Borland®

볼랜드 Kylix와 IBM DB2 7.2를 이용한 데이터베이스 애플리케이션 개발

Paul Yip, DB2 Business Partner Enablement Services
IBM Toronto Labs

Ramesh Theivendran, Technical Lead, Connectivity
Borland RAD Products

목차

서론

볼랜드 Kylix는 리눅스와 윈도우 플랫폼(볼랜드 Delphi 및 C++Builder)을 위한 사용자와 상호작용하며 이벤트 기반의 데이터베이스 애플리케이션을 빠르게 개발하는 데에 사용할 수 있는 가볍고 크로스플랫폼이며 데이터베이스 독립적이며 비연결 데이터액세스 모델(dbExpress)을 제공하는 RAD (Rapid Application Development) 툴입니다.

IBM DB2 Universal Database (DB2)는 세계에서 가장 많은 고급 애플리케이션 개발자들이 사용하는 업계를 주도하는 데이터베이스 관리 시스템입니다. DB2는 리눅스, AIX, HP-UX, Sun Solaris 및 윈도우 OS를 포함한 다양한 플랫폼에서 지원됩니다 (모두 동일한 코드에 기반하고 있습니다.).

Kylix와 DB2를 결합시키면 애플리케이션을 한번 작성하여 리눅스와 윈도우에서 최적의 성능과 유연성을 위하여 코드의 변경 없이 네이티브하게 실행되도록 배포할 수 있습니다.

Kylix™

white paper

이 글은 Kylix와 DB2 Universal Database를 가장 잘 통합할 수 있는 방법을 알리기 위한 것입니다. 애플리케이션을 실제로 개발하기 전에 이 글을 검토함으로써 애플리케이션을 디자인할 때 이 글에서 설명하는 최적의 실천 방안을 고려해 볼 것을 권합니다.

이 글은 완성된 것이 아닙니다. 경험이나 외부로부터 배워가면서 새로운 토픽, 요령, 아이디어 등이 추가될 것입니다. 의견이나 제안이 있으신 분은 Paul Yip (ypaul@ca.ibm.com) 또는 Ramesh Theivendran (rtheivendran@borland.com)에게 연락하시기 바랍니다.

이 글은 두 부분으로 나뉘어 있습니다. 1부는 Kylix와 dbExpress 데이터베이스 액세스 레이어의 토대를 설명하고, 2부는 시나리오/해법의 접근 방식을 사용하여 Kylix와 DB2를 통합하기 위한 가장 좋은 방법을 강조함으로써 문제의 식별과 가능한 해법의 발견을 더 쉽게 하였습니다.

이 글의 대상

이 글은 DB2를 위한 Kylix 애플리케이션을 작성하고자 하는 애플리케이션 개발자들과, 다른 데이터베이스를 위하여 작성된 애플리케이션을 DB2를 위하여 최적화하는 방법을 알고자 하는 사람들을 위한 것입니다. 여기서 우리는 여러분이 DB2 UDB v7.2 Personal Edition (PE)이나 그 이상의 버전으로 작업하는 것으로 가정합니다. FixPak 3이 적용된 DB2 UDB v7.1은 DB2 UDB v7.2와 같으며, 이 제품들에 대한 가장 최신의 패치를 사용할 것을 권장합니다. 여기서 DB2 UDB Enterprise-Extended Edition (EEE)의 자세한 내용을 다루지 않지만, 애플리케이션은 일반적으로 EE에서 EEE로 수정하지 않아도 이식성이 있습니다.

이 글을 읽는 사람은 Kylix와 SQL을 잘 알고 있으며 기본적인 DB2 기술을 갖추고 있는 것으로 가정합니다.

1부: dbExpress를 이용한 n-티어 데이터 액세스

데이터 액세스, 데이터 리모팅과 데이터 조작은 n-티어 데이터 액세스 모델의 세 가지 주요 요소입니다. Kylix와 Delphi, C++Builder에는 데이터베이스 연결을 위한 컴포넌트들을 많이 가지고 있으므로, 개발자들이 n-티어 데이터베이스 애플리케이션을 작성하기가 아주 쉽습니다.

