데이터베이스 어플리케이션 제작의 기초

(Basics of Database Application Development)

이번 장부터 시작되는 3 부에서는 델파이로 작성하는 어플리케이션으로, 가장 흔하면서도 중요하게 사용되고 있는 데이터베이스 어플리케이션을 제작하는 방법에 대해서 다루게 된다.
델파이가 첫 버전으로 세상에 선보일 당시에는 윈도우용 데이터베이스 어플리케이션을 제작할 수 있는 몇가지 툴이 있었다. 당시에 데이터베이스 에플리케이션을 작성하는 SQL Windows 와 델파이를 많은 곳에서 비교하였다. 필자도 초기버전과 델파이 1.0 을 비교하면서 주저없이 델파이를 선택하였다. 그 이유는 SQL Windows 는 Case 도구였고, 델파이는 볼랜드 파스칼 위에 Case 도구 엔진을 올려놓은 라이브러리의 확장판과 같았기 때문이었다.
델파이는 매우 강력한 데이터베이스 어플리케이션을 제작할 수 있는 언어이자 도구이다.
데이터베이스는 데이터를 저장하기 위한 일반적인 수단을 제공하는 것이고, 데이터베이스 엔진은 이러한 데이터베이스에 데이터를 저장하고, 읽어오고, 여러가지로 처리하는 방법을 제공한다. 델파이에서는 기존에 존재하는 데이터베이스 엔진과의 접속은 물론, 자체적으로 꽤 높은 성능의 데이터베이스 엔진을 제공하고 있다.
이번 장에서는 그 첫번째로 데이터베이스에 대한 전체적인 개념과 데이터베이스 어플리케이션의 구조 등에 대해서 알아보고, 델파이를 이용하여 데이터베이스 어플리케이션을 작성하는 간단한 방법을 익히도록 한다.
델파이의 컴포넌트 중에서 데이터베이스와 관련되어 있는 컴포넌트 팔레트의 페이지는 3 가지라고 보면 된다. 주된 것은 Data Access 페이지와 Data Controls 페이지이지만, 최근의 멀티-tier 데이터베이스 어플리케이션을 작성할 때 사용하는 MIDAS 페이지의 컴포넌트 들도 중요하다.
이 중에서 Data Access 페이지의 컴포넌트 들은 어플리케이션이 데이터베이스에 접근하여 데이터를 읽고, 쓰게 해주는 역할을 한다. 그리고, Data Controls 페이지에 있는 컨트롤 들에게 데이터를 전달하여, 여기에 데이터를 표시할 수 있도록 하는 것이다.

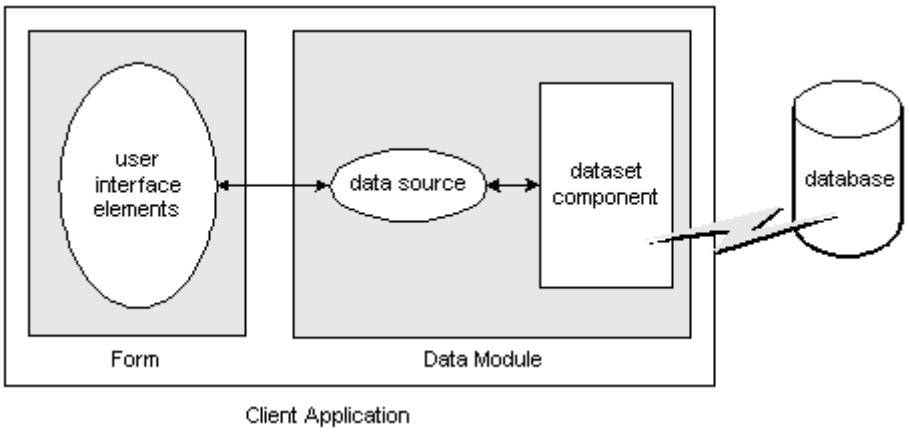
데이터베이스 어플리케이션 아키텍처 (Database Application architecture)

델파이의 데이터베이스 어플리케이션은 사용자 인터페이스 요소 들과 데이터베이스를 관리하는 컴포넌트, 그리고 데이터 세트라고 불리는 데이터베이스에 저장된 테이블 등의 데이터를 나타내는 컴포넌트 들로 구성된다.
이때 데이터 모듈(data module)에 데이터베이스 접근 컴포넌트 들을 분리하면 사용자 인터페이스 부분과 데이터 접근 부분을 보다 효율적으로 관리할 수 있다. 특히 잘 디자인된 폼이나 데이터 모듈이 있으면, 이를 객체 저장소(Object repository)에 저장해 두었다가 사용

한다면 제작하는 어플리케이션의 전체적인 인터페이스를 일관되게 유지할 수 있다.
데이터베이스 어플리케이션의 아키텍처는 실제로 사용하게 되는 데이터베이스의 종류가 무엇이며, 데이터베이스 정보를 공유하게 될 사용자가 몇 명인지, 그리고 사용할 정보의 종류에 따라서 몇 가지 카테고리로 나누어 생각해볼 수 있다.
만약 수 명의 사용자가, 서로 공유하지 않는 정보를 사용하는 데이터베이스 어플리케이션을 작성한다면 1-tiered 기반의 로컬 데이터베이스를 사용하면 될 것이다. 이 방법은 데이터가 로컬에 저장되므로 속도라는 측면과 데이터베이스 서버를 따로 사지 않아도 되므로 경제성이 좋다. 그렇지만, 이 경우에도 얼마나 많은 양의 데이터와 테이블을 사용해야 되는지에 따라서 결정이 달라질 수 있다.
2-tiered 어플리케이션은 1-tiered 기반의 어플리케이션에 비해 보다 많은 사용자를 지원하며, 커다란 원격 데이터베이스를 사용할 수 있기 때문에 로컬 데이터베이스의 한계를 극복할 수 있다.
데이터베이스 정보에 몇 개의 테이블의 관계가 복잡하게 얽혀있고, 계속적으로 클라이언트 수가 증가하는 추세라면 이럴 때에는 멀티-tiered 데이터베이스 어플리케이션을 선택해야 한다. 멀티-tiered 어플리케이션에는 데이터베이스의 상호작용과 데이터 간의 관계에 대한 부분을 중앙집중적으로 관리하는 로직을 미들웨어라는 형식으로 제공하게 된다. 이렇게 함으로써 서로 다른 클라이언트 어플리케이션 들이 데이터 로직이 호환된다면 같은 데이터를 사용하도록 할 수도 있고, 클라이언트 어플리케이션을 작고 가볍게(thin) 만들어 줄 수도 있다. 가벼운 클라이언트 어플리케이션의 장점은 설치도 쉽고, 데이터베이스 접속 소프트웨어를 포함하지 않으므로 환경 설정이나 관리도 대단히 쉬운 잇점이 있다. 또한, 멀티-tiered 어플리케이션은 데이터 처리 작업을 여러 시스템에 분산시켜 처리할 수 있기 때문에 수행 성능도 높일 수 있다.

- 확장성에 대한 계획 수립 (Planning for scalability)

VCL 의 데이터인식 컨트롤(data-aware controls)을 이용하면, 데이터베이스와 데이터베이스에 저장된 데이터의 행동을 추상화할 수 있기 때문에, 쉽게 확장이 가능한 어플리케이션으로 개발할 수 있다. 개발하는 어플리케이션의 모델이 1-tiered, 2-tiered, 멀티-tiered 중 어느 것이라도, 사용자 인터페이스 부분을 다음 그림과 같이 데이터 접근층(data access layer)과 분리할 수 있다.



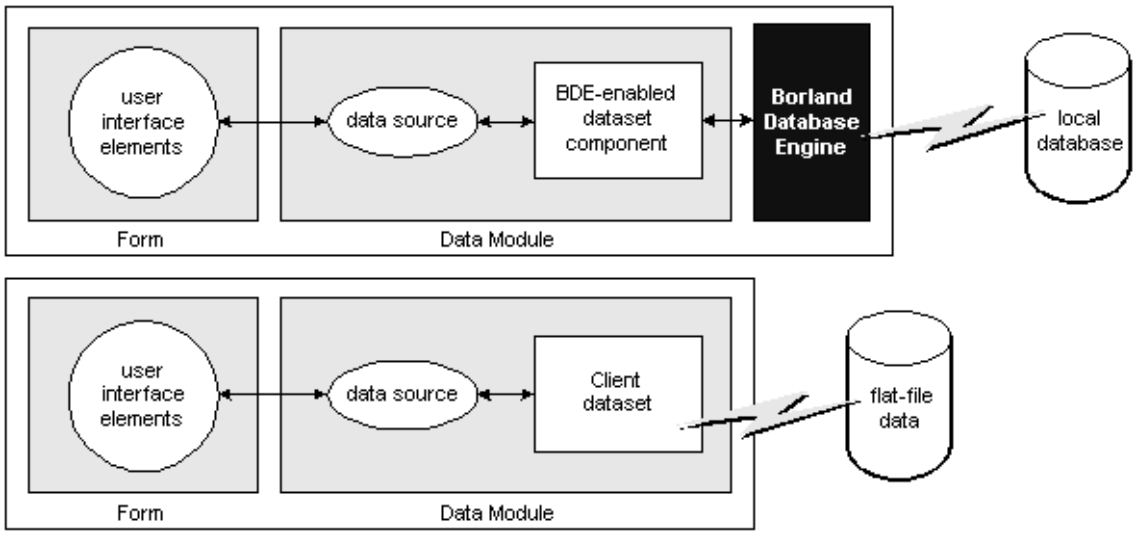
이 그림에서 폼은 사용자 인터페이스를 가리킨다. 폼에는 데이터 컨트롤과 다른 사용자 인터페이스 요소가 포함되는데, 데이터 컨트롤은 사용자 인터페이스를 데이터베이스의 데이터베이스의 정보에 접속하도록 해 준다. 이때 데이터 컨트롤과 연결되는 데이터 소스를 데이터 세트라고 한다. 데이터 소스와 데이터 세트를 데이터 모듈에 분리하면, 폼에는 어떤 방법으로 어플리케이션을 작성하더라도 변경될 부분이 없다. 다만 데이터 세트만 변경해 주면 되는 것이다.
데이터 모듈이 처음 소개 된 것이 델파이 2.0 버전 부터 이므로, 꽤 오랜 시간동안 사용되어온 개념이지만 아직도 생각보다 많이 사용되지 않고 있는 듯하다. 그렇지만, 지금까지 설명한 중요성을 염두에 두고 가능한 반드시 데이터 모듈을 사용하는 것을 습관처럼 하도록 권하고 싶다.

참고: 일부 사용자 인터페이스 요소 들은 확장하고자 할 때, 고려해 주어야 하는 경우가 있다. 예를 들어, 데이터베이스의 보안을 확보하기 위해, 데이터베이스에 접근할 때 로그인 할 수 있도록 하는 방법 등을 들 수가 있을 것이다.

