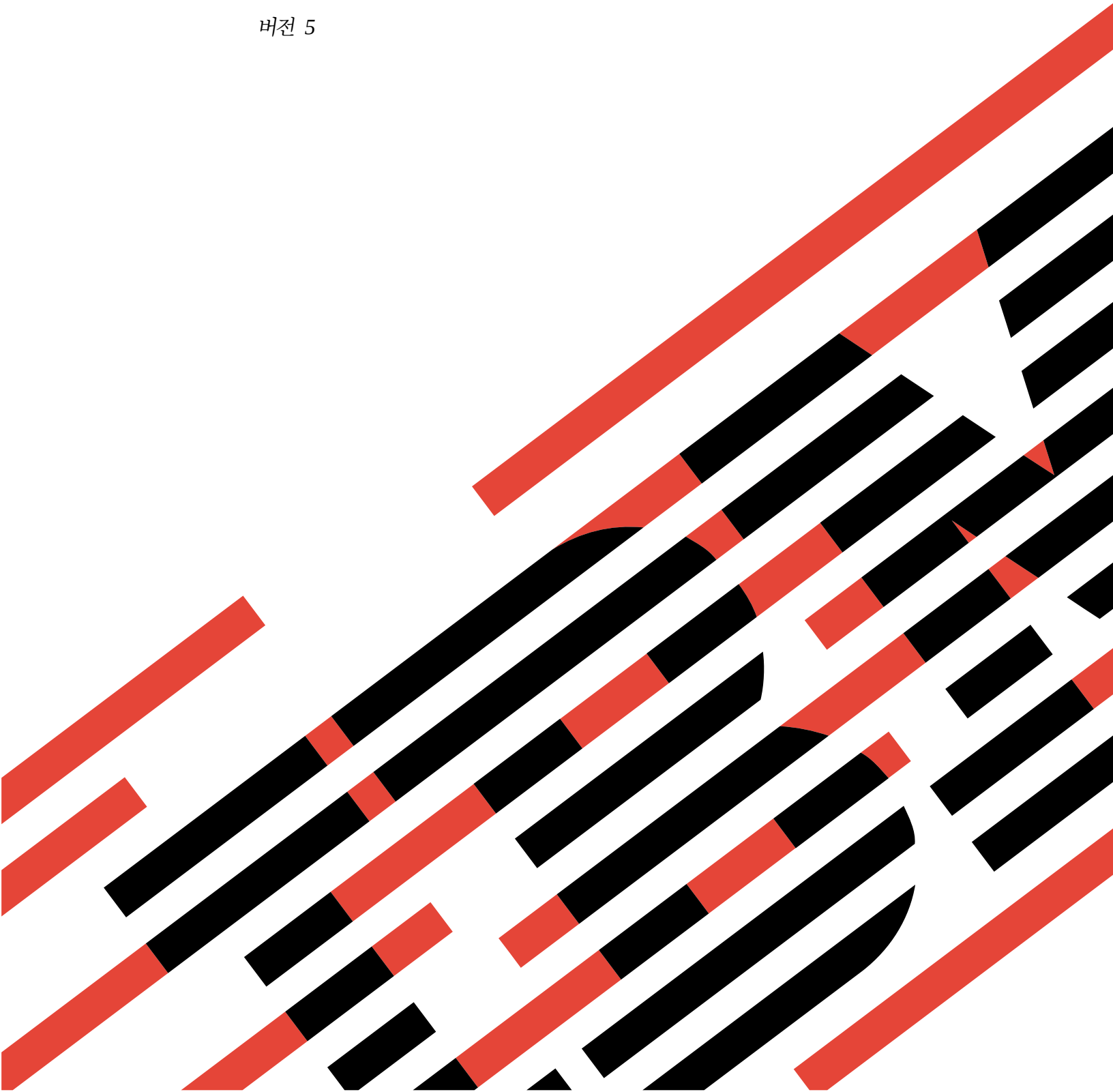
IBM

eserver

iSeries

iSeries용 DB2 Universal Database SQL 프로그래밍 개념

버전 5



IBM

@server

iSeries

iSeries용 DB2 Universal Database SQL 프로그래밍 개념

버전 5

목차

iSeries용 DB2 UDB SQL 프로그래밍 개념 정보

이 책은 프로그래머와 데이터베이스 관리자를 나타내는 기본적인 SQL 프로그래밍 개념을 설명합니다.

- iSeries용 DB2 UDB 사용권 프로그램을 사용하는 방법
- 데이터베이스에서 자료에 액세스하는 방법
- SQL문을 포함하는 어플리케이션 프로그램을 준비, 실행 및 테스트하는 방법

어플리케이션 프로그래밍 환경에서 구현하기 위한 iSeries용 DB2 UDB SQL 지침과 예에 대한 자세한 내용은 iSeries Information Center에서 다음의 책을 참조하십시오.

- SQL 참조서
- 호스트 언어를 위한 iSeries용 DB2 UDB SQL 프로그래밍
- iSeries용 DB2 UDB 데이터베이스 성능 및 조회 최적화
- SQL 호출 레벨 인터페이스(ODBC)
- SQL 메세지 및 코드

이 안내서에 대한 자세한 정보는 다음의 주제를 참조하십시오.

- 『이 책의 사용자』
- x 페이지의 『SQL문 예와 관련된 가정』
- x 페이지의 『코드 면책사항 정보』
- xi 페이지의 『이 안내서의 구문 다이어그램 해석 방법』
- xii 페이지의 『SQL 프로그래밍 개념 정보 V5R2 버전의 새로운 사항』

이 책의 사용자

이 안내서는 iSeries용 COBOL, iSeries용 ILE COBOL, iSeries PL/I, iSeries용 ILE C, ILE C++, REXX, RPG III(iSeries용 RPG의 일부) 또는 iSeries용 ILE RPG 언어에 익숙하며 이러한 언어를 사용하여 프로그램을 만들 수 있으며 기본 데이터베이스 어플리케이션을 이해할 수 있는 어플리케이션 프로그래머와 데이터베이스 관리자들이 사용해야 합니다.

또한 다음 절을 참조하십시오.

- x 페이지의 『SQL문 예와 관련된 가정』
- x 페이지의 『코드 면책사항 정보』
- xi 페이지의 『이 안내서의 구문 다이어그램 해석 방법』

SQL문 예와 관련된 가정

이 책에 나와 있는 SQL문의 샘플은 부록 A, iSeries용 DB2 UDB 샘플 표를 기본으로 한 것으로써, 다음을 전제로 한 것입니다.

