

학습서: EGL 및 SQL을 사용하여 관계형 데이터베이스에 액세스

버전 7.1

학습서: EGL 및 SQL을 사용하여 관계형 데이터베이스에 액세스

버전 7.1

주!

이 정보와 이 정보가 지원하는 제품을 사용하기 전에, 73 페이지의 『주의사항』의 일반 정보를 읽으십시오.

이 개정판은 새 개정판에 별도로 명시되어 있지 않는 한, Rational Business Developer 버전 7.1 및 모든 후속 릴리스와 수정판에 적용됩니다.

© Copyright International Business Machines Corporation 2000, 2008. All rights reserved.

목차

EGL 및 SQL을 사용하여 관계형 데이터베이스에 액세스	1	학습 체크포인트	50
소개	1	학습 7: 예외 처리 및 롤백	50
학습 1: 데이터베이스 연결 설정	2	SQL 데이터 액세스의 예외 처리	50
EGL 프로젝트 작성	2	데이터베이스에 변경사항 롤백	53
데이터베이스 가져오기 및 연결	4	학습 체크포인트	56
테스트 프로그램 실행	8	요약	56
학습 2: EGL의 기본 SQL 조작	10	문제점 해결	56
CRUD 조작 검토	13	런타임 오류 메시지	59
학습 체크포인트	19	자원	59
학습 3: open 및 forEach 로 커서 관리	19	학습 2에서 완성된 CRUDTest.egl 파일	60
단계적으로 결과 세트 이동	22	학습 3에서 완성된 selectItems.egl 파일	62
학습 체크포인트	25	학습 4A에서 완성된 selectItems.egl 파일	63
학습 4: prepare 를 사용하여 동적으로 조회 생성	25	학습 4B에서 완성된 selectItems.egl 파일	65
prepared 문의 일반 오류	26	학습 5에서 완성된 printCustomerOrders.egl 및	
학습 4A: prepared 문 사용	27	records.egl 파일	68
학습 4B: prepare 문의 <u>호스트</u> 변수	30	학습 6에서 완성된 CustomerTableOperations.egl	
학습 체크포인트	34	및 duplicateTable.egl 파일	69
학습 5: 테이블 결합을 사용하여 더욱 복잡한 데이터		학습 7에서 완성된 exceptionHandlingTest.egl 파	
검색	34	일	71
학습 6: 임의의 SQL 코드 실행	39	부록. 주의사항	73
SQL 빌더를 사용하여 SQL 문 작성	39	상표	75
execute 사용	47		

EGL 및 SQL을 사용하여 관계형 데이터베이스에 액세스

이 학습서에서는 EGL 및 Workbench의 기타 도구를 사용하여 관계형 데이터베이스에 대한 작업을 수행합니다. EGL을 사용하여 생성되는 기본 SQL 코드를 사용하는 방법과 고유 SQL 코드를 작성하여 해당 사용자 정의 SQL과 EGL 응용프로그램을 통합하는 방법을 배웁니다.

학습 목표

이 학습서는 다음 주제를 포함하며, EGL을 사용하여 관계형 데이터베이스에 액세스하는 중급 수준의 방법을 다룹니다.

- EGL을 사용하여 네 개의 기본 데이터 액세스 조작(create, read, update 및 delete)을 수행하는 방법
- EGL로 작성하는 SQL 코드를 사용자 정의하는 방법
- EGL 코드 내부 또는 외부에서 사용자 정의된 SQL 문을 작성하고 실행하는 방법
- 데이터 액세스 응용프로그램에서 오류를 처리하는 방법

소요 시간

90분

관련 정보

EGL 소개(빠른 시작 안내서)

EGL로 Hello World 서비스 작성

EGL로 Hello World 프로그램 작성

EGL을 사용하여 JSF 검색 페이지 빌드



PDF 버전 보기

소개

이 학습서에서는 EGL 및 Workbench의 기타 도구를 사용하여 관계형 데이터베이스에 대한 작업을 수행합니다. EGL을 사용하여 생성되는 기본 SQL 코드를 사용하는 방법과 고유 SQL 코드를 작성하여 해당 사용자 정의 SQL과 EGL 응용프로그램을 통합하는 방법을 배웁니다.

EGL(Enterprise Generation Language)은 완전한 기능의 응용프로그램을 빠르게 작성하는 데 사용할 수 있는 개발 환경 및 프로그래밍 언어로서, 사용자가 소프트웨어 기술이 아닌 사용자의 코드와 연관된 비즈니스 문제점에 집중할 수 있도록 도와줍니다.

학습 목표

이 학습서는 다음 주제를 포함하며, EGL을 사용하여 관계형 데이터베이스에 액세스하는 중급 수준의 방법을 다룹니다.

- EGL을 사용하여 네 개의 기본 데이터 액세스 조작(create, read, update 및 delete)을 수행하는 방법
- EGL로 작성하는 SQL 코드를 사용자 정의하는 방법
- EGL 코드 내부 또는 외부에서 사용자 정의된 SQL 문을 작성하고 실행하는 방법
- 데이터 액세스 응용프로그램에서 오류를 처리하는 방법

소요 시간

이 학습서를 완료하는 데 약 90분이 소요됩니다. 이 학습서와 관련된 다른 개념도 참조하는 경우 더 많은 시간이 소요될 수 있습니다.

스킬 레벨

중급

전제조건

이 학습서를 시작하려면 SQL 및 관계형 데이터베이스에 대해 어느 정도의 지식이 있어야 합니다. 또한, *EGL 소개(빠른 시작 안내서)*와 같은 학습서에서 EGL의 기초를 학습한 경우에는 EGL 코드 예제를 더 쉽게 이해할 수 있습니다.

예상 결과

선택사항: 학습서를 완료했을 때 예상되는 결과의 설명(텍스트), 멀티미디어에 대한 링크, 스크린샷 또는 코드 스니펫을 제공할 수 있습니다. 다시 말해, 사용자가 보여주거나 증명 또는 설명할 수 있는 최종 제품이 존재할 경우 이전에 설명하지 않았다면 여기서 설명할 수 있습니다.

학습 1: 데이터베이스 연결 설정

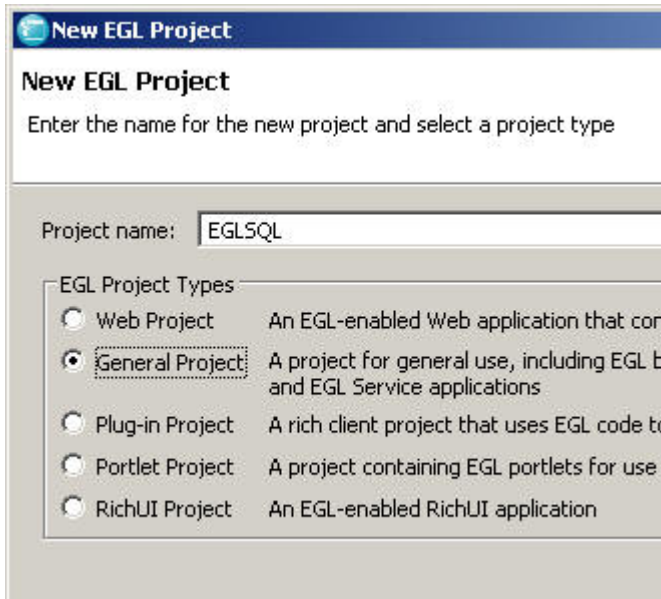
데이터베이스에 액세스하는 첫 번째 단계는 연결 설정입니다. 해당 연결에서는 EGL을 사용하여 데이터 파트 및 논리 파트의 시작 세트를 작성하고 연결을 테스트하는 간단한 데이터 액세스 프로그램을 작성합니다.

EGL 프로젝트 작성

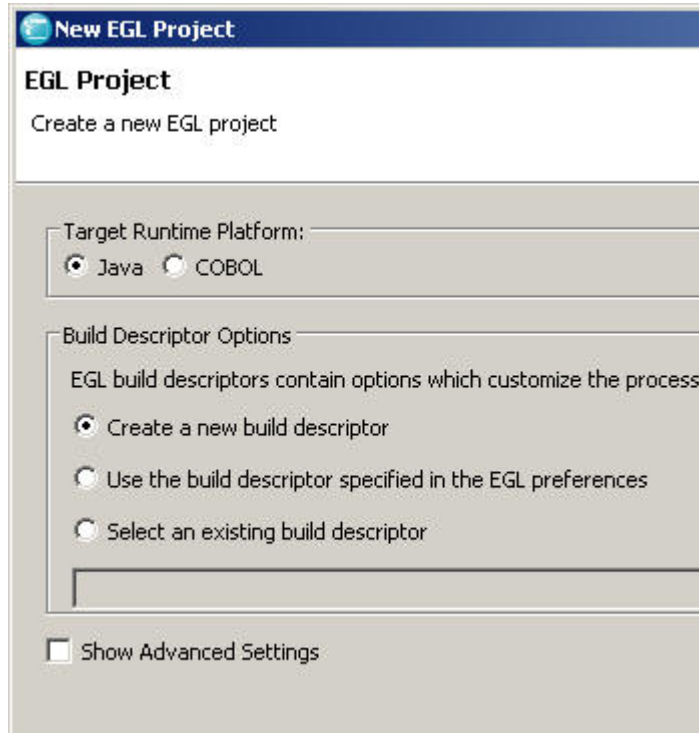
우선, 데이터 액세스 코드를 보유할 새 EGL 프로젝트가 필요합니다.

1. 다음과 같이 EGL Perspective로 전환하십시오.
 - a. 창 → **Perspective** 열기 → 기타를 클릭하십시오.
 - b. Perspective 열기 창에서 **EGL**을 클릭하십시오. Perspective 목록에서 EGL이 표시되지 않으면 모두 표시 선택란을 선택하십시오.
 - c. 확인을 클릭하십시오.

2. EGLSQL이라는 새 EGL 프로젝트(EGL 웹 프로젝트가 아님)를 작성하십시오.
 - a. 파일 → 새로 작성 → 프로젝트를 클릭하십시오.
 - b. 새 프로젝트 창에서 **EGL**을 펼친 다음 **EGL 프로젝트 마법사**를 클릭하십시오.
 - c. 다음을 클릭하십시오.
 - d. 프로젝트 이름 필드에 EGLSQL을 입력하십시오.
 - e. **EGL 프로젝트 유형**에서 일반 프로젝트를 클릭하십시오. 새 EGL 프로젝트 창은 다음과 같이 표시됩니다.



- f. 다음을 클릭하십시오.
- g. 두 번째 페이지에서 대상 런타임 플랫폼 아래 **Java**가 선택되어 있으며 빌드 설명자 옵션 아래 새 빌드 설명자 작성이 선택되어 있는지 확인하십시오. 새 EGL 프로젝트 창은 다음과 같이 표시됩니다.



h. 완료를 클릭하십시오.

새 프로젝트가 작성되어 프로젝트 탐색기 보기에 표시됩니다.

데이터베이스 가져오기 및 연결

이 학습서는 *EGL 소개(빠른 시작 안내서)* 학습서에서 사용한 샘플 Derby 데이터베이스를 사용합니다. 이 단계에서는 이 데이터베이스의 사본을 사용자 컴퓨터에 압축 해제합니다. 또한 데이터 액세스 응용프로그램 마법사를 사용하여 데이터베이스에 연결하고 해당 데이터베이스를 기반으로 데이터 파트와 논리 파트를 작성합니다. 해당 파트에 대한 자세한 정보는 *EGL 소개(빠른 시작 안내서)*를 참조하십시오.

1. 다음 링크를 클릭하여 컴퓨터의 임시 폴더에 샘플 데이터베이스를 다운로드하십시오.

샘플 데이터베이스

다운로드한 위치만 기억한다면 데이터베이스를 어디에 저장해도 상관없습니다.

또는, 다음 위치의 제품 설치 디렉토리에 이 샘플 데이터베이스가 있습니다.

`shared_resources/plugins/com.ibm.etools.egl.tutorial0001.doc_version/resources/EGLDerbyR7.zip`

`shared_resources`

프로젝트의 공유 자원 디렉토리(예: Windows 시스템의 경우 `C:\Program Files\IBM\SDP70Shared` 또는 Linux 시스템의 경우 `/opt/IBM/SDP70Shared`)입니다. 현재 제품을 설치하기 전에 EGL이 포함된 이전 버전의 IBM 제품이 설치되어 있는 경우 이전 설치에서 설정한 공유 자원 디렉토리를 지정해야 합니다.

version

점으로 구분된 세 개의 숫자(문자열 분리 문자, 플러그인이 빌드된 날짜와 시간)를 포함하는 플러그인의 설치 버전(예: 7.0.0.RFB_20070120_1300)입니다. 둘 이상이 있는 경우 최신 버전을 사용하십시오.

2. 컴퓨터의 위치(예: C:\databases)에 데이터베이스의 압축을 푸십시오. 다운로드한 위치만 기억한다면 데이터베이스를 어디에 저장해도 상관없습니다.
3. 파일 → 새로 작성 → 기타를 클릭하십시오. 새로 작성 창이 열립니다.
4. EGL을 펼치고 EGL 데이터 액세스 응용프로그램을 클릭하십시오.
5. 다음을 클릭하십시오.
6. 프로젝트 정의 설정 페이지의 프로젝트 이름 목록에서 **EGLSQL**을 선택하십시오.
7. 데이터베이스 연결 목록 옆에서 새로 작성을 클릭하십시오.
8. 새 연결 창의 데이터베이스 관리자 선택 아래에서 **Derby**를 펼치고 **10.1**을 클릭하십시오.
9. 데이터베이스 위치 필드에서 이전에 압축을 푼 ZIP 파일에 포함된 EGLDerbyR7 폴더를 찾으십시오. 예를 들어, EGLDerbyR7.zip 파일의 압축을 C:\databases에 푼 경우 데이터베이스 위치 필드는 C:\databases\EGLDerbyR7이어야 합니다.

사용자 ID 또는 암호 필드는 변경하지 않아도 됩니다.

10. 클래스 위치 필드에 derby.jar 파일의 경로를 입력하십시오.

다음과 같은 방법으로 이 파일을 찾을 수 있습니다.

- IBM® WebSphere® Application Server, 버전 6.1을 설치한 경우 서버에 포함된 Derby 버전을 사용할 수 있습니다. 다음 폴더에 derby.jar 파일이 있습니다.

`install_location/runtimes/base_v61/derby/lib`

- 제품과 함께 데이터베이스 도구를 설치한 경우 다음 위치에 파일이 있습니다.

`shared_resources/plugins/com.ibm.datatools.db2.cloudscape.driver_version/driver`

`shared_resources`

프로젝트의 공유 자원 디렉토리(예: Windows 시스템의 경우 C:\Program Files\IBM\SDP70Shared 또는 Linux 시스템의 경우 /opt/IBM/SDP70Shared)입니다. 현재 제품을 설치하기 전에 EGL이 포함된 이전 버전의 IBM 제품이 설치되어 있는 경우 이전 설치에서 설정한 공유 자원 디렉토리를 지정해야 합니다.

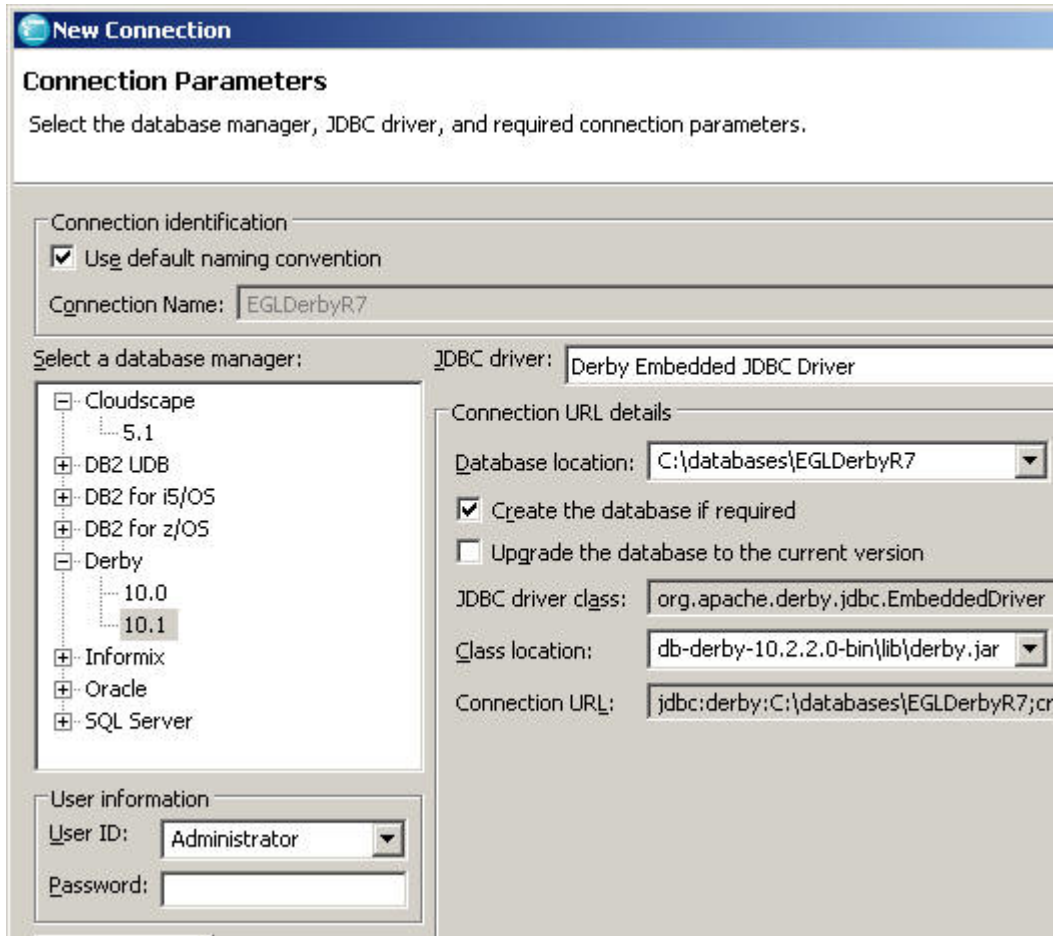
version

점으로 구분된 세 개의 숫자(문자열 분리 문자, 플러그인이 빌드된 날짜와 시간)를 포함하는 플러그인의 설치 버전(예: 7.0.0.RFB_20070120_1300)입니다. 둘 이상이 있는 경우 최신 버전을 사용하십시오.

- 위의 두 위치에서 파일을 찾을 수 없으면 제품 설치 디렉토리에서 derby.jar 파일을 검색해 보십시오. 다르게 설치했다면 다른 위치에 파일이 있을 수도 있습니다.

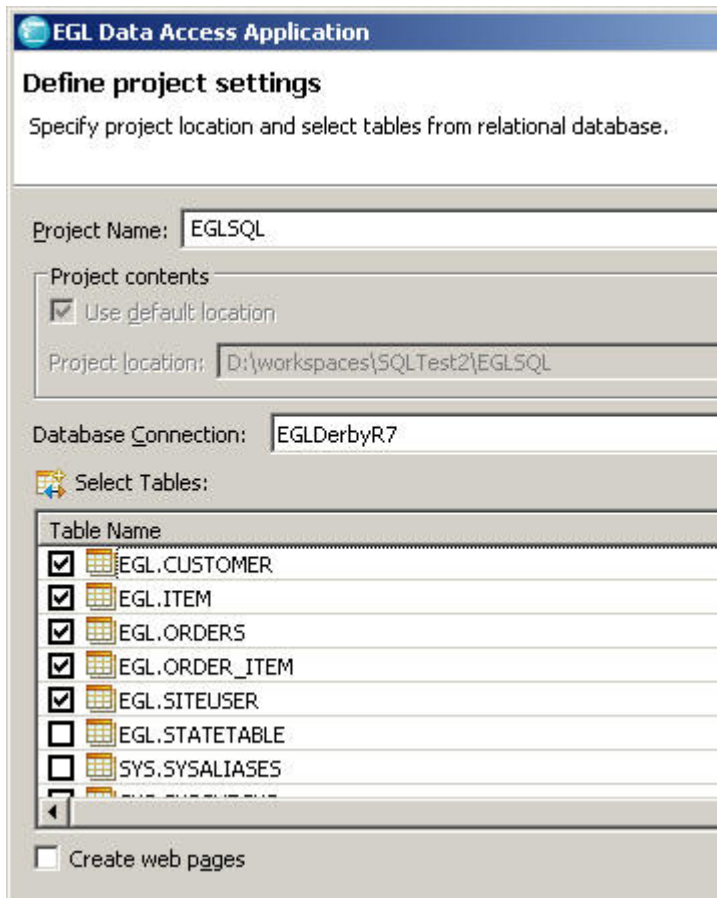
- 컴퓨터에서 이 파일을 찾을 수 없다면 Derby 웹 사이트(<http://db.apache.org/derby/>)에서 직접 다운로드하십시오. 가장 최근에 릴리스된 Derby 버전을 다운로드한 다음 사용자의 컴퓨터에 derby.jar 파일의 압축을 해제하십시오.

새 연결 창은 다음과 같으며 데이터베이스 위치 및 클래스 위치 필드에 사용자 고유 작업공간과 위치 정보가 있습니다.



11. 기본 이름 지정 규칙 사용 선택란이 선택되었는지 확인하십시오.
12. 완료 버튼을 클릭하십시오. 이제, 데이터베이스에 연결을 설정했습니다. 데이터베이스의 모든 테이블은 마법사 맨 아래 테이블 이름 아래 나열됩니다. 일부 테이블은 메타데이터만 포함하고 있으므로, 모든 테이블의 데이터 파트를 작성하지는 않을 것입니다.
13. 테이블 선택에서 다음 테이블 옆의 선택란만 선택하십시오.
 - EGL.CUSTOMER
 - EGL.ITEM
 - EGL.ORDERS
 - EGL.ORDER_ITEM
 - EGL.SITEUSER
 - EGL.STATETABLE

현재까지 설정한 EGL 데이터 액세스 응용프로그램 마법사의 모습은 다음과 같습니다.



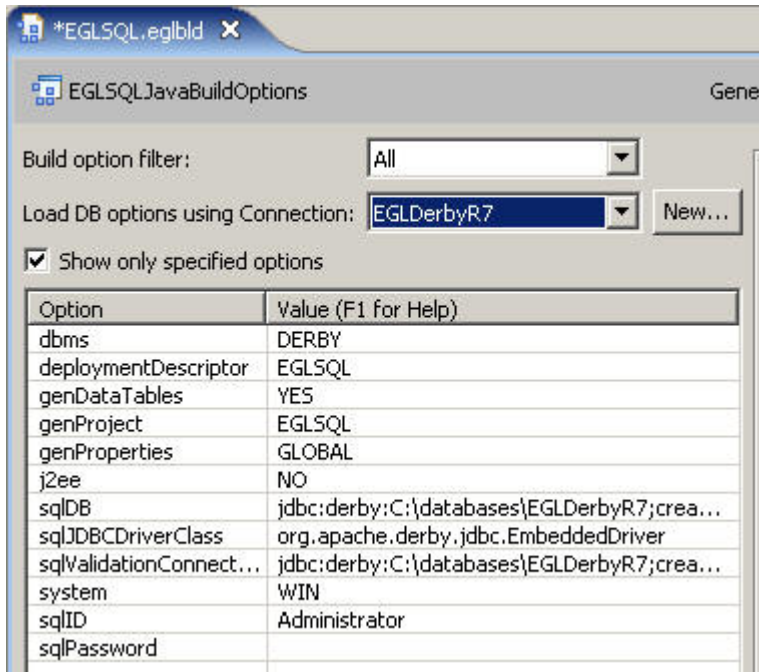
14. 데이터베이스 연결 필드의 연결 이름을 적어 두십시오. 이후에 이 연결 이름이 필요합니다. 일반적으로, 연결 이름은 데이터베이스의 이름을 따르므로 EGLDerbyR7이라고 지정합니다. 그러나 이 작업공간에서 EGL 소개(빠른 시작 안내서) 학습서를 이미 완료했다면 이 이름의 데이터베이스 연결이 이미 작성되었으므로 EGL은 새 연결의 이름을 EGLDerbyR71로 지정합니다.
15. 웹 페이지 작성 선택란을 지우십시오.
16. 프로젝트 이름 필드에 프로젝트 **EGLWeb**이 나열되어 있는지 확인하십시오.

