에 True
iFldCount	Word	필드 정의의 수
percFldOp	pCROpType	필드 작업의 배열
pfldDesc	pFLDDesc	필드 디스크립터 배열
iLdxCount	Word	인덱스의 수
perIdxOp	pCROpType	인덱스 작업 배열
pidxDesc	pIDXDesc	인덱스 디스크립터 배열
iSecRecCount	Word	보안 값의 수 (Paradox 전용)
psecDesc	pSECDesc	보안 디스크립터 배열(Paradox 전용)
iValChkCount	Word	유효성 검사의 수(Paradox 및 SQL 전용)
pecrValChkOp	pCROpType	유효성 검사 작업 배열
pvchkDesc	pVCHKDesc	유효성 검사 디스크립터 배열(Paradox 및 SQL 전용)
iRintCount	Word	참조 무결성 수(Paradox 전용)
pecrRintOp	pCROpType	참조 무결성 작업 배열
printDesc	pRINTDesc	참조 무결성 지정시의 수(Paradox 전용)
iOptParams	Word	선택형 파라미터의 수
pfldOptParams	pFLDDesc	선택형 파라미터를 위한 필드 디스크립터의 배열
pOptData	Pointer	선택형 파라메터의 값

- 실제 테이블을 생성한다.

다음의 코드들은 앞에서 만들어진 소스 코드를 통하여 해당 필드에 값을 채우고 인덱스와 참조 무결성, 유효성 검사에 값을 채운 다음 테이블 정보를 넣는다.

```
procedure StoredNamedb;
begin
  DefField('Name', fldZSTRING, 0, 0, 20, 0);
```

```
  DefField('Address', fldZSTRING, 0, 1, 100, 0);
  DefField('Num', fldINT32, 0, 2, 1, 0);
  DefIndex('', '', '', '', '', [1],
           0, 0, 1, 20, 0, 4096, 1, True, True, False, True, False, False, False, False);
  DefTable('NAMEDB.DB', 'PARADOX', '', 3, 1, 0, 0);
end;

procedure StoredTeldb;
begin
  DefField('Name', fldZSTRING, 0, 0, 20, 0);
  DefField('Tel1', fldZSTRING, 0, 1, 20, 0);
  DefField('Tel2', fldZSTRING, 0, 2, 20, 0);
  DefIndex('', '', '', '', '', [1, 2, 3],
           0, 0, 3, 60, 0, 4096, 1, True, True, False, True, False, False, False, False);
  DefRefInt(0, 1, 1, [1], [1], 'nameRef', 'NAMEDB.DB',
            rintDEPENDENT, rintCASCADE, rintRESTRICT);
  DefTable('TELDB.DB', 'PARADOX', '', 3, 1, 0, 1);
end;

procedure StoredNiknamedb;
begin
  DefField('Name', fldZSTRING, 0, 0, 20, 0);
  DefField('NikName', fldZSTRING, 0, 1, 20, 0);
  DefIndex('', '', '', '', '', [1, 2],
           0, 0, 2, 40, 0, 4096, 1, True, True, False, True, False, False, False, False);
  DefValCheck(0, 1, [0], [0], [0],
              False, False, False, False, '', 'nameDB.DB', lkupHELP);
  DefTable('NIKNAMEDB.DB', 'PARADOX', '', 2, 1, 1, 0);
end;
```

이제 실제 MakeAllTables 프로시저를 통하여 테이블을 만들어 낸다. 이때 개발자는 TDatabase 컴포넌트를 하나 생성하여 원하는 앨리어스를 선택한다.

```
procedure MakeAllTables (dbDatabase: TDatabase);
var
  iTables: StoredTables;
```

```
begin
  dbDatabase.Connected := True;        // 연결한 다음
  Check(DbiGetDirectory(dbDatabase.Handle,False,szDirectory));
          //데이터베이스 컴포넌트에서 해당 디렉토리를 읽어 낸다.
          //StoredNamedb 와 StoredTeldb 와 StoredNikNamedb 를 순서대로 호출한 다음
  if DBICreateTable(dbDatabase.Handle,false,TableDesc) = DBIERR_FILEEXISTS then
          //DBICreateTable 을 통하여 TableDesc 의 값으로 테이블을 생성한다.
  begin
    //에러가 발생하였으므로 작업하던 내용을 모두 지운다.
  end;
end;
```