BDE 에 기반을 둔 어플리케이션을 개발한다면, 1-tiered 에서 2-tiered 모델로 확장하는 것이 매우 쉽다. 단지 데이터 세트에 있는 몇 가지 프로퍼티를 로컬 데이터베이스에서 SQL 서버에 접속하도록 바꾸기만 하면 그걸로 끝이다.
또한, 플랫 파일에 기반을 둔 데이터베이스 어플리케이션을 개발하면, 이 모델은 쉽게 멀티-tiered 모델로 확장할 수 있다. 이것은 이들의 구조가 동일한 클라이언트 데이터 세트 컴포넌트를 사용하기 때문이다. 또한, 플랫 파일 어플리케이션과 멀티-tiered 클라이언트 어플리케이션을 동시에 지원하도록 어플리케이션을 제작할 수도 있는데, 이러한 모델을 서류가방(briefcase) 모델이라고 한다.
만약, 결국에는 멀티-tiered 어플리케이션으로의 확장을 염두에 두고 있다면, 처음에 데이터베이스 어플리케이션을 제작할 때부터 이를 고려하는 것이 좋다. 사용자 인터페이스는 반드시 데이터 모듈과 분리하여 제작하고, 동시에 미들 tier 에 위치할 모든 로직 들은 따로 분리하여 데이터 모듈로 관리하도록 한다. 또한, 사용자 인터페이스 요소를 데이터 세트에 접속하도록 할 때에도 처음부터 클라이언트 데이터 세트를 경유하여 BDE 에 접속하도록 분리된 데이터 모듈에 작성하면, 단순히 접속할 대상을 수정해 주는 것으로 확장이 가능하게 할 수 있다.

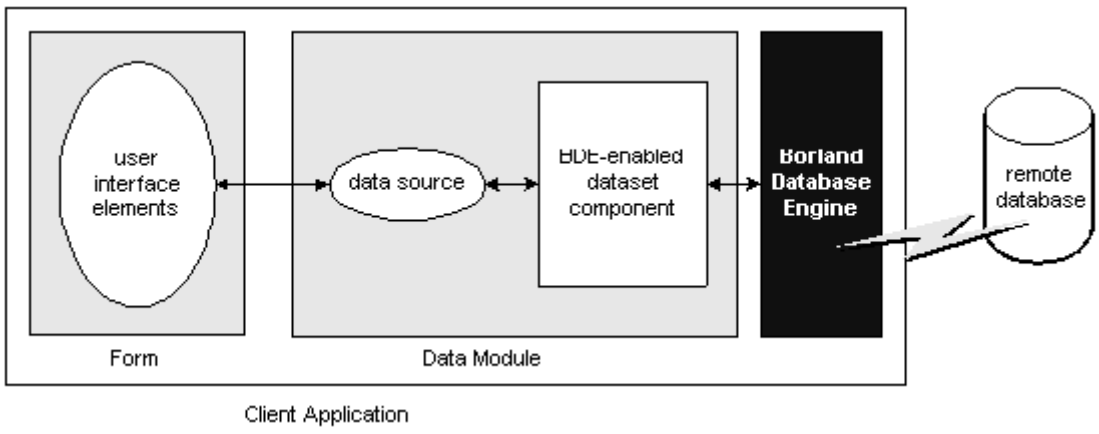
- 1-tiered 데이터베이스 어플리케이션

1-tiered 데이터베이스 어플리케이션은 어플리케이션과 데이터베이스가 같은 파일 시스템을 사용한다. 로컬 데이터베이스나 파일을 이용하여 데이터베이스 정보를 저장하게 된다. 이때 어플리케이션은 사용자 인터페이스 부분과 데이터 접근 기전(BDE 를 사용하거나, 플랫 파일을 사용)으로 구성되며, 데이터 세트 컴포넌트의 종류가 무엇이냐에 따라서 로컬 데이터베이스와 플랫 파일 데이터베이스로 구분할 수 있다. 로컬 데이터베이스는 BDE 를 사용하여 파라독스, 디베이스, 엑세스, 폭스 프로와 같은 상용 로컬 데이터베이스에 접속하게 되며, 플랫 파일은 자체적인 파일을 사용한다.



- 2-tiered 데이터베이스 어플리케이션

2-tiered 데이터베이스 어플리케이션에서는 클라이언트 어플리케이션이 데이터와 원격 데이터베이스 서버에 직접 상호작용할 수 있는 인터페이스를 제공해야 한다.

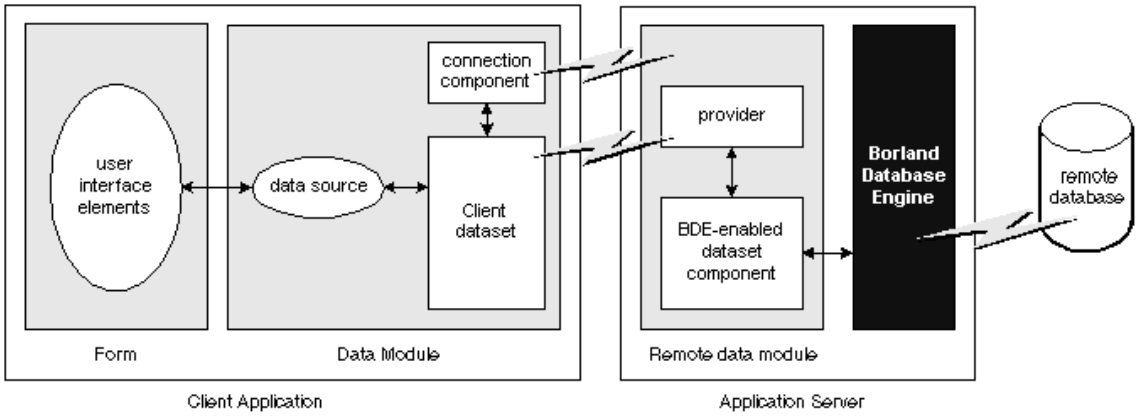


이 모델에서는 모든 어플리케이션이 데이터베이스 클라이언트가 된다. 클라이언트는 서버

에게 정보를 요구하거나, 업데이트된 정보를 데이터베이스 서버로 전송한다. 서버는 여러 클라이언트의 요구를 동시에 처리하고, 이들이 데이터에 접근하고 업데이트할 수 있도록 관리한다.

- 멀티-tiered 데이터베이스 어플리케이션

멀티-tiered 데이터베이스 어플리케이션은 어플리케이션을 서로 다른 기계에 존재할 수 있도록 쪼개어 관리하게 된다. 클라이언트 어플리케이션은 데이터에 대한 사용자 인터페이스를 제공한다.
클라이언트는 모든 데이터에 대한 요구와 업데이트된 정보를 어플리케이션 서버(리모드 데이터 브로커라고도 한다)에 전송하게 된다. 어플리케이션 서버는 원격 데이터베이스 서버와 직접 통신을 하거나, 다른 커스텀 데이터 세트를 통해 데이터베이스 서버에 접속하게 된다 보통 이 모델에서의 클라이언트와 어플리케이션 서버, 원격 데이터베이스 서버는 다른 기계에 위치시키는 것이 좋다 다음 그림은 멀티-tiered 어플리케이션과 BDE 의 관계를 나타낸 것이다



어플리케이션 서버와 클라이언트 어플리케이션을 모두 델파이를 이용해서 작성할 수 있다. 클라이언트 어플리케이션은 표준 데이터인식 컨트롤을 이용하여 하나 이상의 클라이언트 데이터 세트 컴포넌트에 데이터 소스를 통해 접속하게 되고, 여기서 데이터를 얻어서 보여주고, 편집할 수 있게 된다. 각각의 클라이언트 데이터 세트는 IProvider 인터페이스를 통해 어플리케이션 서버와 통신을 하게 되는데, IProvider 인터페이스는 원격 데이터 모듈(remote data module)의 한 부분으로 TCP/IP, DCOM, MTS, CORBA 등의 다양한 프로토콜을 이용하여 통신을 할 수 있다. 이러한 프로토콜은 클라이언트 어플리케이션에서 사용하는 접속 컴포넌트와 서버 어플리케이션의 원격 데이터 모듈의 종류에 따라 결정된다.
어플리케이션 서버는 2 가지 방법으로 IProvider 인터페이스를 사용할 수 있다. 어플리케이션 서버에 Provider 객체가 있으면, 이 객체를 이용하여 IProvider 인터페이스를 생성하

게 된다. 이 방법이 앞의 그림에서 표현한 모델이다. 어플리케이션 서버가 이런 컴포넌트를 포함하고 있지 않은 경우에는 BDE 가 가능한 데이터 세트에서 IProvider 인터페이스를 생성하게 된다. 그렇지만, provider 컴포넌트를 사용하는 것이 보다 확실하게 인터페이스를 사용할 수 있는 방법이다. 두가지 방법 모두, 모든 데이터는 클라이언트 어플리케이션과 어플리케이션 서버 사이에 인터페이스를 통해서 통신하게 된다.

데이터베이스의 특징 (Database Features)

일반적으로 데이터베이스를 관리하는 DBMS 에는 여러 명의 사용자가 동시에 접근하여, 대용량의 데이터를 다루게 되므로 이를 보다 효과적이고, 견고하게 관리하기 위해 공통적인 특징을 가지게 된다.
흔히, 이런 공통적인 특징을 어떤 DBMS 가 더 잘 지원하는지 여부를 가지고 벤치마크 테스트를 하기도 하며, 개발자에게 편리하면서도 강력한 기능을 지원하는 DBMS 가 좋은 DBMS 라고 말할 수 있다.
그렇다면, DBMS 가 갖추고 있어야 할 데이터베이스의 특징과 데이터베이스 어플리케이션을 제작할 때에 여기에 대해서 어떤 것을 고려해야 하는지 알아보도록 하자. 이들에 대한 자세한 접근은 다음 장에서 다루게 되겠지만, 기초적인 부분을 미리 소개한다. 참고로 DBMS 에는 엑세스나 파라독스와 같이 로컬 환경에 기반을 둔 로컬 DBMS 와 오라클이나 인터베이스와 같이 네트워크 환경에 기반을 둔 원격 SQL 서버 DBMS 가 있다.