데이터 액세스

Kylix와 Delphi 6, C++Builder6는 새로운 dbExpress를 도입했는데, 이것은 크로스플랫폼이고, 데이터베이스 독립적이며, 동적인 SQL 처리를 위한 확장 가능한 인터페이스입니다. Borland Database Engine(BDE), ODBC, JDBC 및 ADO 등과 같은 다른 데이터 액세스 기술과 비교해 볼 때, dbExpress의 장점은 다음과 같습니다.

dbExpress는 단방향 커서만을 반환하므로 조회된 데이터에 대해서 캐싱을 하지 않습니다. DataSnap 컴포넌트는 ClientDataset 기술과 함께 각 클라이언트에서의 결과 세트에 대한 캐싱, 스크롤링, 인덱싱, 필터링에 사용할 수 있습니다.

dbExpress는 메타데이터 캐싱을 하지 않으며 디자인타임에서의 메타데이터 액세스 인터페이스는 핵심적인 데이터 액세스 인터페이스를 사용하여 구현됩니다. DataSnap 컴포넌트는 최소한의 런타임 메타데이터를 추출함으로써 데이터 세트가 데이터베이스로 효율적으로 롤백하도록 합니다.

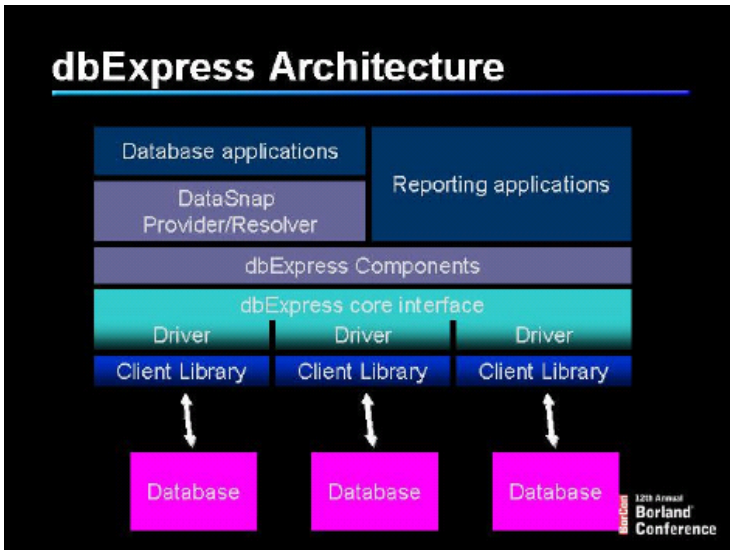
SQL 데이터베이스의 런타임 성능은 내비게이션, Blob 액세스, 메타데이터 추출에 필요한 내부 쿼리 생성의 영향을 받습니다. dbExpress는 사용자가 요청한 쿼리만을 실행하므로 추가적인 쿼리를 도입하지 않고 데이터베이스 액세스를 최적화합니다.

dbExpress는 내부적으로 레코드 버퍼를 관리하며 클라이언트에게 개별적인 필드 값을 제공합니다. 이것은

클라이언트에서 할당한 레코드 버퍼를 관리하는 것보다 오류 발생 가능성이 적습니다.

요약하면, dbExpress의 특징은 다음과 같습니다.

- 향상된 런타임 성능
- 적은 환경설정
- 배포가 용이함
- 데이터베이스 기능에 대한 향상된 액세스
- 새로운 데이터 소스에 대한 손쉬운 적응



데이터 리모팅과 조작

Kylix와 Delphi/C++Builder에 포함된 DataSnap 구조는 n-티어 모델에서 데이터 리모팅과 조작에 필요한 기능을 제공합니다. 데이터셋 프로바이더, 리졸버 및 ClientDataset은 DataSnap의 세 가지 주요 컴포넌트입니다. dbExpress로부터의 데이터는 XML이나 이진 형식의 데이터 패킷으로 스트리밍 되어 클라이언트 애플리케이션으로 전달됩니다. ClientDataset은 클라이언트 애플리케이션에 데이터를 보여주고 데이터에 대한 변경 사항을 추적합니다. 클라이언트가 업데이트를 계속 유지하려고 하는 경우에는 변경 사항이 델타 패킷(XML 또는 이진 형식)으로 스트리밍되어 리졸버로 보내어집니다. 마지막으로 리졸버는 이러한 변경 사항을 받아서 데이터베이스 서버에 데이터를 저장합니다.

