데이터베이스 어플리케이션과 SQL

이번 장에서는 델파이에서 지원하는 기본적인 컴포넌트를 이용하여 프로그래밍하는 방법과 SQL 문장을 사용하여 데이터를 조작하는 방법에 대하여 알아보겠다. 앞에서도 간단히 언급한 바 있지만 필자는 데이터 인식 컴포넌트를 그다지 좋아하지는 않는다.
필자가 아는 몇몇 프로그래머들은 기본적으로 델파이에서 지원하는 DBLabel 이나 DBEdit 와 같이 간단한 컨트롤을 사용하는 것조차도 싫어한다.
델파이로 DB 프로그래밍을 하는 방법에는 1, 2, 멀티-tier 환경의 DB 프로그래밍으로 나누어 볼 수 있다. 이들의 차이는 DB 를 어떤 방법으로 접근하느냐에 달려있다. 하지만, 기본적인 비즈니스 규칙을 적용하는 방법은 동일한데, 보통 다음의 5 가지를 고려하면 된다.

1. 자료를 데이터베이스에서 읽어오는 방법
2. 기본 데이터베이스 관련 유틸리티 사용방법
3. 자료 처리시의 에러처리 방법
4. 자료를 화면에 표시하는 방법
5. 사용자의 요구에 따라 프린터로 출력하는 방법

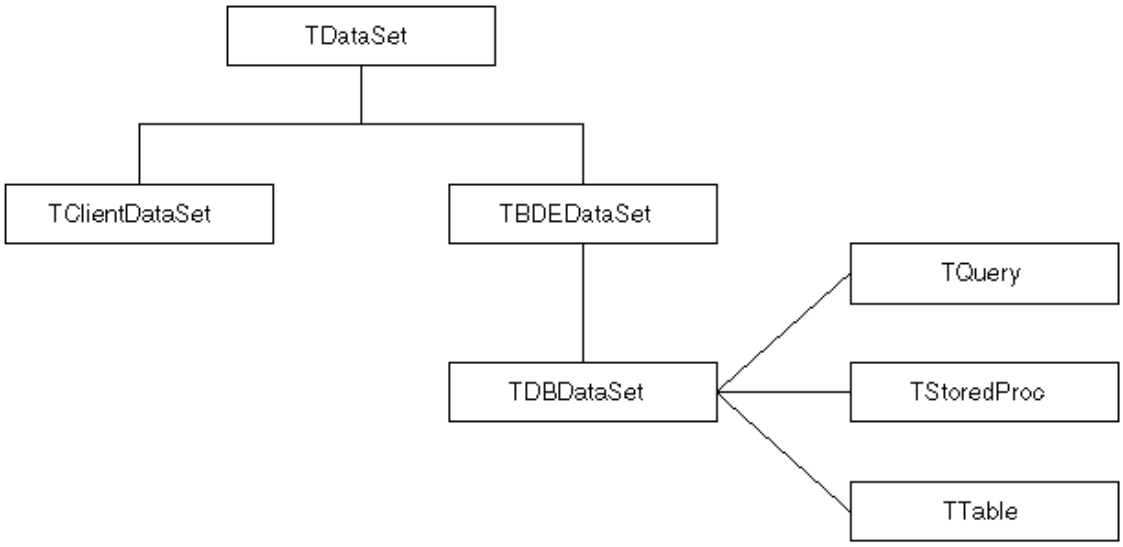
이 내용을 바탕으로 하여 이번 장을 진행하도록 한다. 이 중에서 데이터베이스 관련 유틸리티의 사용방법에 대해서는 앞 장에서 다룬 바 있으므로 이를 참고하기 바란다.

자료를 데이터베이스에서 읽어오는 방법

자료를 읽어오는 대상이 되는 것은 기본적으로 TDataSet 의 구조가 기본이 된다. 그러므로, TDataSet 의 내용을 파악할 필요가 있다.

- TDataSet 컴포넌트

데이터 세트란 결과적으로 델파이에서 지원하는 관계형 데이터베이스의 테이블(열과 행으로 구분되는)을 지칭하는 것이다. 델파이의 Data Access 팔레트에서 다양한 데이터 세트 컴포넌트를 만날 수 있는데, 이러한 컴포넌트 들은 결국 관계형 데이터베이스의 테이블을 읽어오기 위한 것이며, 근본적인 차이는 없다.
가장 기본적인 데이터 세트의 계층도를 살펴보면 다음과 같다.



이 계층도를 살펴보면 TDataSet 클래스는 모든 테이블의 상위에 존재한다. 델파이에서 사용되는 모든 데이터 세트 관련 컴포넌트는 다음과 같은 기능을 지원한다.

구 분	프로퍼티/메소드
기본 기능	Active, Open, Close
검색 기능	First, Last, Next, Prior, MoveBy, Refresh, EOF, BOF, and, IsEmpty
편집 기능	Edit, Insert, InsertRecord, Append, AppendRecord, Delete, Post, Cancel, Midufied, State, ChechBrowseMode, SetFields
북마크 기능	BookmarkValid, CompareBookmarks, GotoBookmark, FreeBookmark, GetBookmark, Bookmark
데이터 컨트롤 지원 기능	ControlsDisabled, DIsableControls, EnableControls, ISLinkedTo
이벤트	BeforeOpen, AfterOpen, BeforeClose, AfterClose, BeforsInsert, AfterInsert, BeforeEdit, AfterEdit, BeforePost, AfterPost, BeforeCancel, AfterCancel, BeforeDelete, AfterDelete, BeforsScroll, AfterScroll, onCalcFiels, onDeleteError, onEditError, onNewRecord, onPostError
필드지원 기능	FieldByname, FieldByNumber, FieldDefs, FieldCount, Fields, FIeldValue, DefaultFields, FindField, GetFieldList, GetFieldNames, UpdateRecord, ClearFields
하위레벨지원 기능	ActiveBuffer, CursorPosChanged, GetCurrenRecord, Translate, RecordSize

- TDataSet 컴포넌트의 주요 기능

TDataSet 클래스의 주요한 기능에 대해서 그룹을 나누어 알아보도록 하자.

1. Active, Open, Close

실제로 Active := True 와 Open 문장은 동일한 동작을 수행한다. 이렇게 한 이유는 어플리케이션의 디자인 시와 런타임 환경에서 동일한 동작을 수행하기 위한 하나의 방편이다. 그에 비해, TQuery 의 형태인 경우에는 Open 문으로 수행하지 못하는 몇가지 작업이 있다. TQuery 에는 ExecSQL 이라는 메소드가 있는데, 이 메소드는 데이터를 조작하는 문장이 Insert 문이나 Delete, Update 문인 경우에 ExecSQL 을 사용한다. 이 경우 수행되는 내용을 Active 프로퍼티를 통하여 검사할 수 있다.