- 데이터베이스 보안 (Database security)

데이터베이스에는 정보가 저장되는데, 이 정보가 기업 기밀 등과 같이 중요한 것인 경우 이를 보호할 수 있는 보안 스키마가 제공되어야 한다. 파라독스나 디베이스와 같은 로컬 DBMS 는 보안을 테이블 또는 필드 레벨에서만 제공할 수 있다. 즉, 사용자가 보호된 테이블에 접근할 경우에 패스워드를 묻도록 할 수 있다. 일단 사용자가 인증(authentication)이 되면 이들은 허용된 필드에만 접근할 수 있다. 이에 비해 대부분의 SQL 서버는 처음에 데이터베이스 서버에 접근할 때 사용자 이름과 패스워드를 요구한다.
데이터베이스 어플리케이션을 디자인 할 때에는 데이터베이스 서버에 의해 요구되는 인증의 종류에 대해 고려해야 한다. 만약 사용자가 패스워드를 사용하지 않도록 해야 한다면, 패스워드를 요구하지 않는 데이터베이스 서버를 사용하거나, 패스워드를 프로그램에서 제공하도록 만들어야 한다. 그렇지만, 프로그램을 통해서 패스워드를 제공하도록 하면, 패스워드가 어플리케이션에 의해 읽혀지고 노출되지 않도록 주의해야 할 것이다.
그리고, 사용자가 패스워드를 입력하도록 할 경우에는, 언제 패스워드를 요구할 것인지를 생각해야 한다. 만약, 현재는 로컬 데이터베이스를 사용하고 있지만 이후 SQL 서버로 확장해야 한다면, 테이블에 접근하기 전에 패스워드를 요구하도록 어플리케이션을 제작할 수

도 있겠다.
만약, 어플리케이션이 여러 개의 보호된 시스템이나 데이터베이스에 접근해야 되기 때문에, 여러 차례 패스워드를 입력해야 한다면, 사용자에게 하나의 마스터 패스워드를 제공하여 필요한 시스템과 데이터베이스에 접근할 수 있도록 하는 것이 현명하다. 즉, 마스터 패스워드를 입력하면 어플리케이션이 패스워드를 프로그램적으로 입력하도록 하는 것이다.
멀티-tiered 어플리케이션에서는 서로 다른 보안 모델을 동시에 사용하기를 원할 수도 있다. 예를 들어, 미들-tier 를 조절하기 위해 CORBA 나 MTS 를 사용할 수도 있고, 미들-tier 가 데이터베이스 서버에 로그인 하는 작업을 모두 처리하도록 할 수도 있다.

- 트랜잭션 처리 (Transactions)

트랜잭션은 한 번에 처리되어야 할 명령의 그룹이다. 이들이 한 번에 처리되어 완전히 데이터를 업데이트 된 경우 이를 'committed' 라고 하며, 처리에 실패해서 전체 명령이 하나도 실행되지 않은 상태로 돌아가는 것을 'rolled back'이라고 한다.
트랜잭션은 데이터베이스 명령이 실행 중에 일어날 수 있는 여러 가지 하드웨어 실패 등에 대한 고려를 하기 위해 디자인 된 것이다. 가장 흔히 예를 드는 것으로는 은행의 온라인 업무를 들 수 있다. 또한, 트랙잭션은 SQL 서버의 일치성(concurrency)을 보장하는 기초가 된다. SQL 서버는 내부적으로 트랜잭션을 스케쥴하여 여러 명의 사용자가 데이터를 처리할 때 한 사람이 데이터를 변경할 때 다른 사람이 동시에 접근하여 데이터가 깨지지 않도록 관리한다.
로컬 데이터베이스의 경우 트랜잭션을 지원하지 않는 것이 많지만, BDE 드라이버를 이용하면 제한적이지만 일부에서 트랜잭션을 지원하게 할 수 있다. 또한, SQL 서버와 ODBC 와 호환되는 데이터베이스는 대개 데이터베이스 서버가 트랜잭션을 직접 지원한다.
멀티-tiered 어플리케이션의 경우에는 트랜잭션을 데이터베이스 차원에서 지원하지 않고 어플리케이션 서버에서 지원하도록 하면 여러 개의 데이터베이스에 대한 접근을 포함한 트랜잭션을 처리할 수 있다. 예를 들어, 인적 사항은 오라클이 관리하게 하고, 물건 들에 대한 재고 정보를 인터베이스가 관리하게 하는 것과 같은 분산 트랜잭션 처리가 가능하다.

- 데이터 사전 (Data Dictionary)

어떤 데이터베이스를 사용하든, 어플리케이션은 데이터 사전에 접근할 수 있다. 데이터 사전은 어플리케이션과는 독립적으로 사용자가 조절할 수 있는 저장 영역으로 필드의 속성 세트를 확장하거나 데이터의 내용을 설명하는 등의 일을 할 수 있는 곳이다.
예를 들어, 회계 어플리케이션을 개발할 때 화폐 단위의 포맷을 여러 가지로 표현해야 한다면, 이를 위해 여러 가지 특수한 필드 속성을 생성할 수 있다.
데이터 세트를 디자인 시에 생성하는 경우에는 오브젝트 인스펙터를 이용해서 필드의 내용

을 일일이 설정하기 보다는 데이터 사전에서 확장된 필드 속성을 이용하도록 하는 것이 더욱 편리하고 효율적이다.
클라이언트/서버 환경에서는 데이터 사전의 위치를 추가적인 정보의 공유를 위해 원격 서버에 위치시킬 수 있다. 데이터 사전에 대한 자세한 내용은 다음 장의 SQL 탐색기에 대한 설명을 할 때 다루게 될 것이므로 이를 참고하기 바란다.
데이터 사전에 대한 프로그래밍 인터페이스는 drintf.pas 유닛에서 제공하는데, 여기에는 다음과 같은 메소드 들이 있다.

루 틴	설 명
DictionaryActive	데이터 사전이 활성화되어 있는지 나타낸다.
DictionaryDeactivate	데이터 사전을 비활성화 시킨다.
IsNullID	주어진 ID 가 Null ID 인지 나타낸다.
FindDatabaseID	주어진 앨리어스의 데이터베이스 ID 를 반환한다.
FindTableID	주어진 데이터베이스의 테이블 ID 를 반환한다.
FindFieldID	주어진 테이블의 필드 ID 를 반환한다.
FindAttrID	명명된 속성 세트에 대한 ID 를 반환한다.
GetAttrName	주어진 ID 에 해당되는 속성 세트의 이름을 반환한다.
GetAttrNames	사전에서의 각 속성 세트에 대한 콜백을 실행한다.
GetAttrID	지정된 필드에 대한 속성 세트 ID 를 반환한다.
NewAttr	필드 컴포넌트에서 새로운 속성 세트를 생성한다.
UpdateAttr	필드의 프로퍼티와 맞는 속성 세트를 업데이트한다.
CreateField	저장된 속성에 기초한 필드 컴포넌트를 생성한다.
UpdateField	지정된 속성 세트에 맞는 필드의 프로퍼티를 변경한다.
AssociateAttr	주어진 필드 ID 와 속성 세트를 연관시킨다.
UnassociateAttr	필드 ID 와 속성 세트의 연관을 제거한다.
GetControlClass	지정된 속성 ID 에 대한 컨트롤 클래스를 반환한다.
QualifyTableName	사용자 이름에 적당한 테이블 이름을 반환한다.
HasConstraints	사전에 있는 데이터 세트가 constraints 를 가지고 있는지 나타낸다.
UpdateConstraints	import 된 데이터 세트의 constraints 를 업데이트 한다.
UpdateDataset	데이터 세트를 사전의 constraints 와 현재 설정을 업데이트 한다.

- 참조 무결성, 저장 프로시저와 트리거 (Referential integrity, stored procedures, and triggers)

모든 관계형 데이터베이스는 앞에서 설명한 바와 같이 어플리케이션이 데이터를 저장하고,

처리하는 공통적인 형태를 가지고 있다. 이 밖에도 데이터베이스는 다음과 같은 특징을 가지고 있는데, 이들은 데이터베이스 별로 조금씩 차이가 날 수 있다.

1. 참조 무결성 (Referential integrity)

참조 무결성은 테이블 사이의 마스터/디테일 관계를 제거하는 방법을 제시한다. 사용자가 마스터 테이블에서 디테일 레코드가 있는 필드를 삭제하려고 할 때에는 참조 무결성 규칙에 따라서, 삭제를 못하게 하거나 자동으로 디테일 레코드를 삭제하도록 할 수 있다.

2. 저장 프로시저 (Stored procedures)

저장 프로시저는 명명되어 저장된 SQL 문장의 세트이다. 저장 프로시저는 보통 서버에서 흔히 행해지는 데이터베이스 작업을 저장했다가 이를 실행하여, 레코드 세트를 반환하게 된다.

3. 트리거 (Triggers)

트리거는 특정 명령에 의해 자동으로 실행되는 SQL 문장의 세트이다.

데이터베이스의 선택 (Select Database)

델파이 데이터베이스 어플리케이션 개발자는 BDE 와 BDE 이외의 다른 데이터베이스 접근 방법들 중에서 선택의 기로에 놓이는 경우가 많다. 델파이의 경우 기본적으로 제공되는 BDE 이외에도 아폴로, 타이탄과 같은 써드 파티에서 제공하는 여러 가지 데이터베이스 엔진들이 있기 때문에, 이를 선택하는 것이 중요한 고려 사항이 되기도 한다.
선택의 기준이 되는 핵심은 간단하다. 어플리케이션의 요구사항이 어느 정도 되며, 이를 해결하기 위한 솔루션이 어떤 것이 있는가 ?, 그리고, 특정 기술의 장점이 있는 경우의 솔루션은 어떤 것이 있는가 ? 하는 것이다.
이를 위해 여러 가지를 고려해보아야 하는데, 특정 데이터베이스 포맷을 지원해야 하는지 ?, 특정 DBMS 에 접근해야 하는지 ?, 또한 어플리케이션에 적합한 데이터베이스의 종류는 어떤 것인지 ? 따위의 것들이다.
이를 단계별로 고려해야 할 점들을 나열해 보면 다음과 같다.

- 1 단계: 사용할 데이터베이스의 종류를 선택한다.

첫 단계로는 어플리케이션에 필요한 데이터베이스의 종류가 어떤 것인지를 결정하는 것이다.

고려해야 할 것으로는 다음과 같은 것들이 있다.

1. 로컬 데이터베이스 (Local Database)

   데이터가 데이터를 사용하는 컴퓨터, 그리고 이를 사용하는 사용자와 같은 컴퓨터에만 위치할 경우. 동시에 여러 명의 사용자가 접근하지 않아도 될 때 선택한다.

2. 다중 사용자 파일 서버 시스템 (Multi-user File Server)

   파일 서버에 데이터가 위치하고, 모든 처리 과정은 클라이언트에서 일어나는 모델이다. 파일을 열고, 몇 바이트를 읽고 하는 명령과 같이 저수준 요구 사항이 사용자의 컴퓨터와 파일 서버에서 전송되게 된다. 파일 서버 데이터베이스는 데이터의 안정성이 낮다. 만약 사용자의 컴퓨터가 어떤 이유에서든 데이터베이스 업데이트를 완료하지 못하면, 데이터베이스는 불안정한 상황에 놓이게 되고, 이럴 경우 데이터베이스의 수리를 위해서는 모든 사용자가 데이터베이스 어플리케이션에서 빠져나가야 한다.
   그러므로, 파일 서버 데이터베이스는 다음과 같이 제한된 경우에만 사용하는 것이 좋다.

   1) 어플리케이션이 분초를 다투는 상황이 없어야 한다.
   2) 보안이 그렇게 중요하지 않아야 한다.
   3) 사용자 수가 15 명을 넘지 않는다.
   4) 예산이 부족한 등의 클라이언트/서버 솔루션을 선택하지 못할 이유가 있을 경우

3. 다중 사용자 클라이언트/서버 데이터베이스 시스템

   클라이언트가 서버에 쿼리 문장과 같이 고수준의 요구를 할 수 있는 시스템이다. 그러므로, 많은 양의 데이터 처리가 서버에서 일어나게 되며, 처리된 데이터가 클라이언트에 전송되는 형태를 가지는 데이터베이스 시스템이다. 클라이언트는 데이터베이스 파일에 직접 접근할 수 없기 때문에, 데이터베이스 파일이 망가질 염려가 없으며, 사용자 수가 많아질수록 효율이 증가한다. 앞에서도 언급했지만, 클라이언트/서버 데이터베이스를 선택해야 하는 경우는 다음과 같다.