주: 아직 완료를 클릭하지 마십시오. 이 마법사에서 설정을 하나 더 변경해야 합니다.

17. 다음을 클릭하여 필드 정의 페이지로 이동하십시오. 이 페이지를 사용하여 데이터베이스 테이블에 키 필드를 추가할 수 있습니다. 이 페이지의 설정을 변경하지 마십시오. 데이터베이스의 각 테이블에 이미 키 필드가 있습니다.
18. 다음을 한번 더 클릭하여 프로젝트 작성 옵션 정의 페이지로 이동하십시오.
19. 프로젝트 작성 옵션 정의 페이지에서 스키마로 테이블 이름 규정 선택란을 선택하십시오.
20. 완료를 클릭하십시오. EGL은 데이터베이스에 액세스하는 데 사용할 수 있는 다양한 데이터 파트와 논리 파트를 작성합니다. 이 학습서의 나머지 부분에서 이 파트를 광범위하게 사용합니다.

런타임 시 이 데이터베이스 연결을 사용하도록 빌드 설명자를 구성해야 합니다.

21. EGLSQL/EGLSource/EGLSQL.eglbuild에 있는 프로젝트의 기본 빌드 설명자를 여십시오. 이 파일을 두 번 클릭하면 EGL 빌드 파트 편집기에서 열립니다.
22. EGL 빌드 파트 편집기의 연결을 사용하여 DB 옵션 로드 옆에서 데이터베이스 연결 이름을 선택하십시오. 이 이름은 이전 단계에서 기록한 데이터베이스 연결의 이름입니다. 빌드 설명자 옵션은 해당 연결을 사용하는 데 필요한 값으로 설정됩니다.



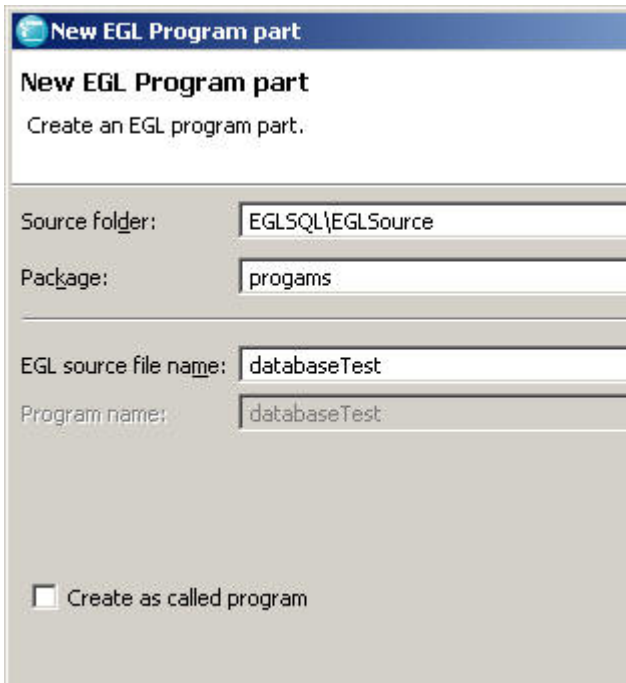
23. 빌드 설명자를 저장하고 닫으십시오.
24. Derby 드라이버를 프로젝트의 Java™ 빌드 경로에 추가하십시오.
 - a. EGLSQL 프로젝트를 마우스 오른쪽 단추로 클릭한 다음 특성을 클릭하십시오.
 - b. 특성 창에서 **Java** 빌드 경로를 클릭하십시오.
 - c. Java 빌드 경로 페이지에서 라이브러리 탭으로 이동하십시오.
 - d. 라이브러리 탭에서 외부 JAR 추가를 클릭하십시오.
 - e. JAR 선택 창에서 Derby.jar 파일을 선택하고 열기를 클릭하십시오.
 - f. 완료를 클릭하십시오.
 - g. 확인을 클릭하십시오.

테스트 프로그램 실행

데이터베이스 연결이 작동하는지 확인하려면 다음 단계에 따라 테스트 프로그램을 실행하고 콘솔로 데이터베이스 행을 인쇄하십시오.

1. 파일 → 새로 작성 → 프로그램을 클릭하십시오.
2. 새 EGL 프로그램 파트 창에서 패키지 필드에 programs를 입력하십시오.

3. **EGL 소스 파일 이름 필드**에 databaseTest를 입력하십시오. 새 EGL 프로그램 파트 창의 모습은 다음과 같습니다.



4. **완료**를 클릭하십시오. 새 프로그램이 작성되어 편집기에서 열립니다.
5. 새 프로그램의 모든 코드를 다음 코드로 바꾸십시오.

```
package progrms;

import eglderbyr7.data.Customer;

program databaseTest type BasicProgram {}

function main()
  customer Customer;
  open allResults for customer;

  SysLib.writeStdout("#::" Customer name");
  forEach (customer)
    SysLib.writeStdout(customer.CustomerId::" "
      ::customer.FirstName::" " ::customer.LastName);
  end
end

end
```

6. 프로그램을 저장하십시오.
7. 프로젝트 탐색기 보기에서 EGLSQL 프로젝트를 마우스 오른쪽 단추로 클릭한 다음 **생성**을 클릭하여 전체 프로젝트를 생성하십시오.

프로그램을 실행하려면 데이터베이스에서 데이터베이스 도구의 연결을 끊어야 합니다. Derby는 한 번에 하나의 연결만을 허용하며 EGL이 데이터 파트를 작성하고 빌드 설명자의 값을 설정하는 데 사용한 도구는 사용자가 연결을 닫을 때까지 연결된 상태로 남아 있습니다.

8. 런타임 시 프로그램에서 데이터베이스를 사용할 수 있도록 데이터베이스 연결을 닫으십시오.
 - a. 창 → 보기 표시 → 기타를 클릭하고 보기 목록에서 데이터 → 데이터베이스 탐색기를 선택한 다음 확인을 클릭하여 데이터베이스 탐색기 보기를 여십시오.
 - b. 데이터베이스 탐색기 보기에서 연결을 펼치십시오. 사용자의 데이터베이스 연결이 EGL 빌드 설명자에 선택한 이름과 같은 이름(예: EGLDerbyR7)으로 연결 아래에 나열됩니다.
 - c. 데이터베이스 연결을 마우스 오른쪽 단추로 클릭한 다음 연결 끊기를 클릭하십시오.
 - d. 데이터베이스 탐색기 보기를 닫으십시오. 나중에 학습서에서 이 보기를 포함하여 데이터 도구에 대해 자세히 설명합니다.
9. 다음과 같이 테스트 프로그램을 실행하십시오.
 - a. EGLSQL/JavaSource/programs로 가십시오. JavaSource 폴더에 생성된 프로그램이 있습니다.
 - b. databaseTest.java 파일을 마우스 오른쪽 단추로 클릭한 다음 실행 도구 → Java 응용 프로그램을 클릭 하십시오.

테스트 프로그램을 실행할 때 콘솔 보기에 인쇄된 데이터베이스의 CUSTOMER 테이블 콘텐츠가 표시되어야 합니다.



```
<terminated> databaseTest [Java Application]
# Customer name
1 Fred Filibuster
2 Andy Lundquist
3 Billie Kingman
4 Francine Liebowitz
5 Ornette Coleman
6 Ellen Springsteen
7 Martin St. Louis
8 Sergey Sharov
9 Melodie Hudak
```

콘솔 보기에 출력 대신 오류 메시지가 표시되면 56 페이지의 『문제점 해결』을 확인하십시오.

학습 2: EGL의 기본 SQL 조작

EGL 소개(빠른 시작 안내서)를 완료하면 관계형 데이터베이스에 액세스하는 데 필요한 EGL 파트와 데이터베이스에서 데이터를 읽고 쓰는 데 사용할 수 있는 명령을 배우게 됩니다. 이 학습에서 해당 개념을 보다 자세히 다룹니다.

이전 학습에서나 지난 학습서에서 데이터 액세스 응용프로그램 마법사를 실행한 경우 EGL이 데이터베이스의 정보를 기반으로 데이터 파트와 논리 파트를 작성합니다.

작성된 가장 중요한 데이터 파트는 SQLRecord 파트입니다. 이 레코드 파트의 필드는 데이터베이스의 열과 관련됩니다. 이 레코드를 기반으로 변수를 작성하면 해당 변수를 사용하여 데이터베이스에 신규 또는 기존 행을 나타낼 수 있습니다. 데이터 액세스 응용프로그램 마법사가 작성한 각 SQLRecord는 단일 테이블의 열과만 일치하지만 나중에 이 학습서에서 두 테이블 이상의 데이터를 단일 레코드로 병합하는 방법에 대해 설명할 것입니다(SQL에서는 테이블 결합이라고 함).

예를 들어, 이 학습서에서는 Customer SQLRecord를 사용할 것입니다. EGLSource/eglderbyr7/data/customer.egl 파일에서 이 레코드를 찾을 수 있습니다.

```
record Customer type sqlRecord {
    tablenames=[["EGL.CUSTOMER"]],
    keyItems=[CustomerId]
}

CustomerId CustomerId {column="EGL.CUSTOMER.CUSTOMER_ID";
FirstName FirstName {column="EGL.CUSTOMER.FIRST_NAME",
    sqlVariableLen=yes, maxLen=30, isSqlNullable=yes};
LastName LastName {column="EGL.CUSTOMER.LAST_NAME",
    sqlVariableLen=yes, maxLen=30, isSqlNullable=yes};
...
Directions Directions {column="EGL.CUSTOMER.DIRECTIONS",
    sqlVariableLen=yes, maxLen=255, isSqlNullable=yes};
end
```

tablenames=[["EGL.CUSTOMER"]] 코드는 이 레코드가 연관된 데이터베이스의 테이블을 지정하며 각 필드의 열 특성은 필드가 연관된 테이블의 열을 지정합니다. keyItems=[CustomerId] 코드는 CustomerId 필드와 데이터베이스의 EGL.CUSTOMER.CUSTOMER_ID 열(CustomerId와의 연결로 인해)이 이 테이블의 1차 키임을 나타냅니다.

데이터 액세스 응용프로그램 마법사는 레코드 배열과 레코드에서 CRUD(Create, Read, Update 및 Delete) 조작을 수행하는 여러 라이브러리도 작성합니다. 이 라이브러리는 EGL 명령을 사용하여 CRUD 조작을 직접 수행하는 고급 방법입니다. 이 학습서에서는 EGL 명령을 직접 사용합니다. 그러나, 재사용을 위해 로직을 라이브러리로 구분하는 것이 편리합니다. 네 가지 EGL CRUD 명령은 다음과 같습니다.

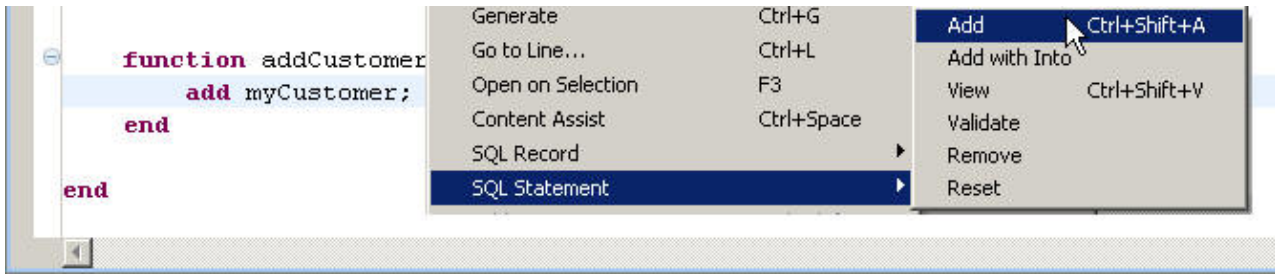
표 1. EGL CRUD 명령 및 대응하는 SQL 명령

EGL 명령	EGL 예제	SQL 명령
add	add myRecord;	INSERT
get	get myRecord;	SELECT
replace	replace myRecord;	UPDATE
delete	delete myRecord;	DELETE

이 명령 중 하나를 사용하면 EGL이 해당 명령에서 기본 SQL 문을 생성합니다. 예를 들어, 다음 **add** 문이 있습니다.

```
add myCustomer;
```

myCustomer 레코드에 커서를 두고 마우스 오른쪽 단추를 클릭한 다음 **SQL** 문 → 추가를 클릭하면 EGL 편집기가 **add** 문을 펼쳐 SQL 코드를 표시하며 SQL 코드가 명시적으로 표시됩니다.



```
add myCustomer with
#sql{
  insert into EGL.CUSTOMER
    (EGL.CUSTOMER.CUSTOMER_ID, EGL.CUSTOMER.FIRST_NAME,
     EGL.CUSTOMER.LAST_NAME, EGL.CUSTOMER.PASSWORD,
     EGL.CUSTOMER.PHONE, EGL.CUSTOMER.EMAIL_ADDRESS,
     EGL.CUSTOMER.STREET, EGL.CUSTOMER.APARTMENT,
     EGL.CUSTOMER.CITY, EGL.CUSTOMER."STATE",
     EGL.CUSTOMER.POSTALCODE, EGL.CUSTOMER.DIRECTIONS)
  values
    (:myCustomer.CustomerId, :myCustomer.FirstName,
     :myCustomer.LastName, :myCustomer.Password,
     :myCustomer.Phone, :myCustomer.EmailAddress,
     :myCustomer.Street, :myCustomer.Apartment,
     :myCustomer.City, :myCustomer.State,
     :myCustomer.Postalcode, :myCustomer.Directions)
};
```

내용은 변경되지 않습니다. EGL이 **add** 문에서 생성한 SQL 코드를 드러냈을 뿐입니다. 이제, 코드가 명시적이므로 EGL 명령에서 생성된 기본 SQL을 편집할 수 있습니다.

SQL INSERT 문의 VALUES절은 새 데이터베이스 행(이 경우, myCustomer 레코드의 필드)에 삽입할 값을 나열합니다. **#sql** 블록의 각 값 앞에는 콜론(:)이 오는데, 이는 값이 SQL 값이 아니라 EGL 값을 나타냅니다. 명시적 SQL 코드에 사용하는 이 EGL 값은 **호스트 변수**라고 합니다.

작동하는 호스트 변수의 또 다른 예로서, 다음은 **get** 문을 사용하여 데이터베이스에서 일련의 레코드를 제거하는 함수입니다.

```
function getCustFromState(customerState CHAR(2) in,
  customers Customer[] out)

  get customers with
  #sql{
    select *
    from EGL.CUSTOMER
    where EGL.CUSTOMER."STATE" like :customerState
  };

end
```

이 함수는 2바이트 CHAR 매개변수를 받은 다음 해당 매개변수를 호스트 변수로 사용하여 해당 두 문자에 일치하는 STATE 필드가 있는 레코드를 검색합니다. EGL이 명시적 SQL 코드와 작동할 수 있도록 하려면 호스트 변수를 주의하여 사용하십시오.

CRUD 조작 검토

이 절에서는 **get**, **add**, **replace** 및 **delete**를 사용하여 EGL로 기본 데이터베이스 조작을 수행합니다. 이 절은 선택사항입니다. 네 개의 기본 EGL 데이터 액세스 명령문 및 해당 SQL 대응 명령에 익숙하다면 이 절을 건너뛰십시오.

1. 다음과 같이 CRUDTest.egl이라는 새 EGL 프로그램을 작성하십시오.
 - a. 프로젝트 탐색기 보기에서 EGLSQL 프로젝트를 마우스 오른쪽 단추로 클릭한 다음 새로 작성 → 프로그램을 클릭하십시오. 새 EGL 프로그램 파트 창이 열립니다.
 - b. 패키지 필드에 programs라는 이름을 입력하십시오.
 - c. EGL 소스 파일 이름 필드에 CRUDTest라는 이름을 입력하십시오.
 - d. 완료를 클릭하십시오.

새 EGL 프로그램이 작성되어 편집기에서 열립니다.

2. 기본 EGL 코드를 다음 코드로 바꾸십시오.

```
package programs;

import eglderbyr7.data.*;

program CRUDTest type BasicProgram {}

function main()

    try
        // Print the starting records in the database.
        SysLib.writeStdout("Contents of database before any SQL operations:");
        printAllCustomers();

        // What to do if an error happens.
        onException (exception SQLException)
            handledDBException(exception);
        end
    end

    // This function prints out the contents of the CUSTOMER
    // table in the database.
    function printAllCustomers()

        allCustomers Customer[0];

        get allCustomers;
        for (counter int from 1 to allcustomers.getSize())
            SysLib.writeStdout(allCustomers[counter].CustomerId::" " ::
                allCustomers[counter].FirstName::" " ::
                allCustomers[counter].LastName);
        end
    end
end
```

```

        SysLib.writeStdout("#n");

    end

    //This function responds to errors accessing the database.
    function handleDBException(exception SQLException)
        SysLib.writeStdout("Error accessing database. "::
            "See Troubleshooting section");
        SysLib.writeStdout(exception.message);
    end

end

```

현재, 이 프로그램은 데이터베이스의 전체 고객 목록을 인쇄할 뿐입니다. 여기에는 `handleDBException`이라는 함수가 포함되어 데이터베이스와 작업할 때 EGL에서 발생하는 모든 문제점을 처리합니다(나중에 이 학습서에서 오류 처리에 대해 자세히 설명함). 이 프로그램을 지금 실행하여 변경 전의 데이터베이스 상태를 확인할 수 있습니다.

3. 프로그램을 저장하고 생성하십시오.
4. 다음과 같이 프로그램을 실행하십시오.
 - a. 프로젝트 탐색기 보기에서 EGLSQL/JavaSource/programs를 펼치십시오.
 - b. `CRUDTest.java` 파일을 마우스 오른쪽 단추로 클릭한 다음 실행 도구 → **Java 응용프로그램**을 클릭하십시오.

오류 없이 프로그램이 실행되면 콘솔 보기가 데이터베이스의 콘텐츠를 표시합니다. 콘솔 보기의 결과는 다음과 유사합니다.

Contents of database before any SQL operations:

```

1 Fred Filibuster
2 Andy Lundquist
3 Billie Kingman
4 Francine Liebowitz
5 Ornette Coleman
6 Ellen Springsteen
7 Martin St. Louis
8 Sergey Sharov
9 Melodie Hudak

```

콘솔 결과에 오류가 포함된 경우 프로그램을 정확하게 복사했는지 확인한 후 56 페이지의 『문제점 해결』을 확인하십시오.

5. 다음과 같이 마지막 **end** 문 앞에 데이터베이스에 새 행을 삽입하는 함수를 추가하십시오.

```

function addCustomer()
    customerToAdd Customer;
    customerToAdd.CustomerId = 50;
    customerToAdd.FirstName = "John";
    customerToAdd.LastName = "Doe";

    // Insert the record into the database
    try
        add customerToAdd;
        SysLib.writeStdout("Contents of database after adding a new record:");
        printAllCustomers();
    end
end

```

```

onException(exception SQLException)
    handleDBException(exception);
end

```

end

이 함수는 **add**를 사용하여 레코드를 데이터베이스에 삽입합니다. **add**문 뒤의 내재적 SQL 코드를 보려면 커서를 add customerToAdd; 행의 customerToAdd 키워드에 직접 두고 마우스 오른쪽 단추로 클릭한 다음 **SQL** 문 → 추가를 클릭하십시오. 이제, 명령문은 다음과 같습니다.

```

try
    add customerToAdd with
    #sql{
        insert into EGL.CUSTOMER
            (EGL.CUSTOMER.CUSTOMER_ID, EGL.CUSTOMER.FIRST_NAME,
            EGL.CUSTOMER.LAST_NAME, EGL.CUSTOMER.PASSWORD,
            EGL.CUSTOMER.PHONE, EGL.CUSTOMER.EMAIL_ADDRESS,
            EGL.CUSTOMER.STREET, EGL.CUSTOMER.APARTMENT,
            EGL.CUSTOMER.CITY, EGL.CUSTOMER."STATE",
            EGL.CUSTOMER.POSTALCODE, EGL.CUSTOMER.DIRECTIONS)
        values
            (:customerToAdd.CustomerId, :customerToAdd.FirstName,
            :customerToAdd.LastName, :customerToAdd.Password,
            :customerToAdd.Phone, :customerToAdd.EmailAddress,
            :customerToAdd.Street, :customerToAdd.Apartment,
            :customerToAdd.City, :customerToAdd.State,
            :customerToAdd.Postalcode, :customerToAdd.Directions)
    };

```

레코드 변수에 직접 커서를 두고 마우스 오른쪽 단추를 클릭한 다음 **SQL** 문 → 보기를 클릭하여 SQL 코드를 명시적으로 만들지 않은 상태에서 해당 코드를 미리볼 수도 있습니다. 이와 같이 EGL 코드에서 SQL 코드에 대한 작업을 수행하려면 EGL 편집기의 커서가 명령문에 있어야 합니다.

- 다음과 같이 프로그램의 main() 함수에서 새 함수를 호출하십시오.

```

function main()

    // Print the starting records in the database.
    SysLib.writeStdout("Contents of database before any SQL operations:");
    printAllCustomers();

    // Add a record.
    addCustomer();

end

```

- 프로그램을 저장하고 생성하여 실행하십시오. 다음 출력은 테이블에 레코드를 추가하기 전/후 테이블에 표시되는 콘텐츠입니다.

```

Contents of database before any SQL operations:
1 Fred Filibuster
2 Andy Lundquist
3 Billie Kingman
4 Francine Liebowitz
5 Ornette Coleman
6 Ellen Springsteen
7 Martin St. Louis

```

8 Sergey Sharov
9 Melodie Hudak

Contents of database after adding a new record:

1 Fred Filibuster
2 Andy Lundquist
3 Billie Kingman
4 Francine Liebowitz
5 Ornette Coleman
6 Ellen Springsteen
7 Martin St. Louis
8 Sergey Sharov
9 Melodie Hudak
50 John Doe

8. 데이터베이스에서 레코드를 삭제하는 함수를 다음과 같이 마지막 **end** 문 앞에 추가하십시오.

```
function deleteCustomer()  
    customerToDelete Customer;  
    customerToDelete.CustomerId = 50;  
  
    // Delete the record.  
    try  
        delete customerToDelete noCursor;  
        SysLib.writeStdout("Contents of database after deleting the record:");  
        printAllCustomers();  
    onException(exception SQLException)  
        handleDBException(exception);  
    end  
  
end
```

9. `main()` 함수에서 레코드를 추가하는 함수 호출을 레코드를 삭제하는 함수 호출로 바꾸십시오. `CUSTOMER_ID` 열은 테이블의 1차 키이며 테이블에 이미 `CUSTOMER_ID`가 50으로 설정된 행이 있으므로 레코드를 다시 추가하려고 하면 오류가 발생합니다.

```
function main()  
  
    // Print the starting records in the database.  
    SysLib.writeStdout("Contents of database before any SQL operations:");  
    printAllCustomers();  
  
    // Delete a record.  
    deleteCustomer();  
  
end
```

10. 프로그램을 저장하고 생성하여 실행하십시오. 다음 출력은 테이블에서 레코드를 삭제하기 전/후 테이블에 표시되는 콘텐츠입니다.