2. TDataSet 의 조작

이번에는 데이터를 검색하고 환경을 조절하는 방법에 대해서 알아보자.
First, Next, Prior, MoveBy(수치)는 레코드의 행을 이동하는 메소드이다. 그리고 BOF 와 EOF 는 레코드의 처음과 끝을 참조하기 위한 Boolean 형태의 프로퍼티이다.
해당 레코드의 필드를 조작하기 위해서는 다음과 같은 프로퍼티를 이용한다.

```
테이블.Fields[n].AsString
테이블.FieldByName( '필드명' ).AsString;
테이블.FieldValues['필드명']
테이블필드명.Value (이 경우는 필드 에디터에서 필요한 필드를 추가해야 사용할 수 있다)
```

이런 4 가지 표현방식은 모두 동일하다. 개발자는 어떠한 방식을 사용하여도 좋으나 가장 무난한 방법은 FieldByName 을 사용하는 것이다.
Insert, Edit, Post, Delete, Cancel 메소드 들은 데이터를 추가하거나 삭제하고 취소하는 메소드 들이다. 이들 메소드는 자료를 추가하고 자료를 변경하고 자료를 지우는데 사용되는데, 이 작업은 데이터 세트의 커서의 위치에서 이루어지는데 커서를 이동하는 방법은 위에서 설명한 레코드 행을 이동하는 방법(Insert, Edit, Post, Delete, Cancel 등)을 사용하면 된다.

3. 자료를 읽어오는 방법

이제 데이터 세트의 기본적인 내용을 살펴보았다. 그러면, TDataSet 에서 파생된 TTable 과 TQuery, TStoredProc 컴포넌트의 기능에 대해서 알아보자.
TQuery 와 TStoredProc 컴포넌트는 거의 동일한 기능을 수행하며, TQuery 와 TTable 의

차이점은 테이블을 보는 관점이 다르다는 것이다. TQuery 컴포넌트에서는 테이블을 바라보는 정의를 SQL 문장으로 기술하며 사용자가 원할 경우에 사용자가 원하는 시점을 사용하기 위해 SQL 문장의 다양한 기능을 이용하여 Join 이나 View 등을 통하여 2 차원적인 테이블을 그때그때 만들어 낼 때 많이 사용한다. TTable 의 경우에는 테이블의 형태가 고정되어 있는 경우가 많거나, 마스터/디테일 기능을 이용할 경우에 많이 사용된다.
델파이에서 데이터베이스를 관리할 때에는 앨리어스를 많이 사용하는데, 이 기능은 여러 RDBMS 서버에서 유래된 것으로, 데이터베이스라고 하는 데이터 공간을 만들어 고유의 명칭을 부여한 다음 해당 데이터공간에 원하는 테이블을 생성하는 방법으로 데이터를 관리하는 것이다. 이 방식은 하나의 목표로 사용되는 테이블 들의 구분과 물리적인 위치를 고정함으로써 데이터베이스의 성능을 향상하는 경우에 사용되었다. MS-SQL 서버인 경우에는 데이터베이스의 크기를 고정하고 단일의 파일을 생성하여 사용한다. 인터베이스의 경우에도 단일 파일로 만들어진 데이터베이스를 사용한다.
그에 비해, 파라독스나 디베이스 등의 경우 많은 테이블이 존재할 수 있고, 이러한 테이블을 단일 형태로 관리할 수 있는 방법으로 이러한 앨리어스를 사용하게 된다. 그에 비해, 엑세스의 경우에는 하나의 앨리어스가 하나의 파일로 존재한다.
델파이에서는 이러한 데이터베이스를 관리하기 위한 방법으로 TDatabase 컴포넌트와 TSession 컴포넌트를 지원한다. 이 컴포넌트 들은 앨리어스를 관리하며 해당 데이터베이스에 접근하기 위해 로그인 기능도 부여한다. 일반적으로 TDatabase 컴포넌트를 이용하여 앨리어스에 접근하고 가상 데이터베이스의 이름을 부여한 다음, 이 가상 데이터베이스 컴포넌트의 이름을 통하여 지정된 데이터 세트 컴포넌트의 데이터베이스 테이블에 접근할 수 있다.

4. 데이터의 검색

테이블은 방대한 자료를 가지고 있으며 기본적인 이동명령어인 First, Next, Prior, MoveBy 문장을 사용하여 자료를 이동하며 각각의 필드에 있는 값을 읽어 낼 수 있다. 하지만 테이블에 많은 데이터가 쌓이므로 이러한 이동 명령어만 이용하여 차례로 비교하는 방법을 사용하면 비효율적이 아닐 수 없다.
이럴 때에는 각각의 테이블에 인덱스를 사용하는데, 다음의 방법은 인덱스가 존재한다면 델파이 내에서 데이터를 검색하는 가장 빠른 방법이다.

```
Table1.SetRangerStart;
Table1.FieldByName('필드명').AsString := '검색값';
Table1.SetRengeEnd;
Table1.FieldByName('필드명').AsStrign := '검색값';
Table1.ApplyRange;
```

이 문장들에 의해 필드명의 시작 값과 끝 값을 지정하면, 해당 테이블의 인덱스를 참조하여 빠르게 원하는 레코드들을 배열한다. 인덱스가 존재한다면 가장 빠른 방법이다. 그렇지만 이 방법은 TTable 에서만 적용되는 방법이다.
다른 방법으로는 Filter 를 사용할 수 있다. 이 방법은 TTable 과 TQuery 에서 모두 사용이 가능하다.

```
Table1.Filtered := True;
Table1.Filter := '(필드명 >= 값 ) and ( 필드명 <= 값 )';
```

이렇게 필터기능을 사용한 다음 FindFirst, FIndLast, FindNext, FIndPrior 을 사용하면 원하는 레코드를 찾을 수 있다.
또한, 인덱스가 지원되는 TTable 이나 TClientDataSet 와 같은 컴포넌트에서는 다음과 같이 EditKey, FindKey, FindNearest, GotoKey, GotoNearest, SetKey 메소드를 사용하여 레코드를 검색할 수도 있다.

```
Table1.SetKey;
Table1.FieldByname('필드명').AsString := '검색값';
if not Table1.GotoKey then ShowMessage('찾지못함');
```