   1) 어플리케이션의 작업이 분초를 다툴 경우
   2) 보안이 중요한 경우
   3) 15 명 이상의 사용자가 있는 경우
   4) 앞으로 1)~3)의 경우가 나타날 가능성이 있는 경우

4. 웹에 기초한 시스템

분산된 데이터 접근을 위해 웹에 기초한 솔루션은 배포와 확장성이란 측면에서 장점이 많은 시스템이다. 그렇지만, 여기에 대해서는 이 책의 범위를 넘기 때문에 자세한 언급을 하지 않는다. 중요한 것은 웹 환경의 경우 동시에 많은 양의 트랜잭션이 일어날 수 있으며, 멀티 쓰레드 환경에 적합한 데이터베이스와 데이터베이스 접근 소프트웨어를 이용해야 한다는 것이다.

5. 멀티-tier 데이터베이스 시스템

클라이언트 소프트웨어와 데이터베이스 서버 사이에 하나 이상의 어플리케이션 서버가 존재하는 모델이다. 볼랜드의 MIDAS 가 대표적인 시스템이다. 써드 파티에서는 현재 특별히 지원되는 모델이 없다.

- 2 단계: 데이터베이스 포맷/서버의 선택

일단 사용할 데이터베이스의 종류를 결정했으면, 실제로 어떤 데이터베이스를 사용할 것인지 구체적으로 결정해야 한다. 여기에는 특별한 선택의 기준이 존재하기 보다는, 선호도와 친밀도 등의 여러 가지를 고려해 보아야 한다. 파일 서버나 로컬 데이터베이스를 사용하기로 결정하였다면 데이터베이스 포맷을 선택하면 BDE 를 쓸 것인지, 아니면 다른 써드 파티 도구를 사용할 것인지를 결정하는데 도움이 된다. 예를 들어, DBF 파일에 기초한 도구를 사용하기 원한다면, DBF 포맷을 지원하도록 해야 한다. 파일 서버 어플리케이션의 경우에는 표준적인 데이터베이스 포맷을 이용하는 것이 이후의 확장성을 위해 권장된다.
클라이언트/서버 시스템이라면 데이터베이스 서버를 선택해야 하는데 각 제품마다의 특징이 다르기 때문에, 자신의 요구에 맞는 제품을 선택해야 할 것이다.

- 3 단계: 데이터베이스에 접근할 수 있는 BDE 대체 엔진을 선택한다.

일단 데이터베이스 포맷이나 데이터베이스 서버를 선택하고 나면, 이를 지원하는 써드 파티 엔진을 고를 수 있다. 같은 데이터베이스를 지원하는 엔진도 그 지원의 정도나 수행 능력에 다소간의 차이를 보이게 되며, 가격에도 차이가 있으므로 이를 잘 고려해서 선택해야 할 것이다. 여기에서 고려해야 할 것들은 다음과 같은 것들이 있다.

1. 포맷과 데이터베이스 서버 종류

2. 수행 능력, vendor 의 지원

3. 배포:
   쉬운 배포가 되어야 한다. 어떤 제품은 완전히 실행 파일에 포함되는 것도 있고, DLL 을 요구하는 경우도 있다. 클라이언트/서버 솔루션의 경우에는 데이터베이스에 접근할 수 있는 드라이버를 요구하기도 하며 ODBC, ADO, BDE 와 같은 미들웨어 베이스의 솔루션은 적절한 설치를 요구하는 경우도 있다. 참고로 BDE 는 2~3 MB 정도 크기의 DLL 을 요구한다.

4. 데이터 컨트롤의 사용 여부:
   GUI 어플리케이션을 개발할 때, 데이터 컨트롤을 제공하는지 여부는 개발에 상당한 영향을 미친다. 이를 지원하는 레벨은 데이터 컨트롤을 지원하지 않는 경우와 자신의 데이터 컨트롤을 제공하는 경우, 그리고 데이터 컨트롤을 바로 사용할 수 있는 경우로 나눌 수 있다. vendor 에서 제동되는 컨트롤만 써야 하는 경우는 없는 것보다는 낫지만, 그리 좋은 방법은 못된다. 델파이 2 까지만 해도 표준 데이터 컨트롤을 이용하도록 개발하는 것이 어려웠지만, 델파이 3 의 가상 데이터 세트를 이용하면 비교적 쉽게 데이터 컨트롤 들을 지원할 수 있으므로 최근의 BDE 대체 엔진들은 대부분 데이터 컨트롤을 사용할 수 있다.

5. 리포트 인쇄:
   데이터 컨트롤과 마찬가지로 QuickReports, Piparti 등의 다른 리포팅 도구를 지원하는지 여부도 중요한 고려 대상이 된다.

6. 데이터베이스 서버의 접근도:
   BDE 대체 엔진 중에 단순히 데이터베이스 서버에 접속하는 것만 고려할 것이 아니라, 어떤 제품들은 그 이상을 지원하므로 이를 고려한다. 예를 들어, Interbase Objects 나 Direct Oracle Access 와 같은 제품은 데이터베이스 서버의 API 를 직접 이용할 수도 있다. 이것이 그렇게 중요한 것은 아니지만, 특정 서버를 사용하는 경우라면 이런 제품을 고려해 보는 것도 괜찮다. 하지만 반드시 생각해야 하는 것은, 이렇게 특정 서버에 적합한 제품을 이용해 프로그램을 개발한 경우 차후에 데이터베이스 서버를 바꿀 경우가 생긴다면 고쳐야 할 사항이 많아진다는 점을 염두에 두어야 할 것이다.

7. Vendor 의 지원 정책와 버그 수정:
   아무래도 같은 나라에 있거나, 지원이 쉬운 회사의 제품을 선택하는 것이 좋을 것이다. 또한, 쉽게 E 메일을 보낼 수 있고, 뉴스 그룹 등의 사용자에 대한 지원이 많은 회사의 제품에 장점이 있다. 또한 모든 소프트웨어에는 버그가 있기 마련이므로, 얼마나 제품에 성의를 가지고 버그를 고쳐주는지 여부도 중요하다.

델파이 데이터베이스 컴포넌트

델파이의 컴포넌트 팔레트의 Data Access 페이지에는 데이터베이스에 대해 작업을 하는 컴포넌트 들이 들어 있다. 여기에 있는 것은 대개 데이터베이스 연결, 테이블과 쿼리 등의 요소들을 캡슐화 한 것이다. 또한, Data Controls 페이지에는 데이터베이스를 눈으로 보고, 수정할 수 있는 데이터 컨트롤 들이 위치하고 있다.
델파이에서 데이터베이스에 접근하려면, DataSource 컴포넌트에 의해 연결할 수 있는 데이터 소스가 있어야 한다. 데이터 소스로는 테이블, 쿼리, 또는 저장 프로시저(stored procedure) 등이 될 수 있으며, 이들은 Data Access 페이지의 TTable, TQuery, TStoredProcedure 등으로 캡슐화 되어 있다. 그러므로, 데이터베이스를 연결할 때에는 데이터 컨트롤은 TDataSource 와 연결하고, TDataSource 의 DataSet 프로퍼티로 데이터 소스로 사용되는 데이터 세트 컴포넌트를 연결하면 기본적인 연결이 끝난다.
참고로, 멀티-tier 데이터베이스 어플리케이션을 사용하거나 플랫 파일을 사용할 때에는 데이터 소스로 TClientDataSet 컴포넌트를 이용해야 한다.

- 테이블과 쿼리, 저장 프로시저

가장 흔히 사용되는 데이터 세트 컴포넌트는 Table 이다. 테이블 객체는 실제 데이터베이스 테이블을 가리킨다. Table 컴포넌트를 사용할 때에는 DatabaseName 프로퍼티에 사용하고자 하는 데이터베이스의 이름을 지시해 주어야 한다. 여기에는 앨리어스 또는 테이블 파일 등이 들어있는 디렉토리 패스를 입력하거나, 사용된 TDatabase 컴포넌트로 설정해야 한다. 일단 DatabaseName 이 설정되면, 오브젝트 인스펙터에 현재 데이터베이스에서 사용이 가능한 테이블의 이름을 목록으로 보여주며 여기에서 테이블을 선택하여 TableName 프로퍼티를 선택해야 한다.
테이블에 못지 않게 많이 사용되는 것이 Query 컴포넌트이다. 쿼리는 SQL 언어 명령으로 이루어져 있다. Query 컴포넌트에서의 테이블은 SQL 프로퍼티 안에 저장되어 있는 SQL 문장의 실행 결과에 의해서 사용된다.
저장 프로시저는 TStoredProc 컴포넌트에 의해 지원된다. 이것은 SQL 서버 데이터베이스의 로컬 프로시저들을 가리키는 것으로, 이러한 프로시저를 실행시켜서 데이터베이스 테이블의 폼 안에 결과를 얻을 수 있다.

- 기타 데이터 접근 컴포넌트

Table, Query, StoredProc, DataSource 컴포넌트 외에도 몇 가지 컴포넌트 들이 있다. Database 컴포넌트는 트랜잭션 제어, 보안, 그리고 연결을 제어하는 목적으로 사용된다.

이 컴포넌트는 일반적으로 클라이언트/서버 어플리케이션 안에서 원격 데이터베이스에 연결하기 위해서만 사용되거나, 여러 개의 폼에서 같은 데이터베이스에 연결되는 일을 방지하기 위해 사용된다.
Session 컨트롤은 기존의 데이터베이스와 앨리어스의 목록과 데이터베이스 로그인을 수정하기 위한 이벤트를 포함해서 어플리케이션의 데이터베이스 연결에 대한 제어 방법을 제공하는 컴포넌트이다. BatchMove 컴포넌트는 복사, 붙이기, 업데이트, 값 지우기 등의 배치 작업을 하나 이상의 데이터베이스에 대해 수행할 때 사용된다.
UpdateSQL 컴포넌트는 SQL 문장을 작성해서, 읽기 전용 쿼리를 사용하고 있을 때 다양한 업데이트 작업을 수행할 수 있게 해준다. 이 컴포넌트는 테이블이나 쿼리의 UpdateObject 프로퍼티의 값으로 사용된다.