Contents of database before any SQL operations:

1 Fred Filibuster
2 Andy Lundquist
3 Billie Kingman
4 Francine Liebowitz
5 Ornette Coleman
6 Ellen Springsteen
7 Martin St. Louis

```
8 Sergey Sharov
9 Melodie Hudak
50 John Doe
```

Contents of database after deleting the record:

```
1 Fred Filibuster
2 Andy Lundquist
3 Billie Kingman
4 Francine Liebowitz
5 Ornette Coleman
6 Ellen Springsteen
7 Martin St. Louis
8 Sergey Sharov
9 Melodie Hudak
```

11. 데이터베이스에서 레코드를 검색하고 갱신하는 함수를 다음과 같이 마지막 **end** 문 앞에 추가하십시오.

```
function updateCustomer()
    customerToUpdate Customer;
    customerToUpdate.CustomerId = 50;

    // Retrieve the row.
    get customerToUpdate;

    // Change values in the record variable.
    customerToUpdate.FirstName = "Johnny";
    customerToUpdate.LastName = "Five";

    // Update the row in the database.
    try
        replace customerToUpdate noCursor;
        SysLib.writeStdout("Contents of database after updating the record:");
        printAllCustomers();
    onException(exception SQLException)
        handleDBException(exception);
    end

end
```

12. 프로그램의 main() 함수에서 추가, 바꾸기 및 삭제 함수를 순서대로 호출하십시오.

```
function main()

    // Print the starting records in the database.
    SysLib.writeStdout("Contents of database before any SQL operations:");
    printAllCustomers();

    // Add a record.
    addCustomer();

    // Update the record.
    updateCustomer();

    // Delete a record.
    deleteCustomer();

end
```

13. 프로그램을 저장하고 생성하여 실행하십시오. 다음 각각의 출력은 레코드를 추가 후, 갱신 후 및 삭제한 후 프로그램의 시작에 표시되는 테이블 콘텐츠입니다.

Contents of database before any SQL operations:

```
1 Fred Filibuster
2 Andy Lundquist
3 Billie Kingman
4 Francine Liebowitz
5 Ornette Coleman
6 Ellen Springsteen
7 Martin St. Louis
8 Sergey Sharov
9 Melodie Hudak
```

Contents of database after adding a new record:

```
1 Fred Filibuster
2 Andy Lundquist
3 Billie Kingman
4 Francine Liebowitz
5 Ornette Coleman
6 Ellen Springsteen
7 Martin St. Louis
8 Sergey Sharov
9 Melodie Hudak
50 John Doe
```

Contents of database after updating the record:

```
1 Fred Filibuster
2 Andy Lundquist
3 Billie Kingman
4 Francine Liebowitz
5 Ornette Coleman
6 Ellen Springsteen
7 Martin St. Louis
8 Sergey Sharov
9 Melodie Hudak
50 Johnny Five
```

Contents of database after deleting the record:

```
1 Fred Filibuster
2 Andy Lundquist
3 Billie Kingman
4 Francine Liebowitz
5 Ornette Coleman
6 Ellen Springsteen
7 Martin St. Louis
8 Sergey Sharov
9 Melodie Hudak
```

CRUDTest.egl 파일의 전체 코드가 완성되었습니다. 파일에 빨간색 X 기호로 표시된 오류가 나타나면 코드가 60 페이지의 『학습 2에서 완성된 CRUDTest.egl 파일』의 코드와 일치하는지 확인하십시오.

학습 체크포인트

이 학습에서는 EGL 및 SQL을 사용하여 관계형 데이터베이스에 액세스하는 기초적인 방법을 배웠습니다.

또한, 다음 개념도 이해하게 되었을 것입니다.

- SQLRecord 파트 및 데이터베이스 테이블과의 해당 관계
- 데이터베이스 조작을 수행하는 네 개의 기본 EGL 문
- 명시적 SQL
- 호스트 변수

네 가지 기본 EGL 데이터 액세스 명령문에 대한 자세한 정보는 다음 도움말 항목을 참조하십시오.

- add
- delete
- get
- replace

학습 3: open 및 forEach로 커서 관리

커서는 책갈피와 같이 동작하는 SQL 구성으로서, 행을 가리키고 사용자가 한 번에 하나씩 일련의 검색 결과(즉, 테이블 행)를 단계적으로 이동할 수 있도록 합니다. EGL이 커서를 관리할 수 있지만 커서를 이용하도록 **open** 및 **forEach** 문도 제공합니다.

이전 학습에서 **delete** 문이 키워드 **noCursor**를 포함하고 있다는 것을 인지했을 것입니다. **get** 문이 전체 테이블에서 하나 이상의 행을 선택하는 반면, **delete**문은 일반적으로 이전 명령문에서 작성한 커서가 표시하는 행에 대해서만 작동합니다. **noCursor** 키워드는 해당 커서가 존재하지 않으며 명령문이 기본 **get** 문과 같은 방식으로 **WHERE**절을 사용하여 행을 식별해야 한다는 것을 나타냅니다.

EGL **open** 문은 커서를 작성하고 커서가 이동할 수 있는 하나 이상의 행 세트 또는 결과 세트를 표시합니다. 커서에 직접 액세스하지 않고 다른 EGL 문을 사용하여 커서가 표시하는 대로 결과 세트에서 다음 행을 검색하거나 커서를 사용하여 한 번에 한 행을 단계적으로 이동하여 전체 결과 세트에서 조작을 수행합니다.

EGL로 커서 및 결과 세트를 작성하려면 **open** 문을 사용하여 하나 이상의 행을 식별한 다음 새 결과 세트에 이름과 임시 변수를 제공하여 커서가 표시하는 행을 다음에 넣으십시오.

```
tempItem Item;  
open allExpensiveItems for tempItem;
```

이 예제에서 tempItem은 데이터베이스의 ITEMS 테이블을 표시하는 SQLRecord 파트의 단일 레코드 변수입니다. for tempItem 코드는 EGL이 커서를 사용하여 한 번에 하나씩 이 레코드에 행을 이동하도록 지정합니다. open allExpensiveItems 코드는 결과 세트에 이름을 부여합니다. allExpensiveItems는 엄격히 말해서 변수가 아니라 ID입니다.

get 문과 같이 EGL은 **open** 문에서 SQL SELECT 문을 작성하여 명시적으로 만든 후 편집할 수 있습니다. 위의 **open** 문의 기본 SELECT 문 예제는 다음과 같습니다.

```
tempItem Item;

open expensiveItems for tempItem with
#sql{
  select
    EGL.ITEM.ITEM_ID, EGL.ITEM.NAME, EGL.ITEM.IMAGE,
    EGL.ITEM.PRICE, EGL.ITEM.DESCRPTION
  from EGL.ITEM
  where
    EGL.ITEM.ITEM_ID >= :tempItem.ItemId
  order by
    EGL.ITEM.ITEM_ID asc
};
```

이 기본 WHERE절은 각 항목의 ITEM_ID 값이 이전 값보다 커야함을 지정합니다(EGL은 SQL DECLARE CURSOR 및 OPEN CURSOR 문을 관리하며 명시적 SQL에 표시하지 않음).

결과 세트에 추가할 특정 행을 선택하려면 **get** 문을 사용하는 경우와 같이 WHERE절을 편집해야 합니다. 예를 들어, 주어진 변수보다 비용이 큰 항목만을 선택할 수 있습니다.

```
tempItem Item;
maxCost Price = 500;

open expensiveItems for tempItem with
#sql{
  select
    EGL.ITEM.ITEM_ID, EGL.ITEM.NAME, EGL.ITEM.IMAGE,
    EGL.ITEM.PRICE, EGL.ITEM.DESCRPTION
  from EGL.ITEM
  where
    EGL.ITEM.PRICE >= :maxCost
  order by
    EGL.ITEM.ITEM_ID asc
};
```

이제, expensiveItems 결과 세트에는 500으로 설정된 maxCost 변수 이상인 PRICE 열이 있는 데이터베이스의 모든 행이 포함됩니다.

이 행에 액세스하려면 루프의 코드 블록을 실행하여 결과 세트의 각 항목에 해당 코드를 적용하는 **forEach** 문을 사용하십시오. 루프의 각 반복 중에 EGL은 커서가 표시하는 행을 임시 변수로 이동시킨 다음 커서를 결과 세트의 다음 행으로 이동시킵니다. 예를 들어, 다음 코드는 가격이 500을 넘는 각 항목의 값을 인쇄합니다.

```
tempItem Item;
maxCost Price = 500;

open expensiveItems for tempItem with
#sql{
  select
    EGL.ITEM.ITEM_ID, EGL.ITEM.NAME, EGL.ITEM.IMAGE,
    EGL.ITEM.PRICE, EGL.ITEM.DESCRPTION
  from EGL.ITEM
  where
```

```

        EGL.ITEM.PRICE >= :maxCost
    order by
        EGL.ITEM.ITEM_ID asc
};

foreach (tempItem)
    SysLib.writeStdout(tempItem.Name :: " " ::
        tempItem.Description :: " $" :: tempItem.Price);
end

```

get next를 사용하여 수동으로 결과를 단계별로 이동할 수 있습니다. 이 경우, 예를 들어 결과 세트에 추가 결과가 있으면 0으로 설정되는 **sysVar.sqlData.sqlcode**의 값을 테스트하여 결과 세트에 추가 결과가 있는지를 발견해야 합니다.

```

tempItem Item;
maxCost Price = 500;

open expensiveItems for tempItem with
    #sql{
        select
            EGL.ITEM.ITEM_ID, EGL.ITEM.NAME, EGL.ITEM.IMAGE,
            EGL.ITEM.PRICE, EGL.ITEM.DESCRPTION
        from EGL.ITEM
        where
            EGL.ITEM.PRICE >= :maxCost
        order by
            EGL.ITEM.ITEM_ID asc
    };

// Retrieve the first row.
get next tempItem;

// As long as there are more rows,
// move the next row into the temporary variable
// and print its values.
while (sysvar.sqlData.sqlcode == 0)
    SysLib.writeStdout(tempItem.Name :: " " ::
        tempItem.Description :: " $" :: tempItem.Price);
    get next tempItem;
end

```

get 문에는 열기 커서 및 결과 세트와 함께 사용할 수 있는 기타 다양한 조작이 있습니다.

- **get next**를 사용할 때 이전 예제에서처럼 임시 변수가 아니라 결과 세트를 지정할 수 있습니다. 예를 들어, **get next from expensiveItems**은 다음 행을 결과 세트에서 임시 변수로 이동합니다. 이 코드는 **get next tempItem**과 동등합니다. 마찬가지로, **forEach** 문에서 결과 세트 이름을 사용할 수 있습니다. **forEach(from expensiveItems)**는 **forEach(tempItem)**와 같습니다.
- **get current**는 커서를 다음 행으로 이동하지 않고 현재 커서가 표시한 행을 검색합니다.
- **get first** 및 **get last**는 결과에서 각각 첫 번째 행과 마지막 행을 검색합니다.
- **get previous**는 **get next**의 반대로 커서 앞의 행을 검색합니다.

- **get absolute(position)**는 결과 세트에서 특정 위치의 행을 검색합니다. 예를 들어, `get absolute(5) tempItem`은 결과 세트에서 다섯 번째 행을 검색하며 `get absolute(-2) tempItem`은 마지막에서 두 번째 행을 검색합니다.
- **get relative(position)**는 커서의 현재 위치와 관련된 특정 위치의 행을 검색합니다. 예를 들어, `get relative(3) tempItem`은 현재 커서 위치 다음의 세 번째 행을 검색하며 `get relative(-2) tempItem`은 현재 커서 위치 앞의 두 번째 행을 검색합니다. `get relative(0) tempItem`은 `get current tempItem`과 동등합니다.

get next 이외의 해당 위치 관련 **get** 문을 사용하려면 시작부터 끝까지 한 번에 한 행씩 단계적으로 이동하지 말고 커서의 위치를 수동으로 변경하려 하는지 또는 결과 세트가 **화면 이동** 가능인지를 지정해야 합니다. 다음 예제에서와 같이 **scroll** 키워드를 **open** 문에 추가하십시오.

```
open expensiveItems scroll for tempItem;
```

여러 조건 중 임의의 조건(예: 커서가 결과 세트의 끝에 있으므로 결과를 리턴하지 않는 **get next** 문, 같은 결과 세트에 있는 다른 **open** 문 또는 프로그램의 끝)이 발생하면 EGL이 자동으로 커서를 닫습니다. 커서 및 결과 세트에 대한 작업을 완료하고 나면 다음 예에서와 같이 EGL **close** 문을 사용하여 닫을 수 있습니다.

```
close expensiveItems;
```

커서 및 결과 세트를 닫고 나면 커서에 의존하는 **get next** 및 기타 명령문은 더 이상 작동하지 않습니다.

단계적으로 결과 세트 이동

이 학습에서는 **forEach** 문을 사용하여 결과 세트를 단계적으로 이동하여 데이터베이스에서 특정 가격 이상의 항목을 검색할 것입니다.

1. `programs` 패키지에 `selectItems`라는 새 프로그램을 작성하십시오.
2. 프로그램에서 기본 코드를 제거하여 다음과 같이 표시되도록 하십시오.

```
package programs;

program selectItems type BasicProgram {}

    function main()
    end

end
```

3. 다음과 같이 데이터베이스에 액세스하는 동안 EGL에 발생할 수 있는 오류를 처리하는 함수를 추가하십시오.

```
function handleDBException(exception SQLException)
    SysLib.writeStdout("Error accessing database. "::
        "See Troubleshooting section");
    SysLib.writeStdout(exception.message);
end
```

나중에 이 학습서에서 데이터베이스 액세스 시 발생하는 오류 처리에 대해 자세히 설명할 것입니다.

4. 다음과 같이 최대 비용의 `Price` 매개변수를 기반으로 데이터베이스에서 행을 검색하는 함수를 추가하십시오.

```
function printExpensiveItems(maxPrice Price in)
end
```

Price dataItem 파트는 DECIMAL 기본 유형에 따라 다릅니다. 이 유형은 eglderbyr7/primitivetypes/data 파일에 정의되어 있습니다.

5. 콘텐츠 지원을 사용하여 함수 선언에서 Price 매개변수를 추가하지 않은 경우 다음 **import** 문을 파일의 맨 위 **package** 문 바로 아래 추가하십시오.

```
import eglderbyr7.primitivetypes.data.*;
```

콘텐츠 지원을 사용하면 처음 몇 개의 문자를 입력한 다음 Ctrl+Space를 눌러 키워드 이름 및 유형을 완료할 수 있습니다. 콘텐츠 지원을 사용하여 유형(예: Price dataItem 파트)을 입력하면 콘텐츠 지원이 파일에 적절한 **import** 문을 추가합니다.

6. 새 함수에서 **open** 문을 추가하여 결과 세트와 커서를 작성하십시오. 커서가 표시하는 행을 보유할 임시 Item 변수도 필요합니다.

```
tempItem Item;
open expensiveItems for tempItem;
```

7. 다시 한 번, 콘텐츠 지원을 사용하여 Item 레코드 유형을 삽입하지 않은 경우 **import** 문을 추가하여 Record 파트를 범위 안에 가져오도록 하십시오.

```
import eglderbyr7.data.Item;
```

8. **open** 문에 커서를 두고 마우스 오른쪽 단추를 클릭한 다음 **SQL** 문 → 추가를 클릭하여 SQL 코드가 명시적으로 표시되도록 하십시오. **open** 문은 다음과 같습니다.

```
open expensiveItems for tempItem with
#sql{
  select
    EGL.ITEM.ITEM_ID, EGL.ITEM.NAME, EGL.ITEM.IMAGE,
    EGL.ITEM.PRICE, EGL.ITEM.DESCRPTION
  from EGL.ITEM
  where
    EGL.ITEM.ITEM_ID >= :tempItem.ItemId
  order by
    EGL.ITEM.ITEM_ID asc
};
```

9. 명시적 SQL 코드에서 WHERE절을 변경하여 maxPrice 변수 이상인 EGL.ITEM.PRICE 열 값이 함수에 전달된 행만을 선택하십시오.

```
open expensiveItems for tempItem with
#sql{
  select
    EGL.ITEM.ITEM_ID, EGL.ITEM.NAME, EGL.ITEM.IMAGE,
    EGL.ITEM.PRICE, EGL.ITEM.DESCRPTION
  from EGL.ITEM
  where
    EGL.ITEM.PRICE >= :maxPrice
  order by
    EGL.ITEM.ITEM_ID asc
};
```

maxPrice 변수 앞에 콜론을 포함하여 해당 변수가 SQL 키워드나 값이 아닌 EGL 변수임을 표시하십시오.

10. 다음과 같이 **open** 문 다음에 항목에 대한 정보를 콘솔에 인쇄하는 **forEach** 루프를 추가하십시오.

```
forEach (tempItem)
  SysLib.writeStdout(tempItem.Name :: " " ::
    tempItem.Description :: " $" :: tempItem.Price);
end
```

전체 함수는 다음과 같습니다.

```
function printExpensiveItems(maxPrice Price in)

tempItem Item;
open expensiveItems for tempItem with
  #sql{
    select
      EGL.ITEM.ITEM_ID, EGL.ITEM.NAME, EGL.ITEM.IMAGE,
      EGL.ITEM.PRICE, EGL.ITEM.DESCRPTION
    from EGL.ITEM
    where
      EGL.ITEM.PRICE >= :maxPrice
    order by
      EGL.ITEM.ITEM_ID asc
  };

forEach (tempItem)
  SysLib.writeStdout(tempItem.Name :: " " ::
    tempItem.Description :: " $" :: tempItem.Price);
end
```

11. main 함수에서 함수를 호출하여 maxPrice 값을 전달하십시오.

```
function main()
  try
    printExpensiveItems(200);
  onException(exception SQLException)
    handleDBException(exception);
  end
end
```

12. 프로그램을 생성하고 실행하십시오.

프로그램은 데이터베이스에서 판매 중인 항목 중에서 가격이 함수에 전달된 값 이상인 항목의 이름을 인쇄합니다. 예를 들어, 함수에 200을 전달하면 다음 결과가 발생합니다.

```
Laptop PC Great little machine. Take it anywhere with you. $1111.11
Combination Safe Keep your valuables safe and secure. Put 'em in here. $222.22
Desktop PC Lots'a power, at the right price. Includes monitor. $888.55
```

selectItems.egl 파일의 전체 코드가 생성되었습니다. 파일에 빨간색 X 기호로 표시된 오류가 나타나면 코드가 62 페이지의 『학습 3에서 완성된 selectItems.egl 파일』의 코드와 일치하는지 확인하십시오.

학습 체크포인트

이 학습에서는 **open** 및 **forEach**를 사용하여 한 번에 한 행씩 결과 세트에 대해 작업하는 방법에 대해 배웁니다.

커서를 사용하여 한 번에 한 행씩 결과 세트에 대한 작업을 수행하는 것이 **get**을 사용하여 전체 결과 목록을 레코드 배열로 검색하는 것보다 편리하고 효율적입니다.

자세한 정보는 **open** 및 **forEach**의 도움말 항목을 참조하십시오.

학습 4: *prepare*를 사용하여 동적으로 조회 생성

EGL **prepare** 문을 사용하여 동적으로 SQL 문을 생성할 수 있습니다.

이전 학습에서 데이터베이스에서 가격이 매개변수 이상인 항목을 선택하는 함수를 작성했습니다.

```
function printExpensiveItems(maxPrice Price in)

tempItem Item;
open expensiveItems for tempItem with
    #sql{
        select
            EGL.ITEM.ITEM_ID, EGL.ITEM.NAME, EGL.ITEM.IMAGE,
            EGL.ITEM.PRICE, EGL.ITEM.DESCRPTION
        from EGL.ITEM
        where
            EGL.ITEM.PRICE >= :maxPrice
        order by
            EGL.ITEM.ITEM_ID asc
    };

forEach (tempItem)
    SysLib.writeStdout(tempItem.Name :: " " ::
        tempItem.Description :: " $" :: tempItem.Price);
end
```

매개변수 이하인 가격의 항목을 표시하려는 경우를 고려해보십시오. 또는, 최대 및 최소 가격 사이의 항목을 표시하려는 경우를 고려해보십시오. SQL 문의 WHERE절에서의 비교는 하드 코딩되어 있으므로 **open** 문에서 명시적 SQL을 사용하여 세 함수(각 경우에 대해 하나씩)를 코딩해야 합니다.

보다 유연하게 SELECT 문을 작성하는 방법으로는 EGL **prepare** 문을 사용하여 런타임 시 조회를 조합하는 방법이 있습니다. **prepare** 문을 사용할 때 우선 다음과 같이 SQL 코드 텍스트가 들어 있는 STRING 변수를 작성합니다.

```
selectStatement STRING =
    "select * from EGL.ITEM where EGL.ITEM.PRICE >= 500";
```

그런 다음, **prepare** 문을 사용하여 문자열을 SQL 문으로 변환합니다. 명령문의 이름과 선택적으로 레코드 변수를 지정하여 명령문이 작동할 SQLRecord 부분을 표시해야 합니다.

```
tempItem Item;
prepare preparedStatement from selectStatement for tempItem;
```

그런 다음, 다음과 같이 EGL **get**, **open** 또는 **execute** 문의 명시적 SQL 대신 prepared 문을 사용할 수 있습니다(이후에 **execute**에 대해 자세히 학습).

```
open expensiveItems for tempItem with preparedStatement;
```

prepared 문의 일반 오류

이 방식으로 문자열에서 SQL 문을 작성하는 것은 SQL에 능숙한 프로그래머를 위한 고급 기술입니다. EGL은 설계 시 SQL 문이 정확한지 확인하지 않으므로 런타임 시 문자열이 올바른 SQL 문으로 해석되는지 확인해야 합니다. 다음은 prepared 문을 사용하여 작업할 때 일반적으로 발생하는 실수 목록입니다.

호스트 변수

prepared 문에서는 명시적 SQL 문에서와 같이 호스트 변수를 사용할 수 없습니다. 예를 들어 다음과 같습니다.

```
tempItem Item;
maxPrice Price = 500;
selectStatement STRING =
    "select * from EGL.ITEM where EGL.ITEM.PRICE >= :maxPrice";
prepare preparedStatement from selectStatement for tempItem;
```

이 경우, 실제 SQL 문은 :maxPrice 문자열을 호스트 변수로 사용하여 maxPrice 변수의 값으로 바꾸지 않고 있는 그대로를 포함합니다. 결과적으로 잘못된 SQL 문이 생성됩니다.

대신, 변수값을 문자열에 삽입해야 합니다.

```
tempItem Item;
maxPrice Price = 500;
selectStatement STRING =
    "select * from EGL.ITEM where EGL.ITEM.PRICE >= " :: maxPrice;
prepare preparedStatement from selectStatement for tempItem;
```

이 경우, EGL이 문자열을 작성할 때 maxPrice 변수의 값이 해석되어 다음 SQL 문이 작성됩니다.

```
select * from EGL.ITEM where EGL.ITEM.PRICE >= 500
```

나중에 prepared 문에서 변수를 사용하는 방법의 대안에 대해 설명할 것입니다.

공백 지정

SQL 문에서 키워드 사이에 공백을 두어 분리하도록 하십시오. 예를 들어, 여러 문자열 값을 병합 연산자(::<=)로 병합하여 준비하는 다음과 같은 SQL 문이 있습니다.

```
incorrectSQL STRING;
incorrectSQL = "select * from EGL.ITEM where";
incorrectSQL ::= "EGL.ITEM.PRICE >= :maxPrice";
incorrectSQL ::= "order by EGL.ITEM.PRICE";
```

이 예제는 다음과 같은 잘못된 SQL 문을 생성합니다.

```
select * from EGL.ITEM whereEGL.ITEM.PRICE >= :maxPriceorder by EGL.ITEM.PRICE
```

이 명령문은 WHERE 뒤와 ORDER BY 앞에 공백이 필요합니다.