Table1.GotoKey 메소드를 사용하면 해당 값을 가진 레코드로 이동하며, 값이 존재하지 않는다면 False 가 반환된다. 참고로, GotoNearest 메소드를 사용하면 해당 값과 가장 근사치인 값의 레코드로 이동한다. FindKyt 와 FindNearest 의 경우는 Setkey 와 GotoKey, Nearest 를 합쳐놓은 것과 동일하다.
TQuery 컴포넌트를 사용할 경우에는 생각외로 간단하다. 해당 SQL 문장을 다음과 같이 기술하면 된다.

```
with Query1 do
beign
  Active := False;
  Sql.Clear;
  Sql.Add( 'select * from 테이블명 where 필드명 조건 값' );
  Active := True 또는 Open 또는 ExecSQL;
end;
```

TQuery 나 기타 SQL 을 사용하는 컴포넌트를 사용할 경우 Active 나 Open/Close 와

ExecSQL 의 차이점은 단순히 데이터를 표시하는 것인지, 아니면 데이터를 조작한 후에 데이터를 리턴하느냐에 달렸다. SQL 문장 중에 데이터를 조작하는 INSERT, DELETE, UPDATE 문장의 경우 ExecSQL 을 사용하여야 한다. 그리고 SELECT 문장의 경우 원하는 데이터를 리턴 받아야 하므로 일반적인 TTable 과 동일하게 Active, Open/Close 를 사용한다.

5. Locate 메소드의 사용

Locate 는 커서를 찾는 범주에 맞는 첫번째 행에 위치시킨다. 간단하게 찾을 컬럼의 이름과 비교할 필드 값, 옵션을 넘겨주면 된다. 예를 들어 다음의 코드는 CustTable 의 Company 컬럼의 값이 "Professional Divers, Ltd."인 첫번째 행에 커서를 위치시킨다.

```
var
  LocateSuccess: Boolean;
  SearchOptions: TLocateOptions;
begin
  SearchOptions := [loPartialKey];
  LocateSuccess := CustTable.Locate('Company', 'Professional Divers, Ltd.',
    SearchOptions);
end;
```

Locate 메소드가 해당되는 레코드를 찾으면 그 첫번째 레코드가 현재 레코드가 되며, 리턴값으로 True 가 넘어온다. 해당되는 레코드가 없으면 현재 레코드는 바뀌지 않으며 리턴값으로 False 가 넘어온다.
Locate 메소드는 여러 개의 컬럼과 여러 개의 값들을 비교할 때 유용하다. 검색 값이 가변형(variant)이기 때문에 여러가지 데이터 형을 이용할 수 있다. 여러 개의 컬럼을 지정해서 검색하려면 각각의 아이템들을 세미콜론으로 분리한다.
검색 값이 가변형이기 때문에 여러 개의 값을 넘길 때에는 가변형 배열을 이용하거나, VarArrayOf 함수를 이용해서 가변형 배열을 만들어야 한다. 다음 문장은 여러 컬럼에 여러 값을 이용해 검색을 하는 예이다.

```
with CustTable do
 Locate('Company;Contact;Phone', VarArrayOf(['Sight Diver','P']), loPartialKey);
```

6. Lookup 메소드의 사용

Lookup 메소드는 검색 조건에 맞는 첫번째 행을 찾아서 데이터 세트와 관련된 lookup 필드와 calculated 필드의 재계산까지 실행하고, 하나 이상의 필드의 값을 가변형으로 돌려준다. Lookup 메소드는 Locate 메소드와 달리 커서를 움직이지 않고, 값만을 돌려준다.
검색할 컬럼과 필드 값을 넘겨주면 되는데, 예를 들어 다음의 코드는 CustTable 의 Company 컬럼의 값이 "Professional Divers, Ltd."인 회사 이름, 담당자, 전화번호를 돌려준다.

```
var
  LookupResults: Variant;
begin
  with CustTable do
    LookupResults := Lookup('Company', 'Professional Divers, Ltd.', 'Company;
        Contact; Phone');
end;
```

Lookup 메소드는 지정된 필드에서 조건에 맞는 첫번째 레코드의 값을 돌려준다. 하나 이상의 값을 요구할 때에는 가변형 배열(variant array)로 돌려준다. 조건에 맞는 레코드가 없으면 Null 값을 돌려준다.
Lookup 메소드도 Locate 메소드와 마찬가지로 여러 개의 컬럼과 여러 개의 값들을 비교할 수 있다. 다음 문장은 여러 컬럼에 여러 값을 이용해 검색을 하는 예이다.

```
var
  LookupResults: Variant;
begin
  with CustTable do
LookupResults := Lookup('Company; City', VarArrayOf(['Sight Diver',
                    'Christiansted']), 'Company; Addr1; Addr2; State; Zip');
end;
```

- 레코드의 표시와 복귀

특정 레코드를 표시하면 원할 때에 그 레코드로 바로 돌아갈 수 있게 된다. 이런 기능을 지원하기 위해 TDataSet 와 그 자손들은 북마크를 지원한다. 북마크 기능은 Bookmark 프로퍼티와 5 개의 메소드로 구현된다.
Bookmark 프로퍼티는 어플리케이션에 있는 북마크 중 현재의 북마크를 가리킨다. 북마크를 하나 추가할 때마다 그것이 현재의 북마크가 된다.

TDataSet 은 가상 북마크 메소드를 구현한다. TBDEDataSet 에서 북마크 메소드를 다시 구현했는데, 이들은 다음과 같다.

1. BookmarkValid: 지정된 북마크가 쓰이고 있는지 알아본다.
2. CompareBookmarks: 2 개의 북마크가 같은 것인지 비교한다.
3. GetBookmark: 현재의 위치를 북마크에 할당한다.
4. GotoBookmark: 이전에 GetBookmark 에 의해 생성된 북마크로 돌아간다.
5. FreeBookmark: 이전에 GetBookmark 에 의해 할당된 북마크를 해제한다.

북마크를 만들려면 먼저 일종의 포인터 변수인 TBookmark 형의 변수를 선언하고, GetBookmark 를 호출하여 데이터 세트의 특정 위치를 값으로 할당하면 된다.
GotoBookmark 를 호출하여 특정 레코드로 이동하기 전에 BookmarkValid 를 호출하여 북마크가 레코드를 가리키고 있는지 확인할 수 있다. 북마크가 레코드를 가리키고 있으면 True 가 돌아온다.
다음은 북마크의 사용 예이다.

```
procedure DoSomething (const Tbl: TTable)
var
  Bookmark: TBookmark;
begin
  Bookmark := Tbl.GetBookmark; {변수에 메모리와 값을 할당}
  Tbl.DisableControls;
  try
    Tbl.First;
    while not Tbl.EOF do
    begin
      {여기에 작업할 코드를 쓴다.}
     …
      Tbl.Next;
    end;
  finally
    Tbl.GotoBookmark(Bookmark);
    Tbl.EnableControls;
    Tbl.FreeBookmark(Bookmark); {북마크에 할당된 메모리 해제}
  end;
end;
```

- SQL 에 대하여 ...