2부: Kylix와 DB2의 통합 요령

여기에서는 DB2와 Kylix의 통합을 진척시킬 수 있는 것으로서 애플리케이션의 성능을 향상시킬 수 있으면서도 덜 알려진 중요한 내용을 다룹니다. 시나리오/해법의 접근 방식을 사용하여 주제를 제시하고 있으므로 문제를 식별하고 가능한 해법을 발견하기가 더 쉽습니다.

다중 사용자 애플리케이션에서 locking/hanging 문제가 발생합니다

설명:

잠금 대기는 같은 데이터를 공유하는 다중 사용자 애플리케이션에 있어서 자주 발생하기는 하지만 피할 수 없는 문제는 아닙니다. 어느 한 사용자가 자주 커밋하지 않고 업데이트하는 경우 그 데이터를 필요로 하는 다른 사용자들은 첫번째 사용자가 변경 사항을 커밋할 때까지 잠길 수 있습니다. 이 문제에 대한 가능한 해법은 많으며, 다음과 같은 규칙을 활용합니다:

- 작업에 필요한 이상의 데이터를 사용자에게 보여주거나 제공하지 말 것
- 가능하면 애플리케이션 작업을 자주 커밋하도록 할 것
- 사용자에게 트랜잭션의 시작과 종료 제어 능력을 허용하지 말 것
- 가능하면 테이블의 고유 키를 사용하여 업데이트할 것
- 쿼리 애플리케이션에 필요한 최소한의 격리 수준을 사용할 것

잠금 대기는 예를 들어 다음과 같은 사건들이 연속해서 일어날 때 발생할 수 있습니다.

사용자 1: 테이블 T1에 있는 행을 즉시 커밋하지 않고 업데이트한 후에 점심 먹으러 나갔다가 쇼핑까지 한다.

사용자 2: 사용자 1이 방금 업데이트한 바로 그 행을

선택하는 SELECT 문장을 내었지만 사용자 1이 작업을 커밋할 때까지 DB2가 데이터로 응답할 수 없으므로 애플리케이션 잠금 대기를 겪게 된다. 잠금 대기는 사용자 1의 애플리케이션이 커밋될 때(아마도 몇 시간 후)까지 계속된다. (기본 설정은 무한정)

사용자 1: 트랜잭션을 커밋한다. (또는 롤백을 실행한다)

사용자 2: 애플리케이션을 재개한다.

만약 애플리케이션 지체가 생겨서 그 원인이 데이터베이스 잠금 대기 때문인지 알고 싶으면 다음과 같이 입력합니다.

```
db2 list applications show detail
```

애플리케이션의 연결 및 상태의 목록이 나타날 것입니다. ***lock-wait*** 상태의 연결은 잠금 쟁탈로 인한 지체가 있는 것입니다.

참고: 이 명령을 위해서는 SYSADM, SYSCTRL 또는 SYSMAINT 특권이 필요하며, 원격 데이터베이스로 작업하는 경우에는 먼저 인스턴스에 ATTACH할 필요가 있습니다.

해법 1: 자주 커밋하고 짧은 트랜잭션만 사용한다.

가장 좋은 해법은 모든 트랜잭션이 애플리케이션에 의해서 제어되도록 하고, 짧으며(1초 이하로 실행), 자주 커밋되도록 하는 것입니다. 특히 데이터가 자주 업데이트/삽입되고 여러 사용자가 공유해야 하는 경우 더욱 그렇습니다.

해법 2: Uncommitted Read (UR) 문장수준 격리를 사용한다.

비즈니스 로직이 허용하거나 위험을 견딜 수 있는 경우, 문장수준 격리(statement level isolation)를 사용합니다. DB2가 사용하는 기본 격리는 Cursor Stability이며, 이것은 애플리케이션이 지저분한 데이터(다른 애플리케이션이 업데이트 또는 삽입한 커밋되지 않은 데이터)를 보도록 허용하지 않습니다.