올바르고 적절한 명령문

EGL은 prepared 문의 유효성을 검증하지 않으므로 명령문이 올바른 SQL이 되었는지 사용자가 확인해야 합니다.

명시적 SQL과 같이 prepared SQL 문은 함께 사용되는 EGL 문에 적합해야 합니다. 예를 들어, EGL **get** 및 **open** 문은 결과 세트를 예상합니다. 이 명령문 중 하나와 함께 prepared 문을 사용하려면 SQL 코드가 결과 세트를 리턴해야 합니다. 이러한 이유로 인해 SQL INSERT 또는 DELETE 문을 준비하여 해당 명령문을 **get** 또는 **open**과 사용할 수 없습니다.

학습 4A: prepared 문 사용

이 학습에서는 하나 또는 둘 모두가 널(null)이 될 수 있는 두 개의 매개변수를 승인하도록 이전 학습에서 작성한 프로그램을 변경할 것입니다. 함수는 널(null)이 아닌 매개변수를 기반으로 조회를 작성하여 특정 가격 범위의 항목을 리턴합니다.

1. selectItems.egl 파일을 여십시오.
2. 다음과 같이 널 입력 가능 Price 매개변수를 받는 printItemRange라는 함수를 추가하십시오.

```
function printItemRange(minPrice Price? in, maxPrice Price? in)
end
```

매개변수 유형 다음의 물음표(?)는 변수가 널값을 받을 수 있다는 것을 나타냅니다. 이 방식으로 함수가 결과 범위의 최소값 및 최대값 중 하나 또는 둘 모두를 승인하게 됩니다.

3. 다음과 같이 SQL 문을 보유할 STRING 변수를 작성하고 SELECT 문의 시작을 이 변수에 삽입하십시오.

```
selectStatement STRING = "select * from EGL.ITEM where ";
```

다음 단계는 적절한 WHERE절을 판별하는 것입니다. 다음과 같이 네 가지의 가능한 경우가 있습니다.

- 두 매개변수 모두 널입니다. 이 경우, 함수는 가격이 0 이상인 모든 항목을 인쇄하므로 WHERE절이 where EGL.ITEM.PRICE > 0이어야 합니다.
- 두 매개변수가 지정됩니다. 이 경우, 함수가 SQL BETWEEN 키워드를 사용하여 매개변수와 같거나 그 사이에 있는 값을 인쇄합니다. WHERE절은 where EGL.ITEM.PRICE between :minPrice and :maxPrice여야 합니다.
- 최대값 매개변수만 지정되었습니다. 이 경우, 함수는 가격이 최대값 이하인 모든 항목을 인쇄합니다. where EGL.ITEM.PRICE <= :maxPrice.
- 최소값 매개변수만 지정되었습니다. 이 경우, 함수는 가격이 최소값 이상인 모든 항목을 인쇄합니다. where EGL.ITEM.PRICE >= :minPrice.

EGL에서 여러 방식으로 이 결정을 내릴 수 있지만, 세 개의 중첩된 if 문을 사용하는 방법이 간단합니다.

4. 다음 코드를 함수에 추가하여 WHERE절을 완료하십시오.

```

if (minPrice == NULL && maxPrice == NULL)
    // Two empty parameters;
    // return all rows with PRICE greater than zero.
    selectStatement ::= "EGL.ITEM.PRICE > 0 ";
else
    if (minPrice != NULL && maxPrice != NULL)
        // Both parameters were specified.
        selectStatement ::= "EGL.ITEM.PRICE between " :: minPrice :: " and " :: maxPrice :: " ";
    else
        // Only one parameter was specified.
        if (minPrice == NULL)
            // Maximum was specified.
            selectStatement ::= "EGL.ITEM.PRICE <= " :: maxPrice :: " ";
        else
            // Minimum was specified.
            selectStatement ::= "EGL.ITEM.PRICE >= " :: minPrice :: " ";
        end
    end
end
end

```

5. 다음과 같이 ORDER BY절로 명령문을 완료하십시오.

```
selectStatement ::= "order by EGL.ITEM.PRICE";
```

6. SQL 문이 올바른지 확인하려면 콘솔에 인쇄하십시오.

```
SysLib.writeStdout("Completed query: "::selectStatement);
```

7. prepared 문이 액세스할 테이블에서 작성된 레코드를 참조하여 사용할 명령문을 준비하십시오.

```

tempItem Item;
prepare preparedStatement
    from selectStatement
    for tempItem;

```

8. **open** 문에서 명시적 SQL 대신 prepared 문을 사용하십시오.

```
open expensiveItems for tempItem with preparedStatement;
```

9. 다음과 같이 조회 결과를 인쇄하십시오.

```

foreach (tempItem)
    SysLib.writeStdout(tempItem.Name :: " " ::
        tempItem.Description :: " $" :: tempItem.Price);
end

```

전체 함수는 다음과 같습니다.

```

function printItemRange(minPrice Price? in, maxPrice Price? in)

    selectStatement STRING = "select * from EGL.ITEM where ";

    if (minPrice == NULL && maxPrice == NULL)
        // Two empty parameters;
        // return all rows with PRICE greater than zero.
        selectStatement ::= "EGL.ITEM.PRICE > 0 ";
    else

        if (minPrice != NULL && maxPrice != NULL)
            // Both parameters were specified.
            selectStatement ::= "EGL.ITEM.PRICE between " :: minPrice :: " and " :: maxPrice :: " ";
        else

```

```

    // Only one parameter was specified.
    if (minPrice == NULL)
        // Maximum was specified.
        selectStatement ::= "EGL.ITEM.PRICE <= " :: maxPrice :: " ";
    else
        // Minimum was specified.
        selectStatement ::= "EGL.ITEM.PRICE >= " :: minPrice :: " ";
    end
end

end

// Regardless of parameters, sort by price.
selectStatement ::= "order by EGL.ITEM.PRICE";

// Print completed SQL code.
SysLib.writeStdout("Completed query: "::selectStatement);

// Prepare the statement to be used.
tempItem Item;
prepare preparedStatement
from selectStatement
for tempItem;

// Use the prepared statement in place of explicit SQL.
open expensiveItems for tempItem with preparedStatement;

// Print the results to the console.
foreach (tempItem)
    SysLib.writeStdout(tempItem.Name :: " " ::
        tempItem.Description :: " $" :: tempItem.Price);
end

end

```

10. 다음과 같이 프로그램의 main 함수에서 함수를 호출하십시오.

```

function main()
    minPrice, maxPrice Price?;
    minPrice = 50;
    maxPrice = 500;
    try
        printItemRange(minPrice, maxPrice);
    onException(exception SQLException)
        handledDBException(exception);
    end
end
end

```

11. 프로그램을 생성하고 실행하십시오.

위 예제의 매개변수 값을 사용하여 함수가 가격이 50 - 500인 항목을 인쇄합니다.

```

Completed query: select * from EGL.ITEM where EGL.ITEM.PRICE
    between 50.00 and 500.00 order by EGL.ITEM.PRICE
Serial Mouse Great little mouse. All kinds'a applications. $65.22
Keyboard Got QWERTY? If not, go get this awesome flat keyboard. $78.99
Watch Keep time - and look rakish! $99.01
Shredder Don't let that politically-sensitive document fall into the wrong hands! $198.99
Combination Safe Keep your valuables safe and secure. Put 'em in here. $222.22

```

다음 예제에서와 같이 매개변수 값을 편집하여 널 값을 포함하는 다른 범위를 시도할 수 있습니다.

```

function main()
  minPrice, maxPrice Price?;
  minPrice = 500;
  maxPrice = null;
  try
    printItemRange(minPrice, maxPrice);
  onException(exception SQLException)
    handleDBException(exception);
end
end

```

500과 널 값을 전달하면 다음과 같은 결과가 발생합니다.

```

Completed query: select * from EGL.ITEM where
EGL.ITEM.PRICE >= 500.00 order by EGL.ITEM.PRICE
Desktop PC Lots'a power, at the right price. Includes monitor. $888.55
Laptop PC Great little machine. Take it anywhere with you. $1111.11

```

selectItems.egl 파일의 전체 코드가 생성되었습니다. 파일에 빨간색 X 기호로 표시된 오류가 나타나면 코드가 63 페이지의 『학습 4A에서 완성된 selectItems.egl 파일』의 코드와 일치하는지 확인하십시오.

학습 4B: *prepare* 문의 호스트 변수

이전 예제의 SQL 문은 WHERE절의 변수가 고정되었다는 점에서 한계가 있습니다. 명령문에서 사용할 변수를 알기 전에 prepared 문을 작성할 수도 있습니다. 이 유형의 명령문은 다소 복잡하지만 명령문을 준비한 다음에 세부사항을 입력하려는 경우에는 유연성을 제공할 수 있습니다.

이전 예제에서 나머지 조회 문자열과 병합하기 위해 변수값을 해석해야 했습니다.

```

selectStatement STRING = "select * from EGL.ITEM where ";
...
selectStatement ::= "EGL.ITEM.PRICE between " :: minPrice :: " and " :: maxPrice :: " ";

```

prepared 문을 작성할 때 명령문에 물음표(?)를 두어 나중에 값을 입력할 것이라는 것을 나타낼 수 있습니다.

```

selectStatement2 STRING = "select * from EGL.ITEM where " ::
  "EGL.ITEM.PRICE between ? and ? " ::
  "order by EGL.ITEM.PRICE";

```

```

tempItem Item;
prepare preparedStatement2
  from selectStatement2
  for tempItem;

```

이제, prepared 문에는 두 호스트 변수의 위치가 있으며 BETWEEN절에서 물음표로 표시됩니다. prepared 문을 사용할 준비가 되면 **using** 문을 사용하여 값을 삽입하십시오. 예를 들어, 5 - 500의 가격이 있는 행을 검색하려면 다음과 같이 리터럴 5와 500을 명령문에 전달하십시오.

```

open expensiveItems2 for tempItem with preparedStatement2
  using 5, 500;

```

물론 다음과 같이 **using** 키워드와 함께 변수를 사용할 수도 있습니다.

```

open expensiveItems2 for tempItem with preparedStatement2
  using minPrice, maxPrice;

```

명령문을 한 번 조합한 다음 서로 다른 값을 사용하여 여러 번 호출하려는 경우, 이 방식으로 prepared 문에서 호스트 변수를 사용하면 매우 유용합니다. 예를 들어, 이 함수는 같은 prepared 문을 세 번 사용하여 서로 다른 세 범위의 값을 인쇄합니다.

```
function printRanges()

    tempItem Item;
    selectStatement string = "select * from EGL.ITEM where " ::
        "EGL.ITEM.PRICE between ? and ? " ::
        "order by EGL.ITEM.PRICE";

    prepare preparedStatement from selectStatement for tempItem;

    SysLib.writeStdout("\nPrinting values between 0 and 200");
    open expensiveItems for tempItem with preparedStatement using 0, 200;
    foreach(tempItem)
        SysLib.writeStdout(tempItem.Name :: " " :: tempItem.Description ::
            " $" :: tempItem.Price);
    end

    SysLib.writeStdout("\nPrinting values between 200 and 500");
    open expensiveItems for tempItem with preparedStatement using 200, 500;
    foreach(tempItem)
        SysLib.writeStdout(tempItem.Name :: " " :: tempItem.Description ::
            " $" :: tempItem.Price);
    end

    SysLib.writeStdout("\nPrinting values between 500 and 1,000");
    open expensiveItems for tempItem with preparedStatement
        using 500, 1000;
    foreach(tempItem)
        SysLib.writeStdout(tempItem.Name :: " " :: tempItem.Description ::
            " $" :: tempItem.Price);
    end

end
```

이 방식으로 호스트 변수를 사용하면 각 경우의 명령문을 수정하는 대신 단일 prepared 문을 사용하여 이전 섹션에서 프로그램을 다시 쓸 수 있습니다.

1. 이전 섹션에서와 같이 selectItems.egl 파일에서 최대값 및 최소값 매개변수를 승인하는 새 함수를 추가하십시오.

```
function printItemRange2(minPrice Price? in, maxPrice Price? in)

end
```

이 함수에서는 먼저 명령문을 준비한 다음 변수를 나중에 삽입합니다.

2. 다음과 같이 prepared 문을 보유할 string 변수를 작성한 다음 **prepare** 문을 사용하여 준비하십시오.

```
selectStatement STRING = "select * from EGL.ITEM where " ::
    "EGL.ITEM.PRICE between ? and ? " ::
    "order by EGL.ITEM.PRICE";
```

```
tempItem Item;
prepare preparedStatement
  from selectStatement
  for tempItem;
```

이 prepared 문에 두 매개변수를 전달하여 검색의 경계를 나타낼 것입니다. 최대 가격이 지정되지 않은 경우 일단 명령문을 준비하고 나면 EGL.ITEM.PRICE between ? and ?절을 EGL.ITEM.PRICE >= ?로 변환할 수 없으므로 검색할 상위 경계값을 알아야 합니다. 따라서, 테이블의 가장 비싼 가격을 판별하여 상위 경계값으로 사용해야 합니다. SQL MAX() 함수를 사용하여 이 상위 경계값을 쉽게 판별할 수 있습니다.

3. 다음과 같이 Item 변수를 작성하고 MAX() 함수를 사용하여 데이터베이스에서 가장 비싼 가격의 항목을 검색하여 레코드 변수의 Price 필드에 두십시오.

```
get mostExpensiveItem
  into mostExpensiveItem.Price
  with
    #sql{
      select
        max(EGL.ITEM.PRICE)
      from EGL.ITEM
    };
```

이제 mostExpensiveItem 레코드는 Items 테이블의 가장 높은 가격을 포함합니다.

4. 지정된 매개변수를 판별하고 그에 알맞는 prepared 문을 사용하십시오.

```
if (minPrice == NULL && maxPrice == NULL)
  // Two empty parameters;
  open expensiveItems for tempItem with preparedStatement
    using 0, mostExpensiveItem.Price;
else
  if (minPrice != NULL && maxPrice != NULL)
    // Both parameters were specified.
    open expensiveItems for tempItem with preparedStatement
      using minPrice, maxPrice;
  else
    // Only one parameter was specified.
    if (minPrice == NULL)
      // Maximum was specified.
      open expensiveItems for tempItem with preparedStatement
        using 0, maxPrice;
    else
      // Minimum was specified.
      open expensiveItems for tempItem with preparedStatement
        using minPrice, mostExpensiveItem.Price;
    end
  end
end

end
```

5. 다음과 같이 **forEach**를 사용하여 결과를 인쇄하십시오.

```

forEach (tempItem)
  SysLib.writeStdout(tempItem.Name :: " " ::
    tempItem.Description :: " $" :: tempItem.Price);
end

```

전체 함수는 다음과 같습니다.

```

function printItemRange2(minPrice Price? in, maxPrice Price? in)

  selectStatement STRING = "select * from EGL.ITEM where " ::
    "EGL.ITEM.PRICE between ? and ? " ::
    "order by EGL.ITEM.PRICE";

  // Prepare the statement to be used.
  tempItem Item;
  prepare preparedStatement
    from selectStatement
    for tempItem;

  mostExpensiveItem Item;
  get mostExpensiveItem
    into mostExpensiveItem.Price
    with
    #sql{
      select
        max(EGL.ITEM.PRICE)
      from EGL.ITEM
    };

  if (minPrice == NULL && maxPrice == NULL)
    // Two empty parameters;
    open expensiveItems for tempItem with preparedStatement
      using 0, mostExpensiveItem.Price;
  else

    if (minPrice != NULL && maxPrice != NULL)
      // Both parameters were specified.
      open expensiveItems for tempItem with preparedStatement
        using minPrice, maxPrice;
    else
      // Only one parameter was specified.
      if (minPrice == NULL)
        // Maximum was specified.
        open expensiveItems for tempItem with preparedStatement
          using 0, maxPrice;
      else
        // Minimum was specified.
        open expensiveItems for tempItem with preparedStatement
          using minPrice, mostExpensiveItem.Price;
      end
    end

  end

end

// Print the results to the console.
forEach (tempItem)
  SysLib.writeStdout(tempItem.Name :: " " ::

```

```
tempItem.Description :: " $" :: tempItem.Price);
end
```

```
end
```

6. 다음과 같이 이전 함수가 아니라 새 함수를 사용하도록 main 함수를 수정하십시오.

```
function main()
  minPrice, maxPrice Price?;
  minPrice = 50;
  maxPrice = 500;
  try
    printItemRange2(minPrice, maxPrice);
  onException(exception SQLException)
    handleDBException(exception);
  end
end
```

7. 널(null) 값을 포함하는 서로 다른 값의 minPrice 및 maxPrice를 사용하여 새 프로그램을 저장하고 생성하여 실행하십시오. 결과는 이전 함수와 같지만 함수가 결과에 도달하는 방식은 다릅니다.

selectItems.egl 파일의 전체 코드가 생성되었습니다. 파일에 빨간색 X 기호로 표시된 오류가 나타나면 코드가 65 페이지의 『학습 4B에서 완성된 selectItems.egl 파일』의 코드와 일치하는지 확인하십시오.

학습 체크포인트

올바른 prepared 문을 작성하는 것은 복잡할 수 있습니다. 하지만 그 결과로 생성된 명령문은 명시적 SQL 문으로 하드 코딩된 명령문보다 더 유연합니다. 이후의 학습에서는 다른 방식으로 SQL 문을 사용자 정의할 수 있는, SQLRecord 파트의 기본 SQL 코드 설정에 대해 설명합니다.

prepared 문에 대한 자세한 정보는 prepare를 참조하십시오.

학습 5: 테이블 결합을 사용하여 더욱 복잡한 데이터 검색

지금까지는 각 데이터 액세스 함수가 오직 한 테이블에서만 행을 검색했습니다. 이 학습에서는 EGL과 SQL 테이블 결합을 사용하여 한 번에 여러 테이블에 대한 작업을 수행하는 방법을 설명합니다.

SQL 테이블 결합에 대한 전체 설명은 이 학습서의 범위를 벗어납니다. 쉽게 말해서, 테이블 결합은 둘 이상의 관련된 테이블로부터 결과 세트를 조합합니다.

예를 들어, 데이터베이스의 CUSTOMERS 테이블은 고객의 성 및 이름과, 1차 키로서 ID 번호를 포함합니다 (기타 여러 컬론 포함).

표 2. CUSTOMER 테이블의 샘플 데이터

CUSTOMER_ID	FIRST_NAME	LAST_NAME
1	Fred	Filibuster
2	Andy	Lundquist
3	Billie	Kingman

ORDERS 테이블에는 1차 키로서 주문 ID 번호와 주문 양을 포함합니다. 테이블에는 주문을 한 고객의 ID 번호도 포함되며 해당 열은 외부 키가 됩니다.

표 3. ORDERS 테이블의 샘플 데이터

ORDER_ID	CUSTOMER_ID	ORDER_AMOUNT
1	1	111.11
2	1	222.22
3	1	333.33

이 경우, CUSTOMER_ID 열이 두 테이블 사이의 관계를 작성하므로 이 열은 테이블 결합에 사용할 좋은 후보입니다. 이 학습에서는 이 두 테이블을 사용자 정의된 SQLRecord 파츠와 결합하여 고객의 주문과 해당 고객을 일치시키는 결과 세트를 생성합니다.

테이블 결합은 복잡한 데이터베이스 오퍼레이션이므로 데이터베이스의 동작에 대해 잘 이해하고 있어야 하며 출력을 주의하여 테스트해야 합니다. 이 학습에서 표시되는 바와 같이 예상 데이터를 항상 얻지는 못합니다.

- 다음과 같이 새 SQLRecord 파츠를 보유할 새 EGL 소스 파일을 작성하십시오.
 - 프로젝트 탐색기 보기에서 EGLSQL 프로젝트를 마우스 오른쪽 단추로 클릭한 다음 새로 작성 → EGL 소스 파일을 클릭하십시오.
 - 새 EGL 소스 파일 창에서 패키지 필드에 data라는 이름을 입력하십시오.
 - EGL 소스 파일 이름 필드에 records를 입력하십시오.
 - 완료를 클릭하십시오.

새 EGL 소스 파일이 작성되어 편집기에서 열립니다.

- 새 파일에서 **package** 문 아래 새 SQLRecord의 이름을 입력하여 CUSTOMER와 ORDERS 테이블을 결합하십시오.

```
record OrderCustomerJoin type SQLRecord
{
  tableNames = [{"EGL.CUSTOMER", "C"}, {"EGL.ORDERS", "O"}]
}
end
```

이 레코드의 첫 번째 행은 기타 SQLRecord 파츠와 동일합니다. 레코드의 이름과 SQLRecord 스테레오 타입을 지정합니다. 두 번째 행은 이 레코드가 연관될 테이블을 **tableNames** 특성 값으로 지정합니다. 이전 학습에서 작업한 단일 테이블 레코드는 이 특성에서 한 테이블만 나열합니다(예: Orders 레코드의 ORDERS 테이블).

```
record Orders type sqlRecord {
  tablename=["EGL.ORDERS"]
}
...
```

작성 중인 OrderCustomerJoin 레코드는 두 테이블 모두에서 데이터를 검색하도록 **tableNames** 특성에 두 테이블을 나열해야 합니다. 구분을 쉽게 하기 위해 레코드는 각 테이블 이름의 별명을 포함합니다 (ORDERS의 경우 "O" 및 CUSTOMERS의 경우 "C"). 이러한 별명을 통해 레코드에서 테이블 이름을 보다 쉽게 참조할 수 있습니다.

이제, 레코드 파츠의 기본 정보를 지정했으므로 EGL이 나머지 파츠를 채울 수 있습니다. 그러나, 우선 EGL이 연결 정보를 검색할 데이터베이스를 지정해야 합니다.

- 다음과 같이 EGL의 기본 SQL 데이터베이스로 Derby 데이터베이스를 설정하십시오.
 - 창 → 환경 설정을 클릭하십시오.

- b. 환경 설정 창에서 **EGL**을 펼치고 **SQL 데이터베이스 연결**을 클릭하십시오.
 - c. **연결 목록**에서 **EGLDerbyR7** 또는 **EGLDerbyR71**이라는 데이터베이스 연결을 선택하십시오.
 - d. **확인**을 클릭하십시오.
4. 다시 **records.egl** 파일에서 커서를 레코드의 이름에 두십시오.
 5. **OrderCustomerJoin** 레코드의 이름에 커서를 둔 상태로 마우스 오른쪽 단추를 클릭한 다음 **SQL 레코드** → **SQL 검색**을 클릭하십시오. EGL SQL 검색 기능
 6. 암호 프롬프트에서 사용자 이름 및 암호의 기본값을 그대로 두십시오. 샘플 데이터베이스에는 특정 사용자 이름이나 암호가 필요하지 않습니다. EGL SQL 검색 기능이 **keyItems** 특성과 테이블의 행 필드를 포함하여 나머지 **SQLRecord** 파트를 작성합니다.

```
record OrderCustomerJoin type SQLRecord
{tableNames = [{"EGL.CUSTOMER", "C"}, {"EGL.ORDERS", "O"}],
keyItems=[CUSTOMER_ID, ORDER_ID]}

CUSTOMER_ID int
{column="C.CUSTOMER_ID", isReadOnly=yes};
FIRST_NAME string
{column="C.FIRST_NAME", isReadOnly=yes,
isSqlNullable=yes, sqlVariableLen=yes, maxlen=30};
LAST_NAME string
{column="C.LAST_NAME", isReadOnly=yes,
isSqlNullable=yes, sqlVariableLen=yes, maxlen=30};
...
ORDER_ID int
{column="O.ORDER_ID", isReadOnly=yes};
orders_CUSTOMER_ID int
{column="O.CUSTOMER_ID", isReadOnly=yes,
isSqlNullable=yes};
ORDER_AMOUNT decimal(8,2)
{column="O.ORDER_AMOUNT", isReadOnly=yes,
isSqlNullable=yes};
ORDER_DETAILS string
{column="O.ORDER_DETAILS", isReadOnly=yes,
isSqlNullable=yes, sqlVariableLen=yes, maxlen=111};
end
```

각 필드의 열 특성 값은 **tableNames** 특성에 지정된 대로 테이블의 별명으로 시작됩니다. 이 별명을 사용하여 특정 필드가 연관되는 테이블을 추적할 수 있습니다.