세부적인 SQL 문장을 사용하기 전에 델파이에서 지원하는 인터베이스를 살펴보자, 델파이 C/S 버전에서는 C/S 프로그램을 제작하기 편하도록 테스트용 인터베이스 서버가 같이 지원된다. 인터베이스는 다른 RDB 와는 다르게 리모트나 로컬의 DB 를 접근할 경우에 앨리어스에서 해당 데이터베이스를 지정하기 위해서 컴퓨터이름과 해당 디렉토리를 모두 기술 하여 주어야 한다. 이것이 불편하게 느껴질 수도 있지만, 물리적인 데이터베이스를 관리하기가 편하므로 유닉스나 윈도우 NT 의 경우 하드가 모자라서 확장할 경우에 데이터베이스를 간단하게 이동을 한 뒤에 해당 BDE 세팅에서 위치만 변경해주면 변경이 쉽게 된다는 장점도 있다.
로컬 인터베이스의 기본적인 User Name 은 SYSDBA, Password 는 masterkey 이다.
인터베이스 그룹의 인터베이스 서버 관리자(Server Manager) 프로그램은 인터베이스의 전반적인 데이터베이스를 다루는 DBA 의 입장에서 인터베이스를 다룰 수 있도록 해준다.
File/Server Login 작업에서는 리모트와 연결 데이터베이스에 접근하는데, 여기에서 앞서 설명한 SYSDBA/masterkey 를 이용한다. 그리고, Tasks/User Security 메뉴에서 새로운 사용자를 추가하는데, 이렇게 추가된 사용자명을 사용하여 프로그램에서 인터베이스에 접근하게 된다.
윈도우 ISQL(Windows ISQL)은 콘솔 SQL(Console SQL)을 지원하는 프로그램으로 데이터베이스나 테이블을 추가/수정/삭제할 수 있는 SQL 문장으로 인터베이스를 조작할 수 있도록 해준다. File/Create Database 메뉴를 선택하면 원하는 위치에 데이터베이스를 생성할 수 있으며, File/Drop Database 메뉴를 선택하면 데이터베이스를 삭제할 수 있다.
참고로, 인터베이스는 기본적으로 gdb 라는 확장자명을 사용하며, 모든 SQL 문장을 이용한 작업을 수행하려면 Commit 명령을 사용해야 한다.
기본적인 SQL 문장의 문법은 다음과 같다.

1. 테이블 생성

create table 테이블명(필드명 속성 형태 키형태, ...)으로 기술하면 되는데 다음의 예를 살펴보자.

```
create table data (code char(5) not null primary key, name varchar(20), age integer)
```

이 코드는 data 라는 테이블을 생성하는데 기본키로는 code 라는 char(5)형태의 널(null)값이 존재하지 않는 필드를 사용하고, 그 밖에 name 이라는 varchar 20 자리, age 라는 정수형 필드를 가지는 테이블을 만들려는 SQL 문장이다. 이런 작업은 SQL 문장을 이용하여

수행할 수도 있고, 데이터베이스 데스크탑을 이용할 수도 있다.

2. 레코드 추가

insert into 테이블명 (필드값) values (넣는 값)

3. 레코드 수정

update from 테이블명 set 필드명 = 값 where 조건

4. 레코드 삭제

Delete from 테이블명 where 조건

해당 테이블에 해당 조건에 맞는 레코드를 삭제한다.

5. 레코드 검색

Select 선택필드 into 변수명 또는 레코드명 from 테이블 where 조건

선택필드에서 * 가 기술되면 전체 필드를 지칭하며 필요한 필드명을 나열하면 해당 필드로 구성된 테이블 뷰를 만들게 된다. 이때에 필드의 개수가 적을수록 해당 SQL 문장의 동작 속도는 빨라지게 된다.
참고로 TQuery 컴포넌트에는 Prepare 라는 메소드가 존재하는데, 이 메소드는 SQL 문장을 수행하기 전에 해당 문장을 최적으로 수행할 수 있는 환경을 구성한다. 이 문장은 꼭 사용하는 것이 좋다.
테이블 절에는 여러 개의 테이블을 기술할 수 있는데, 다음과 같이 사용할 수 있다.

Select 테이블명 A.필드명, 테이블명 B.필드명 from 테이블명 A, 테이블명 B

이렇게 하면, 두개 이상의 파일을 Join 하여 하나의 뷰로 볼 수 있도록 하여 준다.

6. Like 연산자의 활용

앞에서 언급한 SQL 문장의 조건절에는 다음의 Like 연산자를 사용할 수 있다. Like 연산자는 해당 필드에 속해있는 값들 중에 비슷한 값을 찾아내는 연산자로 도스 명령어를 사용할

때의 *(와일드카드)와 비슷하다.
SQL 문장 내에 %를 사용하게되면 Like 연산자가 동작하게 된다.

Select * from 테이블 where 필드 like '신%';

이 문장은 해당 필드에 속해 있는 값들 중에 '신'으로 시작되는 데이터를 찾을 것이다.

7. Order By 절의 활용

OrderBy 절은 레코드를 정렬하는 경우에 사용한다.

Select * from 테이블 order by 필드명

오름차순과 내림차순은 Asce 와 Desc 를 해당 필드명 뒤에 붙여주면 해당 형식으로 정렬된다.

8. Group By 절의 활용

선택된 필드를 그룹단위로 묶어서 중복된 것만 보여준다.

Select 필드값, sum(필드값) "합계", avg(필드값) "평균" from 테이블명 group by 테이블명

이렇게 하면 해당 테이블명이 중복된 값의 합계와 평균을 출력한다.

SQL 문장에 대해서 자세히 알기 위해서는 별도로 RDB 와 SQL 에 대해서 다루고 있는 서적을 한권 정도 참고하기를 권하고 싶다. SQL 문장은 기본적으로 표준화가 되어 있기 때문에, 이를 익혀두면 많은 부분에서 도움이 될 것이다. 그 밖에 몇 가지 유용한 SQL 문장에 대해서는 다음 장에서도 소개할 것이다.