데이터베이스와 앨리어스의 이해 (Understanding Database and Alias)

데이터베이스는 정보를 테이블에 저장한다. 그 밖에도, 데이터베이스에 포함된 정보에 대한 테이블과 인덱스처럼 테이블에 의해 사용되는 객체, 저장 프로시저와 같은 SQL 객체 등이 포함되어 있다.
각각의 BDE 를 이용한 데이터 세트 컴포넌트에는 DatabaseName 이라는 프로퍼티가 있다. 이 프로퍼티는 데이터 세트에 있는 정보를 담고 있는 테이블을 포함한 데이터베이스를 지정한다. 어플리케이션을 설정할 때에는 데이터 세트를 바인드하기 전에 반드시 이 프로퍼티를 이용해서 데이터베이스를 지정해 주어야 한다.
데이터베이스는 BDE 앨리어스(alias)를 가지고 있다. 보통 이런 BDE 앨리어스를 DatabaseName 프로퍼티의 값으로 지정하게 된다. BDE 앨리어스는 데이터베이스에다가 데이터베이스의 환경 정보를 합쳐 놓은 것이라고 생각하면 된다. 앨리어스와 연관된 환경 정보는 데이터베이스 종류에 따라 다르다.
예를 들어, 파라독스 데이터베이스에서는 데이터베이스를 어떤 디렉토리에 저장하고 있는지를 알기만 하면 되지만, 사이베이스에서는 서버에 대한 네트워크 주소, 데이터베이스 이름과 사용자 ID, 패스워드 등의 정보를 알고 있어야만 한다.
BDE Administration 도구나 SQL 탐색기를 이용해서 이런 각각의 BDE 앨리어스를 생성하고, 관리할 수 있다.
데이터베이스에 대해서는 독립된 컴포넌트인 TDatabase 를 사용할 수도 있다. 데이터베이스 컴포넌트를 어플리케이션에 추가하지 않으면, 데이터 세트의 DatabaseName 프로퍼티를 바탕으로 임시 컴포넌트가 하나 생성되어 사용된다.

데이터베이스에 접속하기 (Connecting to databases)

델파이 어플리케이션이 데이터베이스에 접속할 때, 이런 접속 과정은 앞에서도 언급한

TDatabase 컴포넌트가 관장하게 된다. 데이터베이스 컴포넌트는 BDE 세션을 이용해서 데이터베이스 서버에 접속한다. 데이터베이스 컴포넌트는 이 밖에도 BDE 에 기초한 어플리케이션에서의 트랙잭션 관리와 연관된 테이블의 캐쉬 업데이트(cached update)를 할 때 사용된다.

- 데이터베이스 컴포넌트

어플리케이션의 데이터베이스 접속은 데이터베이스 컴포넌트에 의해 관리된다. 비록 명시적으로 데이터베이스 컴포넌트를 어플리케이션에 추가하지 않았다고 하더라도 런타임에서 임시로 생성되어 접속을 관리한다. 임시 데이터베이스 컴포넌트 들은 필요할 때마다 생성되어 데이터베이스 접속의 세세한 부분을 개발자가 제어하지 않더라도, 대개의 전형적인 데스크탑 데이터베이스 어플리케이션을 지원한다. 그렇지만, 클라이언트/서버 어플리케이션을 제작할 경우에는 개발자가 데이터베이스 컴포넌트를 이용해서 관리하는 것이 좋다.

1. 임시 데이터베이스 컴포넌트의 사용

임시 데이터베이스 컴포넌트는 필요에 따라 자동으로 생성된다. 예를 들어, TTable 컴포넌트를 폼에 올려 놓고, 적당히 프로퍼티를 설정해서 테이블을 열면, 델파이는 임시 데이터베이스 컴포넌트를 생성한다.
임시 데이터베이스 컴포넌트의 주요 프로퍼티 들은 현재의 세션에 의해서 결정된다. 예를 들어, 세션의 KeepConnections 프로퍼티가 True(디폴트) 데이터베이스 접속이 데이터 세트가 닫혀도 계속 유지된다. 이 경우에 DropConnections 메소드를 호출해야 접속이 종료된다. 또한, 세션의 디폴트 OnPassword 이벤트는 어플리케이션이 데이터베이스에 접속하고자 할 때 패스워드를 요구하면 표준 패스워드 입력 상자를 보여준다
임시 데이터베이스 컴포넌트에 의해 설정되는 디폴트 프로퍼티는 일반적이고, 합리적인 값으로 설정되어 있기 때문에, 복잡한 클라이언트/서버 어플리케이션과 같이 많은 사용자와 데이터베이스 접속에 서로 다른 요구 사항을 필요로 하는 데에는 적합하지 않다.

2. 디자인 시의 데이터베이스 컴포넌트 생성

데이터베이스 컴포넌트를 디자인 시에 생성하면, 초기 프로퍼티를 쉽게 설정할 수 있고, OnLogin 이벤트를 이용하여 데이터베이스 컴포넌트가 서버에 처음 접속할 때의 접속 문제를 컨트롤 할 수 있다.

3. 런타임에서의 데이터베이스 컴포넌트 생성

데이터베이스 컴포넌트를 런타임에서 생성하는 경우는 얼마나 많은 수의 데이터베이스 컴포넌트가 필요한 지 모르지만 어플리케이션이 데이터베이스 연결을 관리하기 위한 데이터베이스 컴포넌트가 반드시 필요할 때를 들 수 있다. 데이터베이스 컴포넌트를 런타임에서 생성할 때에는 항상 유일한 이름을 부여해야 하며, 세션과 연관시켜야 한다.
컴포넌트는 TDatabase.Create constructor 를 호출하면 생성된다. 이때 데이터베이스의 이름과 세션의 이름을 모두 제시해야 한다. 다음의 함수는 런타임에서 데이터베이스 컴포넌트를 생성해주는 함수이다.

```
function RunTimeDbCreate(const DatabaseName, SessionName: string): TDatabase;
var
  TempDatabase: TDatabase;
begin
  TempDatabase := nil;
  try
    //세션이 존재하면 이를 활성화하고, 아니면 새로운 세션을 생성한다.
    Sessions.OpenSession(SessionName);
    with Sessions do
      with FindSession(SessionName) do
        Result := FindDatabase(DatabaseName);
        if Result = nil then
        begin
          //새로운 데이터베이스 컴포넌트를 생성한다.
          TempDatabase := TDatabase.Create(Self);
          TempDatabase.DatabaseName := DatabaseName;
          TempDatabase.SessionName := SessionName;
          TempDatabase.KeepConnection := True;
        end;
        Result := OpenDatabase(DatabaseName);
      end;
    end;
  except
    TempDatabase.Free;
    raise;
  end;
end;
```

다음 코드는 이 함수를 이용하여 디폴트 세션에 대한 데이터베이스 컴포넌트를 생성하는 예를 보여준다.

```
var
  MyDatabase: array[1..10] of TDatabase;
  MyDbCount: Integer;
begin
  MyDbCount := 1;
  ...
  //데이터베이스 컴포넌트를 런타임에서 생성한다.
  begin
    MyDatabase[MyDbCount] := RunTimeDbCreate('MyDb' + IntToStr(MyDbCount), '');
    Inc(MyDbCount);
  end;
  ...
end;
```

데이터 컨트롤의 활용 (Using data controls)

대부분의 데이터 인식 컴포넌트 들은 데이터 세트에 저장된 정보를 대표한다. 이들의 종류로는 하나의 필드를 보여 주는 여러 컨트롤 들과 DBGrid 와 같이 여러 레코드를 한번에 보여주는 컨트롤, 그리고 레코드 사이의 이동과 여러가지 작업을 하도록 제공되는 시각적 도구인 DBNavigator 등이 있다.

- 공통적인 데이터 컨트롤의 특징

데이터 컨트롤은 데이터 세트의 현재 레코드의 필드의 내용을 보여주고, 편집할 수 있도록 해준다. 다음에 데이터 컨트롤의 종류와 이들의 특징을 나열해 보았으니, 참고하기 바란다.

데이터 컨트롤	설 명
TDBGrid	정보를 테이블 형태로 보여 준다. 그리드의 열은 데이터 세트의 컬럼, 행은 레코드를 나타낸다.
TDBNavigator	데이터 세트의 데이터 레코드 사이를 이동, 레코드 업데이트, 삭제 등의 작업을 할 수 있는 비주얼 도구
TDBText	데이터를 라벨의 형태로 보여 준다.
TDBEdit	데이터를 에디트 박스로 나타낸다.

TDBMemo	메모나 BLOB 필드에 저장된 데이터를 메모 형태로 보여준다.
TDBImage	데이터 필드의 내용을 그래픽으로 보여 준다.
TDBListBox	필드에 업데이트할 수 있는 아이템의 리스트를 보여 준다..
TDBComboBox	TDBListBox 와 같으나 직접 텍스트를 입력할 수 있다.
TDBCheckBox	Boolean 필드의 값을 나타내는 체크 박스
TDBRadioGroup	여러 값 중에서 하나의 값만 선택할 수 있는 라디오 그룹 형태의 데이터
TDBLookupListBox	필드의 값에 기초한 다른 데이터 세트의 값을 찾을 수 있다.
TDBLookupComboBox	TDBLookupListBox 와 같으나 직접 텍스트를 입력할 수 있다.
TDBCtrlGrid	여러 데이터 컨트롤을 설정할 수 있는 그리드이다.
TDBRichEdit	데이터 필드를 포맷된 메모의 형태로 보여 준다.

데이터 컨트롤은 디자인 시부터 데이터를 인식할 수 있다. 이렇게 하려면, 데이터 컨트롤의 DataSource 프로퍼티를 활성화된 데이터 소스로 설정하면 되는데, 프로퍼트를 설정하는 것과 동시에 데이터 세트의 내용을 볼 수 있다.

1. 데이터 세트와 데이터 컨트롤의 연결

데이터 컨트롤을 데이터 세트와 연결할 때에는 데이터 소스 컴포넌트를 이용한다. 데이터 소스 컴포넌트는 데이터 세트와 데이터 컨트롤의 통로 역할을 한다.
일단 DataSource 프로퍼티를 설정했으면, DataField 프로퍼티를 설정할 수 있다. 여기서 컨트롤에 보여줄 데이터 필드를 선택한다. DataField 프로퍼티는 TDBGrid, TDBCtrlGrid, TDBNavigator 와 같은 컴포넌트에는 적용되지 않는다.
그리고, 데이터 세트의 Active 프로퍼티를 True 로 설정하면 컨트롤에서 데이터를 볼 수 있을 것이다.