7. **records.egl** 파일을 저장하되 닫지는 마십시오.
8. **printCustomerOrders**라는 프로그램 패키지에서 새 프로그램 파트를 작성하십시오.
9. 새 프로그램에서 기본 코드를 제거하십시오.
10. 프로그램의 **main** 함수에서 사용자가 방금 작성한 **OrderCustomerJoin** 레코드를 기반으로 배열 변수를 작성하십시오.

```
allCustomerOrders OrderCustomerJoin[0];
```

Ctrl+Space를 눌러 레코드 파트 유형을 삽입하도록 콘텐츠 지원을 사용하십시오. 콘텐츠 지원을 사용하지 않는 경우 **import** 문을 파일의 맨 위 **package**문 바로 아래에 수동으로 추가해야 합니다.

```
import data.OrderCustomerJoin;
```

11. **open** 및 **forEach** 문을 사용하여 데이터베이스에서 행을 검색하고 콘솔에 값을 인쇄하십시오.

```
tempRec OrderCustomerJoin;
open resultSet for tempRec;
forEach (tempRec)
    SysLib.writeStdout(tempRec.CUSTOMER_ID :: " " ::
        tempRec.LAST_NAME :: " " :: tempRec.ORDER_ID);
end
```

12. 프로그램을 저장하고 생성하여 실행하십시오. 사용자가 테이블 결합의 함정을 잘 이해하고 있다면, 이 테이블 결합의 오류와 출력이 그리 유용하지 않다는 것을 알아챌 것입니다.

```
1 Filibuster 1
1 Filibuster 2
1 Filibuster 3
...
1 Filibuster 22
1 Filibuster 23
2 Lundquist 1
2 Lundquist 2
2 Lundquist 3
2 Lundquist 4
...
2 Lundquist 22
2 Lundquist 23
3 Kingman 1
3 Kingman 2
...
```

데이터베이스가 테이블 사이의 논리적 교차 참조를 작성하지 않았습니다. 그 대신, 가능한 모든 조합으로 CUSTOMERS 테이블의 각 행을 ORDERS 테이블의 각 행과 쌍으로 연결합니다. 이 결과에서 각 고객은 모두를 주문한 것으로 기록되며, 이 기록은 올바르지 않습니다. **open** 문 뒤에서 SQL 코드를 명시적으로 표시되게 만들면 SQL이 WHERE절을 사용하지 않고 두 테이블 모두에서 모든 행을 선택하는 것을 확인할 수 있습니다.

```
open resultSet for tempRec with
    #sql{
        select
            C.CUSTOMER_ID, C.FIRST_NAME, C.LAST_NAME, C.PASSWORD,
            C.PHONE, C.EMAIL_ADDRESS, C.STREET, C.APARTMENT,
            C.CITY, C.STATE, C.POSTALCODE, C.DIRECTIONS,
            O.ORDER_ID, O.CUSTOMER_ID, O.ORDER_AMOUNT,
            O.ORDER_DETAILS, O.ORDER_DATE, O.ORDER_STATUS
        from EGL.CUSTOMER C, EGL.ORDERS O
        order by
            C.CUSTOMER_ID, O.ORDER_ID asc
    };
```

테이블을 효과적으로 결합하려면 테이블 유니온을 리턴하지 않고 테이블을 논리적으로 병합하는 방법을 데이터베이스에 알려야 합니다. 명시적 SQL 코드를 편집하여 이 작업을 수행할 수 있지만, 이 경우 레코드를 사용할 때마다 작업을 수행하거나 정확하지 않은 데이터를 검색할 수도 있다는 위험이 있습니다. 대신, 내재적 SQL 코드가 항상 효과적인 테이블 결합을 수행하도록 이 레코드의 기본 선택 조건을 설정합니다.

13. **open** 문 뒤에서 SQL 코드를 명시적으로 만든 경우 레코드 이름에 커서를 두고 마우스 오른쪽 단추로 클릭한 다음 **SQL** 문 → 제거를 클릭하여 다시 내재적으로 만드십시오.
14. records.egl 파일의 레코드 파트 정의로 되돌아가십시오.
15. 다음과 같이 레코드 파트 정의에 **defaultSelectCondition** 특성을 추가하십시오.

```
record OrderCustomerJoin type SQLRecord
{
  tableNames = ["EGL.CUSTOMER", "C"], ["EGL.ORDERS", "O"],
  keyItems=[CUSTOMER_ID, ORDER_ID],
  defaultSelectCondition = #sqlCondition{ condition }}

```

defaultSelectCondition 특성은 이 레코드와 **get** 또는 **open**을 사용할 때 내재적 SQL 코드와 기본 명시적 SQL 코드에 WHERE절을 지정합니다.

16. **#sqlCondition{}** 블록에서 기본 "condition" 텍스트를 바꾸고 이 두 테이블 사이의 SQL 테이블 결합에 올바른 WHERE절을 입력하여 전체 테이블 이름이 아니라 테이블 별명을 사용하도록 하십시오.

```
defaultSelectCondition = #sqlCondition{ O.CUSTOMER_ID = C.CUSTOMER_ID }
```

키워드 WHERE를 추가할 필요가 없습니다. 이 경우에는 EGL이 WHERE를 SQL 코드에 자동으로 삽입합니다.

17. 파일을 저장하십시오. 이제, **open** 문 뒤에서 SQL 코드를 명시적으로 만든 경우 다음과 같이 표시됩니다.

```
open resultSet for tempRec with
  #sql{
    select
      C.CUSTOMER_ID, C.FIRST_NAME, C.LAST_NAME, C.PASSWORD,
      C.PHONE, C.EMAIL_ADDRESS, C.STREET, C.APARTMENT,
      C.CITY, C.STATE, C.POSTALCODE, C.DIRECTIONS,
      O.ORDER_ID, O.CUSTOMER_ID, O.ORDER_AMOUNT,
      O.ORDER_DETAILS, O.ORDER_DATE, O.ORDER_STATUS
    from EGL.CUSTOMER C, EGL.ORDERS O
    where
      O.CUSTOMER_ID = C.CUSTOMER_ID
    order by
      C.CUSTOMER_ID, O.ORDER_ID asc
  };

```

이제, SQL 코드는 리턴된 행을 CUSTOMER 및 ORDERS 테이블 모두에 같은 고객 번호가 있는 행으로만 제한하는 WHERE절을 포함합니다.

18. 프로그램을 저장하고 생성하여 실행하십시오.

효과적으로 테이블을 결합하면 이 프로그램의 결과는 각각의 주문을 한 고객을 표시합니다.

```
1 Filibuster 1
1 Filibuster 2
1 Filibuster 3
...
1 Filibuster 11
1 Filibuster 18
2 Lundquist 7
2 Lundquist 8
2 Lundquist 12

```

2 Lundquist 15
2 Lundquist 20
3 Kingman 13
4 Liebowitz 14
6 Springsteen 22
7 St. Louis 16
8 Sharov 17
9 Hudak 23

이 학습에서 사용하는 두 파일의 전체 코드를 살펴보십시오. 파일에 빨간색 X 기호로 표시된 오류가 나타나면 사용자의 코드가 68 페이지의 『학습 5에서 완성된 printCustomerOrders.egl 및 records.egl 파일』의 코드와 일치하는지 확인하십시오.

학습 6: 임의의 SQL 코드 실행

이 학습에서는 EGL에서 보다 유연하게 SQL 코드를 사용하는 방법에 대해 설명합니다.

지금까지는 EGL 문과 일치하는 SQL 문을 사용하도록 제한되었습니다. 예를 들어, **get** 문과 연관된 SQL 코드를 편집할 수 있지만 SQL 코드는 **SELECT** 문으로만 제한되어 있습니다. 다시 말해, **get** 문의 명시적 SQL 코드에 **SQL DELETE** 문을 넣을 수 없습니다. 명시적 SQL을 편집할 수는 있지만 SQL 코드는 여전히 EGL 키워드와 일치해야 합니다.

그렇다면 직접적인 EGL 유사문이 없는 SQL 문을 어떻게 사용할 수 있을까요? 사용자는 Workbench 도구를 통해 직접적으로 또는 EGL **execute** 문을 통해 EGL 키워드와 연관되지 않은 SQL 문을 실행할 수 있습니다.

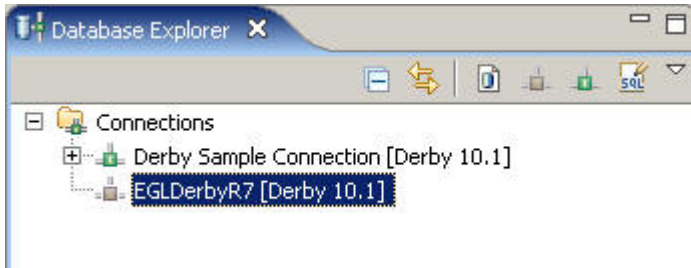
SQL 빌더를 사용하여 SQL 문 작성

Workbench에는 EGL 응용프로그램 외부의 데이터베이스와 작업하는 데 사용할 수 있는 여러 도구가 포함되어 있습니다. 이 절에서는 SQL 빌더를 사용하여 보다 직접적으로 샘플 데이터베이스에 대해 작업합니다.

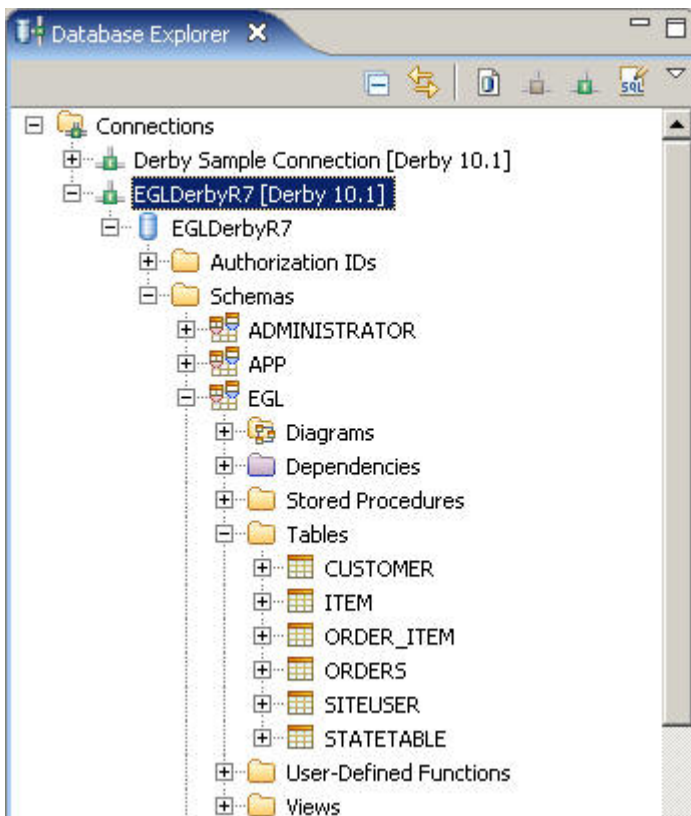
1. 다음과 같이 데이터 Perspective로 전환하십시오.
 - a. 창 → **Perspective** 열기 → 기타를 클릭하십시오.
 - b. Perspective 열기 창에서 데이터를 클릭하십시오.
 - c. 확인을 클릭하십시오.

데이터베이스 작업에 사용되는 보기가 포함되어 있는 데이터 Perspective가 열립니다. 주로 데이터베이스 탐색기 보기를 사용하여 작업을 수행할 것입니다.

2. 데이터베이스 탐색기 보기에서 연결을 펼치십시오. 데이터베이스 탐색기 보기는 테스트에 사용하는 샘플 Derby 연결과 데이터베이스에 대한 프로젝트의 연결을 EGLDerbyR7로 표시합니다.



3. EGLDerbyR7 연결을 마우스 오른쪽 단추로 클릭한 다음 다시 연결을 클릭하십시오. 이제 데이터베이스 탐색기 보기가 데이터베이스에 연결되므로 데이터 Perspective의 도구를 사용하여 데이터베이스에 대한 작업을 수행할 수 있습니다. 그러나, Derby는 데이터베이스에 대한 연결을 한 번에 하나씩만 허용하므로 데이터베이스 탐색기 보기가 연결된 경우 EGL 프로그램은 데이터베이스에 연결할 수 없습니다.
4. 데이터베이스 권한 부여 창에서 확인을 클릭하십시오.
5. **EGLDerbyR7 → EGLDerbyR7 → 스키마 → EGL → 테이블**을 펼치십시오. 이제, 데이터베이스 탐색기 보기가 이 학습서에서 다루는 데이터베이스의 테이블을 나열합니다. 다른 테이블은 메타데이터이며 이 학습서에는 중요하지 않습니다.



6. CUSTOMERS 테이블을 마우스 오른쪽 단추로 클릭한 다음 데이터 → 샘플 콘텐츠를 클릭하십시오. 데이터 출력 보기는 CUSTOMERS 테이블의 데이터를 표시합니다.

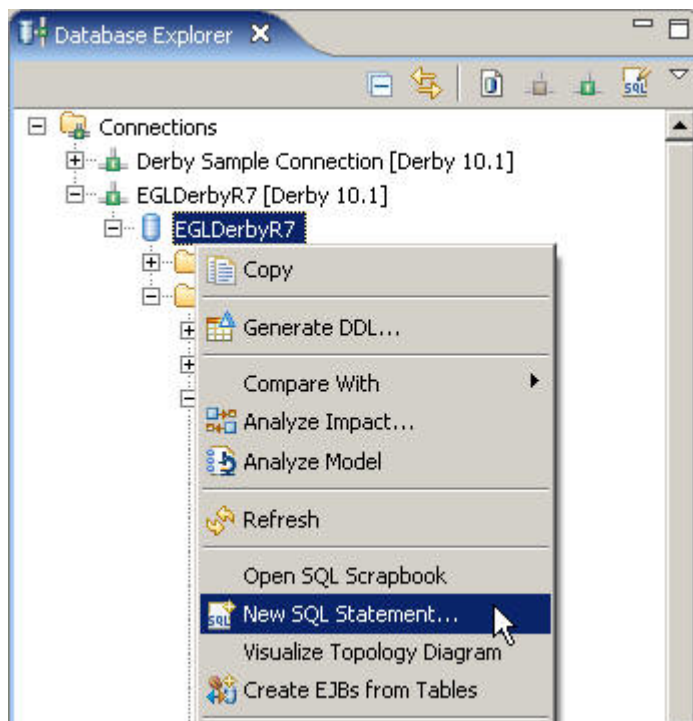
Sample Contents

Messages Parameters Results Profiling Data

CUSTOMER...	FIRST_NAME	LAST_NAME	PASSWORD	PHONE	EMAIL_AD...	STREET	APARTM
1	Fred	Filibuster	dsds	(201)652-3...	f_filiBust@...	14 Maple A...	1A
2	Andy	Lundquist	bbb	(221)545-5...	alundy@yn...	1338 Dean ...	2B
3	Billie	Kingman	ccc	(261)898-4...	bjPride@ec...	14 Mayberr...	TGIF
4	Francine	Liebowitz	aa	(414)923-2...	obscureRef...	1234 Politic...	
5	Ornette	Coleman	eee	(518)132-4...	horns@jazz...	5829 Two P...	
6	Ellen	Springsteen	23	(712)469-5...	musicNot@...	88 Allen Str...	4A
7	Martin	St. Louis	StLouis	(670)250-5...	iAmFst@nhl...	54 Belvede...	
8	Sergey	Sharov	ds	(413)452-1...	s_sharov@...	14 Belmont ...	
9	Melodie	Hudak	afda	(860)742-0...	mhj@firest...	43 Spring S...	

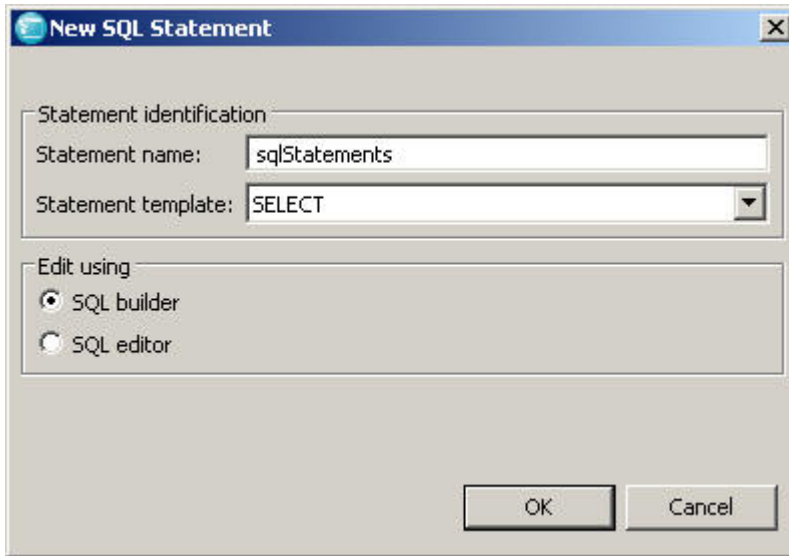
데이터 출력 보기는 EGL **get** 문 뒤의 내재적 SELECT 문과 같이 기본 SELECT 문을 실행합니다. 보다 직접적으로 데이터베이스에 대한 작업을 수행하고 SQL 코드를 EGL 프로그램에 추가하기 전에 테스트하려면, SQL 빌더를 사용하여 대화식으로 SQL 명령을 작성하고 실행할 수 있습니다.

- 데이터베이스 탐색기 보기에서 파란색 데이터베이스 아이콘이 표시하는 EGLDerbyR7 데이터베이스를 마우스 오른쪽 단추로 클릭한 다음 새 **SQL** 문을 클릭하십시오.



- 새 SQL 문 창에서 명령문의 이름을 sqlStatements로 지정하십시오.

9. 명령문 템플릿 필드는 SELECT로 설정하고 편집 도구 단일 선택 단추 그룹은 SQL 빌더로 설정된 상태로 두십시오.

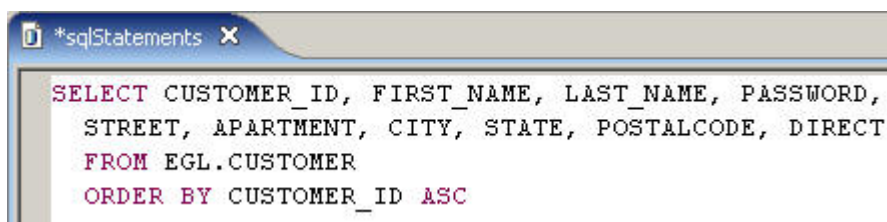


10. 확인을 클릭하십시오. SQL 빌더가 열립니다. 이 편집기에서 SQL 문을 작성한 다음 응용프로그램에 추가하기 전에 테스트할 수 있습니다.
11. 예를 들어, 고객 레코드의 기본 SQL 문을 SQL 빌더에 붙여넣으십시오. EGL 편집기의 명시적 SQL에서 이 명령문(또는 기타 임의의 SQL 문)을 복사하거나 다음 코드를 사용할 수 있습니다.

```
select
  EGL.CUSTOMER.CUSTOMER_ID, EGL.CUSTOMER.FIRST_NAME,
  EGL.CUSTOMER.LAST_NAME, EGL.CUSTOMER.PASSWORD,
  EGL.CUSTOMER.PHONE, EGL.CUSTOMER.EMAIL_ADDRESS,
  EGL.CUSTOMER.STREET, EGL.CUSTOMER.APARTMENT,
  EGL.CUSTOMER.CITY, EGL.CUSTOMER."STATE",
  EGL.CUSTOMER.POSTALCODE, EGL.CUSTOMER.DIRECTIONS
from EGL.CUSTOMER
order by
  EGL.CUSTOMER.CUSTOMER_ID asc
```

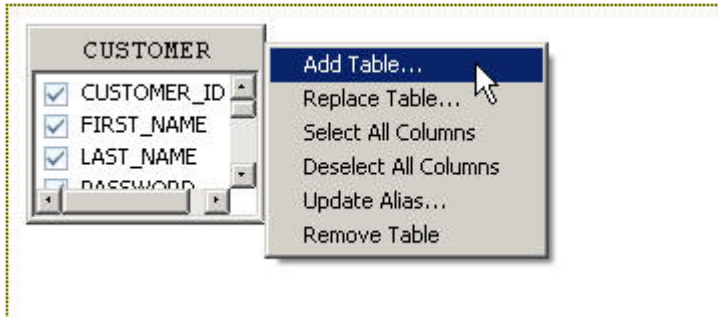
SQL 빌더는 다양한 방식으로 SQL 코드를 표시합니다.

- SQL 빌더의 맨 위에 있는 코드 편집기를 사용하여 EGL 편집기처럼 같이 SQL 문을 편집할 수 있습니다. 그러나, 이 편집기에는 SQL 문을 처리하는 추가 옵션이 있습니다. 예를 들어, EGL 코드에서 이 편집기로 기본 SQL 문을 붙여넣을 때 SQL 빌더는 이 명령문을 단순화하고 다시 포맷합니다.



또한, 이 SQL 편집기는 SQL 문의 콘텐츠 지원도 제공합니다. EGL 콘텐츠 지원과 같이 Ctrl+Space를 눌러 제안사항 목록을 표시할 수 있습니다.

- 맨 위에서 두 번째 섹션은 명령문 및 해당 열과 관련된 테이블의 그래픽 보기를 표시합니다. 열 옆의 선택란을 사용해 SELECT 문에서 열을 선택하거나 지울 수 있으며 편집기 영역을 마우스 오른쪽 단추로 클릭한 다음 테이블 추가를 클릭하여 명령문을 새 테이블에 추가할 수 있습니다. 테이블을 마우스 오른쪽 단추로 클릭하여 테이블의 옵션 목록도 표시할 수 있습니다.



- 맨 아래 섹션은 명령문의 테이블 보기를 표시합니다. 명령문 편집 방법을 안내하는 테이블의 템플릿을 사용할 수 있습니다.

Columns	Conditions	Groups	Group Conditions
Column	Alias	Output	Sort Type
CUSTOMER.CUSTOMER_ID		☒	
CUSTOMER.FIRST_NAME		☒	
CUSTOMER.LAST_NAME		☒	
CUSTOMER.PASSWORD		☒	

예를 들어, 다음과 같이 SQL 빌더의 테이블 영역을 사용하여 편집기의 SELECT 문에 WHERE절을 추가할 수 있습니다.

12. SQL 빌더의 조건 탭으로 이동하십시오.
13. 조건 탭에서 다음과 같이 명령문에 새 WHERE절을 추가하십시오.
 - a. 조건 탭에서 열 옆의 빈 행을 클릭하십시오.
 - b. 열의 빈 셀에서 CUSTOMER.POSTALCODE를 선택하십시오.
 - c. 연산자에서 **LIKE**를 선택하십시오.
 - d. 값에 다음 코드를 입력하십시오.