다른 테이블의 데이터 import

테이블 컴포넌트의 BatchMove 메소드를 이용하면 다른 테이블의 데이터를 가져올 수 있다. BatchMove 는 테이블 간의 레코드를 복사하거나, 다른 테이블에서 일어난 레코드를 업데이트, 추가, 삭제하는 기능을 할 수 있다.
BatchMove 는 2 개의 파라미터를 가진다. 데이터를 가져올 테이블의 이름과 어떤 동작을 할 것인지에 대한 모드가 그것이다. 다음 테이블은 BatchMove 의 모드에 대한 값이다.

값	의 미
batAppend	소스 테이블의 레코드를 테이블의 끝에 추가
batAppendUpdate	소스 테이블의 모든 레코드를 추가하고, 테이블에 존재하는 같은 레코드를 업데이트 한다.
batCopy	소스 테이블의 모든 레코드를 복사한다.
BatDelete	소스 테이블에 있는 모든 레코드를 삭제한다.
BatUpdate	소스 테이블의 레코드에 기준하여 레코드를 업데이트 한다.

다음 문장은 현재 테이블의 레코드를 Customer 테이블의 레코드로 업데이트 한다.

```
Table1.BatchMove('CUSTOMER.DB', batUpdate);
```

BatchMove 는 리턴값으로 성공적인 동작을 한 레코드의 수를 돌려준다.

주의: batCopy 로 레코드를 복사하면, 과거의 레코드에 덮어쓰게 된다.

같은 데이터베이스 테이블에 연결된 테이블들의 동기화

하나 이상의 테이블 컴포넌트가 같은 테이블에 연결되어 있으면서 데이터 소스 컴포넌트를 공유하지 않으면 각각의 테이블은 데이터와 레코드에 대한 자신의 뷰를 가지게 된다. 사용자가 이런 테이블 컴포넌트에 접근하게 되면 컴포넌트의 현재 레코드가 서로 달라지게 된다. 이런 상황에서 현재의 레코드를 같게 유지하는 메소드가 있는데, GotoCurrent 메소드가 그것이다.
다음 문장은 CustomerTableOne 테이블 컴포넌트의 현재 레코드를 CustomerTableTwo 테이블 컴포넌트의 현재 레코드와 같게 설정한다.

```
CustomerTableOne.GotoCurrent(CustomerTableTwo);
```

서로 다른 폼에 있는 테이블 컴포넌트를 동기화할 때에는 다음 문장과 같이 폼의 이름을 앞에 적어주면 된다.

```
CustomerTableOne.GotoCurrent(Form2.CustomerTableTwo);
```

양방향 커서의 비활성화

BDE 의 양방향 커서를 비활성화할 때 UniDirectional 프로퍼티를 사용한다. 디폴트 값은 False 로 결과 세트(result set)의 커서를 레코드의 전후로 움직일 수 있다는 것을 의미한다. 양방향 커서를 사용하면 작업을 할 때 약간의 오버헤드를 가중시키기 때문에 쿼리의 속도가 다소 느려지게 된다. 그러므로, 쿼리의 속도를 증진시키려면 UniDirectional 프로퍼티를 True 로 설정하여 커서가 앞으로만 움직이도록 제한하는 것이 좋다.
다음 코드는 쿼리를 실행하기 전에 UniDirectional 을 설정하는 예이다.

```
if not CustomerQuery.Prepared then
begin
  CustomerQuery.UniDirectional := True;
  CustomerQuery.Prepare;
end;
CustomerQuery.Open;
```