예: 다음 쿼리의 결과를 DBGrid Control에 표시하고자 하는 경우를 가정해봅시다.

```
SELECT * FROM T1
```

예상 결과에는 다른 사람이 업데이트하고 있는 과정에 있는 행이 많이 포함되어 있을 수 있습니다. 쿼리가 현재 작성된 방식에서는 DBGrid에서 리프레시가 있을 때마다 애플리케이션은 모든 애플리케이션이 커미될 때까지 잠금 대기 상태에 들어갈 것입니다. 따라서, 이에 대한 가능한 해법은 쿼리를 다음과 같이 수정하는 것입니다.

```
SELECT * FROM T1 WITH UR
```

UR 문장수준 격리가 있는 SELECT 문장은 절대 잠금 대기되지 않습니다. 하지만 다른 사용자들이 커밋하지 않은 데이터의 커밋되지 않았거나 변경되었거나 새로운 행을 볼 수도 있습니다. 그러나, 이 방법은 잠재적으로 많은 문장의 구문을 변경해야 하는 경우에는 어수선해 질 수 있습니다.

해법 3: 필요 이상의 연결을 사용하지 않는다.

DB2에서의 기본 격리 수준은 Cursor Stability이며, 이것은 지저분한 데이터를 보도록 허용하지 않습니다. 사용자 혼자서 데이터를 보기 위해서 연결 하나를 사용하고 업데이트를 위해서 별도의 연결을 사용하지 말아야 합니다. 한 사용자의 애플리케이션이 n개의 연결을 유지하고 있는 경우 DB2에는 마치 n명의 사용자가 연결되어 있는 것처럼 보이므로 각자의 동작으로부터 보호되게 될 것입니다.

해법 4: DB2_RR_TO_RS 레지스트리 변수를 사용한다.

다음키 잠금(DB2에서 Repeatable Read 격리 수준을 지원하기 위해 사용됨)이 잠금 대기의 원인이 되는 경우도 있습니다. 애플리케이션이 Repeatable Read 격리 수준을 사용하지 않는다면 DB2_RR_O_RS 레지스트리 변수를 다음키 잠금을 감소시키도록 설정해도 안전합니다. 이러한

변경은 인스턴스 전체에 걸친 것이므로 해당 인스턴스에 있는 모든 데이터베이스에 적용됩니다. 애플리케이션이 Repeatable Read 격리 수준을 사용하려는 경우에는 자동으로 Read Stability 격리 수준으로 내려갑니다.

다음키 잠금을 감소시키려면 커맨드 라인에서 다음과 같이 입력합니다.

```
db2set DB2_RR_TO_RS=yes
```

그 후, 인스턴스를 다시 시작하면 변경 사항이 적용됩니다.

```
db2stop
db2start
```

해법 5: 잠긴 행을 인덱스로 무시한다.

SQL을 작성한 후에 최소한의 결과 셋 행만을 읽어들이도록 인덱스를 생성합니다. 그러면 사용 가능한 인덱스들을 이용하여 배타적인 행 잠금을 피할 수 있는 확률이 커집니다. 아이덴티티 값을 나타내는 열에 대한 인덱스를 생성하는 것은 분명히 이점이 있습니다. 작은 테이블에서 유니크하지 않은 열에 대해 인덱스를 생성하는 것은 이점이 크지 않지만 잠금 대기의 확률은 감소시킵니다.

아래와 같이 id 열에 인덱스가 있고 val 열에 인덱스가 없는 작은 테이블을 예를 들어 봅니다.

T1

id	val
1	a
2	b
3	c
4	d
5	e

사용자가 두 명이라고 가정하면 다음과 같은 일련의 이벤트가 발생합니다.

사용자 1: `UPDATE T1 SET val = 'X' where id=3`
(아직 커미트하지 않음)

사용자 2: `SELECT id from T1 where val= 'e'`
(잠금 대기됨)

DB2에서 쿼리의 셋을 생성할 수 있는 유일한 방법은 tablescan(각 행을 처음부터 끝까지 읽는 것)을 이용하는 것이므로, 사용자 2는 사용자 1이 커미트할 때까지 잠금 대기될 것입니다. 이 작업 중에 사용자 2는 사용자 1에 의해서 잠긴 행에서 대기하게 됩니다. 테이블이 작기 때문에 보통의 경우라면 val 열에 대해 인덱스는 그다지 의미가 없을 것입니다. 하지만, 이런 경우에는 val 열에 대하여 인덱스를 생성하면 인덱스를 필요한 행을 직접 추출하는 데에 사용할 수 있으므로 사용자 2가 잠금 대기 없이 작업을 진행할 수 있습니다.

인덱스를 생성한 후에는 인덱스를 사용할 수 있다는 것을 DB2가 알 수 있도록 테이블에 대하여 반드시 'runstats'를 실시해야 합니다. 스토어드 프로시저(또는 정적 SQL 애플리케이션)를 이용하는 경우에는 해당 패키지를 재구축 또는 재 바인드합니다.