1%

퍼센트 기호(%)는 SQL의 와일드 카드입니다. 이 WHERE절은 명령문을 숫자 1로 시작하는 우편번호를 가진 고객으로 제한합니다.

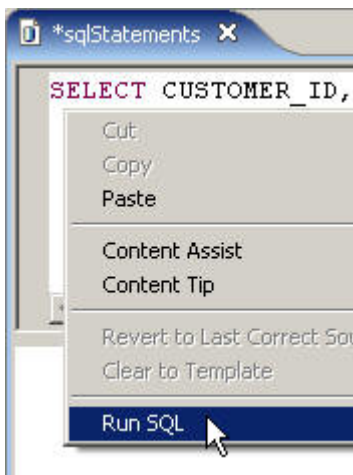
선택 조건은 테이블 영역의 다음과 같습니다.

Columns	Conditions	Groups	Group Conditions
Column	Operator	Value	AND/OR
POSTALCODE	LIKE	'1%'	

조건 추가를 완료하고 다른 위치에 초점을 설정하면 SQL 빌더가 편집기에서 SELECT 문의 코드를 갱신합니다.

```
SELECT CUSTOMER_ID, FIRST_NAME, LAST_NAME,
STREET, APARTMENT, CITY, STATE, POSTALCOI
FROM EGL.CUSTOMER
WHERE POSTALCODE LIKE '1%'
```

14. SQL 빌더의 맨 위에 있는 코드 편집기를 마우스 오른쪽 단추로 클릭한 다음 SQL 실행을 클릭하십시오.



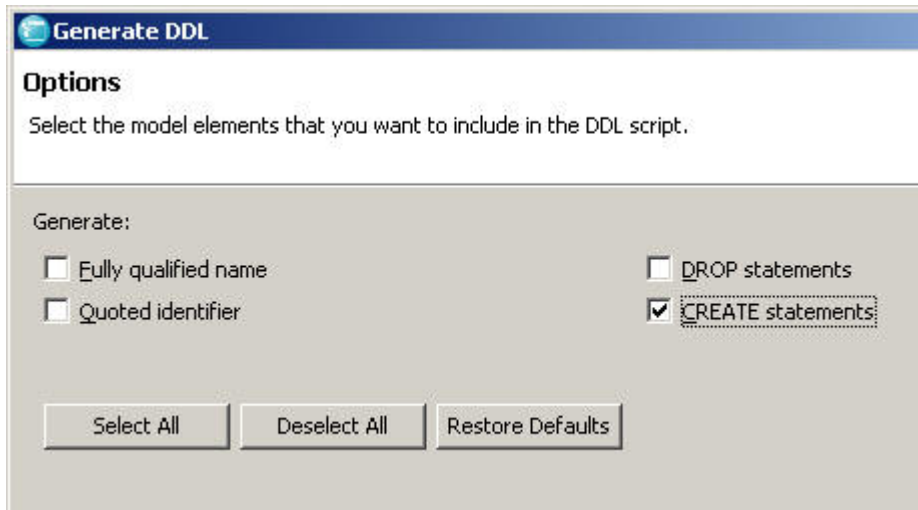
데이터 출력 보기는 명령문의 결과를 표시합니다. 이 경우, 1로 시작하는 우편번호를 가진 고객을 표시합니다.

Messages	Parameters	Results	Profiling Data	
CUSTOMER...	FIRST_NAME	LAST_NAME	PASSWORD	PHONE
4	Francine	Liebowitz	aa	(414)923-2...
8	Sergey	Sharov	ds	(413)452-1...

SQL에 익숙한 경우 여기에 고유 SQL 문을 작성한 다음 응용프로그램에서 사용하기 전에 테스트할 수 있습니다.

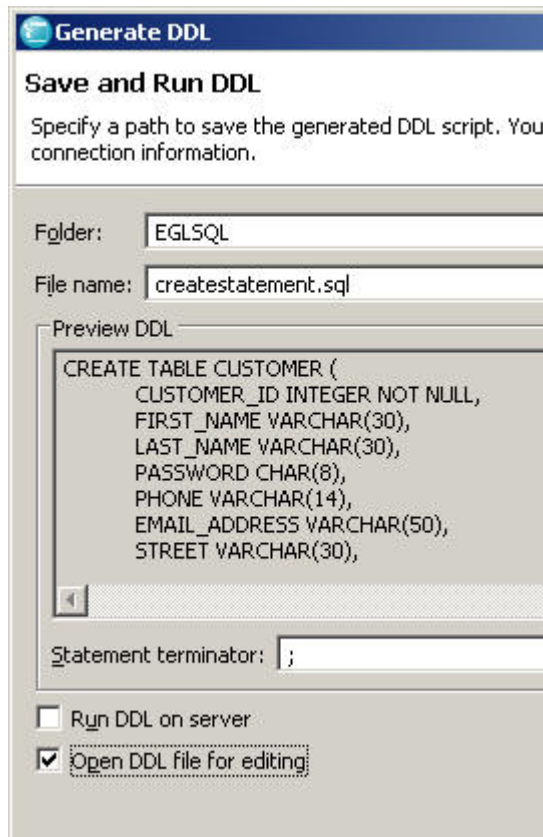
15. SQL 빌더 사용을 완료하면 코드를 파일로 저장하거나 버릴 수 있습니다.
16. 보다 강력한 예제로서, CUSTOMER 테이블의 디자인을 복사하여 새 테이블을 작성할 SQL 문을 작성하십시오.

- a. 데이터베이스 탐색기 보기에서 CUSTOMER 테이블을 마우스 오른쪽 단추로 클릭한 다음 DDL 생성을 클릭하십시오.
- b. DDL 생성 창에서 CREATE 문 선택란을 선택하고 다른 선택란은 지우십시오.



- c. 다음을 클릭하십시오.
- d. 오브젝트 페이지에서 모두 선택을 클릭하십시오.
- e. 다음을 클릭하십시오.
- f. 폴더 필드에서 EGLSQL 프로젝트를 선택하십시오.
- g. DDL 파일 저장 및 실행 페이지에서 파일의 이름을 createtable.sql로 지정하십시오.
- h. 서버에서 DDL 실행 선택란을 지우십시오.

- i. 편집을 위해 **DDL 파일 열기** 선택란을 선택하십시오. DDL 저장 및 실행 페이지는 다음과 같습니다.



Generate DDL

Save and Run DDL

Specify a path to save the generated DDL script. You connection information.

Folder: EGLSQL

File name: createstatement.sql

Preview DDL

```
CREATE TABLE CUSTOMER (  
    CUSTOMER_ID INTEGER NOT NULL,  
    FIRST_NAME VARCHAR(30),  
    LAST_NAME VARCHAR(30),  
    PASSWORD CHAR(8),  
    PHONE VARCHAR(14),  
    EMAIL_ADDRESS VARCHAR(50),  
    STREET VARCHAR(30),  
);
```

Statement terminator: ;

☐ Run DDL on server

☒ Open DDL file for editing

j. 다음을 클릭하십시오.

k. 완료를 클릭하십시오.

SQL 빌더처럼 작동하지만 코드 편집 영역만 있으며 추가 그래픽 영역은 없는 SQL 문 편집기에 새 SQL 문이 열립니다.

17. 명령문을 편집하여 작성된 테이블의 이름을 CUSTOMER가 아닌 EGL.CUSTOMERTEMP로 변경하십시오. 편집한 SQL 코드는 다음과 같습니다.

```
CREATE TABLE EGL.CUSTOMERTEMP (  
    CUSTOMER_ID INTEGER NOT NULL,  
    FIRST_NAME VARCHAR(30),  
    LAST_NAME VARCHAR(30),  
    PASSWORD CHAR(8),  
    PHONE VARCHAR(14),  
    EMAIL_ADDRESS VARCHAR(50),  
    STREET VARCHAR(30),  
    APARTMENT VARCHAR(10),  
    CITY VARCHAR(30),  
    STATE CHAR(2),  
    POSTALCODE VARCHAR(10),  
    DIRECTIONS VARCHAR(255)  
);
```

18. 명령문을 저장하십시오.

19. 데이터베이스 탐색기 보기에서 EGLDerbyR7 데이터베이스를 마우스 오른쪽 단추로 클릭한 다음 연결 끊기를 클릭하여 데이터베이스에서 연결을 끊으십시오.

이제, 데이터베이스의 새 테이블을 작성할 SQL 문이 생겼습니다. 그러나, EGL에는 SQL CREATE 문과 동등한 명령문이 없으며 **get**은 데이터베이스로부터의 결과 세트를 요구하므로 **get**과 같은 명령문의 명시적 SQL에 이 명령문을 삽입할 수 없습니다. 다음 섹션에서는 **execute**를 사용하여 EGL에서 이 SQL 명령문을 실행할 것입니다.

execute 사용

데이터 Perspective에서 도구를 사용하여 새 CREATE 문을 실행할 수 있지만 EGL 응용프로그램에서 사용자 정의 SQL 코드를 실행해야 하는 경우도 있습니다. EGL 응용프로그램에 사용자 정의된 SQL 코드를 삽입할 때마다 예상한대로 작업을 수행하는지 철저히 테스트해야 합니다.

EGL **execute** 문을 사용하여 SQLRecord 변수를 처리하지 않고 SQL 코드를 실행할 수 있습니다. **execute**를 사용하여 다양한 SQL 문(CREATE, ALTER, and DROP TABLE, INSERT, DELETE 및 UPDATE)을 실행할 수 있습니다. 예를 들어, 다음과 같이 데이터베이스에서 테이블을 제거할 수 있습니다.

```
execute #sql{
    DROP TABLE OLD_TABLE;
}
```

execute 문은 일부 SQL 문(특히, SELECT 및 OPEN)을 지원하지 않습니다. **execute**와 작동하거나 작동하지 않는 SQL 문 목록은 SQL의 **execute** 고려사항을 참조하십시오.

다음과 같이 **execute**를 사용하여 스토어드 프로시저를 호출할 수도 있습니다.

```
execute #sql{
    call aStoredProcudure( :parameterVar)
};
```

Derby는 대부분의 데이터베이스 관리 시스템과 같은 방식으로 스토어드 프로시저를 지원하지 않으므로 이 학습서는 스토어드 프로시저를 다루지 않습니다. 대신, 이 절에서는 **execute**를 사용하여 데이터베이스에 테이블을 작성한 다음 SQLRecord 파트를 사용하지 않고 이 테이블을 데이터로 채웁니다.

1. EGL Perspective로 되돌아가십시오.
2. libraries 패키지에서 CustomerTableOperations.egl이라는 파일의 새 라이브러리 파트를 작성하십시오.
3. execCreateTable, execInsertIntoTable 및 execDropTable이라는 새 함수를 라이브러리에 삽입하십시오. 각 함수는 매개변수로 StatusRec 레코드 변수를 받습니다.

```
function execCreateTable(status StatusRec inOut)
end

function execInsertIntoTable(status StatusRec inOut)
end
```

```
function execDropTable(status StatusRec inOut)

end
```

StatusRec 레코드 변수는 데이터베이스 오퍼레이션의 성공 또는 실패에 대한 정보를 기록합니다.

4. 콘텐츠를 지원을 사용하여 매개변수를 추가하지 않은 경우 다음 **import** 문을 파일의 맨 위 **package** 문 바로 아래 추가하십시오.

```
import egllderbyr7.StatusRec;
```

5. execCreateTable 함수에서 **execute** 문을 사용하여 이전 절에서 작성한 CREATE TABLE 문을 실행하십시오.

```
function execCreateTable(status StatusRec inOut)
    try
        execute
            #sql{
                CREATE TABLE EGL.CUSTOMERTEMP (
                    CUSTOMER_ID INTEGER NOT NULL,
                    FIRST_NAME VARCHAR(30),
                    LAST_NAME VARCHAR(30),
                    PASSWORD CHAR(8),
                    PHONE VARCHAR(14),
                    EMAIL_ADDRESS VARCHAR(50),
                    STREET VARCHAR(30),
                    APARTMENT VARCHAR(10),
                    CITY VARCHAR(30),
                    STATE CHAR(2),
                    POSTALCODE VARCHAR(10),
                    DIRECTIONS VARCHAR(255)
                )
            };
        onException (exception SQLException)
            ConditionHandlingLib.HandleException(status, exception);
    end
end
```

이 함수가 테이블을 작성한 다음 데이터베이스에서 데이터 파트 및 논리 파트와 함께 작성된 HandleSuccess 함수를 사용하여 모든 오류를 처리합니다.

6. 콘텐츠를 지원을 사용하여 이 코드를 추가하지 않은 경우 다음 **import** 문을 파일의 맨 위에 추가하십시오.

```
import egllderbyr7.ConditionHandlingLib;
```

7. execDropTable 함수에서, 데이터베이스에서 테이블을 제거하는 **execute** 문을 추가하십시오.

```
function execDropTable(status StatusRec inOut)
    try
        execute #sql{
            DROP TABLE EGL.CUSTOMERTEMP
        } ;
        onException (exception SQLException)
            ConditionHandlingLib.HandleException(status, exception);
    end
end
```

8. `execInsertIntoTable` 함수에서, `CUSTOMER` 테이블에서 `CUSTOMERTEMP` 테이블로 레코드를 이동하는 **execute** 문을 추가하십시오.

```
function execInsertIntoTable(status StatusRec inOut)
try
    execute #sql{
        INSERT INTO EGL.CUSTOMERTEMP
        SELECT * FROM EGL.CUSTOMER
    };
onException (exception SQLException)
    ConditionHandlingLib.HandleException(status, exception);
end
end
```

이 코드는 `CUSTOMER`에서 `CUSTOMERTEMP`로 각 레코드를 복사하기만 합니다. 원하는 경우 복사하려는 특정 레코드만 선택하도록 `WHERE`절을 추가할 수 있습니다.

9. 라이브러리를 저장하고 생성하십시오.
10. `programs` 패키지에서 `duplicateTable.egl`이라는 파일에 새 프로그램 파트를 작성하십시오.
11. 새 프로그램에서 새 테이블을 작성한 다음 레코드를 삽입하는 함수를 호출하십시오.

```
package programs;

import eglderbyr7.StatusRec;
import libraries.CustomerTableOperations;

program duplicateTable type BasicProgram {}

    status StatusRec;

    function main()
        CustomerTableOperations.execCreateTable(status);
        CustomerTableOperations.execInsertIntoTable(status);
    end

end
```

12. 프로그램을 생성하고 실행하십시오.
13. 데이터 Perspective로 돌아온 다음 데이터베이스 탐색기 보기에서 데이터베이스에 다시 연결하십시오.
14. 이전 절에서와 같이 새 테이블의 데이터 미리보기를 수행하여 새 테이블과 해당 데이터를 확인하십시오.
- a. **EGLDerbyR7** → **EGLDerbyR7** → 스키마 → **EGL** → 테이블을 펼치십시오.
 - b. `CUSTOMERTEMP` 테이블을 마우스 오른쪽 단추로 클릭한 다음 데이터 → 샘플 콘텐츠를 클릭하십시오.

이제, 새 테이블의 데이터를 확인할 수 있습니다.

Messages	Parameters	Results	Profiling Data	
CUSTOMER...	FIRST_NAME	LAST_NAME	PASSWORD	PHONE
1	Fred	Filibuster	dsds	(201)652-3...
2	Andy	Lundquist	bbb	(221)545-5...
3	Billie	Kingman	ccc	(261)898-4...
4	Francine	Liebowitz	aa	(414)923-2...
5	Ornette	Coleman	eee	(518)132-4...
6	Ellen	Springsteen	23	(712)469-5...
7	Martin	St. Louis	StLouis	(670)250-5...
8	Sergey	Sharov	ds	(413)452-1...
9	Melodie	Hudak	afda	(860)742-0...

이제, 새 테이블과 작동할 기타 명령문을 작성하거나 `execDropTable` 함수를 호출하여 데이터베이스에서 제거할 수 있습니다. 예를 들어, `CREATE TABLE` 문과 함께 작성된 `ALTER TABLE` 및 `CREATE INDEX` 문을 실행할 수도 있습니다. 하지만, 이 경우 Derby가 1차 키 제한조건의 중복된 이름을 허용하지 않으므로 제한조건 및 색인의 이름을 바꾸어야 합니다. 데이터베이스에 액세스하는 EGL 코드를 실행하기 전에 데이터베이스 탐색기 보기에서 데이터베이스에 대한 연결을 끊도록 하십시오.

이 학습에서 사용하는 두 파일의 전체 코드를 살펴보십시오. 파일에 빨간색 X 기호로 표시된 오류가 나타나면 사용자의 코드가 69 페이지의 『학습 6에서 완성된 CustomerTableOperations.egl 및 duplicateTable.egl 파일』의 코드와 일치하는지 확인하십시오.

학습 체크포인트

이 학습에서는 Workbench의 일부 데이터베이스 관리 도구 및 해당 도구를 **execute**문과 결합하여 EGL 응용 프로그램에서 매우 다양한 SQL 문과 실행하는 방법에 대해 배웠습니다.

Workbench의 데이터 액세스 도구에 대한 자세한 정보는 SQL 문에 대한 작업을 참조하십시오. EGL **execute** 문에 대한 자세한 정보는 `execute`를 참조하십시오.

학습 7: 예외 처리 및 롤백

올바르지 않은 데이터 검색, 올바르지 않은 데이터 삽입 또는 응용프로그램과 데이터 소스 사이 불일치를 예방하기 위해 데이터 소스에 액세스하는 응용프로그램에서 오류를 적절히 처리하는 것은 매우 중요합니다. 이 학습에서는 관계형 데이터베이스에 액세스할 때 오류에 응답하는 데 사용할 수 있는 몇 가지 도구에 대해 설명합니다.

SQL 데이터 액세스의 예외 처리

이 학습서의 예제에서는 종종 **try** 블록에 데이터 액세스 명령문을 둡니다.

```
function execDropTable(status StatusRec inOut)
try
    execute #sql{
        DROP TABLE EGL.CUSTOMERTEMP
    } ;
```

```

onException (exception SQLException)
    ConditionHandlingLib.HandleException(status, exception);
end
end

```

try 블록에 코드를 넣으면 EGL이 해당 코드를 일반적으로 실행하려 합니다. EGL에 오류가 발생하면 **try** 블록에서 코드 실행을 중지하고 **try** 블록에 남아 있는 모든 명령문을 건너뛴 다음 **onException** 블록으로 직접 이동합니다. **onException** 블록에 코드를 넣어 오류에서 복구하거나 문제점을 정정하거나 로그 파일에 오류 정보를 쓸 수 있습니다.

try 블록에서 EGL에 오류가 발생하면 예외 레코드도 작성하여 오류에 대한 정보를 제공합니다. 해당 레코드를 **onException** 블록에 전달합니다. 오류의 원인을 판별하고 오류에서 복구하는 데 도움이 되도록 **onException** 블록에서 해당 레코드의 정보를 사용할 수 있습니다. 위의 예제에서 **onException** 블록은 오류를 처리하는 함수에 예외 레코드를 전달합니다.

오류 유형에 따라 예외 레코드에는 여러 스테레오타입 중 하나가 있을 수 있습니다. 위의 예제에서 **onException** 블록은 SQL 오류 및 기타 관계형 데이터베이스 오류용인 **SQLException** 스테레오타입의 예외 레코드를 요구합니다. EGL에 널값 참조 등과 같은 다른 유형의 오류가 발생하면 다른 유형의 예외 레코드가 작성됩니다(널값 참조의 경우, **NullPointerException** 스테레오타입이 있는 예외 레코드).

다음 예제에서와 같이 다른 유형의 오류를 각각 다르게 처리하도록 단일 **try** 블록 다음에 여러 **onException** 블록을 둘 수 있습니다.

```

function execDropTable(status StatusRec inOut)
    try
        execute #sql{
            DROP TABLE EGL.CUSTOMERTEMP
        };
        onException (exception SQLException)
            ConditionHandlingLib.HandleException(status, exception);
        onException (exception AnyException)
            HandleOtherError(exception);
    end
end

```

이 예제에서 첫 번째 **onException** 블록은 **SQLException** 레코드를 요구하며 두 번째 블록은 **AnyException** 스테레오타입을 요구합니다. 이 경우, EGL에 SQL 오류가 발생하면 첫 번째 **onException** 블록이 실행됩니다. EGL에 다른 유형의 오류가 발생하면 두 번째 블록이 실행됩니다. **AnyException** 스테레오타입은 스테레오타입에 관계없이 모든 오류에 응답하는 데 사용하는 일반 사용자 예외 레코드 스테레오타입입니다.

모든 예외 레코드에는 최소 두 개의 필드(오류의 오류 코드가 있는 **messageID** 필드 및 문제점의 간략한 설명이 있는 **message** 필드)가 있습니다. 스테레오타입에 따라 예외 레코드에는 다른 필드가 있을 수 있습니다. 예를 들어, **SQLRecord** 스테레오타입에는 명령문의 상태가 있는 **CHAR(5)** 값을 보유하는 **sqlState** 필드와 DBMS에서의 **INT** 리턴 코드를 보유하는 **sqlCode** 필드가 있습니다.

올바르지 않은 정보를 전달하고, 존재하지 않는 레코드를 삭제하거나 갱신 또는 동일한 1차 키가 있는 레코드를 추가하려고 시도하여 예외 처리를 시도할 수 있습니다. 다음 예제에서는 데이터베이스의 기존 행과 같은 ID 번호가 있는 새 고객 레코드를 추가하려 합니다. 따라서, 1차 키와 충돌을 일으키며 EGL이 SQLException 예외를 처리하도록 프롬프트됩니다.

1. programs 패키지에 이름이 exceptionHandlingTest인 새 프로그램을 작성하십시오.
2. 다음과 같이 새 프로그램의 main 함수에서 고객 레코드 변수를 작성하여 정보(데이터베이스에 이미 있는 ID 번호 포함)를 지정하십시오.

```
package programs;

import egllderbyr7.data.Customer;

program exceptionHandlingTest type BasicProgram {}

    function main()
        customer Customer;
        customer.FirstName = "John";
        customer.LastName = "Doe";
        customer.CustomerId = 1;
    end

end
```

3. **try** 블록에서 다음과 같이 데이터베이스에 레코드를 추가하십시오.

```
try
    add customer;
    SysLib.writeStdout("Customer added successfully.");
end
```

오류에 응답하는 일치하는 **onException** 블록이 없으므로 이 **try** 블록은 그리 유용하지 않습니다.