이종(heterogeneous) 쿼리의 생성

델파이는 하나 이상의 데이터베이스의 테이블들을 사용하는 이종 쿼리를 지원한다. 예를

들어 오라클, 사이베이스의 테이블과 로컬 디베이스 테이블을 포함한 쿼리가 가능하다. 이렇게 하려면 각각의 서버에서만 지원하는 확장 SQL 문법을 쓰면 안된다.
이종 쿼리를 생성하려면 로컬 디렉토리를 참조하는 BDE 앨리어스를 정의하고 쿼리 컴포넌트의 데이터베이스 Name 프로퍼티를 그 앨리어스로 설정한다. BDE 앨리어스의 정의는 데이터베이스 탐색기(Database Explorer)를 이용하면 된다. 각각의 데이터베이스에 대해 분리된 BDE 앨리어스를 정의하고, SQL 프로퍼티에서 실행할 SQL 문장을 지정한다. Params 프로퍼티를 설정하고 Prepare, Open, ExecSQL 로 쿼리를 실행하면 된다.
예를 들어, CUSTOMER 테이블을 가지고 있는 오러클 데이터베이스의 앨리어스를 Oracle1, ORDERS 테이블을 가지고 있는 사이베이스 데이터베이스의 앨리어스를 Sybase1 이라고 할 때 두 테이블을 이용한 간단한 쿼리를 아래에 적어 보았다.

```
SELECT CUSTOMER.CUSTNO, ORDERS.ORDERNO
FROM ":Oracle1:CUSTOMER", ":Sybase1:ORDERS"
WHERE CUSTNO = 1503
```

그 밖의 유용한 팁들 ...

- TDBNavigator 버튼을 동적으로 제어

TDBNavCracker(DBNavigator1).Buttons[nbEdit].Enabled := true/fasle 로 조절한다.

- 테이블이 비정상 종료할 때를 대비하여 자료를 강제로 저장하는 방법

function DbiSaveChanges(hCursor: hDBICur): DBIResult stdcall;
를 이용한다.

예) DbiSaveChanges(TTable.Handle);

- 파라독스의 Auto Increment 형의 필드작성

TQuery 를 가지고 SQL 의 CREATE TABLE 명령을 사용한다.

```
with Query1 do
begin
  DatabaseName := 'DBDemos';
  with SQL do
```

```
  begin
    Clear;
    Add('CREATE TABLE "PDoxTbl.db" (ID AUTOINC,');
    Add('Name CHAR(255),');
    Add('PRIMARY KEY(ID))');
    ExecSQL;
    Clear;
    Add('CREATE INDEX ByName ON "PDoxTbl.db" (Name)');
    ExecSQL;
  end;
end;
```

- DBGrid 에서 다중 선택된 것 찾아내기

```
var
   x: word;
   TempBookmark: TBookMark;
 begin
   DBGrid1.Datasource.Dataset.DisableControls;
         //SelectedRows 프로퍼티에 DBGrid 에서 다중 선택된 레코드들의
         //북마크(bookmark)를 가지고 있다(TBookmarkList)
   with DBgrid1.SelectedRows do
     if Count > 0 then
     begin
       TempBookmark := DBGrid1.Datasource.Dataset.GetBookmark;
       for x := 0 to Count - 1 do
       begin
         if IndexOf(Items[x]) > -1 then
         begin
           DBGrid1.Datasource.Dataset.Bookmark := Items[x];
           Showmessage(DBGrid1.Datasource.Dataset.Fields[0].AsString);
         end;
       end;
     end;
   DBGrid1.Datasource.Dataset.GotoBookmark(TempBookmark);
   DBGrid1.Datasource.Dataset.FreeBookmark(TempBookmark);
```

```
    DBGrid1.Datasource.Dataset.EnableControls;
 end;
```

- DBGrid 에서 선택된 필드의 타이틀을 굵게 표시하기

ColEnter 이벤트과 ColExit 이벤트를 사용한다.

```
procedure TForm1.DBGrid1ColEnter(Sender: TObject);
begin
          //선택된 필드의 타이틀 색을 파란색으로 굵게 ...
  DBGrid1.Columns[DBGrid1.SelectedIndex].Title.Font.Color := clRed;
  DBGrid1.Columns[DBGrid1.SelectedIndex].Title.Font.Style :=
    DBGrid1.Columns[DBGrid1.SelectedIndex].Title.Font.Style + [fsBold];
  DBGrid1.Repaint;
end;

procedure TForm1.DBGrid1ColExit(Sender: TObject);
begin
          //원래대로...
  DBGrid1.Columns[DBGrid1.SelectedIndex].Title.Font.Color := clBlack;
  DBGrid1.Columns[DBGrid1.SelectedIndex].Title.Font.Style :=
  DBGrid1.Columns[DBGrid1.SelectedIndex].Title.Font.Style - [fsBold];
  DBGrid1.Repaint;
end;
```

- DBGrid 내에서 메모 필드의 내용을 보려면 ?