- 대화식 SQL 환경에서 표시되거나 ILE C 또는 COBOL로 기록됩니다. EXEC SQL 및 END-EXEC는 COBOL 프로그램에서 SQL문을 구분하기 위해 사용합니다. COBOL 프로그램에서 SQL문을 사용하는 방법에 대한 설명은 ″COBOL 어플리케이션에 SQL문 코드화″에 제공되어 있습니다. ILE C 프로그램에서 SQL문을 사용하는 방법에 대한 설명은 ″C 어플리케이션에 SQL문 코드화″에 제공되어 있습니다.
- 각 SQL 예제는 여러 행으로 표시되며 명령문의 각 절은 하나의 개별 행으로 표시됩니다.
- SQL 키워드는 강조표시됩니다.
- 부록 A, iSeries용 DB2 UDB 샘플 표에 제공된 표 이름들은 스키마 CORPDATA를 사용합니다. 샘플 표에 없는 표 이름은 사용자가 작성한 스키마를 사용해야 합니다.
- 계산되는 열은 괄호 ()와 브라켓 []으로 묶입니다.
- SQL 명명 규칙이 사용됩니다.
- APOST 및 APOSTSQL 사전컴파일러 옵션은 COBOL에서 디폴트 옵션이 아니더라도 옵션으로 가정됩니다. SQL 및 호스트 언어 명령문 내의 문자 스트링 리터럴은 어포스트로피(’)로 구분됩니다.
- 다른 주(note)가 없는 한, *HEX의 정렬 순서가 이용됩니다.
- 대부분의 예제에는 SQL문의 완전한 구문이 표시되어 있지 않습니다. 이 안내서에 설명된 명령문에 대한 완전한 설명과 구문에 대해서는 SQL 참조서를 참조하십시오.

이러한 가정은 예제가 나올 때마다 언급됩니다.

이 책은 어플리케이션 프로그래머를 위한 것이므로 대부분의 예제가 어플리케이션 프로그램에서 작성된 것처럼 표시됩니다. 그러나 많은 예제는 다소 변경될 수 있으며 대화식 SQL을 사용하여 대화식으로 수행이 가능합니다. 대화식 SQL 사용시, SQL문의 구문은 동일한 명령문이 프로그램에 삽입될 때의 형식과 약간 차이가 납니다.

코드 면책사항 정보

이 문서에는 프로그래밍 예제가 들어 있습니다.

IBM은 귀하에게 유사한 기능을 귀하의 특정 요구에 맞게 조정하여 생성할 수 있도록 모든 프로그래밍 코드 예제를 사용할 수 있는 비독점적인 저작권 사용권을 부여합니다.

모든 샘플 예제는 IBM에 의해 예시 목적으로만 제공됩니다. 이러한 예제는 모든 조건 하에서 철저히 테스트된 것은 아닙니다. 따라서 IBM은 이들 프로그램의 신뢰성, 실용성 또는 기능에 대해 보증할 수 없습니다.

여기에 포함된 모든 프로그램은 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여 어떠한 종류의 보증 없이 "현상태대로" 제공됩니다.

이 안내서의 구문 다이어그램 해석 방법

이 책에서는 다음과 같은 구조를 사용하여 구문을 서술합니다.

- 구문 다이어그램은 왼쪽에서 오른쪽으로, 위에서 아래로 그리고 다음과 같은 선의 경로를 따라서 읽도록 하십시오.

  ▶▶── 기호는 명령문의 시작을 나타냅니다.

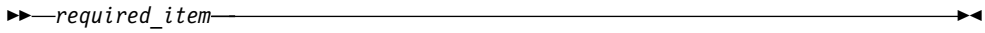
  ──▶ 기호는 명령문 구문이 다음 행에 계속됨을 나타냅니다.

  ▶── 기호는 명령문이 이전 행에서 계속된 것임을 나타냅니다.

  ──▶◀ 기호는 명령문의 끝을 나타냅니다.

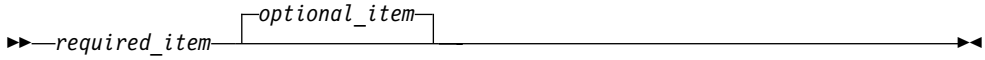
  명령문 전체를 나타내는 경우가 아닌 구문 단위의 다이어그램에서는 ▶── 기호로 시작하여 ──▶ 기호로 끝납니다.

- 필수 항목은 수평선(기본 경로)상에 표시됩니다.



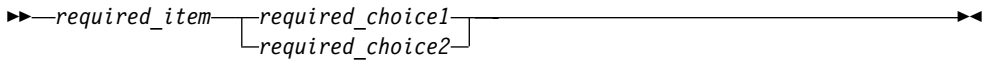
- 생략 가능한 항목은 기본 경로 밑에 표시됩니다.

  생략 가능한 항목이 기본 경로 위에 표시되면, 그 항목은 명령문의 실행에 아무런 영향을 미치지 않으며 읽기용으로만 사용되는 것입니다.

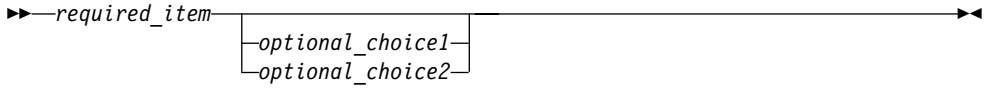


- 둘 이상의 항목에서 선택할 경우에는 항목들이 세로로 스택에 표시됩니다.

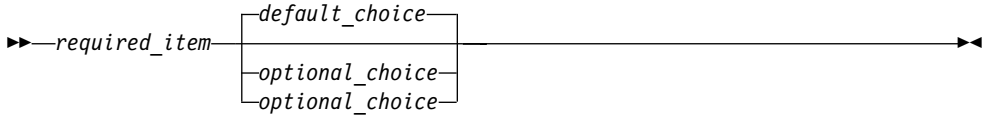
  항목 중 하나를 선택*해야* 하는 경우, 스택의 한 항목이 기본 경로에 나타납니다.



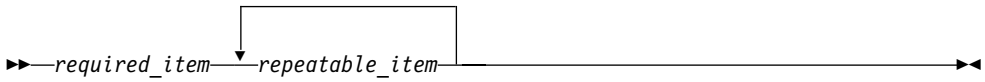
  항목 중에서 반드시 하나를 선택하지 않아도 되는 경우에는 전체 스택이 기본 경로 밑에 표시됩니다.



  항목 중 하나가 디폴트이면 그 디폴트는 기본 경로 위에 표시되고 나머지 선택사항들은 기본 경로 밑에 표시됩니다.