4. **try** 블록을 닫는 **end** 앞에 **onException** 블록을 추가하여 EGL이 **try** 블록의 코드를 실행하려 할 때 발생하는 SQLException을 처리하십시오.

```
try
    add customer;
    SysLib.writeStdout("Customer added successfully.");
onException(ex SQLException)
    SysLib.writeStdout("SQL exception " :: ex.messageID);
    SysLib.writeStdout("message = " :: ex.message);
    SysLib.writeStdout("SQLCode = " :: ex.sqlCode);
    SysLib.writeStdout("SQLState = " :: ex.sqlState);
end
```

전체 프로그램은 다음과 같습니다.

```
package programs;

import egllderbyr7.data.Customer;

program exceptionHandlingTest type BasicProgram {}

    function main()
        customer Customer;
        customer.FirstName = "John";
```

```

customer.LastName = "Doe";
customer.CustomerId = 1;

try
    add customer;
    SysLib.writeStdout("Customer and order added successfully.");
onException(ex SQLException)
    SysLib.writeStdout("SQL exception " "::ex.messageID);
    SysLib.writeStdout("message = " "::ex.message);
    SysLib.writeStdout("SQLCode = " "::ex.sqlCode);
    SysLib.writeStdout("SQLState = " "::ex.sqlState);
end

```

end

end

5. 프로그램을 저장하고 생성하여 실행하십시오. 콘솔이 **onException** 블록에서 출력을 표시합니다.

```

SQL exception EGL0504E
message = EGL0504E ADD: The statement was aborted
because it would have caused a duplicate key value
in a unique or primary key constraint or unique
index identified by 'SQL070307095259210' defined
on 'CUSTOMER'.[sqlstate:23505][sqlcode:20000]
EGL0002I The error occurred in the
exceptionHandlingTest program processing the main function.
SQLCode = 20000
SQLState = 23505

```

데이터베이스에 변경사항 롤백

Derby를 포함하여, 대부분의 관계형 데이터베이스는 복구 가능 자원입니다. 복구 가능한 자원을 변경할 때 변경사항을 저장하는 **확약**을 실행해야 변경사항이 영구적으로 됩니다. 이 방식으로, 변경사항을 **확약**하기 전에 예외가 발생하는 경우 데이터베이스의 변경사항을 되돌릴 수 있습니다.

sqlCommitControl 빌드 설명자 옵션을 NOAUTOCOMMIT로 설정하여 EGL이 자동으로 각 변경사항을 **확약**하지 않도록 할 수 있습니다. 자동 **확약**을 꺼져 있는 상태라면 수동으로 **sysLib.commit()**을 호출하여 변경사항을 **확약**하고 **sysLib.rollback()**을 호출하여 마지막 **확약** 이후로 변경된 사항을 되돌릴 수 있습니다. **sysLib.commit()**으로 변경사항을 **확약**하지 않으면 EGL이 항상 실행 단계의 끝(예: main 프로그램의 끝)에서 변경사항을 **확약**합니다.

예를 들어, 데이터베이스의 여러 사항을 변경한다고 가정하십시오(예: 새 고객 추가 및 해당 고객의 첫 번째 주문에 대한 주문 정보 입력). 일부 해당 오퍼레이션이 다른 오퍼레이션이 실패하기 전에 성공하면 주문이 없는 고객이 남거나 주문을 한 고객이 없는 주문이 남을 수 있습니다. 이런 경우, 다음과 같이 **onException** 블록에 롤백을 넣어 변경사항을 되돌리고 데이터베이스 외부에 불완전한 정보를 보존하십시오.

1. **sqlCommitControl** 빌드 설명자 옵션을 NOAUTOCOMMIT로 설정하십시오.

- a. 프로젝트 탐색기 보기에서 EGLSQL 프로젝트의 빌드 설명자를 두 번 클릭하여 EGL 빌드 파트 편집기로 여십시오. 이 파일은 EGLSQL/EGLSource/EGLSQL.eglbld에 있습니다.
- b. 빌드 파트 편집기에서 지정된 옵션만 표시 선택란을 지우십시오.

- c. 옵션에서 **sqlCommitControl**을 찾으십시오.
- d. **sqlCommitControl**의 값을 NOAUTOCOMMIT로 설정하십시오.

주: 편집을 위해 **sqlCommitControl** 옵션을 열려면 해당 옵션 옆의 값 열을 천천히 두 번 클릭하십시오. 값 열을 빠르게 세 번 클릭할 수도 있습니다.

- e. 빌드 설명자를 저장하고 닫으십시오.
2. 이전 절에서 작성한 **exceptionHandlingTest** 프로그램을 여십시오.
 3. 이미 프로그램에 있는 고객 레코드에 주문 레코드를 추가하십시오.

```
program exceptionHandlingTest type BasicProgram {}
```

```
function main()
  customer Customer;
  customer.FirstName = "John";
  customer.LastName = "Doe";
  customer.CustomerId = 1;

  customerOrder Orders;
  customerOrder.CustomerId = 1;
  customerOrder.OrderId = 50;
  customerOrder.OrderAmount = 0;
  customerOrder.OrderDetails =
    "This is my first order, so get it right!";

  ...

end
```

4. **try** 블록에서 먼저 주문을 추가한 다음 고객을 추가하십시오.

```
try
  add customerOrder;
  SysLib.writeStdout("Order added successfully.");
  add customer;
  SysLib.writeStdout("Customer added successfully.");
end
```

이 경우, 중복된 1차 키로 인해 주문은 추가되지만 고객은 추가되지 않습니다. 따라서, **onException** 블록에서 롤백을 호출하여 **ORDERS** 테이블의 변경사항을 되돌릴 수 있습니다.

5. 다음과 같이 **try** 블록 다음에 롤백을 포함하여 **onException** 블록을 추가하십시오.

```
try
  add customerOrder;
  SysLib.writeStdout("Order added successfully.");
  add customer;
  SysLib.writeStdout("Customer added successfully.");
onException(ex SQLException)
  SysLib.writeStdout("SQL exception " & ex.messageID);
  SysLib.writeStdout("message = " & ex.message);
  SysLib.writeStdout("Performing rollback.");
  SysLib.rollback();
end
```

이제, **add** 문 중 하나가 실패하면 **onException** 블록이 실행되어 데이터베이스의 변경사항을 되돌리려고 시도합니다.

- 예외가 발생하기 전에 추가된 주문을 롤백하여 성공적으로 제거했는지 여부를 테스트하는 코드를 **onException** 블록 다음에 추가하십시오. 주문 레코드가 여전히 데이터베이스에 있는지 확인하려면 다음과 같이 **noRecordFound**와 비교해 보십시오.

```
tempOrder Orders;  
tempOrder.OrderId = 50;  
get tempOrder;  
  
if (tempOrder is noRecordFound)  
    SysLib.writeStdout("The order was rolled back successfully.");  
else  
    SysLib.writeStdout("The order is still in the database");  
end
```

데이터베이스 호출 상태를 확인하는 또 다른 방법으로서 **sqlLib.sqlData** 시스템 변수의 값을 테스트합니다. 이 시스템 변수는 레코드이며 해당 필드는 마지막 SQL 데이터베이스 오퍼레이션의 성공 또는 실패에 대한 정보를 저장합니다. 특히, **sqlLib.sqlData.sqlcode** 필드는 오퍼레이션이 성공적으로 완료되어 결과를 리턴하면 0을 포함하며 오퍼레이션이 성공적으로 완료되었지만 **SELECT** 문이 결과를 리턴하지 않으면 100을 포함합니다. 따라서, 다음 코드를 사용하여 행이 발견되었는지 여부를 확인할 수도 있습니다.

```
if (sqlLib.sqlData.sqlcode == 0)  
    SysLib.writeStdout("The order is still in the database");  
else  
    if (sqlLib.sqlData.sqlcode == 100)  
        SysLib.writeStdout("The order was rolled back successfully.");  
    end  
end
```

- 프로그램을 생성하고 실행하십시오.

다음 프로그램 출력은 주문이 데이터베이스에 추가되고 예외가 발생한 다음 롤백하여 주문이 성공적으로 제거되었음을 나타냅니다.

```
Order added successfully.  
SQL exception EGL0504E  
message = EGL0504E ADD: The statement was aborted  
because it would have caused a duplicate  
key value in a unique or primary key constraint  
or unique index identified by 'SQL070307095259210'  
defined on 'CUSTOMER'.[sqlstate:23505][sqlcode:20000]  
EGL0002I The error occurred in the exceptionHandlingTest  
program processing the main function.  
Performing rollback.  
The order was rolled back successfully.
```

exceptionHandlingTest.egl 파일의 코드가 완성되었습니다. 파일에 빨간색 X 기호로 표시된 오류가 나타나면 코드가 71 페이지의 『학습 7에서 완성된 **exceptionHandlingTest.egl** 파일』의 코드와 일치하는지 확인하십시오.

학습 체크포인트

이 학습에서는 SQL 오퍼레이션으로 오류를 처리하는 기초 작업에 대해 배웠습니다. 문제점을 예상하고 응용 프로그램이 복구할 수 있도록 예외 처리를 제공하는 것은 매우 중요합니다.

예외를 초래할 수 있는 런타임 오류에 대한 자세한 정보는 EGL Java 런타임 오류 코드를 확인하십시오. EGL 이 작성할 수 있는 예외 레코드 목록은 EGL 코어 예외 레코드를 확인하십시오.

요약

학습한 내용

이 학습서를 완료하여 다음 개념 및 태스크에 대해 배웠습니다.

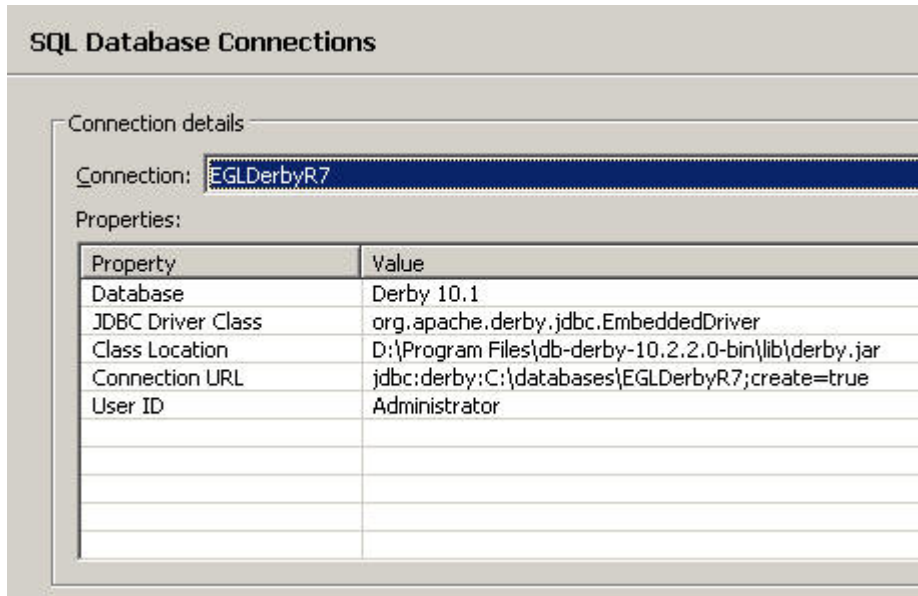
- **get** 및 **add**와 같은 명령문을 사용하여 EGL의 데이터베이스에 대한 기초 작업 수행
- 고유 SQL 문 작성 방법(예: 명시적 SQL, SQL 빌더, SQLRecord 파트의 기본 선택 조건, prepared 문 및 **execute** 문 등을 사용).
- 테이블 결합을 수행하여 복잡한 데이터에 대한 작업을 처리하는 방법
- 명시적 SQL 및 prepared 문의 호스트 변수
- 데이터 액세스 응용프로그램에서 오류를 처리하는 방법

문제점 해결

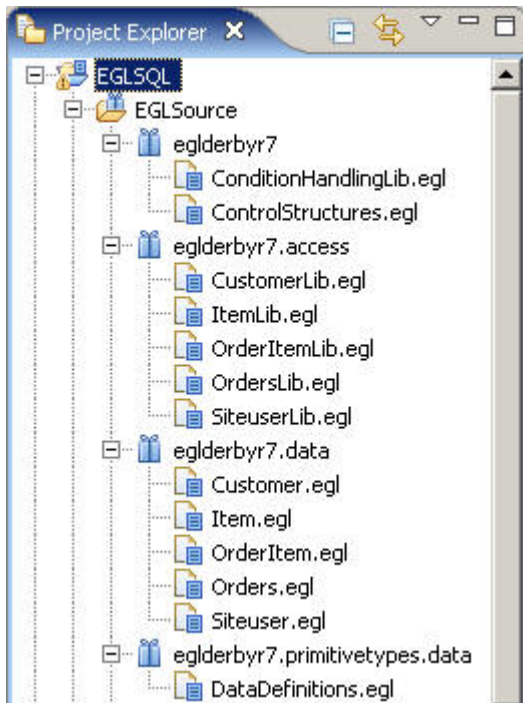
이 절에서는 학습서 응용프로그램과 작업할 때 발생할 수 있는 문제점을 나열합니다. 다음은 예상한 결과를 얻지 못한 경우 문제점을 수정하기 위해 시도할 수 있는 사항입니다.

- 코드 편집기에 빨간색 X 아이콘 또는 EGL 생성 결과 보기에서 오류로 표시된 유효성 검증 또는 생성 오류가 EGL 코드에 없는지 확인하십시오. EGL 생성 결과 보기에 초록색 체크 표시 아이콘이 아닌 빨간색 X 아이콘이 있는 소스 파일이 나열된 경우 이 파일은 제대로 생성되지 않은 것입니다. 이 학습서에서 사용하는 대부분의 EGL 코드가 참조서의 맨 끝에 포함되어 있습니다. 59 페이지의 『자원』을 참조하십시오.
- 모든 파일이 저장되었는지 확인한 다음 프로젝트를 마우스 오른쪽 단추로 클릭한 후 생성을 클릭하여 전체 프로젝트를 생성하십시오.
- 다음을 포함하여 2 페이지의 『학습 1: 데이터베이스 연결 설정』의 모든 단계를 제대로 수행했는지 확인하십시오.
 - 프로젝트의 빌드 설명자를 열고 연결을 사용하여 DB 옵션 로드 목록에서 데이터베이스 연결을 선택하여 모든 빌드 설명자 옵션을 설정하십시오. 빌드 설명자를 변경한 다음 프로젝트를 저장하고 생성하십시오.
 - Derby JAR 파일을 프로젝트의 Java 빌드 경로에 추가하십시오.
- 다음을 수행하여 데이터베이스 연결이 올바른지 확인하십시오.
 1. 창 → 환경 설정을 클릭하십시오.
 2. 환경 설정 창에서 EGL을 펼치고 SQL 데이터베이스 연결을 클릭하십시오.

3. 연결 목록에서 연결을 선택하십시오. 샘플 데이터베이스로의 연결을 작성하지 않은 경우(예: *EGL 소개 (빠른 시작 안내서)*에서), 일반적으로 이 연결의 이름은 EGLDerbyR7입니다. 이미 작성한 경우, 연결의 이름은 EGLDerbyR71입니다.
4. 해당 연결의 특성이 컴퓨터에 있는 관련 파일의 위치와 일치하는지 확인하십시오. derby.jar 파일과 EGLDerybyR7 폴더(데이터베이스 포함)의 사용자 고유 위치를 포함하는 연결은 다음과 같아야 합니다.



5. 특성이 파일의 실제 위치와 일치하지 않으면 편집을 클릭하고 설정을 정정한 다음 연결 테스트를 클릭하여 해당 연결이 작동하는지 확인하십시오.
- 다음과 같이 데이터 파트와 논리 파트가 올바르게 작성되었는지 확인하십시오.
 - 다음 그림에서와 같이 eglderbyr7 패키지에 모든 파일이 있는지 확인하십시오.



- 예를 들어, eglderbyr7.data 패키지의 Customer.egl 파일을 열고 고객 레코드 파트와 다음 레코드를 비교하십시오.

```
record Customer type sqlRecord {
    tablename=[["EGL.CUSTOMER"]],
    keyItems=[CustomerId]
}

CustomerId CustomerId {column="EGL.CUSTOMER.CUSTOMER_ID"};
FirstName FirstName {column="EGL.CUSTOMER.FIRST_NAME",
    sqlVariableLen=yes, maxLen=30, isSqlNullable=yes};
LastName LastName {column="EGL.CUSTOMER.LAST_NAME",
    sqlVariableLen=yes, maxLen=30, isSqlNullable=yes};
Password Password {column="EGL.CUSTOMER.PASSWORD",
    maxLen=8, isSqlNullable=yes};
Phone Phone {column="EGL.CUSTOMER.PHONE",
    sqlVariableLen=yes, maxLen=14, isSqlNullable=yes};
EmailAddress EmailAddress {column="EGL.CUSTOMER.EMAIL_ADDRESS",
    sqlVariableLen=yes, maxLen=50, isSqlNullable=yes};
Street Street {column="EGL.CUSTOMER.STREET",
    sqlVariableLen=yes, maxLen=30, isSqlNullable=yes};
Apartment Apartment {column="EGL.CUSTOMER.APARTMENT",
    sqlVariableLen=yes, maxLen=10, isSqlNullable=yes};
City City {column="EGL.CUSTOMER.CITY", sqlVariableLen=yes,
    maxLen=30, isSqlNullable=yes};
State State {column="EGL.CUSTOMER.₩STATE₩", maxLen=2,
    isSqlNullable=yes};
Postalcode Postalcode {column="EGL.CUSTOMER.POSTALCODE",
    sqlVariableLen=yes, maxLen=10, isSqlNullable=yes};
Directions Directions {column="EGL.CUSTOMER.DIRECTIONS",
    sqlVariableLen=yes, maxLen=255, isSqlNullable=yes};
end
```

사용자의 파트가 올바르지 않으면 EGL 데이터 액세스 응용프로그램 마법사를 다시 한 번 단계별로 주의하여 수행한 후 해당 파트를 다시 작성한 다음 올바르지 않은 파트를 겹쳐쓰십시오. 예를 들어, 레코드 파트의 **tableNames** 특성과 해당 레코드 파트의 필드에 있는 열 특성에 각 테이블 이름의 EGL 접두부가 없다면, 이는 곧 스키마로 테이블 이름 규정 선택란을 선택하지 않은 것입니다.

런타임 오류 메시지

다음은 이 학습서에서 프로그램을 실행하는 동안 콘솔 보기에 표시될 수 있는 일부 오류 메시지 목록입니다.

EGL0505E Cannot connect to jdbc:derby:C:\databases\EGLDerbyR7;create=true: Failed to start database 'C:\databases\EGLDerbyR7', see the next exception for details.

데이터베이스를 기반으로 파트를 작성하는 데이터 도구가 데이터베이스에 연결되어 있어 EGL 프로그램이 데이터베이스에 연결할 수 없습니다. 데이터베이스 탐색기 보기에서 데이터 도구 연결을 닫은 다음 8 페이지의 『테스트 프로그램 실행』에 설명된 대로 EGL 프로그램을 다시 실행해 보십시오.

EGL0507E An error occurred while loading the JDBC drivers. Error: egl.core.RuntimeException: EGL0009E A value for the vgj.jdbc.drivers property is required.

빌드 설명자가 올바르게 설정되지 않았습니다. 빌드 설명자를 열고 연결을 사용하여 DB 옵션 로드 목록에서 연결을 선택하여 빌드 설명자 옵션의 값을 로드하십시오.

EGL0507E An error occurred while loading the JDBC drivers. Error: java.lang.ClassNotFoundException: org.apache.derby.jdbc.EmbeddedDriver

EGL이 derby.jar 파일을 찾을 수 없습니다. 4 페이지의 『데이터베이스 가져오기 및 연결』에 설명된 대로 프로젝트의 Java 빌드 경로에 파일을 추가했는지 확인하십시오.

자원

이 학습서는 다음 자원을 사용합니다.

- 다음 위치에 있는 샘플 Derby 데이터베이스

shared_resources/plugins/com.ibm.etools.egl.tutorial0001.doc_version/resources/EGLDerbyR7.zip
shared_resources

프로젝트의 공유 자원 디렉토리(예: Windows 시스템의 경우 C:\Program Files\IBM\SDP70Shared 또는 Linux 시스템의 경우 /opt/IBM/SDP70Shared)입니다. 현재 제품을 설치하기 전에 EGL이 포함된 이전 버전의 IBM 제품이 설치되어 있는 경우 이전 설치에서 설정한 공유 자원 디렉토리를 지정해야 합니다.

version

점으로 구분된 세 개의 숫자(문자열 분리 문자, 플러그인이 빌드된 날짜와 시간)를 포함하는 플러그인의 설치 버전(예: 7.0.0.RFB_20070120_1300)입니다. 둘 이상이 있는 경우 최신 버전을 사용하십시오.

- 60 페이지의 『학습 2에서 완성된 CRUDTest.egl 파일』
- 62 페이지의 『학습 3에서 완성된 selectItems.egl 파일』
- 63 페이지의 『학습 4A에서 완성된 selectItems.egl 파일』

- 65 페이지의 『학습 4B에서 완성된 selectItems.egl 파일』
- 68 페이지의 『학습 5에서 완성된 printCustomerOrders.egl 및 records.egl 파일』
- 69 페이지의 『학습 6에서 완성된 CustomerTableOperations.egl 및 duplicateTable.egl 파일』
- 71 페이지의 『학습 7에서 완성된 exceptionHandlingTest.egl 파일』

다음은 이 학습서에서 다루는 주제에 대한 도움말 시스템 링크입니다.

- #sql 지시문
- SQL 데이터에 대한 작업
- SQLRecord 스테레오타입
- SQL의 add 고려사항
- SQL의 delete 고려사항
- SQL의 execute 고려사항
- SQL의 forEach 고려사항
- SQL의 get 고려사항
- SQL의 open 고려사항
- SQL의 prepare 고려사항
- SQL의 replace 고려사항
- EGL Java 런타임 오류 코드
- EGL 코어 예외 레코드
- EGL 라이브러리 sqlLib
- commit()
- rollback()
- SQL 문에 대한 작업

학습 2에서 완성된 CRUDTest.egl 파일

이 코드는 학습 2에서 완성된 CRUDTest.egl 파일입니다. 파일에 빨간색 X 기호로 표시된 오류가 나타나면 사용자의 코드가 다음 코드와 일치하는지 확인하십시오.

```
package programs;

import eglderby7.data.*;

program CRUDTest type BasicProgram {}

function main()

    // Print the starting records in the database.
    SysLib.writeStdout("Contents of database before any SQL operations:");
    printAllCustomers();

    // Add a record.
    addCustomer();
```

```

// Update the record.
updateCustomer();

// Delete a record.
deleteCustomer();

end

function addCustomer()
    customerToAdd Customer;
    customerToAdd.CustomerId = 50;
    customerToAdd.FirstName = "John";
    customerToAdd.LastName = "Doe";

    // Insert the record into the database
    try
        add customerToAdd;
        SysLib.writeStdout("Contents of database after adding a new record:");
        printAllCustomers();
    onException(exception SQLException)
        handleDBException(exception);
    end

end

function deleteCustomer()
    customerToDelete Customer;
    customerToDelete.CustomerId = 50;

    // Delete the record.
    try
        delete customerToDelete noCursor;
        SysLib.writeStdout("Contents of database after deleting the record:");
        printAllCustomers();
    onException(exception SQLException)
        handleDBException(exception);
    end

end

function updateCustomer()
    customerToUpdate Customer;
    customerToUpdate.CustomerId = 50;

    // Retrieve the row.
    get customerToUpdate;

    // Change values in the record variable.
    customerToUpdate.FirstName = "Johnny";
    customerToUpdate.LastName = "Five";

    // Update the row in the database.
    try
        replace customerToUpdate noCursor;
        SysLib.writeStdout("Contents of database after updating the record:");
        printAllCustomers();
    onException(exception SQLException)

```

```

        handleDBException(exception);
    end

end

// This function prints out the contents of the CUSTOMER
// table in the database.
function printAllCustomers()

    allCustomers Customer[0];

    get allCustomers;
    for (counter int from 1 to allcustomers.getSize())
        SysLib.writeStdout(allCustomers[counter].CustomerId:" "":
            allCustomers[counter].FirstName:" "":
            allCustomers[counter].LastName);
    end
    SysLib.writeStdout("\n");

end

//This function responds to errors accessing the database.
function handleDBException(exception SQLException)
    SysLib.writeStdout("Error accessing database. "":
        "See Troubleshooting section");
    SysLib.writeStdout(exception.message);
end

end

```

10 페이지의 『학습 2: EGL의 기본 SQL 조작』으로 되돌아가십시오.