2. 데이터 편집과 업데이트

데이터 컨트롤은 데이터를 보여주는 것 뿐만 아니라, 데이터를 편집하고 변경된 사항을 업데이트 하는 용도로 사용할 수 있다. 이를 위해서는 데이터 세트의 상태가 dsEdit 로 설정되어 있어야 한다.
데이터 컨트롤과 연결된 데이터 소스의 AutoEdit 프로퍼티는 데이터 컨트롤에 키보드 입력이나 마우스 이벤트가 있을 때, 자동으로 데이터 세트의 상태를 dsEdit 로 바꿔줄 것인지를 결정한다. AutoEdit 프로퍼티의 디폴트 값이 True 이기 때문에, 데이터 컨트롤에 작업을 하면 데이터 세트의 상태는 dsEdit 로 변경된다. AutoEdit 프로퍼티가 False 일 경우에는 TDBNavigator 컨트롤을 사용할 경우에는 Edit 버튼을 클릭해야 편집 상태로 들어갈 수 있으며, 아니면 런타임에서 데이터 세트의 Edit 메소드 등을 호출해서 상태를 변경시켜야 한

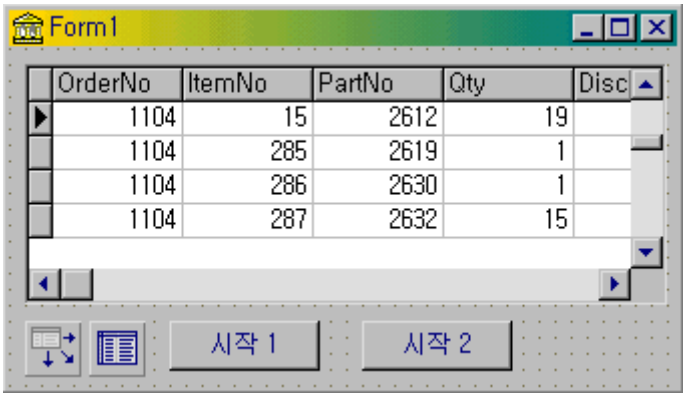
다.
데이터 컨트롤의 ReadOnly 프로퍼티는 컨트롤에 표시되는 데이터를 편집할 수 있는지 여부를 결정하게 된다. 이 프로퍼티의 디폴트 값은 False 로 사용자는 데이터를 편집할 수 있다. 사용자가 데이터를 컨트롤에서 편집하지 못하도록 하려면, 이 값을 True 로 설정해야 한다. 데이터 소스 컴포넌트의 Enabled 프로퍼티도 편집 여부를 결정하는데 중요하다. 이 프로퍼티는 데이터 소스와 연결된 데이터 컨트롤에 값을 표시할 것인지를 결정한다. 이 값이 False 이면 사용자가 값을 편집할 수가 없다. 디폴트 값은 True 이다. 또한, 데이터 세트의 ReadOnly 프로퍼티는 사용자가 데이터 세트의 데이터를 편집할 수 있는지 여부를 결정한다. 디폴트 값은 False 이다.

참고: 데이터 세트에는 읽기 전용, 런타임 프로퍼티인 CanModify 라는 프로퍼티가 있다. 이 값은 편집을 할 수 있을 때 True 로 설정된다.

이렇게 데이터 컨트롤에서 데이터를 편집하면, TDBGrid 를 제외하고는 탭(Tab)키를 누를 때 변경된 내용이 필드 컴포넌트에 복사된다. 탭키를 누르기 전에 Esc 키를 누르면, 데이터 컨트롤은 변경된 내용을 원상복귀시킨다. TDBGrid 컴포넌트는 변경된 내용을 사용자가 다른 레코드로 옮겨갈 때에만 복사한다. 그러므로, 레코드를 변경하기 전에 Esc 키를 누르면 레코드의 모든 변경 사항을 원상복귀한다.

3. 데이터 디스플레이

어플리케이션에서 여러 레코드를 처리하는 경우에, 레코드가 변경될 때마다 이런 내용을 데이터 컨트롤에 표시하려면 화면이 껌뻑이면서 많은 시간이 소요된다. 이럴 때에는 데이터 세트의 DisableControls 메소드를 호출하여 데이터 인식 컨트롤에 레코드 내용을 표시하지 않도록 하는 것이 현명하다. 일단 레코드를 검사하는 부분이 모두 종료되면 다시 EnableControls 메소드를 호출하여 컨트롤에 데이터 세트의 내용을 표시하도록 한다.
지금까지, 설명만 해서 지루할 지 모르겠다. 그러면, 이쯤에서 간단한 예제를 이용하여 데이터 컨트롤을 이용하는 기본적인 방법과 EnableControls 와 DisableControls 메소드를 이용할 때와 이를 그대로 사용할 때의 시간 차이를 재도록 해보자.
먼저 폼에다가 TDBGrid, TDataSource, TTable 컴포넌트를 하나씩 올려 놓고, TButton 컴포넌트를 2 개 올려 놓는다.



Button1, Button2 의 Caption 프로퍼티를 '시작 1', '시작 2'로 설정한다. 여기에서 Button1 을 선택하면 아무런 조치 없이 처음부터 끝까지 레코드를 변경할 것이다. Button2 를 선택하면 데이터 세트(Table1)의 DisableControls, EnableControls 메소드를 사용해서 각각의 시간을 GetTickCount 함수를 이용해서 측정할 것이다.

먼저 테이블과 데이터 컨트롤을 연결하도록 하자. 방법은 간단하다 DBGrid1 의 DataSource 프로퍼티를 DataSource1 으로 설정하고, DataSource1 의 DataSet 프로퍼티를 Table1 으로 설정한다. 실제 테이블을 지정할 차례인데, Table1 의 DatabaseName 에는 DBDEMOS 앨리어스를 선택하여 데모 데이터베이스 테이블을 사용할 수 있도록 한다. TableName 프로퍼티의 드롭-다운 버튼을 클릭하면 여러 테이블 들을 나열하게 되는데, 여기서는 비교적 레코드의 수가 많은 테이블을 선택하는 것이 시간의 차이를 느끼기 좋으므로 레코드 수가 많은 Items.db 테이블을 선택하도록 한다. 이제 Table1 의 Active 프로퍼티를 True 로 설정하면, DBGrid1 컨트롤에 앞의 그림과 같이 레코드가 나타날 것이다.

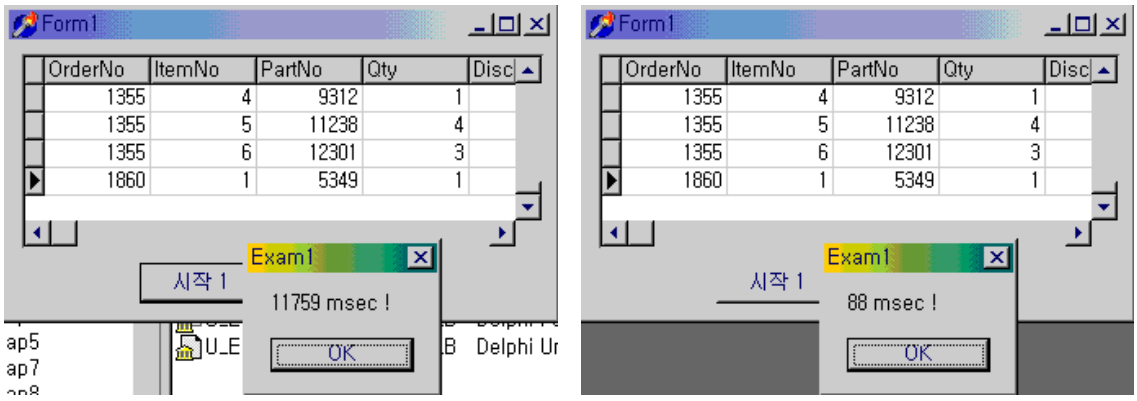
이제 Button1 과 Button2 의 OnClick 이벤트 핸들러를 다음과 같이 작성한다.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Start, Finish: Integer;
begin
  Start := GetTickCount;
  Table1.First;
  while not Table1.Eof do
    Table1.Next;
  Finish := GetTickCount;
  ShowMessage(IntToStr(Finish - Start) + ' msec !');
end;