해법 6: 잠금 대기 시간 경과 조정

DB2는 기본적으로 애플리케이션이 영구적으로 잠금 대기하도록 합니다. 다시 말해서, 애플리케이션은 잠금이 해제될 때까지 무한히 대기할 수 있습니다. 데이터베이스 구성 파일(db cfg)을 업데이트하면 이러한 것을 변경할 수 있습니다.

커맨드 라인에서 데이터베이스 관리자 id를 사용하여 다음과 같은 명령을 입력합니다.

```
db2 update db cfg for <dbname> using
locktimeout 10
```

locktimeout을 10으로 변경하면 트랜잭션이 10초 이상 잠금에서 대기한 애플리케이션은 롤백됩니다.
애플리케이션은 데이터베이스 예외를 처리할 수 있으며 사용자가 나중에 다시 작업을 시도하도록 합니다. 물론

locktimeout의 값은 원하는 대로 정할 수 있습니다 (-1을 사용하면 DB2가 무한 대기의 기본 설정으로 되돌아갑니다).

잘 처리된 롤백이 장시간의 데이터베이스 잠금 대기(이것은 사용자에게 시스템이 중지된 것처럼 보이기 때문에 사용자가 애플리케이션을 죽이고 재부팅할 생각을 하게 됩니다)보다 낫기 때문에 이 값을 설정하면 사용자에게 도움이 됩니다.

참고: 데이터베이스 구성 변경이 적용되려면 모든 애플리케이션이 데이터베이스과의 접속을 중단해야 합니다.

결과 셋이 커서 애플리케이션의 응답이 느려졌습니다

설명:

애플리케이션이 테이블 내용을 수백개(혹은 그 이상)의 행을 결과로 돌려줄 수 있는 쿼리로 사용자에게 볼 수 있도록 허용해야 할 경우가 있습니다. 사용자는 한번에 수십 행 정도만을 처리할 수 있을 수 있습니다. 하지만 쿼리를 어떻게 제한하더라도, 결과 셋은 적절한 양보다 많은 행을 반환할 수 있습니다. 이것은 결과 셋이 클수록 DB2가 데이터를 반환하는 시간도 더 오래 걸리고, Kylix에서도 많은 행을 캐시하는 데 오랜 시간이 걸리므로 사용자에게 좋지 않은 경험이 됩니다.

다음과 같은 일반적인 쿼리에서,

```
SELECT * FROM T1 WHERE condition
```

condition은 결과 셋의 크기를 보장하지 않습니다.

해법 1

쉬운 해결 방법은 쿼리를 다음과 같이 수정하는 것입니다.

```
SELECT * FROM T1 WHERE condition
  OPTIMIZE FOR n ROWS
```

이것은 DB2가 처음의 n개의 유효한 행을 가능한 빨리 가져오도록 하는 액세스 플랜을 이용하여 쿼리를 실행하도록 합니다. 사용자가 처음의 n개의 행으로 작업하는 동안 DB2는 백그라운드에서 나머지 행을 계속 처리합니다. 이 쿼리 모디파이어를 사용할 때에는 나머지 행을 검색하는 비용이 모디파이어를 사용하지 않았을 경우보다 더 클 수 있다는 것을 염두에 두어야 합니다. (중요한 것은 사용자가 느끼는 경험이기 때문에 이것은 중요하지 않을 수 있습니다. 사용자가 처음의 n개의 행을 처리하는 동안 DB2는 백그라운드에서 필요한 행을 이미 모두 성공적으로 검색했을 가능성이 큽니다.)

해법 2

Kylix는 DB2와 연동하기 위해 DB2 CLI 네이티브 인터페이스를 사용하며, 열린 커서는 절대 위치 업데이트를 하기 위해 사용되지 않습니다. 따라서, 우리는 쿼리를 모호하지 않게 할 수 있으며, DB2가 블록 가져오기(block fetching; 행을 하나씩 가져오는 대신 그룹으로 가져옴)를 사용하여 SELECT 성능을 크게 향상시킬 수 있습니다.