학습 3에서 완성된 selectItems.egl 파일

이 코드는 학습 3에서 완성된 selectItems.egl 파일입니다. 파일에 빨간색 X 기호로 표시된 오류가 있으면 사용자의 코드가 다음 코드와 일치하는지 확인하십시오.

```

package programs;

import eglderbyr7.data.Item;
import eglderbyr7.primitivetypes.data.Price;

program selectItems type BasicProgram {}

function main()
    try
        printExpensiveItems(200);
    onException(exception SQLException)
        handleDBException(exception);
    end
end

function printExpensiveItems(maxPrice Price in)

    tempItem Item;
    open expensiveItems for tempItem with
        #sql{

```

```

select
    EGL.ITEM.ITEM_ID, EGL.ITEM.NAME, EGL.ITEM.IMAGE,
    EGL.ITEM.PRICE, EGL.ITEM.DESCRPTION
from EGL.ITEM
where
    EGL.ITEM.PRICE >= :maxPrice
order by
    EGL.ITEM.ITEM_ID asc
};

forEach (tempItem)
    SysLib.writeStdout(tempItem.Name :: " " ::
        tempItem.Description :: " $" :: tempItem.Price);
end

end

function handleDBException(exception SQLException)
    SysLib.writeStdout("Error accessing database. "::
        "See Troubleshooting section");
    SysLib.writeStdout(exception.message);
end

end

```

19 페이지의 『학습 3: **open** 및 **forEach**로 커서 관리』로 되돌아가십시오.

학습 4A에서 완성된 **selectItems.egl** 파일

이 코드는 학습 4A에서 완성된 **selectItems.egl** 파일입니다. 파일에 빨간색 X 기호로 표시된 오류가 나타나면 사용자의 코드가 다음 코드와 일치하는지 확인하십시오.

```

package programs;

import eglDerby7.data.Item;
import eglDerby7.primitiveTypes.data.Price;

program selectItems type BasicProgram {}

function main()
    minPrice, maxPrice Price?;
    minPrice = 50;
    maxPrice = 500;
    try
        printItemRange(minPrice, maxPrice);
    onException(exception SQLException)
        handleDBException(exception);
    end
end

function printItemRange(minPrice Price? in, maxPrice Price? in)

    selectStatement STRING = "select * from EGL.ITEM where ";

    if (minPrice == NULL && maxPrice == NULL)
        // Two empty parameters;
        // return all rows with PRICE greater than zero.

```

```

    selectStatement ::= "EGL.ITEM.PRICE > 0 ";
else
    if (minPrice != NULL && maxPrice != NULL)
        // Both parameters were specified.
        selectStatement ::= "EGL.ITEM.PRICE between ":: minPrice :: " and " :: maxPrice :: " ";
    else
        // Only one parameter was specified.
        if (minPrice == NULL)
            // Maximum was specified.
            selectStatement ::= "EGL.ITEM.PRICE <= " :: maxPrice :: " ";
        else
            // Minimum was specified.
            selectStatement ::= "EGL.ITEM.PRICE >= ":: minPrice :: " ";
        end
    end
end

end

// Regardless of parameters, sort by price.
selectStatement ::= "order by EGL.ITEM.PRICE";

// Print completed SQL code.
SysLib.writeStdout("Completed query: "::selectStatement);

// Prepare the statement to be used.
tempItem Item;
prepare preparedStatement
    from selectStatement
    for tempItem;

// Use the prepared statement in place of explicit SQL.
open expensiveItems for tempItem with preparedStatement;

// Print the results to the console.
forEach (tempItem)
    SysLib.writeStdout(tempItem.Name :: " " ::
        tempItem.Description :: " $" :: tempItem.Price);
end

end

function printExpensiveItems(maxPrice Price in)

tempItem Item;
open expensiveItems for tempItem with
    #sql{
        select
            EGL.ITEM.ITEM_ID, EGL.ITEM.NAME, EGL.ITEM.IMAGE,
            EGL.ITEM.PRICE, EGL.ITEM.DESCRPTION
        from EGL.ITEM
        where
            EGL.ITEM.PRICE >= :maxPrice
        order by
            EGL.ITEM.ITEM_ID asc
    };

forEach (tempItem)

```

```

        SysLib.writeStdout(tempItem.Name :: " " ::
            tempItem.Description :: " $" :: tempItem.Price);
    end

end

function handleDBException(exception SQLException)
    SysLib.writeStdout("Error accessing database. "::
        "See Troubleshooting section");
    SysLib.writeStdout(exception.message);
end

end

```

27 페이지의 『학습 4A: prepared 문 사용』으로 되돌아가십시오.

학습 4B에서 완성된 selectItems.egl 파일

이 코드는 학습 4B에서 완성된 selectItems.egl 파일입니다. 파일에 빨간색 X 기호로 표시된 오류가 나타나면 사용자의 코드가 다음 코드와 일치하는지 확인하십시오.

```

package programs;

import eglderbyr7.data.Item;
import eglderbyr7.primitivetypes.data.Price;

program selectItems type BasicProgram {}

function main()
    minPrice, maxPrice Price?;
    minPrice = 50;
    maxPrice = 500;
    try
        printItemRange2(minPrice, maxPrice);
    onException(exception SQLException)
        handleDBException(exception);
    end
end

function printItemRange2(minPrice Price? in, maxPrice Price? in)

    selectStatement STRING = "select * from EGL.ITEM where " ::
        "EGL.ITEM.PRICE between ? and ? " ::
        "order by EGL.ITEM.PRICE";

    // Prepare the statement to be used.
    tempItem Item;
    prepare preparedStatement
        from selectStatement
        for tempItem;

    mostExpensiveItem Item;
    get mostExpensiveItem
        into mostExpensiveItem.Price
        with
        #sql{
            select
                max(EGL.ITEM.PRICE)
            from EGL.ITEM

```

```

};

if (minPrice == NULL && maxPrice == NULL)
    // Two empty parameters;
    open expensiveItems for tempItem with preparedStatement
        using 0, mostExpensiveItem.Price;
else

    if (minPrice != NULL && maxPrice != NULL)
        // Both parameters were specified.
        open expensiveItems for tempItem with preparedStatement
            using minPrice, maxPrice;
    else
        // Only one parameter was specified.
        if (minPrice == NULL)
            // Maximum was specified.
            open expensiveItems for tempItem with preparedStatement
                using 0, maxPrice;
        else
            // Minimum was specified.
            open expensiveItems for tempItem with preparedStatement
                using minPrice, mostExpensiveItem.Price;
        end
    end

end

// Print the results to the console.
foreach (tempItem)
    SysLib.writeStdout(tempItem.Name :: " " ::
        tempItem.Description :: " $" :: tempItem.Price);
end

end

function printItemRange(minPrice Price? in, maxPrice Price? in)

    selectStatement STRING = "select * from EGL.ITEM where ";

    if (minPrice == NULL && maxPrice == NULL)
        // Two empty parameters;
        // return all rows with PRICE greater than zero.
        SysLib.writeStdout("All items with price greater than zero:");
        selectStatement ::= "EGL.ITEM.PRICE > 0 ";
    else

        if (minPrice != NULL && maxPrice != NULL)
            // Both parameters were specified.
            selectStatement ::= "EGL.ITEM.PRICE between " :: minPrice :: " and " :: maxPrice :: " ";
        else
            // Only one parameter was specified.
            if (minPrice == NULL)
                // Maximum was specified.
                selectStatement ::= "EGL.ITEM.PRICE <= " :: maxPrice :: " ";
            else
                // Minimum was specified.
                selectStatement ::= "EGL.ITEM.PRICE >= " :: minPrice :: " ";
            end
        end
    end

end

```

```

// Regardless of parameters, sort by price.
selectStatement ::= "order by EGL.ITEM.PRICE";

// Print completed SQL code.
SysLib.writeStdout("Completed query: "::selectStatement);

// Prepare the statement to be used.
tempItem Item;
prepare preparedStatement
  from selectStatement
  for tempItem;

// Use the prepared statement in place of explicit SQL.
open expensiveItems for tempItem with preparedStatement;

// Print the results to the console.
foreach (tempItem)
  SysLib.writeStdout(tempItem.Name :: " " ::
    tempItem.Description :: " $" :: tempItem.Price);
end

end

function printExpensiveItems(maxPrice Price in)

tempItem Item;
open expensiveItems for tempItem with
  #sql{
    select
      EGL.ITEM.ITEM_ID, EGL.ITEM.NAME, EGL.ITEM.IMAGE,
      EGL.ITEM.PRICE, EGL.ITEM.DESCRPTION
    from EGL.ITEM
    where
      EGL.ITEM.PRICE >= :maxPrice
    order by
      EGL.ITEM.ITEM_ID asc
  };

foreach (tempItem)
  SysLib.writeStdout(tempItem.Name :: " " ::
    tempItem.Description :: " $" :: tempItem.Price);
end

end

function handleDBException(exception SQLException)
  SysLib.writeStdout("Error accessing database. "::
    "See Troubleshooting section");
  SysLib.writeStdout(exception.message);
end

end

```

30 페이지의 『학습 4B: **prepare** 문의 호스트 변수』로 되돌아가십시오.

학습 5에서 완성된 printCustomerOrders.egl 및 records.egl 파일

이 코드는 학습 5에서 완성된 printCustomerOrders.egl 및 records.egl 파일입니다. 파일에 빨간색 X 기호로 표시된 오류가 있으면 사용자의 코드가 다음 코드와 일치하는지 확인하십시오.

```
[data/records.egl]
```

```
package data;
```

```
record OrderCustomerJoin type SQLRecord
{tableNames = ["EGL.CUSTOMER", "C"], ["EGL.ORDERS", "O"]},
keyItems=[CUSTOMER_ID, ORDER_ID],
defaultSelectCondition = #sqlCondition{ O.CUSTOMER_ID = C.CUSTOMER_ID }}
```

```
CUSTOMER_ID int
{column="C.CUSTOMER_ID", isReadOnly=yes};
FIRST_NAME string
{column="C.FIRST_NAME", isReadOnly=yes,
 isSqlNullable=yes, sqlVariableLen=yes, maxLen=30};
LAST_NAME string
{column="C.LAST_NAME", isReadOnly=yes,
 isSqlNullable=yes, sqlVariableLen=yes, maxLen=30};
PASSWORD string
{column="C.PASSWORD", isReadOnly=yes,
 isSqlNullable=yes, maxLen=8};
PHONE string
{column="C.PHONE", isReadOnly=yes, isSqlNullable=yes,
 sqlVariableLen=yes, maxLen=14};
EMAIL_ADDRESS string
{column="C.EMAIL_ADDRESS", isReadOnly=yes,
 isSqlNullable=yes, sqlVariableLen=yes, maxLen=50};
STREET string
{column="C.STREET", isReadOnly=yes,
 isSqlNullable=yes, sqlVariableLen=yes, maxLen=30};
APARTMENT string
{column="C.APARTMENT", isReadOnly=yes,
 isSqlNullable=yes, sqlVariableLen=yes, maxLen=10};
CITY string
{column="C.CITY", isReadOnly=yes, isSqlNullable=yes,
 sqlVariableLen=yes, maxLen=30};
STATE string
{column="C.STATE", isReadOnly=yes,
 isSqlNullable=yes, maxLen=2};
POSTALCODE string
{column="C.POSTALCODE", isReadOnly=yes,
 isSqlNullable=yes, sqlVariableLen=yes, maxLen=10};
DIRECTIONS string
{column="C.DIRECTIONS", isReadOnly=yes,
 isSqlNullable=yes, sqlVariableLen=yes, maxLen=255};
ORDER_ID int
{column="O.ORDER_ID", isReadOnly=yes};
orders_CUSTOMER_ID int
{column="O.CUSTOMER_ID", isReadOnly=yes,
 isSqlNullable=yes};
ORDER_AMOUNT decimal(8,2)
{column="O.ORDER_AMOUNT", isReadOnly=yes,
 isSqlNullable=yes};
ORDER_DETAILS string
```

```

        {column="O.ORDER_DETAILS", isReadOnly=yes,
          isSqlNullable=yes, sqlVariableLen=yes, maxLen=111};
ORDER_DATE date
        {column="O.ORDER_DATE", isReadOnly=yes,
          isSqlNullable=yes};
ORDER_STATUS string
        {column="O.ORDER_STATUS", isReadOnly=yes,
          isSqlNullable=yes, maxLen=12};
end

[programs/printCustomerOrders.egl]

package programs;

import data.OrderCustomerJoin;

program printCustomerOrders type BasicProgram {}

    function main()

        allCustomerOrders OrderCustomerJoin[0];

        tempRec OrderCustomerJoin;
        open resultSet for tempRec;
        foreach (tempRec)
            SysLib.writeStdout(tempRec.CUSTOMER_ID :: " "
                               :: tempRec.LAST_NAME :: " " :: tempRec.ORDER_ID);
        end

    end

end

```

34 페이지의 『학습 5: 테이블 결합을 사용하여 더욱 복잡한 데이터 검색』으로 되돌아가십시오.

학습 6에서 완성된 CustomerTableOperations.egl 및 duplicateTable.egl 파일

이 코드는 학습 6에서 완성된 CustomerTableOperations.egl 및 duplicateTable.egl 파일입니다. 파일에 빨간색 X 기호로 표시된 오류가 있으면 사용자의 코드가 다음 코드와 일치하는지 확인하십시오.

```

[libraries/CustomerTableOperations.egl]

package libraries;

import egl Derby7.ConditionHandlingLib;
import egl Derby7.StatusRec;

library CustomerTableOperations type BasicLibrary {}

    function execCreateTable(status StatusRec inOut)
        try
            execute
                #sql{
                    CREATE TABLE EGL.CUSTOMERTEMP (
                        CUSTOMER_ID INTEGER NOT NULL,
                        FIRST_NAME VARCHAR(30),
                        LAST_NAME VARCHAR(30),

```

```

        PASSWORD CHAR(8),
        PHONE VARCHAR(14),
        EMAIL_ADDRESS VARCHAR(50),
        STREET VARCHAR(30),
        APARTMENT VARCHAR(10),
        CITY VARCHAR(30),
        STATE CHAR(2),
        POSTALCODE VARCHAR(10),
        DIRECTIONS VARCHAR(255)
    )
};
onException (exception SQLException)
    ConditionHandlingLib.HandleException(status, exception);
end

end

function execInsertIntoTable(status StatusRec inOut)
    try
        execute #sql{
            INSERT INTO EGL.CUSTOMERTEMP
            SELECT * FROM EGL.CUSTOMER
        } ;
        onException (exception SQLException)
            ConditionHandlingLib.HandleException(status, exception);
    end
end

function execDropTable(status StatusRec inOut)
    try
        execute #sql{
            DROP TABLE EGL.CUSTOMERTEMP
        } ;
        onException (exception SQLException)
            ConditionHandlingLib.HandleException(status, exception);
    end
end

end

```

[programs/duplicateTable.egl]

```

package programs;

import eglderbyr7.StatusRec;
import libraries.CustomerTableOperations;

program duplicateTable type BasicProgram {}

    status StatusRec;

    function main()
        CustomerTableOperations.execCreateTable(status);
        CustomerTableOperations.execInsertIntoTable(status);
    end
end

```

```

        //CustomerTableOperations.execDropTable(status);
    end

end

```

39 페이지의 『학습 6: 임의의 SQL 코드 실행』으로 되돌아가십시오.

학습 7에서 완성된 exceptionHandlingTest.egl 파일

이 코드는 학습 7에서 완성된 exceptionHandlingTest.egl 파일입니다. 파일에 빨간색 X 기호로 표시된 오류가 있으면 사용자의 코드가 다음 코드와 일치하는지 확인하십시오.

```

package programs;

import eglderbyr7.data.Customer;
import eglderbyr7.data.Orders;

program exceptionHandlingTest type BasicProgram {}

function main()
    customer Customer;
    customer.FirstName = "John";
    customer.LastName = "Doe";
    customer.CustomerId = 1;

    customerOrder Orders;
    customerOrder.CustomerId = 1;
    customerOrder.OrderId = 50;
    customerOrder.OrderAmount = 0;
    customerOrder.OrderDetails =
        "This is my first order, so get it right!";

    try
        add customerOrder;
        SysLib.writeStdout("Order added successfully.");
        add customer;
        SysLib.writeStdout("Customer added successfully.");
    onException(ex SQLException)
        SysLib.writeStdout("SQL exception ">::ex.messageID);
        SysLib.writeStdout("message = ">::ex.message);
        SysLib.writeStdout("Performing rollback.");
        SysLib.rollback();
    end

    tempOrder Orders;
    tempOrder.OrderId = 50;
    get tempOrder;

    if (tempOrder is noRecordFound)
        SysLib.writeStdout("The order was rolled back successfully.");
    else
        SysLib.writeStdout("The order is still in the database");
    end
end

```

end

end

50 페이지의 『학습 7: 예외 처리 및 롤백』으로 되돌아가십시오.

부록. 주의사항

이 정보는 미국에서 제공되는 제품 및 서비스용으로 작성된 것입니다. IBM은 다른 국가에서 이 문서에 기술된 제품, 서비스 또는 기능을 제공하지 않을 수도 있습니다. 현재 사용할 수 있는 제품 및 서비스에 대한 정보는 한국 IBM 담당자에게 문의하십시오. 이 문서에서 IBM 제품, 프로그램 또는 서비스를 언급했다는 것이 해당 IBM 제품, 프로그램 또는 서비스만을 사용할 수 있다는 것을 의미하지는 않습니다. IBM의 지적 재산권을 침해하지 않는 한, 기능상으로 동등한 제품, 프로그램 또는 서비스를 대신 사용할 수도 있습니다. 그러나 비 IBM 제품, 프로그램 또는 서비스의 운영에 대한 평가 및 검증은 사용자의 책임입니다.

IBM은 이 문서에 설명된 특정 내용을 다루는 응용프로그램에 대해 특허를 보유하고 있거나 현재 특허 출원 중일 수 있습니다. 이 문서를 제공한다고 해서 특허에 대한 라이선스까지 부여하는 것은 아닙니다. 라이선스에 대한 의문사항은 다음으로 문의하십시오.

135-700

서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩

한국 아이.비.엠 주식회사

고객만족센터

전화번호: 080-023-8080

2바이트(DBCS) 정보에 관한 라이선스 문의는 한국 IBM 고객만족센터에 문의하거나 다음 주소로 서면 문의하시기 바랍니다.

IBM World Trade Asia Corporation

Licensing

2-31 Roppongi 3-chome, Minato-ku

Tokyo 106-0032, Japan

다음 단락은 현지법과 상충하는 영국이나 기타 국가에서는 적용되지 않습니다. IBM은 타인의 권리 비침해, 상품성 또는 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여(단, 이에 한하지 않음) 묵시적이든 명시적이든 어떠한 종류의 보증 없이 이 책을 현상태대로 제공합니다. 일부 국가에서는 특정 거래에서 명시적 또는 묵시적 보증의 면책사항을 허용하지 않으므로, 이 사항이 적용되지 않을 수도 있습니다.

이 정보에는 기술적으로 부정확한 내용이나 인쇄상의 오류가 있을 수 있습니다. 이 정보는 주기적으로 변경되며, 이 변경사항은 최신판에 통합됩니다. IBM은 이 책에서 설명한 제품 및/또는 프로그램을 사전 통고없이 언제든지 개선 및/또는 변경할 수 있습니다.

이 정보에서 비IBM의 웹 사이트는 단지 편의상 제공된 것으로, 어떤 방식으로든 이들 웹 사이트를 옹호하고자 하는 것은 아닙니다. 해당 웹 사이트의 자료는 본 IBM 제품 자료의 일부가 아니므로 해당 웹 사이트 사용으로 인한 위험은 사용자 본인이 감수해야 합니다.

(i) 독립적으로 작성된 프로그램과 기타 프로그램(본 프로그램 포함) 간의 정보 교환 및 (ii) 교환된 정보의 상호 이용을 목적으로 정보를 원하는 프로그램 라이선스 사용자는 다음 주소로 문의하십시오.

135-700

서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터

이러한 정보는 해당 조항 및 조건에 따라(예를 들면, 사용료 지불 포함) 사용할 수 있습니다.

이 문서에 기술된 라이선스가 있는 프로그램 및 이 프로그램에 대해 사용 가능한 라이선스가 있는 모든 자료는 IBM이 IBM 기본 계약, IBM 프로그램 라이선스 계약(IPLA) 또는 이와 동등한 계약에 따라 제공한 것입니다.

본 문서에 포함된 모든 성능 데이터는 제한된 환경에서 산출된 것입니다. 따라서 다른 운영 환경에서 얻어진 결과는 상당히 다를 수 있습니다. 일부 성능은 개발 레벨 상태의 시스템에서 측정되었을 수 있으므로 이러한 측정치가 일반적으로 사용되고 있는 시스템에서도 동일하게 나타날 것이라고는 보증할 수 없습니다. 또한, 일부 성능은 추정치일 수도 있으므로 실제 결과는 다를 수 있습니다. 이 문서의 사용자는 해당 데이터를 사용자의 특정 환경에서 검증해야 합니다.

비IBM 제품에 관한 정보는 해당 제품의 공급업체, 공개 자료 또는 기타 범용 소스로부터 얻은 것입니다. IBM에서는 이러한 제품들을 테스트하지 않았으므로, 비IBM 제품과 관련된 성능의 정확성, 호환성 또는 기타 청구에 대해서는 확신할 수 없습니다. 비IBM 제품의 성능에 대한 의문사항은 해당 제품의 공급업체에 문의하십시오.

IBM의 향후 방향 또는 의도에 관한 모든 언급은 별도의 통지없이 변경되거나 철회될 수 있으며 목표 및 목적만을 반영합니다.

이 정보에는 일상의 비즈니스 운영에서 사용되는 자료 및 보고서에 대한 예제가 들어 있습니다. 이들 예제에는 개념을 가능한 완벽하게 설명하기 위해 개인, 회사, 상표 및 제품의 이름이 사용될 수 있습니다. 이들 이름은 모두 가공의 것이며 실제 기업의 이름 및 주소와 유사하더라도 이는 전적으로 우연입니다.

저작권 라이선스:

이 정보에는 여러 운영 플랫폼에서의 프로그래밍 기법을 보여주는 원어로 된 샘플 응용프로그램이 들어 있습니다. 귀하는 이러한 샘플 프로그램의 작성 기준이 된 운영 플랫폼의 응용프로그램 프로그래밍 인터페이스(API)에 부합하는 응용프로그램을 개발, 사용, 판매 또는 배포할 목적으로 추가 비용없이 이들 샘플 프로그램을 어떠한 형태로든 복사, 수정 및 배포할 수 있습니다. 이러한 샘플 프로그램은 모든 조건하에서 완전히 테스트된 것은 아닙니다. 따라서 IBM은 이들 샘플 프로그램의 신뢰성, 서비스 가능성 또는 기능을 보증하거나 암시하지 않습니다.

이러한 샘플 프로그램 또는 파생 제품의 각 사본이나 그 일부에는 반드시 다음과 같은 저작권 표시가 포함되어야 합니다.

© (귀하의 회사명) (연도). 이 코드의 일부는 IBM Corp.의 샘플 프로그램에서 파생됩니다. © Copyright IBM Corp. _연도 입력_. All rights reserved.

이 정보를 소프트키피로 확인하는 경우에는 사진과 컬러 삽화가 제대로 표시되지 않을 수도 있습니다.

상표

다음 용어는 미국 또는 기타 국가에서 사용되는 IBM Corporation의 상표입니다.

IBM

Rational

Java 및 모든 Java 기반 상표는 미국 또는 기타 국가에서 사용되는 Sun Microsystems, Inc.의 상표입니다.

기타 회사, 제품 또는 서비스 이름은 해당 회사의 상표 또는 서비스표입니다.

IBM