procedure TForm1.Button2Click(Sender: TObject);
```

```
var
  Start, Finish: Integer;
begin
  Start := GetTickCount;
  Table1.DisableControls;
  Table1.First;
  while not Table1.Eof do
    Table1.Next;
  Finish := GetTickCount;
  Table1.EnableControls;
  ShowMessage(IntToStr(Finish - Start) + ' msec !');
end;
```

이와 같이 데이터 세트의 레코드들 사이를 이동할 때에는 데이터 세트의 First, Next, Previous, Last 메소드를 사용하며 데이터 세트의 레코드가 첫번째 레코드의 이전으로 이동했다면 Bof, 마지막 레코드의 다음으로 이동했다면 Eof 가 설정되므로 이를 이용하여 검색을 하게 된다.
앞의 이벤트 핸들러 코드에서 대부분의 내용이 비슷하지만, Button2 이벤트 핸들러에서 DisableControls, EnableControls 메소드를 호출한 것만 다르다. 어플리케이션을 호출하여 이들 메소드의 효과를 알아보도록 하자. 필자의 컴퓨터에서 실행한 결과는 다음과 같다.



결과는 무려 100 배가 넘는 속도의 차이가 난다. 이는 데이터 세트 자체의 처리 속도보다, 화면에 내용을 뿌리는 데에 소요되는 시간이 월등히 많다는 것을 의미한다. 그러므로, 많은 양의 레코드를 동시에 처리할 때에는 반드시 DisableControls, EnableControls 메소드를 사용할 것을 권하고 싶다.

4. 데이터 리프레쉬 (Refreshing data)

데이터 세트의 Refresh 메소드는 로컬 버퍼를 비우고, 데이터 세트에서 데이터를 다시 가져오는 메소드이다. 보통 이 메소드는 데이터가 변경되었을 것으로 생각될 때, 컨트롤에 표시되는 내용을 새롭게 하기 위해 사용한다. 흔히 여러 명의 사용자가 동시에 데이터를 편집할 경우에 이런 변경 사항을 빠르게 갱신해줄 필요성이 있다.

참고: 리프레쉬가 간혹 예상치 못한 결과를 가져올 때가 있다. 예를 들어, 사용자가 다른 어플리케이션이 삭제한 레코드를 보고 있다면, refresh 메소드를 호출하면 순간적으로 컨트롤에 보이던 데이터가 모두 사라져 버린다.

- Lookup 리스트와 콤보 박스를 이용한 데이터 표시와 편집

TDBLookupListBox 와 TDBLookupComboBox 컴포넌트는 보통 마스터/디테일 테이블의 관계나, 특정 필드 값에 대한 내용을 찾거나 할 때 유용하게 사용된다.
예를 들어, OrderTable 을 보여주는 order 폼이 있다고 하자. OrdersTable 에는 고객의 ID 에 해당되는 CustNo 필드가 있지만(FK, Foreign Key), 회사 이름, 회사 주소, 집 주소, 전화 번호 등의 고객에 대한 정보는 CustomerTable 에 존재하게 된다. 이때 order 폼에서 고객에 대한 정보를 찾을 때, CustNo 필드가 아닌 회사 이름과 같은 고객 정보를 이용해서 검색을 하고자 한다면 어떻게 해야 할까 ? 이럴 때에는 TDBLookupListBox 를 이용해서 CustomerTable 의 모든 회사 이름을 보여 주게 하고, 사용자가 회사 이름을 선택하면 CustNo 의 값을 이용해서 지정된 Order 를 보여주면 된다.

1. Lookup 필드에 기초한 리스트의 지정

Lookup 필드로 사용할 리스트 박스 아이템을 지정하려면 컨트롤에 연결될 데이터 세트에는 반드시 lookup 필드가 정의되어 있어야 한다. 정의되어 있다면, 리스트 박스의 DataSource 프로퍼티를 lookup 필드를 가지고 있는 데이터 세트의 데이터 소스 컴포넌트로 설정하고, DataField 프로퍼티에서 사용할 lookup 필드를 선택하면 된다.
이렇게 하면, lookup 리스트 박스 컨트롤과 연결된 테이블이 활성화될 때, 컨트롤은 데이터 필드가 lookup 필드라는 사실을 알게 되고, 적절한 값을 표시한다.

2. 다른 데이터 소스에 기초한 리스트의 지정

데이터 세트에 대한 lookup 필드를 정의하지 않은 경우에는 다른 데이터 소스 컴포넌트를

이용해서 이를 처리해야 한다. 일단 리스트 박스의 DataSource 프로퍼티를 컨트롤에 대한 데이터 소스로 설정한다. Lookup 값들을 삽입하여 리스트로 보여줄 필드를 DataField 프로퍼티로 지정하는데, 이 필드는 lookup 필드가 아니다. 그리고, ListSource 프로퍼티를 look up 하기 원하는 값들이 있는 필드를 포함한 데이터 세트에 대한 데이터 소스로 설정한다. Lookup 키로 사용하고자 하는 필드를 KeyField 프로퍼티에 설정한다. 이때 KeyField 프로퍼티로 설정한 필드에 인덱스가 되어 있으면 수행속도가 빠르다. 마지막으로 반환하길 원하는 값을 가진 필드를 ListField 로 설정한다.

DBGrid 의 활용

TDBGrid 컴포넌트는 테이블 형태의 그리드에 레코드를 보여주고, 편집할 수 있는 데이터 컨틀이다. 데이터 컨트롤 중에서도 가장 흔히 사용되는 컨트롤이지만, 과거 델파이 3 까지는 기본으로 제공되는 그리드의 기능이 뛰어나지 않아서 많은 수의 공개, 쉐어 컴포넌트가 소개된 바 있다. 그렇지만, 델파이 4 에서 제공되는 DBGrid 는 비교적 뛰어난 성능을 보여주며 그 동안 필요로 해왔던 기능을 많이 추가하였다.
그리드 컨트롤의 Columns 프로퍼티는 TDBGridColumns 객체의 wrapper 이다.
TDBGridColumns 객체는 그리드 컨트롤의 모든 컬럼을 대표하는 TColumn 객체의 컬렉션으로, 개발자는 컬럼 에디터(Columns editor)를 이용하여 디자인 시에 컬럼의 속성을 설정할 수 있다.
Columns.State 프로퍼티는 그리드에 의해 자동으로 설정되는 런타임 프로퍼티로, 디폴트 값은 그리드에 persistent 컬럼 객체가 존재하지 않거나 데이터 세트에 persistent 필드 컴포넌트가 없다는 의미인 csDefault 이다.

- 디폴트 state 에서 그리드 컨트롤 이용하기

그리드의 Columns.State 프로퍼티의 값이 csDefault 이면, 레코드 들은 그리드의 데이터 세트의 각 필드 들의 프로퍼티에 의해서 나타나게 된다. 그리고, 그리드의 컬럼의 순서는 데이터 세트의 필드 순서와 일치한다. 그리드 컨트롤을 이용해서 동적으로 컬럼을 생성해서 테이블의 내용을 보여주면 보다 다양하고, 다양한 형태로 그리드를 활용할 수 있다. 예를 들어, 파라독스 테이블을 처음에는 보여주다가, SQL 쿼리의 결과를 DataSource 프로퍼티를 변경해서 보여주게 하는 등의 변화를 줄 수 있다.

참고: 그리드의 Columns.State 프로퍼티를 런타임에서 csDefault 로 변경할 경우, 그리드에 있는 모든 컬럼 객체가 삭제된다. 그리고, 동적 컬럼을 생성한다.

- Persistent 컬럼의 이해

그리드에 사용하기 위한 persistent 컬럼 객체를 생성하면, 이들은 그리드의 데이터 세트와 느슨하게 연결되어 있다. 꼭 알아두어야 하는 것은 persistent 컬럼은 연결된 필드 컴포넌트와는 독립적이라는 것이다. 물론 디폴트 값을 필드 컴포넌트에서 가져오지만, 독자적인 형태를 가지고 있다. 만약 persistent 컬럼의 FieldName 프로퍼티가 비어 있거나, 필드의 이름이 현재 연결된 데이터 세트의 필드 이름과 맞지 않는 경우 Field 프로퍼티의 값은 NULL 이 되고, 셀의 내용은 비게 된다. 이렇게 비어 있는 컬럼에 비트맵을 그려 넣거나, 다른 문자를 채우는 등의 작업을 셀의 디폴트 드로잉 메소드를 오버라이드하여 구현할 수도 있다.
어떤 경우에는 2 개 이상의 persistent 컬럼을 같은 필드와 연결할 수도 있다. 예를 들어, 넓은 그리드에 특정 부품의 번호를 좌측과 우측 끝에 같이 보여주고 싶을 경우에는 이들 컬럼의 필드를 같은 부품의 번호를 나타내도록 설정하면 된다.

참고:

Persistent 컬럼은 데이터 세트의 필드와 반드시 연결될 필요가 없고, 여러 개의 컬럼이 하나의 필드를 나타낼 수도 있기 때문에 그리드의 FieldCount 프로퍼티는 그리드의 컬럼 수보다 작을 수 있다. 또한, 현재 선택된 컬럼이 필드와 연결되어 있지 않은 경우에는 SelectedField 프로퍼티가 NULL 로 설정된다.

그리드의 형태를 디자인시에 변경하려면, 일단 컬럼 에디터(Column Editor)를 실행하고, 여기에서 persistent 컬럼 객체의 세트를 생성해야 한다. 이렇게 하면, 그리드의 State 프로퍼티는 자동으로 csCustomized 로 설정된다.
컬럼 에디터에서는 컬럼을 추가, 편집, 삭제할 수 있게 되어 있다. 이런 식으로 컬럼 에디터에서 추가된 컬럼의 이름은 기본적인 디폴트 이름과 순서를 가지게 된다 (예: 0-TColumn). 이렇게 새로운 컬럼에 대한 필드를 설정하기 위해 FieldName 프로퍼티를 나타내고자 하는 필드로 지정한다. 또한, 새로운 컬럼의 타이틀을 설정할 때에는 Title 프로퍼티의 Caption 옵션의 값을 지정하면 된다. 컬럼 에디터에 나타나는 컬럼의 순서는 그리드에 나타나는 컬럼의 순서가 되기 때문에, 컬럼을 드래그-드롭하여 컬럼의 순서를 재배치할 수 있다.
런타임에서 컬럼을 추가할 때에는 다음과 같이 하면 된다.

```
DBGrid1.Columns.Add;
```

또한, persistent 컬럼을 삭제하려면 단순히 컬럼 객체를 해제하면 된다.