- 저장 프로시저(Stored Procedure)의 사용방법

저장 프로시저는 일련의 SQL 문장으로 이루어진 서브루틴과 동일하다. 2-tier 이상의 프로그램을 제작하는 경우에는 많은 테이블을 Open 하여 반복적인 작업을 수행하는 경우가 빈번한데, 해당 테이블마다 SQL 문장을 통하여 Open 한 다음 클라이언트에서 작업을 하게 되면 비효율적인 네트워크 트래픽을 비롯하여 여러가지 자원의 낭비가 발생하게 된다.

이럴 때에는 저장 프로시저를 사용하여 해당 문장을 기술할 수가 있다. 그 밖에도 고정된 SQL 문장으로 클라이언트의 요구를 파라미터로 받아들여서 일반적인 서브루틴처럼 작동하는 모듈을 구성할 경우에도 이와 같은 저장 프로시저를 사용할 수 있습니다.
다음은 저장 프로시저를 만드는 예제 코드이다.

```
Create Procedure Insert_data ( PCode varchar(4), pname varchar(20), pag smallint )
as
begin
      begin
            insert into data values ( :pcode, :pname, :page );
      end
      suspend;
end
```

이런 문장을 수행하면 해당 데이터베이스에 Insert_data 라는 저장 프로시저가 생성된다. 이제 델파이에서 이를 사용하려면 TStoredProc 컴포넌트를 사용하여 호출하면 된다. 사용하는 방법은 다음과 같다.

```
with StoredProc1 do
begin
  StoredProcName := 'insert_data';
  ParamByName( 'pcode' ).AsString := '값';
  ParamByName( 'pname' ).AsString := '값';
  ParamByName( 'page' ).AsInteger := 값;
  ExecProc;
end;
```

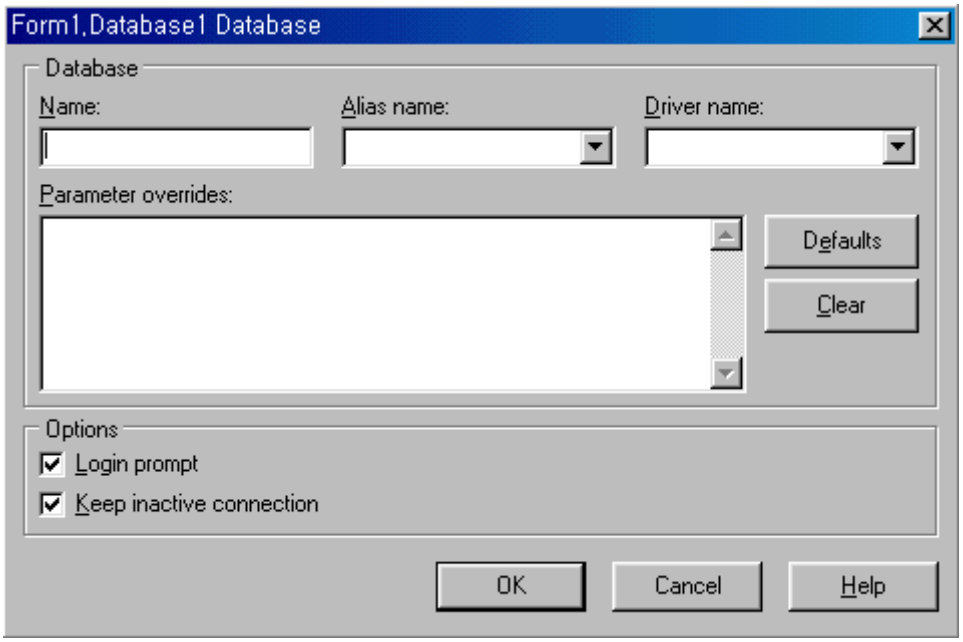
- TDatabase 컴포넌트의 활용

12 장에서도 이 컴포넌트에 대해서는 언급한 바 있지만, 많이 사용되는 컴포넌트인 만큼 이번 장에서도 사용하는 방법에 대해서 간단하게 알아보도록 하자.

1. 동적인 앨리어스의 생성

TDatabase 컴포넌트를 사용하면 존재하는 앨리어스를 사용하여 접근하거나, 또는 실행환경에서 하나씩 BDE 에 앨리어스를 생성시켜주는 불편을 사용하지 않고 수행 중에만 존재하는

앨리어스를 생성하여 사용할 수 있게 해준다.
다음 그림은 TDatabase 컴포넌트를 선택하고, 팝업 메뉴에서 Database Editor ... 이라는 메뉴를 선택하면 나타나는 대화상자이다.



여기에서 Parameter overrides 부분에는 해당 데이터베이스에 접근하기 위한 다양한 값들을 설정할 수 있다. Defaults 버튼를 클릭하면 기본적인 정보가 나타난다. Login Prompt 체크 박스를 설정하면, 사용자에게 UserName 과 Password 를 물어보는 대화 상자를 생략하게 할 수 있다. 또한, Keep inactive connection 체크 박스를 설정하면 데이터베이스를 열 때마다 사용자 이름과 암호의 입력작업을 중복할 필요가 없게 해 준다.

2. 트랜잭션의 관리

TDatabase 컴포넌트를 사용하는 가장 커다란 잇점이 바로 이것이다. 보통 C/S 프로그램을 제작하는 경우에 초보자들이 가장 어렵게 생각하는 부분이 바로 트랜잭션의 관리이다. 트랜잭션이란 SQL 문장으로 기술된 논리적인 하나의 작업 단위를 뜻하는데, 원격 DB 를 사용하는 경우 중간의 네트워크 때문에 자료의 비정상적인 전송이나 오류가 발생하는 경우가 빈번하다. 특히, 테이블이 하나가 아닌, 2~3 개 이상의 테이블의 값을 수정하거나 추가하는 경우에는 데이터의 무결성에 치명적인 문제가 발생할 수 있기 때문에, 이런 트랜잭션이 필수적이라고 할 수 있다.
3 개의 테이블을 수정하는데, 2 개의 테이블만 수정되고 1 개의 테이블의 값이 수정되지 않는다면 이 데이터는 나중에 치명적인 문제를 발생할 수 있다. 그렇기 때문에, 이럴 때에는 이런 수정작업을 하나로 묶어서 사용해야 하는데, 이를 트랜잭션이라 한다.

이렇게 트랜잭션을 적절하게 사용하면 데이터의 하드웨어적인 문제의 일관성(Consistency)과 다중 사용자 접근에 대한 무결성(Integrity)을 보장해 줍니다.

3. 트랜잭션의 관리 방법

트랜잭션의 관리 방법에는 크게 나누어 다음과 같은 2 가지가 있다.