OnDrawCell 이벤트를 사용하면 된다. 물론, 메모 필드를 보여주기 위해서는 TMemoField 객체를 미리 생성해야 한다.

```
procedure TForm1.DBGrid1DrawDataCell(Sender: TObject; const Rect: TRect;
  Field: TField; State: TGridDrawState);
var
  P: array [0..50] of char;
  BS: TBlobStream;
  S: String;
```

```
begin
  If Field is TMemoField then
  begin
    with (Sender as TDBGrid).Canvas do
    begin
      BS := TBlobStream.Create(Table1Notes, bmRead);
      FillChar(P,SizeOf(P),#0);
      BS.Read(P, 50);                    // 50 크기만큼 읽어 들이기.. P 의 배열 크기만큼..
      BS.Free;
      S := StrPas(P);
      while Pos(#13, S) > 0 do S[Pos(#13, S)] := ' ';       //리턴값을 뺀다.
      while Pos(#10, S) > 0 do S[Pos(#10, S)] := ' ';       //LineFeed 값을 뺀다.
      FillRect(Rect);                                       //출력한 값을 초기화한 다음
      TextOut(Rect.Left, Rect.Top, S);                      //메모의 내용을 출력한다.
    end;
  end;
end;
```

- TBlobField 에서 32KB 밖에 데이터를 읽고 쓰지 못하면 ?

이 경우에는 TQuery 의 RequestLive 가 False 인 경우 발생한다. 이때에 데이터를 올바르게 읽고 쓰기 위해서는 BDE 관리자에서 사용하고 있는 데이터베이스 앨리어스에서 BLOB Size 를 실제로 사용되는 데이터 사이즈보다 크게 설정해야 한다.
문제는 BDE 관리자의 환경 설정에서 이미 BLOB 설정이 되어서 만들어진 기존의 앨리어스는 수정된 BLOB Size 가 변경된다고 반영되지 않는 점을 주의해야 한다.

- DBGrid 에서 스크롤 바를 없애는 방법

```
procedure TForm1.DBGrid1DrawColumnCell(Sender: TObject; const Rect: TRect;
  Field: TField; State: TGridDrawState);
begin
  ShowScrollbar(DBGrid1.handle, SB_VERT, False);
end;
```

- DBGrid 입력시에 수치만 입력받게 하는 방법

```
procedure TForm1.DBGrid1KeyPress(Sender: TObject; var Key: Char);
begin
  if DBGrid1.SelectedField.FieldName = 'NO_FIELD' then
  begin
    if Key > #32 then
      if (not (Key in ['0'..'9'])) then Key := #0;
  end;
end;
```

- DBGrid 에서 특정 컬럼만 이동하게 하는 방법

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  TDrawGrid(dbgrid1).FixedCols := 3;              //세번째 컬럼을 고정
  TDrawGrid(dbgrid1).FixedRows := 1;              //첫번째 컬럼을 고정
  TDrawGrid(dbgrid1).FixedColor := clYellow;      //고정 컬럼의 색
end;
```

- RecordCount 프로퍼티의 사용

TQuery 를 사용할 경우에는 RecordCount 는 대다수 오동작을 수행한다. 그러므로, 사용을 자제하는 것이 좋다. 차라리, 첫번째 레코드가 EOF 인지 검사하는 것이 훨씬 동작속도가 빠르다.

정 리 (Summary)

이번 장에서는 데이터베이스 어플리케이션을 개발하면서 유용하게 사용될 수 있는 유용한 팁들에 대해서 다루어 보았다. 여기에 다루지 못한 내용 들이 매우 많지만, 인터넷 등을 뒤져보면 도움이 되는 내용을 많이 찾을 수 있을 것이다.