```
DBGrid1.Columns[5].Free;
```

1. Lookup 리스트 컬럼의 정의

분리된 lookup 테이블에서 값들의 드롭-다운 리스트를 보여주도록 컬럼을 설정하려면 먼저 데이터 세트에서 lookup 필드 객체를 정의해야 한다. 일단 lookup 필드가 정의되면 컬럼의 FieldName 프로퍼티를 lookup 필드로 설정하고, ButtonStyle 프로퍼티를 cbsAuto 로 설정한다. 이렇게 하면, 그리드는 자동으로 콤보 박스와 같은 형태의 드롭-다운 버튼을 이 컬럼의 셀이 에디트 모드로 들어갈 때 보여주게 되며, 이 버튼을 누르면 lookup 필드에 정의된 테이블의 값들을 드롭-다운 리스트에 보여주게 된다.

2. Pick 리스트 컬럼의 정의

Pick 리스트 컬럼은 컬럼 필드가 보통 필드이고, 드롭-다운 리스트가 lookup 테이블이 아닌 PickList 프로퍼티 값들에 의해 마치 lookup 리스트 컬럼처럼 동작한다는 점을 제외하면 lookup 리스트 컬럼과 비슷하다. Pick 리스트 컬럼을 정의하려면, 먼저 컬럼 리스트 박스에서 컬럼을 선택하고, ButtonStyle 프로퍼티를 cbsAuto 로 설정한다. 오브젝트 인스펙터에서 Picklist 프로퍼티를 더블 클릭하여 문자열 리스트 에디터(string list editor)를 불러낸 후, 드롭-다운 리스트에 보여줄 값들을 리스트를 입력한다.

3. 컬럼에 버튼 올려 놓기

컬럼의 정상적인 셀 에디터 우측에 ellipsis 버튼(...)을 올려 놓을 수 있다. Ctrl+Enter 를 누르거나, 이 버튼을 마우스로 클릭하면 OnEditButtonClick 이벤트가 발생한다. 이 버튼을 이용하면 컬럼의 데이터에 대해 보다 자세한 내용을 볼 수 있도록 할 수 있다. 예를 들어, 주문장의 요약 내용을 보여주다가, 합계 컬럼의 ellipsis 버튼을 클릭하면 각 아이템의 내역과 세금 계산 방법 등을 보여주게 할 수 있다. 그래픽 필드의 경우에는 이 버튼을 누르면 이미지를 보여주도록 할 수 있다.
Ellipsis 버튼을 컬럼에 올려 놓으려면, 먼저 컬럼 리스트 박스에서 컬럼을 선택하고 ButtonStyle 프로퍼티를 cbsEllipsis 로 설정한다. 그리고, 마지막으로 OnEditButtonClick 이벤트 핸들러를 작성하면 된다.

4. 주요 컬럼 프로퍼티

컬럼의 프로퍼티는 컬럼의 셀에 데이터를 어떤 식으로 보여줄 것인지를 결정한다. 다음에 컬럼 객체에 대한 주요 프로퍼티에 대해서 설명하였다.

프로퍼티	설 명
Alignment	데이터의 정렬 방법을 결정한다. 연결 필드의 Alignment 프로퍼티가 디폴트 값
ButtonStyle	디폴트 값은 cbsAuto 로 연결된 필드가 lookup 필드이거나, PickList 프로퍼티에 데이터가 있으면 드롭-다운 리스트를 보여준다. cbsEllipsis 로 설정하면, 셀의 우측에 ellipsis 버튼을 보여주는데, 이 버튼을 클릭하면 OnEditButtonClick 이벤트가 발생한다. cbsNone 으로 설정하면 정상적인 에디트 컨트롤로 사용된다.
Color	셀의 배경 색을 결정한다. DBGrid 의 Color 프로퍼티가 디폴트 값이다.
DropDownRows	드롭-다운 리스트에 보여줄 텍스트의 줄 수. 디폴트 값은 7 이다.
Expanded	컬럼이 확장될 수 있는지 결정한다. ADT 나 배열 필드의 컬럼에 적용된다.
FieldName	컬럼과 연결된 필드의 이름을 지정한다.
ReadOnly	True 이면 사용자가 데이터를 편집할 수 없다. 디폴트 값은 False 이다.
Width	컬럼의 폭을 결정한다. 디폴트 값은 연결 필드의 DisplayWidth 값이다.
Font	컬럼 텍스트의 폰트를 지정한다. DBGrid 의 Font 프로퍼티가 디폴트 값이다.
PickList	드롭-다운 리스트에 보여줄 값의 리스트를 결정한다.
Title	선택된 컬럼의 타이틀에 대한 프로퍼티를 설정한다.

- ADT 와 배열 필드 나타내기

데이터 세트의 ObjectView 프로퍼티 값에 따라, 그리드는 ADT 나 배열 필드를 직접 보여주거나, 필드를 펼치거나 접어서 보여줄 수 있다. ObjectView 프로퍼티의 값이 True 이면 객체 필드를 펼치거나 접어서 보여줄 수 있는데, 필드가 펼쳐지면 각각의 자식 필드가 컬럼의 타이틀 바 아래에 나열된다. 필드가 접히면 하나의 컬럼이 자식 필드 들을 포함한 문자열로 나타난다. 필드의 내용을 펼치거나 접는 것은 필드의 타이틀 바에서 화살표를 클릭해서 실행할 수 있다. ObjectView 프로퍼티 값이 False 이면, 각각의 자식 필드가 분리된 컬럼으로 나타난다.

프로퍼티	클래스	설 명
Expandable	TColumn	값이 True 이면 컬럼이 확장되어 분리된 자식 필드를 보여줄 수 있다.
Expanded	TColumn	컬럼이 확장되었는지를 나타낸다.
MaxTitleRows	TDBGrid	그리드에 나타날 수 있는 타이틀 행의 총 수를 결정한다.
ObjectView	TDataSet	필드가 분리된 컬럼으로 보이게 할 지, 객체 필드를 확장하거나 접을 수 있게 할 지를 결정한다.

ParentColumn	TColumn	자식 필드 컬럼을 소유할 TColumn 객체를 지정한다.

- 그리드 옵션의 설정

그리드의 Options 프로퍼티를 이용하면, 그리드의 기본적인 행동양식과 형태를 결정할 수 있다. Options 프로퍼티는 여러 가지 프로퍼티로 구성된 세트이다. 다음에 Options 프로퍼티에서 설정할 수 있는 내용 들을 나열하였다.

옵 션	목 적
dgEditing	디폴트 값은 True 로이다. 그리드에서 레코드를 편집, 삽입, 삭제가 가능하게 할 것인지 여부를 결정한다.
dgAlwaysShowEditor	필드가 선택되면, 자동으로 Edit state 로 될 것인지 여부를 결정한다. 디폴트 값은 False 이다.
dgTitles	그리드 위에 필드 이름을 표시할 것인지 여부를 결정한다. 디폴트 값은 True 이다.
dgIndicator	그리드 좌측에 indicator 컬럼을 표시할 지를 결정한다. 레코드를 삽입하면, 화살표가 별표('*')로 변하고, 편집 시에는 'I'의 형태로 변한다.
dgColumnResize	컬럼 크기를 변경할 수 있는지 여부를 결정한다. 디폴트 값은 True 이다.
dgColLines	컬럼 사이에 라인을 그릴 것인지 여부를 결정한다. 디폴트 값은 True 이다.
dgRowLines	레코드 사이에 라인을 그릴 것인지 결정한다. 디폴트 값은 True 이다.
dgTabs	True(디폴트)이면 필드 사이를 옮길 때 탭키를 사용하며, False 이면 탭키를 누를 때 그리드 컨트롤에서 빠져나간다.
dgRowSelect	True 이면 선택을 할 때 그리드의 한 레코드 전체가 선택되며, False(디폴트)이면 레코드의 특정 필드만 선택된다.
dgAlwaysShowSelection	True(디폴트)이면 다른 컨트롤에 포커스가 있더라도 선택된 부분이 보이게 하며, False 이면 그리드에 포커스가 있을 때에만 보인다.
dgConfirmDelete	레코드를 삭제할 때 물어볼 지 여부를 결정한다. 디폴트 값은 True 이다.
dgCancelOnExit	포커스가 그리드에서 벗어날 때 레코드를 삽입하는 도중 이었다면, 이 레코드의 삽입을 취소할 지 여부를 결정한다. 디폴트 값은 True 이다.
dgMultiSelect	사용자가 그리드에서 인접하지 않은 레코드 들을 여러 개 선택할 수 있는지 여부를 결정한다. 디폴트 값은 False 이다.

- 그리드에서 편집하기

런타임에서 데이터를 수정하고, 삽입할 수 있게 하려면, 데이터 세트의 CanModify 프로퍼티는 True 이어야 하며, 그리드의 ReadOnly 프로퍼티가 False 이어야 한다.

사용자가 레코드를 편집할 때, 각 필드의 변경된 내용은 내부의 레코드 버퍼에 저장된다. 그렇지만, 다른 레코드로 넘어가지 전에는 내용이 post 되지 않는다. 포커스가 그리드에서 다른 폼으로 넘어가더라도 이 내용은 레코드가 변경되기 전에는 post 되지 않는데, 레코드가 post 되기 전에 Esc 를 누르면 변경된 내용을 취소할 수 있다.

- 런타임에서 사용자 행동에 반응하기

그리드를 이용하여 작업을 할 때, 사용되는 이벤트에는 어떤 것들이 있을까 ? 그리드에서는 사용되는 이벤트도 꽤나 많을 것이다. 다음에 그리드에서 흔히 사용되는 이벤트에 대해서 정리해 보았다.

이벤트	설 명
OnCellClick	그리드에서 셀을 클릭하면 발생한다.
OnColEnter	그리드의 컬럼으로 이동할 때 발생한다.
OnColExit	그리드의 컬럼에서 떠날 때 발생한다.
OnColumnMoved	사용자가 컬럼을 새로운 위치로 이동할 때 발생한다.
OnDblClick	그리드를 더블 클릭하면 발생한다.
OnDragDrop	그리드에서 드래그-드롭을 할 때 발생한다.
OnDragOver	그리드에서 드래그를 할 때 발생한다.
OnDrawColumnCell	어플리케이션이 개개의 셀을 그릴 필요가 있을 때 발생한다.
OnDrawDataCell	이전 버전에서 사용하던 이벤트로, 어플리케이션이 State 가 csDefault 일 때 개개의 셀을 그릴 필요가 있을 때 발생한다.
OnEditButtonClick	컬럼에서 ellipsis 버튼을 클릭할 때 발생한다.
OnEndDrag	그리드에서 드래깅을 중지할 때 발생한다.
OnEnter	그리드가 포커스를 가질 때 발생한다.
OnExit	그리드가 포커스를 잃을 때 발생한다.
OnStartDrag	그리드에서 드래깅을 시작할 때 발생한다.
OnTitleClick	컬럼의 타이틀을 클릭할 때 발생한다.

알아두면 유용하다 !

데이터베이스 어플리케이션을 델파이를 이용하여 작성할 경우에 가장 많은 고려 대상이 되는 부분이 프로그램의 사용자 인터페이스이다. 개발기간이 짧고 간단한 어플리케이션의 경우 델파이의 기본적인 데이터 접근 컴포넌트 들과 각종 데이터 인식 컨트롤을 사용하여 프로그램을 디자인하게 된다.
그렇지만, 대형 SQL 을 사용하기 위한 세션의 처리 사용자 로그인 등의 작업을 고려해야

한다면, 이런 데이터 인식 컨트롤을 사용하지 않는 것을 권한다.
델파이 초보자들은 TTable 을 사용하여 데이터 접속을 유지한 상태에서 데이터 인식 컨트롤을 사용하여 프로그램을 작성하는 경우가 많다. 필자가 주변에 확인해본 바로는 대다수의 전문 프로그래머들 역시 데이터 모듈에 TDatabase, TQuery, TDataSource 컴포넌트를 각각 하나씩 올려놓고 데이터 인식 컴포넌트는 직접 사용하지 않는 경우가 많다.
물론, 이런 컨트롤을 잘 사용하면 문제가 없겠지만, 많은 프로그래머 들이 자신들이 직접 각종 컨트롤에 데이터를 표시하도록 코딩하는 이유는 내부적인 기능에 대한 철저한 연구와 윈도우의 고질적인 문제점인 리소스 문제를 해결하기 위함이고, 사용자가 원하는 시점에서 데이터를 조절하기 쉽도록 하기 위해서이다.

대량의 레코드 관리 요령

간단한 형태의 레코드를 다룰 때에는 고려하지 않아도 될 내용을, 레코드의 크기가 커질 때에는 이를 효율적으로 사용하기 위해 조금은 특별하게 처리할 필요가 있다. 이때 많이 사용되는 기법으로는 다음의 두가지가 있다.

- 마스터/디테일 형태

이 형태는 주가 되는 마스터 테이블의 세부적인 내용을 포함하는 디테일 테이블을 연결해서 사용하는 것이다. 예를 들어, 고객관리 프로그램의 경우 고객명단을 확인하면서 각 고객에 대한 정보나, 물건과 돈의 입출금 내역 등을 확인하는 형태를 말한다. 보통 DBGrid 를 많이 이용한다.

- 드릴-다운(Drill-down) 형태

이 형태는 마스터/디테일 연결 방법에서 한단계 발전한 것으로 요즈음 각광 받는 의사결정 시스템의 기초적인 단계로 활용할 수 있다. 마스터/디테일 연결에서는 사용자가 데이터를 바라보는 시점이 개발자가 정해준 시점으로 밖에 볼 수 없으며 다양한 폼으로 구성된 개별적인 데이터로서만 확인할 수 있다.
그렇지만 드릴-다운 형태를이용하면 사용자가 원하는 조건과 형식으로 데이터를 바라볼 수 있다. 델파이에서는 Decision Cube 라는 컴포넌트 세트가 이런 역할을 담당한다.

데이터 분석과 문서 작성

델파이에서는 좀더 통계적인 자료를 시각화 시켜줄 수 있는 TChart 컴포넌트가 제공된다. 이 컴포넌트는 DBAware 를 지원하는 TDBChart 로 확장이 되어있는데, 이 차트를 사용하

면 여러가지 자료들을 각종 그래프로 분석하여 자료를 제시하여 줄 수 있기 때문에 시각적인 효과를 높일 수 있다.
또한, 데이터베이스에 저장된 내용을 문서로 표시하기 위해서는 델파이 초기 버전에서는 쉐어웨어로 팔리던 퀵 리포트를 델파이 3 에서부터 정식 컴포넌트로 등록을 하여, 델파이에서 간단한 문서부터 다양한 문서의 형태까지 지원한다. 또한, 따로 퀵 리포트를 구입하면 여러가지 다양한 리포트 디자이너를 사용할 수 있다.
필자는 소규모의 데이터베이스를 사용하는 경우에는 퀵 리포트를 사용하지만 국내의 현실상 이것 만으로는 만들어 내기가 힘든 문서가 많다고 생각한다. 그렇기 때문에, 대규모의 문서와 일반문서와 똑같은 형태로 구성되는 문서를 만드는 경우에는 Crystal Report 를 유용하게 사용하고 있다.

정 리 (Summary)

이번 장에서는 델파이가 지원하는 데이터베이스 모델과 여러가지 컴포넌트의 사용 방법에 대해서 간단하게 설명하였다. 개념적인 내용을 이해한다면 이번 장의 목적은 달성한 것이라고 보겠다. 다음에 이어지는 여러 장에서는 구체적인 예제를 가지고, 데이터베이스 어플리케이션을 제작하는 기법들을 알아볼 것이다.