- 기본제어

델파이는 기본적으로 트랜잭션을 기초적인 단계에서 수행한다. Post 나 AppendRecord 등과 같은 메소드에서 자동적으로 해당 트랜잭션을 수행하고 commit 하게 된다. 문제는 이런 기본적인 제어방법은 추가/수정/삭제 작업을 할 때의 문제 발생은 막을 수 있지만, 네트워크 트래픽이나 여러 개의 작업 단위로 이루어진 문장에서는 안전성을 보장할 수 없다.

- 개발자가 지정하는 제어방법

메소드	설 명
StartTransation	트랜잭션의 시작 포인트를 설정한다.
Commit	진행된 트랜잭션을 물리적인 데이터베이스에 적용한다.
RollBack	현재 진행된 트랜잭션을 원상태로 복귀시킨다.

참고로, 델파이 4 에서는 로컬 DB 인 파라독스에서도 이러한 트랜잭션을 지원한다.

4. 트랜잭션의 옵션과 지원되는 DB 에서의 동작

델파이에서는 다음과 같은 작업을 통하여 작업을 수행한다.

```
TDatabase.StartTransaction;        // 트랜잭션의 시작을 지시한다.
TDatabase.Commit;                  // 지금까지의 작업을 실제 DB 에 반영한다.
TDatabase.Rollback;                // 지금까지 작업내용을 취소한다.
```

StartTransations 메소드와 Commit, Rollback 은 try ... finally, try ... except 블록을 활용하여 작업을 수행한 다음의 결과에 대해서 작업을 수행할 수 있도록 하는 것이 좋다.
inTransacion 이라는 Boolean 형의 프로퍼티는 실제 트랜잭션의 작업 시에 True 가 선언되면 StartTransation 과 동일한 역할을 하며, Commint 이나 Rollback 과 같은 기능을 하기 위해서는 False 를 설정하면 된다.

트랜잭션에는 3 가지 방식이 있는데 다음과 같다.

레 벨	내 용
TiDirtyRead	현재 작업과 관계없는 내용은 다른 사람들이 읽을 수 있다.
TiReadCommotted	현재 작업이 다른 트랜잭션에서의 연산에 참여할 수 있다.
TiRepeatableRead	현재 작업중인 데이터에 대해 다른 트랜잭션이 참여할 수 없다.

약간의 문제는 이러한 트랜잭션이 각각의 RDBMS 마다 다르게 동작한다는 것이다. 다음의 표는 이러한 차이점에 대해서 나열해 보았다.

서 버	레벨 설정	실제 동작
Oracle	tiDirtyRead	tiReadCommitted tiReadCommitted
	tiReadCommitted	tiRepeatableRead(ReadOnly)
	tiRepeatableRead	
Sybase, MS-SQL	tiDirtyRead	tiReadCommitted tiReadCommitted
	tiReadCommitted	tiRepeatableRead
	tiRepeatableRead	
DB2	tiDirtyRead	tiDirtyRead
	tiReadCommitted	tiReadCommitted
	tiRepeatableRead	tiRepeatableRead
Informix	tiDirtyRead	tiDirtyRead
	tiReadCommitted	tiReadCommitted
	tiRepeatableRead	tiRepeatableRead
인터베이스	tiDirtyRead	tiReadCommitted tiReadCommitted
	tiReadCommitted	tiRepeatableRead
	tiRepeatableRead	
Paradox, dBase, Access, FoxPro	tiDirtyRead	tiDirtyRead
	tiReadCommitted	지원하지 않음
	tiRepeatableRead	지원하지 않음

여기에서 보면 알겠지만 로컬 DB 의 경우 tiDirtyRead 의 옵션 밖에 차이가 나지 않는다. 이 옵션은 ODBC 인 경우에도 동일하다.

5. 트랜잭션의 종류

BDE 설정에서도 해당 데이터베이스의 트랜잭션 동작에 대한 기본설정을 할 수 있다. 이것

이 바로 SQLPASSTHRU MODE 인데, 다음의 3 가지로 지원된다.

모 드	설 명
SHARED AUTOCOMMIT	기본적인 설정으로 레코드(행)단위의 모든 작업에 Commit 을 자동 작동시킨다. SQL 과 BDE 메소드가 동일하게 동작한다.
SHARED NOAUTOCOMMIT	프로그램 내부에서 정확하게 트랜잭션이 기술된 시점에서 트랜잭션을 수행한다. SQL 과 BDE 메소드가 동일하게 동작한다.
NOT SHARED	완전히 별개의 트랜잭션을 기술하며 SQL 과 BDE 는 다른 연결방법을 사용한다.

- 로컬 DB 의 원격 DB 로의 이동

일반적으로 작성된 파라독스나 DBase 형태의 DB 를 상용 DBMS 의 데이터로 전송하는 경우에 해당 프로그램을 작성할 필요 없이 간단한 방법으로 이러한 데이터를 전송할 수 있다. 이 경우에는 개발자는 앨리어스를 2 개 준비하여 Source 에서 Target 으로 데이터를 전송하기만 하면 된다.
이 때 사용하는 것이 바로 데이터 이동 위저드(Data Migration Wizard)인데, 주의할 점은 데이터베이스의 형태가 바뀌므로 전송에 앞서 다음과 같은 몇 가지 사항을 고려해야 한다.

1. 각각의 필드의 속성이 Target 에서 지원되는지 살펴 본다.
2. 필드명중에 해당 DB 의 예약어인 경우가 있으므로, 필드명의 적절하게 변경한다.

필자의 경험상 자료를 올리는 경우에는 인덱스를 일단 모두 삭제한 다음 Upsize 하는 것이 성공확률이 높다.

- TDataSource 의 사용 방법

TDataSource 컴포넌트는 단순히 데이터 세트 컴포넌트와 데이터 인식 컨트롤을 연결하는 연결방법으로만 사용하는 것은 않는다. 여기서는 이 컴포넌트의 몇 가지 유용한 기능에 대하여 간단하게 알아보자

1. 마스터/디테일 연결

마스터/디테일 테이블을 TDataSource 컴포넌트를 이용하여 연결할 수 있다. 이때 디테일 테이블은 반드시 TTable 을 사용해야 한다.

2. 다양한 이벤트의 활용

이벤트	설 명
onDataChange	해당 DataSet 이 변경될 경우에 발생한다.
onStateChange	해당 DataSet 의 작업형태가 변경될 경우에 발생한다.
onUpdateData	물리적인 데이터가 변경될 경우에 발생한다.