- 화살표가 기본 행의 위를 통해 왼쪽으로 되돌아 오면 항목이 반복될 수 있음을 나타냅니다.



반복 화살표에 쉼표가 들어 있으면 반복되는 항목들을 쉼표로 분리해야 합니다.

스택 위의 반복 화살표는 스택 내의 항목들을 반복할 수 있음을 나타냅니다.

- 키워드는 대문자로 표시되며(예를 들어 FROM) 표시된 대로 철자를 정확히 입력해야 합니다. 변수는 모두 영문 소문자로 표시되며(예를 들어 *column-name*) 사용자 제공 이름이나 값을 나타냅니다.
- 구두점, 괄호, 산술 연산자 또는 기타 그와 같은 기호들이 표시되면 구문의 한 부분으로서 반드시 입력해야 합니다.

SQL 프로그래밍 개념 정보 V5R2 버전의 새로운 사항

이 릴리스 정보에 대한 주요 변경사항은 다음과 같습니다.

- 사용자 정의 표 기능. 세부사항은 221 페이지의 『사용자 정의 기능(UDF)』을 참조하십시오.
- 펜스되지 않은 사용자 정의 기능. 세부사항은 272 페이지의 『기능을 펜스 또는 펜스되지 않게 설정』을 참조하십시오.
- 저장점. 세부사항은 354 페이지의 『저장점』을 참조하십시오.
- 임시 표
- 식별 열. 세부사항은 56 페이지의 『식별 열 작성 및 변경』을 참조하십시오.
- 스칼라 부속 선택
- SQL 프로시듀어 디버깅. 세부사항은 185 페이지의 『저장 프로시듀어 디버그』를 참조하십시오.

제 1 장 iSeries용 DB2 UDB 구조화 조회 언어(SQL) 소개

이들 주제에서는 iSeries용 DB2 UDB와 DB2 UDB 조회 관리자 및 SQL 개발 킷 버전 5 사용권 프로그램을 사용한 SQL(구조화 조회 언어)의 iSeries 서버 구현에 대해 설명합니다. SQL은 자료의 관계 모델에 따라 정보를 관리합니다. SQL문은 고급 언어에 삽입되어 동적으로 준비 및 수행되거나 대화식으로 수행될 수 있습니다.

SQL은 데이터베이스 내의 자료로 수행하려는 작업과 작업 수행 조건을 설명하는 명령문과 절로 구성됩니다.

이 주제는 다음에 대해서 설명합니다.

- 『SQL 개념』
- 6 페이지의 『SQL 오브젝트』
- 11 페이지의 『어플리케이션 프로그램 오브젝트』

SQL은 IBM Distributed Relational Database Architecture*(DRDA*)를 사용하여 리모트 관계형 데이터베이스 내의 자료에 액세스할 수 있습니다. 이 기능은 이 안내서의 제 21 장 『분산 관계형 데이터베이스 기능』 주제에서 설명합니다. DRDA에 대해 자세한 내용은 분산 데이터베이스 프로그래밍 책에 들어 있습니다.

SQL 개념

iSeries용 DB2 UDB SQL은 다음의 주요 부분으로 구성됩니다.

- SQL 런타임 지원

  SQL 런타임은 SQL문을 분석하고 모든 SQL문을 수행합니다. 이 지원 사항은 Operating System/400*(OS/400) 사용권 프로그램의 일부로 DB2 UDB 조회 관리자 및 SQL 개발 킷 사용권 프로그램이 설치되어 있지 않은 시스템에서 SQL문을 포함하는 어플리케이션이 수행될 수 있도록 합니다.

- SQL 사전컴파일러

  SQL 사전컴파일러는 호스트 언어에 포함된 SQL문의 사전컴파일을 지원합니다. 다음 언어가 지원됩니다.

  – ILE C
  – iSeries용 ILE C++
  – ILE COBOL
  – iSeries용 COBOL
  – iSeries PL/I
  – RPG III(iSeries용 RPG의 일부)

– ILE RPG

SQL 호스트 언어 사전컴파일러는 SQL문이 들어 있는 어플리케이션 프로그램을 준비합니다. 그런 후 호스트 언어 컴파일러가 사전컴파일된 호스트 소스 프로그램을 컴파일합니다. 사전컴파일에 대한 자세한 내용은 *호스트 언어를 사용한 SQL 프로그래밍* 정보에서 SQL문을 사용한 프로그램 준비 및 실행 주제를 참조하십시오. 사전컴파일러 지원은 DB2 UDB 조회 관리자 및 SQL 개발 킷 사용권 프로그램의 한 부분입니다.

- SQL 대화식 인터페이스

  SQL 대화식 인터페이스를 사용하면 SQL문을 작성하고 수행할 수 있습니다. 대화식 SQL에 대한 자세한 내용은 제 17 장 『대화식 SQL 사용』을 참조하십시오. 대화식 SQL은 DB2 UDB 조회 관리자 및 SQL 개발 킷 사용권 프로그램의 한 부분입니다.

- SQL 스크립트 실행

  iSeries Navigator에서 SQL 스크립트 실행 창을 통해 SQL문의 스크립트를 작성, 편집, 실행 및 문제점 해결할 수 있습니다. SQL 스크립트 실행은 iSeries Navigator의 일부입니다. 추가 정보는 309 페이지의 『SQL 스크립트 실행을 사용하여 데이터베이스 조회』를 참조하십시오.

- SQL문 CL 명령의 실행

  RUNSQLSTM을 사용하면 소스 파일에 저장되어 있는 일련의 SQL문을 실행할 수 있습니다. SQL문 실행 명령에 대한 자세한 내용은 제 18 장 『SQL문 프로세서 사용』을 참조하십시오.

- iSeries용 DB2 조회 관리자

  iSeries용 DB2 조회 관리자는 프롬트 방식 대화형 인터페이스로서 사용자는 이를 통해 자료를 작성하고 추가 및 유지보수하며 데이터베이스에 대한 보고서를 수행할 수 있습니다. 조회 관리자는 DB2 UDB 조회 관리자 및 SQL 개발 킷 사용권 프로그램의 한 부분입니다. 자세한 내용은 조회 관리자 사용법을 참조하십시오.

- SQL REXX 인터페이스

  SQL REXX 인터페이스를 사용하면 REXX 프로시듀어에서 SQL문을 실행할 수 있습니다. REXX 프로시듀어에서의 SQL문 사용에 대한 자세한 정보는 *호스트 언어를 사용한 SQL 프로그래밍* 정보에 있는 REXX 어플리케이션에서 SQL문 코드화 주제를 참조하십시오.