이를 위하여 select 문장에 FOR READ ONLY 모디파이어를 사용합니다. 예를 들면 다음과 같습니다.

```
SELECT * FROM T1 WHERE condition
  FOR READ ONLY
```

이 시나리오에서 해법 1과 해법 2를 다음과 같이 결합할 수 있습니다.

```
SELECT * FROM T1 WHERE condition
  OPTIMIZE FOR n ROWS FOR READ ONLY
```

테이블/뷰를 생성하는 데 사용된 스키마가 사용자 ID와 일치하지 않습니다

설명

이것은 2-티어 애플리케이션(또는 복잡한 배포 조건이 따르는 n-티어 애플리케이션)에 흔한 문제인데, 많은 사용자들이 그들 자신의 고유한 사용자 ID를 가지고

데이터베이스에 연결하지만, 테이블은 모두 단일 스키마로 생성된 경우입니다. 여기에서의 문제는 사용자가 DB2 데이터베이스에 연결할 때 사용자의 스키마가 사용자의 ID를 기본값으로 간주한다는 것입니다. 예를 들어, 해당 애플리케이션에 대한 모든 테이블이 X 스키마에 따라 생성되었고 사용자 USER 1 아래의 쿼리를 실행하는 경우를 가정합니다.

```
SELECT * FROM T1;
```

테이블 이름이 X로 수식되지 않으므로, DB2는 이 명령을 SELECT * FROM USER1.T1라고 해석하게 됩니다.

참고: 이 문제를 해결하기 위해서 애플리케이션에 있는 모든 테이블 이름에 수식명을 붙이려고(하드코드) 할 수도 있습니다. 유연성과 유지성 측면에서 볼 때 이것은 스키마 이름이 절대로 변하지 않을 것이 아닌 한 권장할 만한 방법이 아닙니다(재판매할 애플리케이션을 생성하는 경우에는 가능성이 적습니다). 아래에 제시하는 각 해법에서 쿼리는 완전한 수식명을 사용하지 않는 것으로 가정합니다.

해법 1

모든 새로운 연결에 대하여 다음을 실행합니다.

```
SET CURRENT SCHEMA <대상 스키마>
```

이후의 모든 쿼리는 수식명이 없는 데이터베이스 객체에 대해 이 새 스키마로 간주하게 됩니다.

해법 2

사용자의 액세스가 허용된 테이블에 대한 별명을 생성할 수 있습니다. 예를 들어, 애플리케이션에서 사용하는 테이블의 이름이 아래와 같은 경우,

```
ACCT.table1
ACCT.table2
ACCT.view1
```

각 사용자에 대하여 각 테이블의 별명을 다음과 같이 정합니다.

```
CREATE ALIAS <userid>.table1 FOR ACCT.table1
CREATE ALIAS <userid>.table1 FOR ACCT.table1
CREATE ALIAS <userid>.table1 FOR ACCT.view1
```

해법 3

각 클라이언트 컴퓨터의 sqllib 디렉토리에 있는 DB2 CLI 구성 파일인 db2cli.ini를 수정하여 아래의 내용을 추가합니다.

```
[myAppDB]
CURRENTSCHEMA=X
```

참고: 파일의 마지막 줄 뒤에 빈 줄이 있어야 합니다. 그렇지 않으면 DB2가 db2cli.ini 파일을 올바로 분석하지 못할 수도 있습니다.

위의 예에서 괄호 안의 myAppDB는 데이터베이스 이름입니다. CURRENTSCEMA=X는 연결하는 사용자 ID에 상관 없이 DB2가 자동으로 기본 스키마 이름을 X로 변경하도록 합니다. 이러한 스키마 변경은 설정이 적용된 후에 클라이언트와 모든 새로운 연결에 적용됩니다.

컬럼에 별명을 정하고 싶지만 별명이 있는 컬럼은 업데이트할 수 없습니다

설명

데이터베이스 레벨에서는, 테이블의 컬럼 이름은 간혹 줄여 쓰거나 약자로 쓰는 경우가 있습니다. SELECT 문에서 사용되는 컬럼에 별명(Alias)을 붙여서 컬럼 이름이 사용자에게 더 친숙하게 할 수 있습니다. 예를 들어, 학생 데이터를 가진 테이블은 아래와 같이 생성할 수 있습니다.

```
Create table Student (sid int, fname
  varchar(20), lname varchar(20))
```

데이터를 추출하기 위하여 컬럼 별명을 사용하는 쿼리는 아래와 같습니다.

```
SELECT sid as StudentId, fname as
```

```
FirstName, lname as LastName from STUDENT
WHERE .....
```