- 캐쉬 업데이트(Cached Updates)

트랜잭션과 유사한 기능을 하는 것으로, 개발자가 원하는 테이블에 변경 작업(추가/수정/삭제)작업을 취할 경우에 다중 레코드를 처리하는 것을 허용하여 중간 버퍼에 이러한 작업을 유지한 다음 원하는 시점에 이 작업을 물리적인 데이터베이스에 반영하도록 하는 것을 말한다.
이 기능은 모든 데이터 세트 컴포넌트에서 사용할 수 있는데, CachedUpdates 프로퍼티 값이 True 가 설정되어 있으면 동작이 가능하다. 사용하는 메소드는 다음과 같다.

메소드	설 명
ApplyUpdates	버퍼의 내용을 물리적인 DB 에 저장합니다.
CancelUpdates	현재 버퍼에 저장된 변경내용을 삭제합니다.
RevertRecord	현재 레코드만 원래의 레코드로 변경합니다.

참고로, DataBase1.ApplyUpdates([테이블, 테이블])과 같은 코드를 사용하면 여러 개의 테이블에 있는 캐쉬 업데이트를 하나의 코드로 동작시킬 수 있다. 그러나, CancelUpdates 메소드는 데이터베이스 컴포넌트에서 지원하지 않는다.

- TUpdateSQL 컴포넌트

TQuery, TStoredProc 컴포넌트 이외에 기본적으로 TUpdateSQL 이라는 컴포넌트가 지원된다. TQuery 컴포넌트를 이용하면 Select, Insert, Delete, Update 등의 모든 SQL 문장을 사용할 수 있으며, 심지어는 저장 프로시저나 트리거(Trigger)도 만들 수 있다. 그렇지만, 실제로 SQL 문장을 사용하다 보면 자그마한 리소스나 속도 문제에 민감해지는 경우가 많다. TQuery 의 경우 모든 SQL 문장을 동작시킬 수 있도록 많은 내부코드를 가지고 있기 때문에, 일반적인 SQL 문장을 사용하는 경우에 주로 사용되는 Insert, Update, Delete 의 3 가지 기본적인 작업만 사용할 때 이것을 사용하는 것은 다소 낭비라고 할 수 있다. 이런 경우에 보다 빠른 작업과 관리의 편리성을 위하여 TUpdateSQL 컴포넌트를 사용한다.
이 단순화된 컴포넌트는 필요한 3 가지의 행동코드를 가질 수 있어서, 매번 추가/수정/삭제

할 때마다 SQL 프로퍼티의 내용을 지웠다 추가했다 하는 코드를 줄일 수 있으므로 간단한 저장 프로시저처럼 활용할 수 있다.

- 전역 세션(Global Session)의 기능 활용

델파이에서 현재 사용하고 있는 데이터베이스의 정보나 테이블 정보를 검색해야 할 경우가 있다. 이런 경우에는 세션 컴포넌트를 사용하는데, 기본적으로 세션이 하나 전역으로 선언되어 있고, 이를 디폴트 세션으로 사용하게 된다.
이를 이용하여 Session.GetDriveNames(Strings 객체) 메소드를 사용하면 현재 사용되는 데이터베이스 드라이브 정보를 얻을 수 있으며, Session.GetDataBaseNames(Strings 객체) 메소드를 사용하면 현재 사용되는 데이터베이스명을 얻을 수 있습니다.
마찬가지로, Session.GetTableNames('데이터베이스명', 형태('*.db'),False, False, Strings 객체) 메소드를 이용하면 테이블의 이름을 얻을 수 있다.

- 북마크(Bookmark) 사용하기

북마크는 책갈피라고 생각하면 된다. 원하는 위치를 지정해 놓고 해당위치로 빠르게 이동할 수 있게 해준다.

```
북마크 := Table1.GetBookMark;          //현재의 위치를 저장한다.
Table1.GotoBookmark(북마크);           // 해당 북마크로 이동
Table1.FreeBookmark(북마크);           // 해당 북마크를 삭제
```

자료 처리시의 에러처리 방법

잘 만들어진 데이터베이스 어플리케이션은 예기치 않은 에러 상황이 발생하더라도 이를 고급스럽게 처리하는 방법을 제공해야 한다. 데이터베이스 어플리케이션에서 가장 많이 발생하는 에러는 EDatabaseError 와 EDBEngineError 이다.

- EDatabaseError

실제 DB 프로그래밍을 할 때 만나게 되는 많은 에러 상황은 EDatabaseError 클래스를 사용하여 해결할 수 있다. 이를 처리하는 방법을 pseudo-code 로 나타내면 다음과 같다.

```
try
  ... DB 작업
```

```
except
  on EDatabaseError do
  ...
  raise;
end;
```

즉, DB 작업 중에 문제가 발생할 가능성이 있는 부분을 try...excpet 블록으로 감싸고 해당 부분에서 EDataBaseError 객체로 발생한 에러를 추적하여 대처할 수 있다.
그러나, EDatabaseError 객체만 가지고는 직접적으로 어떤 오류가 발생하였는지를 알 수 없고 실제 내용은 파생된 EDBEngineError 클래스를 참조해야 알아낼 수 있다.
보통의 경우에는 데이터 세트의 OnPostError, OnEditError, OnDeleteError 의 이벤트를 참조하면 대다수 문제가 해결된다. 해당 이벤트에는 문제가 발생된 데이터 세트의 EDatabaseError 객체가 전달되며 문제를 해결하기 위해 TDataAction 데이터 형의 값을 지정하게 된다. 지정할 수 있는 값에는 daFail(실패), daAbort(중단), daRetry(재시도) 중에서 선택할 수 있다.

- EDBEngineError

EDBEngineError 클래스는 Errors 배열에 BDE 에서 발생되는 모든 오류의 값과 내용을 가지고 있다. 그렇기 때문에, 이를 이용하면 보다 구체적인 에러 처리를 할 수 있게 된다. 이 클래스의 멤버에서 ErrorsCount 에는 발생한 에러의 개수가 담겨 있고, Errors 에는 발생된 에러의 내용이 TDBError 의 배열 형태로 들어 있다. 이들의 값에는 다음과 같은 것들이 있다.

속 성	내 용
Category	오류코드 범주
ErrorCode	오류코드 숫자 (DBIResult 형태)
Message	서버에 연결되어 있으면 서버에서 리턴된 에러메시지, 보통은 BDE 의 에러메시지
NativeError	RDB 에서 발생한 에러인지 체크, 0 이 리턴되면 서버오류가 아닌 BDE 오류
SubCode	하위 오류코드

참고로, Category 와 SubCode 는 Byte 형태로 두 코드는 상위와 하위의 에러코드를 지칭하게 된다.
리턴되는 에러코드는 BDE.INT 파일을 참조하면 다양하게 존재하는 에러의 코드를 구체적으로 알 수 있다. 해당 BDE.INT 에는 Delphi4 의 DOC 디렉토리에 333kB 의 파일로 존재하는데, 실제 에러는 Byte 값으로 존재하며 DBI 상에서의 에러는 ErrorCode 의 Word 타입

의 자료를 참조하여 에러를 찾을 수 있다.