- SQL 호출 레벨 인터페이스

  iSeries용 DB2 UDB는 SQL 호출 레벨 인터페이스를 지원합니다. 이를 사용함으로써 모든 ILE 언어의 사용자들은 시스템이 제공하는 서비스 프로그램에 대한 프로시듀어 호출을 통해 직접 SQL 함수를 액세스할 수 있습니다. SQL 호출 레벨 인터페이스를 사용하면 사전컴파일을 하지 않고도 모든 SQL 함수를 실행할 수 있습니다. 이것이 SQL문을 준비하고 SQL문을 실행하며 자료 행을 페치하기 위한 표준 프로

시듀어 호출 세트로서 이를 통해 목록을 액세스하고 프로그램 변수를 출력 열로 바인딩하는 것과 같은 확장 기능도 수행할 수 있습니다.

사용할 수 있는 모든 기능과 구문에 대한 자세한 설명은 SQL 호출 레벨 인터페이스(ODBC) 책을 참조하십시오.

- QSQPRCED API

  이 어플리케이션 프로그램 인터페이스(API)는 확장된 동적 SQL 능력을 제공합니다. SQL문은 SQL 패키지로 준비된 다음 이 API를 사용하여 실행할 수 있습니다. 이 API에 의해 패키지로 준비한 명령문은 패키지나 명령문이 명시적으로 삭제될 때까지 유지됩니다. QSQPRCED는 OS/400 사용권 프로그램의 한 부분입니다. QSQPRCED API에 대한 자세한 내용은 iSeries Information Center의 프로그래밍 절에서 QSQPRCED 주제를 참조하십시오. API에 대한 일반 정보는 iSeries Information Center에서 OS/400 API 주제를 참조하십시오.

- QSQCHKS API

  이 API 구문은 SQL문을 체크합니다. QSQCHKS는 OS/400 사용권 프로그램의 한 부분입니다. QSQCHKS API에 대한 자세한 내용은 iSeries Information Center의 프로그래밍 절에서 QSQCHKS 주제를 참조하십시오. API에 대한 일반 정보는 iSeries Information Center에서 OS/400 API 주제를 참조하십시오.

- DB2 Multisystem

  이와 같은 오퍼레이팅 시스템의 피처로 인해 사용자의 자료를 여러 서버에 분산시킬 수 있습니다. DB2 Multisystem에 대한 자세한 내용은 DB2 Multisystem 책을 참조하십시오.

- DB2 UDB 대칭형 멀티프로세싱

  이와 같은 오퍼레이팅 시스템의 피처는 조회 최적자에게 병렬 처리가 포함되어 있는 자료 검색 방법을 추가로 제공합니다. 대칭형 멀티프로세싱(SMP)은 메모리와 디스크 자원을 공유하는 다수의 프로세서(CPU 및 I/O 프로세서)가 하나의 최종 결과를 위해 동시에 작업하는 단일 시스템에서 이루어집니다. 병렬 처리는 데이터베이스 관리자가 하나의 조회를 처리함에 있어서 다수의 시스템 자원을 동시에 사용할 수 있음을 의미합니다. 병렬 처리 제어 방법에 대한 세부사항은 *데이터베이스 성능 및 조회 최적화* 정보에서 주제, 병렬 처리 제어를 참조하십시오.

자세한 정보는 다음의 절을 참조하십시오.

- 『SQL 관계형 데이터베이스 및 시스템 용어』
- 4 페이지의 『SQL문의 유형』
- 6 페이지의 『SQL 통신 영역(SQLCA)』

SQL 관계형 데이터베이스 및 시스템 용어

자료의 모델에서 모든 자료는 표에 존재하는 것으로 간주됩니다. iSeries용 DB2 UDB 오브젝트는 시스템 오브젝트로 작성되고 유지보수됩니다. 다음 표에서는 시스템 용어와

SQL 관계형 데이터베이스 용어 사이의 관계를 보여줍니다. 고유 파일 인터페이스 사용에 관한 데이터베이스 프로그래밍에 대한 자세한 정보는 데이터베이스 프로그래밍 책을 참조하십시오.

표 1. 시스템 용어와 SQL 용어 사이의 관계

시스템 용어	SQL 용어
라이브러리. 연관 오브젝트를 그룹화하여 이름별로 오브젝트화합니다.	**스키마**. 라이브러리, 저널, 저널 리시버, SQL 카탈로그 및 선택적인 자료 사전으로 구성됩니다. 스키마는 관련된 오브젝트들을 그룹화하고 오브젝트를 이름으로 찾을 수 있도록 합니다.
실제 파일(PF). 일련의 레코드	**표**. 일련의 열과 행
레코드. 일련의 필드	**행**. 일련의 열이 들어 있는 표의 수평 부분
필드. 하나의 자료 유형에 대한 하나 이상의 연관 정보 문자	**열**. 하나의 자료 유형에 대한 표의 수직 부분
논리 파일. 하나 이상의 실제 파일의 필드와 레코드의 서브세트	**보기**. 하나 이상의 표 내의 열과 행의 서브세트
SQL 패키지. SQL문 실행시에 사용된 오브젝트 유형	**패키지**. SQL문 실행시에 사용된 오브젝트 유형
사용자 프로파일	**권한부여 이름 또는 ID**

또한 다음을 참조하십시오.

- 『SQL 및 시스템 명명 규약』

SQL 및 시스템 명명 규약

iSeries용 DB2 UDB 프로그래밍에서 사용될 수 있는 명명 규약에는 두 가지(시스템(*SYS) 및 SQL(*SQL))이 있습니다. 사용되는 명명 규칙은 대화식 SQL 화면에서 사용되는 파일, 표 이름, 용어를 규정하는 방법 등에 영향을 미칩니다. 사용되는 명명 규칙은 SQL 명령의 매개변수에 의해 선택되거나 REXX의 경우 SET OPTION문을 통해 선택됩니다. 자세한 내용은 SQL 참조서의 규정화되지 않은 오브젝트명의 규정을 참조하십시오.

시스템 명명(*SYS): 시스템 명명 규칙에서, SQL문의 표 및 다른 SQL 오브젝트는 다음 양식의 스키마명으로 규정됩니다.

```
schema/table
```

SQL 명명(*SQL): SQL 명명 규칙에서, SQL문의 표 및 다른 SQL 오브젝트는 다음 양식의 스키마명으로 규정됩니다.

```
schema.table
```

SQL문의 유형

다음은 네 가지의 SQL문 기본 유형입니다.