Kylix는 위치 업데이트를 하기 위해 커서를 사용하지 않으므로, 컬럼에 별명이 주어지면 UPDATES를 실행하려고 할 때 문제가 발생한다는 것을 금방 알아차렸을 것입니다. 원래의 SELECT 쿼리에 기초한 직접 UPDATE 문장이 런타임에 생성됩니다. 예를 들어, ApplyUpdates()를 호출하면, 위의 SELECT 문장을 사용했고 행의 처음과 마지막 이름을 studentid=1000으로 수정하려고 하는 경우 다음과 같은 쿼리가 생성됩니다.

```
UPDATE FirstName='foo', LastName='bar'
  WHERE StudentId=1000
```

물론 위의 쿼리는 실제의 컬럼 이름 대신에 별명을 사용하였으므로 실행되지 않습니다.

해법 1

쉬운 해결 방법은 컬럼 이름을 줄여 쓰거나 약자로 쓰지 않는 것입니다. 그러면 절대로 별명이 필요하지 않습니다.

해법 2

DB2(그리고 대부분의 다른 데이터베이스)에서는 특정 조건만 맞으면 뷰를 삭제, 업데이트, 삽입할 수 있습니다. 이러한 조건에 대한 설명을 보려면 DB2 SQL 레퍼런스에서 CREATE VIEW를 찾아보십시오.

위의 시나리오에 따라, 다음과 같이 뷰를 생성할 수 있습니다.

```
Create view student_v as
  SELECT sid as StudentID, fname as
    FirstName, lname as LastName
  FROM STUDENT WHERE ....
```

이제, 새로운 쿼리는 아래와 같은 형태일 것입니다.

```
SELECT * FROM STUDENT_V ..........
```

일반적인 오류와 해법

여기에서는 DB2를 실행하는 중에 Kylix 개발자가 겪을 수 있는 일반적인 문제들을 다룹니다.

테이블/뷰가 존재하지 않습니다

```
(SQL0204N <tabname> is an undefined name. SQLSTATE=42704)
```

DB2를 포함한 일부 데이터베이스 관리 시스템은 테이블 이름의 대소문자를 구분합니다. DB2 객체 이름은 생성할 때 이름을 따옴표로 묶은 경우에만 대소문자를 구분합니다. 예를 들어 다음과 같이 했다면,

```
CREATE TABLE "t1" (c1 int)
CREATE TABLE "T1" (c1 int)
```

두 개의 테이블이 성공적으로 생성됩니다. 하지만, 만약 따옴표를 사용하지 않는다면 DB2는 두번째 라인에서 실패할 것입니다.

DB2 객체를 다루는 데에 있어서 가장 좋은 방안은 쿼리에 대문자 이름을 사용하고 객체를 생성할 때 대소문자 구분에 전혀 의존하지 않는 것입니다 (즉, 따옴표를 사용하지 않는 것입니다). 객체를 생성할 때 따옴표를 전혀 사용하지 않으면 DB2는 마치 대소문자를 구분하지 않는 데이터베이스인 것처럼 동작합니다. Kylix는 객체 이름을 대문자로 쓰도록 요구합니다.

Kylix와 Delphi/C++Builder 개발자들이 알려준 또다른 일반적인 실수는 따옴표를 사용하여 스키마와 테이블을 참조할 때 구문을 잘못 사용하는 것입니다. 예를 들어, abc.tabname이라는 테이블을 참조하기 위해 다음과 같은 SQL을 작성하는 경우가 있습니다.

```
SELECT * from "abc.tabname"
```

이 쿼리는 스키마와 테이블 이름에서 따옴표가 잘못되었기 때문에 실패할 것이며, 올바른 표현은 아래와 같습니다.

```
SELECT * from "abc"."tabname"
```

요약

요약하면, Kylix와 DB2의 결합은 크로스 플랫폼 애플리케이션을 개발하기 위한 강력한 솔루션입니다. 이 문서에서 우리는 Kylix와 DB2를 최적으로 통합함으로써 동시성, 유연성 및 전반적인 성능을 최대화하는 통합할 수 있는 방법을 알아보았습니다.

Borland®

100 Enterprise Way
Scotts Valley, CA 95066-3249
www.borland.com | 831-431-1000