자료를 화면에 제시하는 방법

데이터베이스의 내용을 표시하기 위해서 데이터 인식 컴포넌트들을 사용하면 필요한 데이터베이스의 내용을 제시할 수 있으며 쉽게 데이터베이스의 레코드들간의 이동이 가능하다. 이렇게 데이터 인식 컴포넌트를 이용하여 데이터를 표시하는 방법은 간단히 언급한 바 있고, 델파이의 도움말에도 잘 나와 있으므로 자세한 설명은 생략하도록 한다.
필자는 앞에서도 언급했듯이 데이터 인식 컴포넌트들을 거의 사용하지 않는다. 실제 테이블이나 쿼리를 사용할 때 데이터 인식 컨트롤을 사용하면, 소규모 프로젝트에서는 큰 문제가 되지 않지만, 대규모 프로젝트인 경우에는 많은 시행착오를 겪기 쉽다. 엉뚱한 Access Violation Error 를 비롯하여 레코드의 locking 문제 등이 많이 발생하기 때문에, 필자의 경우에는 StringGrid 와 TListView, TListBox 등의 기본적인 화면용 컴포넌트를 주로 사용한다.
데이터는 TQuery 나 TStoredProc, TUpdateSQL 컴포넌트를 주로 사용한다. 물론, 이러한 방식은 델파이 2 까지 사용되던 기법이었고, 델파이 3 부터는 TClientDataSet 을 많이 활용한다. 전장에서도 충분이 이야기 했지만, TClientDataSet 컴포넌트를 사용하면 필요한 데이터만 클라이언트에서 변경하고 이렇게 변경된 내용을 Delta 프로퍼티에 보관하여 RDB 에 반영할 수 있기 때문에, 많은 잇점이 있다.
데이터 인식 컨트롤을 사용하는 방법이나 화면용 기본 컴포넌트를 이용하여 데이터를 표시하는 방법에 대해서는 특별히 언급하지 않아도 여러 곳에서 정보를 얻을 수 있으므로 여기에 대한 설명은 생략하도록 하며, 개발 시에 유용한 몇 가지 팁들은 18 장에서 소개하였다.

- TeeChart 의 활용

델파이 3 에서부터 포함된 TeeChart 는 델파이의 native VCL 로 막강한 그래프를 만들어낼 수 있는 유용한 컴포넌트 세트이다. 보통의 경우에는 델파이에서 제공되는 위저드를 이용하면 기본적인 그래프는 간단히 생성할 수 있다. 이 방법에 대해서는 기본으로 제공되는 도움말을 참고하면 쉽게 알 수 있을 것이다.

- Decision Cube 의 활용

델파이 3 에서부터 지원되는 Decision Cube 는 의사결정 도구의 결정판이다. 이 도구를 사용하면 드릴-다운(Drilled down)기법을 이용하여 각종 데이터를 조회하여 볼 수 있다. 여기에 대해서는 다음 장에서 자세히 다룰 것이다.

사용자의 요구에 따라 프린터로 출력하는 방법

프린터로 데이터를 출력하는 방법에는 여러가지 방법이 있습니다. 이를 크게 나누어 보면 다음과 같이 3 가지로 생각할 수 있다.

1. 일반적인 프린터 메소드를 사용하는 방법
2. QuickReprot 를 사용하는 방법
3. 다른 서드파티를 사용하는 방법

- 일반적인 프린터 메소드를 사용하는 방법

WriteLn 함수와 Canvas 를 사용하여 원하는 출력물을 만들어 낼 수 있다. WriteLn 의 사용방법은 다음과 같다.

WriteLn(텍스트화일, 문자열, ...);

참고로, 보통 테이블이나 쿼리를 Open 하여 놓은 상태, 특히 데이터 인식 컨트롤을 사용한 경우에는 화면에서 작업하다가 프린팅과 같은 전체 자료를 사용하여 출력하는 경우에는 Bookmark 를 사용하여 작업하던 장소를 저장하고, 테이블이나 쿼리문을 다시 동작하여 사용하면 간단하게 작업할 수 있다. 또한 데이터 인식 컨트롤을 사용할 경우 전체 내용을 검색하면 DataSource 가 연결된 경우에는 해당 데이터의 Refresh 때문에 수행이 느려진다.
이럴 때에는 DisableControls 와 EnableControls 메소드를 활용하여 활성을 조절하면 프로그램의 수행이 빨라진다.
일반적인 문자열을 출력하는 경우에는 WriteLn 도 무방하지만, 색을 지정하거나 그림이나 도형을 출력하는 경우에는 Canvas 를 사용하여 출력하여야 한다. 다음의 코드를 살펴 보자.

```
Printer.BeginDoc;
Printer.Canvas.TextOut(x, y, 문자열);
Printer.EndDoc;
```

이때 주의할 점은 x, y 의 해당 좌표값을 지정하는 것과 좌표값은 Pixel 단위이므로 해당 폰트의 Height 와 Width 를 알아서 계산해 주어야 한다는 점이다. 그리고, 반드시 EndDoc 를 호출하여 끝을 맺어야 한다. 이 부분이 빠지게 되면 무한루프를 돌 우려가 있으니 주의하기 바란다. 이외에도 NewPage, Abort, PageNumber 등을 이용하여 새로운 페이지 출력, 출력 중단, 페이지 번호 등을 알아낼 수 있다.

- 퀵 리포트(Quick Report)의 활용

퀵 리포트는 델파이 3 부터 기본적으로 제공되는 출력 컴포넌트로서 강력한 리포팅 기능을 가지고 있어서 기존의 리포트 스미스를 완전히 대체하였다. 퀵 리포트에 대한 더 자세한 내용은 다음 장에서 다루도록 한다.

정 리 (Summary)

이번 장에서는 델파이에서 지원하는 기본적인 컴포넌트를 이용하여 프로그래밍하는 방법과 기본적인 데이터베이스 어플리케이션을 작성할 때 고려해야 할 여러가지에 대해서 다루어 보았다.
다음 장에서는 이번 장에서 간단히 알아본 Decision Cube 와 퀵 리포트에 대해서 더 자세하게 알아보도록 할 것이다.