- DDL(data definition language)문
- DML(data manipulation language)문
- 동적 SQL문

- 기타 명령문

SQL문은 SQL 스키마 내에 있는지 여부와는 상관없이 외부적으로 설명된 실제 파일과 단일 형식 논리 파일 뿐만 아니라 SQL에 의해 작성된 오브젝트에서 작업할 수 있습니다. 프로그램 서술 파일에 대한 IDDU 사전 정의는 참조되지 않습니다. 프로그램 서술 파일은 단일 열만을 가진 표로서 표시됩니다.

SQL DDL문	**SQL DML문**
ALTER TABLE	CLOSE
COMMENT ON	COMMIT
CREATE ALIAS	DECLARE CURSOR
CREATE DISTINCT TYPE	DELETE
CREATE FUNCTION	FETCH
CREATE INDEX	INSERT
CREATE PROCEDURE	LOCK TABLE
CREATE SCHEMA	OPEN
CREATE TABLE	RELEASE SAVEPOINT
CREATE TRIGGER	ROLLBACK
CREATE VIEW	SAVEPOINT
DECLARE GLOBAL TEMPORARY TABLE	SELECT INTO
DROP ALIAS	SET 변수
DROP DISTINCT TYPE	UPDATE
DROP FUNCTION	VALUES INTO
DROP INDEX	
DROP PACKAGE	
DROP PROCEDURE	
DROP SCHEMA	
DROP TABLE	
DROP TRIGGER	
DROP VIEW	
GRANT DISTINCT TYPE	
GRANT FUNCTION	
GRANT PACKAGE	
GRANT PROCEDURE	
GRANT TABLE	
LABEL ON	
RENAME	
REVOKE DISTINCT TYPE	
REVOKE FUNCTION	
REVOKE PACKAGE	
REVOKE PROCEDURE	
REVOKE TABLE	

동적 SQL문	기타 명령문
DESCRIBE	BEGIN DECLARE SECTION
EXECUTE	CALL
EXECUTE IMMEDIATE	CONNECT
PREPARE	DECLARE PROCEDURE
	DECLARE STATEMENT
	DECLARE VARIABLE
	DESCRIBE TABLE
	DISCONNECT
	END DECLARE SECTION
	FREE LOCATOR
	HOLD LOCATOR
	INCLUDE
	RELEASE
	SET CONNECTION
	SET OPTION
	SET PATH
	SET RESULT SETS
	SET SCHEMA
	SET TRANSACTION
	WHENEVER

SQL DDL문은 53 페이지의 제 4 장『DDL(Data Definition Language)』에 설명되어 있습니다. SQL DML문은 69 페이지의 제 5 장『SELECT문을 사용하여 자료 검색』 및 105 페이지의 제 6 장『SQL 삽입, 갱신 및 삭제』에 설명되어 있습니다. 이 명령문에 대한 완벽한 설명을 SQL 참조서에서 참조할 수 있습니다.

SQL 통신 영역(SQLCA)

SQLCA는 모든 SQL문의 실행 마지막에 갱신되는 변수 세트입니다. 실행가능한 SQL 문이 포함된 프로그램은 정확히 하나의 SQLCA를 제공해야 합니다(독립형 SQLCODE 또는 독립형 SQLSTATE 변수가 대신 사용되는 경우는 제외됩니다). 자세한 내용은 iSeries Information Center의 *SQL 참조서* 책에서 SQL 통신 영역 주제를 참조하십시오.

SQL 오브젝트

SQL 오브젝트는 스키마, 자료 사전, 저널, 카탈로그, 표, 별명, 보기, 색인, 제한사항, 트리거, 저장 프로시듀어, 사용자 정의 기능, 사용자 정의 유형 및 SQL 패키지입니다. SQL은 이러한 오브젝트들을 시스템 오브젝트로 작성하고 유지보수합니다. 다음은 이러한 오브젝트에 대한 간략한 설명입니다.

- 7 페이지의『스키마』
- 7 페이지의『자료 사전』
- 7 페이지의『저널과 저널 리시버』
- 8 페이지의『카탈로그』

- 8 페이지의 『표, 행 및 열』
- 8 페이지의 『별명』
- 8 페이지의 『보기』
- 9 페이지의 『색인』
- 9 페이지의 『제한조건』
- 10 페이지의 『트리거』
- 10 페이지의 『저장 프로시듀어』
- 10 페이지의 『사용자 정의 기능』
- 10 페이지의 『사용자 정의 유형』
- 10 페이지의 『SQL 패키지』

스키마

스키마는 SQL 오브젝트의 논리 그룹을 제공합니다. **스키마**는 라이브러리, 저널, 저널 리시버, 카탈로그로 구성되며 자료 사전도 선택할 수 있습니다. 표, 보기 및 시스템 오브젝트(예: 프로그램)를 시스템 라이브러리로 작성, 이동 또는 복원할 수 있습니다. SQL 스키마에 자료 사전이 포함되지 않은 경우, 모든 시스템 파일이 SQL 스키마로 작성되거나 이동될 수 있습니다. SQL 스키마에 자료 사전이 포함된 경우, 다음과 같은 사항이 가능합니다.

- 멤버가 하나인 소스 실제 파일이나 비소스 실제 파일이 SQL 스키마로 작성, 이동 또는 복원될 수 있습니다.
- 논리 파일은 자료 사전에서 설명될 수 없으므로 SQL 스키마에 배치할 수 없습니다.

여러개의 스키마를 작성하고 소유할 수 있습니다. 용어 *콜렉션*을 *스키마*와 동의어로 사용할 수 있습니다.

자료 사전

스키마가 버전 3 릴리스 1 이전에서 작성되었거나 CREATE SCHEMA문에 WITH DATA DICTIONARY 절이 지정된 경우, 스키마에 자료 사전이 포함됩니다. **자료 사전**은 오브젝트 정의를 포함하는 표 세트입니다. SQL이 사전을 작성하고 나면 사전은 자동으로 시스템에 의해 유지보수됩니다. 사용자는 OS/400 프로그램의 일부인 대화식 자료 정의 유틸리티(IDDU)를 사용하여 자료 사전에 대해 작업할 수 있습니다. IDDU에 대한 자세한 정보는 IDDU 사용  을 참조하십시오.

저널과 저널 리시버

저널과 **저널 리시버**는 데이터베이스 내의 표와 보기에 대한 변경을 기록하는데 사용됩니다. 그런 다음, 저널 및 저널 리시버는 SQL COMMIT, ROLLBACK, SAVEPOINT 및 RELEASE SAVEPOINT문 처리에 사용됩니다. 저널과 저널 리시버는 또한 감사

추적으로 사용되거나 정방향 또는 역방향의 회복을 위해서도 사용될 수 있습니다. 저널링에 대한 자세한 정보는 저널링 주제를 참조하십시오. 확약 제어에 대한 자세한 정보는 확약 제어 주제를 참조하십시오.

카탈로그

SQL **카탈로그**는 표, 보기, 색인, 패키지, 프로시듀어, 파일, 트리거 및 제한사항을 설명하는 표와 보기 세트로 구성됩니다. 이 정보는 라이브러리 QSYS와 QSYS2에 있는 일련의 상호 참조표에 포함됩니다. 각 SQL 스키마에는 스키마의 표, 보기, 색인, 패키지, 파일 및 제한사항에 대한 정보가 들어 있는 카탈로그 표에 대해 작성된 일련의 보기가 있습니다.

카탈로그는 스키마를 작성할 때 자동으로 작성됩니다. 카탈로그를 제거하거나 명시적으로 변경하는 것은 불가능합니다.

SQL 카탈로그에 대한 자세한 내용은 SQL 참조서 책에서 카탈로그 주제를 참조하십시오.

표, 행 및 열

표는 **행**과 **열**로 구성된 자료의 2차원 배열입니다. 행은 하나 이상의 열이 들어 있는 수평 부분입니다. 열은 한 가지 유형의 자료를 한 행 이상 포함하고 있는 수직 부분입니다. 한 열의 모든 자료는 동일한 유형이어야 합니다. SQL에서 표는 키가 있거나 키가 없는 실제 파일입니다. 자료 유형에 대한 설명은 SQL 참조서의 자료 유형 주제를 참조하십시오.

표의 자료는 여러 서버에 분산시킬 수 있습니다. 분산표에 대한 자세한 내용은 DB2 Multisystem을 참조하십시오.

별명

별명은 표 또는 보기에 대한 대체명입니다. 기존 표나 보기를 참조할 수 있는 경우에 별명을 사용하여 표 또는 보기를 참조할 수 있습니다. 추가로, 별명은 표 멤버를 결합하는 데 사용될 수 있습니다. 별명에 대한 자세한 정보는 SQL 참조서의 별명 주제를 참조하십시오.

보기

보기는 어플리케이션 프로그램에 대한 표와 유사하나 자료를 가지지 않습니다. 이는 하나 이상의 표로부터 작성됩니다. 보기에는 주어진 표의 모든 열이나 서브세트의 일부가 포함될 수 있으며 주어진 표의 모든 행이나 서브세트 일부가 포함될 수 있습니다. 열은 지정된 표 내에 있는 것과는 다르게 보기 내에 배열될 수 있습니다. SQL에서 보기는 키가 없는 논리 파일의 특수 형식입니다.

보기에 대한 자세한 내용은 iSeries Information Center에서 *SQL 참조서* 책의 보기를 참조하십시오.

색인

SQL **색인**은 오름차순 또는 내림차순으로 논리적으로 배열된 표 열에 있는 자료의 서브세트입니다. 각 색인은 개별적으로 배열되어 있습니다. 이러한 배열은 순서화(ORDER BY절), 그룹화(GROUP BY절) 및 결합을 위해 사용됩니다. SQL 색인은 키가 있는 논리 파일입니다.

색인은 보다 신속한 자료 검색을 위해 시스템에 의해 사용됩니다. 색인 작성은 선택 사항입니다. 색인을 임의 수 만큼 작성할 수 있습니다. 언제라도 색인을 작성하거나 제거할 수 있습니다. 색인은 시스템에 의해 자동적으로 유지보수됩니다. 그러나 색인이 시스템에 의해 유지보수되므로 많은 수의 색인은 표를 변경하는 어플리케이션 수행에 나쁜 영향을 미칠 수도 있습니다.

효과적인 색인 코드화에 대한 자세한 내용은 iSeries Information Center에서 *데이터베이스 성능 및 조회 최적화* 책의 큰 표에 빠르게 액세스하기 위한 색인 사용 주제를 참조하십시오.

제한조건

제한조건은 데이터베이스 관리자에 의해 강제되는 규칙입니다. iSeries용 DB2 UDB에서는 다음의 제한조건을 사용할 수 있습니다.

- 고유 제한조건

  고유 제한조건은 키 값이 고유한 경우에만 유효하다는 규칙입니다. CREATE TABLE 및 ALTER TABLE문을 사용하여 고유 제한조건을 작성할 수 있습니다. 또한 CREATE INDEX는 고유성을 보장하는 고유 색인도 작성할 수 있지만, 이와 같은 색인은 제한조건이 아닙니다.

  고유 제한조건은 INSERT 및 UPDATE문의 실행시 강제됩니다. PRIMARY KEY 제한조건은 UNIQUE 제한조건의 한 형식입니다. PRIMARY KEY는 어떠한 널 가능 열도 포함할 수 없다는 점이 다릅니다.

- 참조 제한조건(referential constraints)

  참조 제한조건은 다음의 경우에만 외부 키의 값이 유효하다는 규칙입니다.

  – 외부 키의 값이 상위 키의 값으로 나타날 때 또는

  – 외부 키의 일부 구성요소가 널(null)일 때

  참조 제한조건은 INSERT, UPDATE 및 DELETE문의 실행시 강제됩니다.

- 체크 제한조건(Check constraints)

  체크 제한조건은 열 또는 열 그룹에 허용되는 값을 제한하는 규칙입니다. 체크 제한조건은 CREATE TABLE 및 ALTER TABLE문을 사용하여 추가할 수 있습니다. 체크 제한조건은 INSERT 및 UPDATE문의 실행시 강제됩니다. 제한조건을 만족시키려면 표에서 삽입 또는 갱신된 각 행이 TRUE 또는 알 수 없음(널 값으로 인해)으로 지정되어야 합니다.

제한사항에 대한 자세한 내용은 제 10 장『자료 무결성』을 참조하십시오.

트리거

트리거는 지정 이벤트가 지정된 기본표에 대해 발생할 때마다 자동으로 실행되는 조치의 세트입니다. 이벤트는 삽입,갱신,삭제 또는 읽기 조작이 될 수 있습니다. 트리거는 이벤트 이전 또는 이후에 수행될 수 있습니다. iSeries용 DB2 UDB에서는 SQL 삽입, 갱신 및 삭제 트리거와 외부 트리거를 지원합니다. 트리거에 대한 자세한 정보는 이 책의 제 10 장『자료 무결성』을 참조하거나 *데이터베이스 프로그래밍*의 데이터베이스에서 자동 이벤트 트리거 주제를 참조하십시오.

저장 프로시듀어

저장 프로시듀어는 SQL CALL문을 사용하여 호출될 수 있는 프로그램입니다. iSeries용 DB2 UDB는 외부 저장 프로시듀어와 SQL 프로시듀어를 지원합니다. 외부 저장 프로시듀어는 모든 시스템 프로그램 또는 REXX 프로시듀어가 될 수 있습니다. 그러나 System/36 프로그램이나 프로시듀어, 또는 서비스 프로그램은 될 수 없습니다. SQL 프로시듀어는 전체적으로 SQL에 정의되며 SQL 제어문이 들어 있는 SQL을 포함할 수 있습니다. 저장 프로시듀어에 대한 자세한 내용은 이 책에서 제 11 장『저장 프로시듀어』주제를 참조하십시오.

사용자 정의 기능

사용자 정의 기능은 내장 기능과 같이 호출될 수 있는 프로그램입니다. iSeries용 DB2 UDB는 외부 기능, SQL 기능 및 소스 기능을 지원합니다. 외부 기능은 모든 시스템 ILE 프로그램이나 서비스 프로그램이 될 수 있습니다. SQL 기능은 전체적으로 SQL로 정의되며 SQL 제어문을 포함하여 SQL문이 있을 수 있습니다. 소스 기능은 내장되거나 기존의 사용자 정의 기능을 통해 구축됩니다. SQL 또는 외부 기능 중 하나로 스칼라 함수 또는 표 함수를 작성할 수 있습니다. 사용자 정의 기능에 대한 자세한 정보는 259 페이지의 제 13 장『사용자 정의 기능(UDF) 작성』을 참조하십시오.

사용자 정의 유형

사용자 정의된 유형은 사용자가 데이터베이스 관리 시스템에 의해 공급된 자료 유형과 별개로 정의할 수 있는 고유한 자료 유형입니다. 기존 데이터베이스 유형에 대한 1대1 기초의 고유한 자료 유형 맵 사용자 정의 유형에 대한 자세한 정보는 240 페이지의『사용자 정의된 고유한 유형(UDT)』을 참조하십시오.

SQL 패키지

SQL 패키지는 어플리케이션 프로그램 내의 SQL문이 리모트 관계형 데이터베이스 관리 시스템(DBMS)에 결합될 때 생성된 제어 구조를 포함하는 오브젝트입니다. DBMS는 그 제어 구조를 사용해 어플리케이션 프로그램 수행중에 발생된 SQL문을 처리합니다.

SQL 패키지는 관계형 데이터베이스명(RDB 매개변수)이 SQL 작성(CRTSQLxxx) 명령에 지정될 때와 프로그램 오브젝트가 작성될 때 작성됩니다. 또한 패키지는 CRTSQLPKG 명령을 사용하여 작성될 수도 있습니다. 패키지와 분산 관계형 데이터베이스에 대해 자세히 알려면 제 21 장 『분산 관계형 데이터베이스 기능』을 참조하십시오

SQL 패키지는 QSQPRCED API를 사용하여 작성될 수도 있습니다. 이 책에서 언급된 SQL 패키지는 분산 프로그램 SQL 패키지만을 의미합니다. QSQPRCED은 확장 동적 SQL 지원을 제공하는 데 SQL 패키지를 사용합니다. QSQPRCED에 대한 자세한 정보는 iSeries Information Center의 OS/400 API 절에 있는 QSQPRCED 주제를 참조하십시오.

주: 이 명령에서 xxx는 호스트 언어 인디케이터를 나타내는데, ILE C 언어의 경우 CI, iSeries용 ILE C++ 언어의 경우 CPPI, iSeries용 COBOL 언어의 경우 CBL, ILE COBOL 언어의 경우 CBLI, iSeries PL/I 언어의 경우 PLI, iSeries용 RPG 언어의 경우 RPG 및 ILE RPG 언어의 경우 RPGI를 나타냅니다.

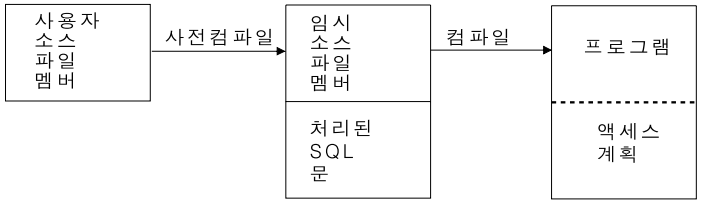
어플리케이션 프로그램 오브젝트

iSeries용 DB2 UDB 어플리케이션 프로그램의 작성 프로세스 결과로 여러 오브젝트가 작성될 수 있습니다. 이 섹션은 간략하게 iSeries용 DB2 UDB 어플리케이션 작성 프로세스에 대해 설명합니다. iSeries용 DB2 UDB는 비ILE 및 ILE 사전처리 컴파일러를 지원합니다. 어플리케이션 프로그램은 분산 프로그램일 수도 있고 비분산 프로그램일 수도 있습니다. iSeries용 DB2 UDB 어플리케이션 프로그램 작성에 대한 추가 내용은 *호스트 언어에 의한 SQL 프로그래밍* 에서 SQL문에 의한 SQL 프로그래밍 주제를 참조하십시오.

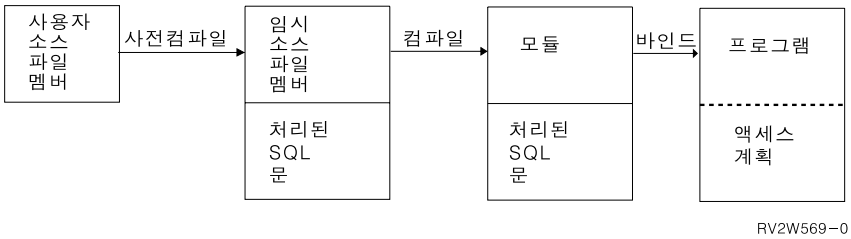
iSeries용 DB2 UDB를 사용할 경우 다음 오브젝트를 관리해야 합니다.

- 소스 원본
- 선택 항목으로서 ILE 프로그램에 대한 모듈 오브젝트
- 프로그램 또는 서비스 프로그램
- 분산 프로그램에 대한 SQL 패키지

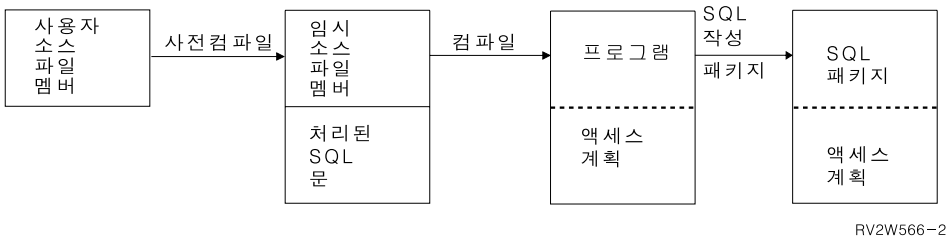
비분산 비ILE iSeries용 DB2 UDB 프로그램으로는 소스 원본과 결과 프로그램만을 관리해야 합니다. 다음은 비분산 비ILE iSeries용 DB2 UDB 프로그램에 대한 사전컴파일 및 컴파일 프로세스중에 발생하는 단계와 각각에 포함된 오브젝트를 보여 줍니다.



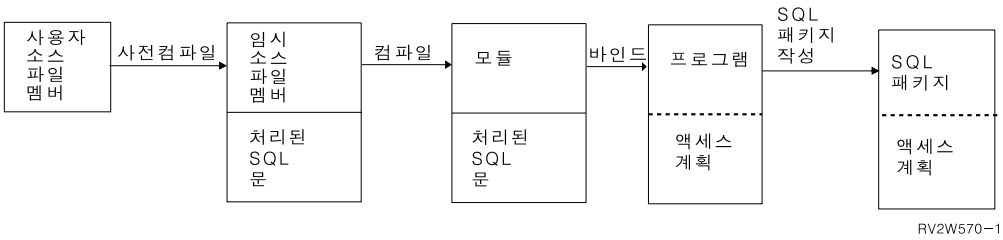
비분산 ILE iSeries용 DB2 UDB 프로그램으로 소스 원본, 모듈 및 결과 프로그램이나 서비스 프로그램을 관리해야 합니다. 다음은 사전컴파일 명령에 OBJTYPE(*PGM)이 지정되었을 때 비분산 ILE iSeries용 DB2 UDB 프로그램에 대한 사전컴파일 및 컴파일 처리시에 포함되는 오브젝트와 발생하는 단계를 보여줍니다.



분산 비ILE iSeries용 DB2 UDB 프로그램으로는 소스 원본, 결과 프로그램 및 결과 패키지를 관리해야 합니다. 다음은 분산 비ILE iSeries용 DB2 UDB 프로그램에 대한 사전컴파일 및 컴파일 처리시에 발생하는 오브젝트와 단계를 보여줍니다.



분산 ILE iSeries용 DB2 UDB 프로그램으로는 소스 원본, 모듈 오브젝트, 결과 프로그램이나 서비스 프로그램 및 결과 패키지를 관리해야 합니다. SQL 패키지는 분산 ILE 프로그램이나 서비스 프로그램에서 각각의 분산 모듈에 대해 작성될 수 있습니다. 다음은 분산 ILE iSeries용 DB2 UDB 프로그램에 대한 사전컴파일 및 컴파일 처리시에 발생하는 오브젝트와 단계를 보여줍니다.



주: iSeries용 DB2 UDB 분산 프로그램 오브젝트와 연관된 액세스 계획은 프로그램이 로컬로 수행될 때까지 작성되지 않습니다.

자세한 정보는 다음의 절을 참조하십시오.

- 『사용자 소스 파일 멤버』
- 『출력 소스 파일 멤버』
- 『프로그램』
- 14 페이지의『모듈』
- 15 페이지의『서비스 프로그램』

사용자 소스 파일 멤버

소스 파일 멤버에는 프로그래머의 어플리케이션 언어와 SQL문이 들어 있습니다. iSeries용 IBM WebSphere Development Studio 사용권 프로그램의 일부인 소스 입력 유틸리티(SEU)를 사용하여 소스 파일 멤버를 작성하고 유지보수할 수 있습니다.

출력 소스 파일 멤버

SQL 사전컴파일은 출력 소스 파일 멤버를 작성합니다. 디폴트로 사전컴파일 프로세스가 QTEMP에 임시 소스 파일 QSQLTxxxxx를 작성하거나 아니면 사용자가 사전컴파일 명령에 영구 파일명으로 출력 소스 파일을 지정할 수 있습니다. 사전컴파일 프로세스가 QTEMP 라이브러리를 사용한다면 시스템은 작업 완료시 자동으로 이 파일을 삭제합니다. 프로그램과 동일한 이름을 가진 멤버는 출력 소스 파일에 추가됩니다. 이 멤버에는 다음 항목들이 있습니다.

- SQL 런타임 지원에 대한 호출. 이는 삽입된 SQL문을 대체합니다.
- 분석되고 구문 검색된 SQL문

사전컴파일러는 디폴트로 호스트 언어 컴파일러를 호출합니다. 사전컴파일러에 대한 자세한 내용은 *호스트 언어를 사용한 SQL 프로그래밍* 정보에서 SQL문을 사용한 프로그램 준비 및 실행 주제를 참조하십시오.

프로그램

프로그램은 사용자가 수행할 수 있는 오브젝트로, 비ILE 컴파일의 경우에는 컴파일 처리의 결과로, ILE 컴파일의 경우에는 결합 처리의 결과로 작성됩니다.

액세스 계획은 삽입된 SQL문을 가장 효율적으로 수행하는 방법을 SQL에 알려주는 일련의 내부구조 및 정보입니다. 이는 프로그램이 제대로 작성된 경우에만 작성됩니다. 액세스 계획은 명령문이 다음과 같으면 SQL문에 대한 프로그램 작성중에 작성되지 않습니다.

- 찾을 수 없는 표 또는 보기를 참조할 경우
- 권한이 없는 사용자가 표 또는 보기를 참조할 경우

그러한 명령문에 대한 액세스 계획은 프로그램이 수행될 때 작성됩니다. 그 때에도 표나 보기를 찾을 수 없거나 권한이 없으면 다음의 SQLCODE가 리턴됩니다. 액세스 계획은 비분산 SQL 프로그램에 대한 프로그램 오브젝트와 분산 SQL 프로그램에 대한 SQL 패키지에서 저장되고 유지보수됩니다.

SQL 패키지

SQL 패키지에는 분산 SQL 프로그램에 대한 액세스 계획이 포함됩니다.

SQL 패키지는 다음의 경우에 작성되는 오브젝트입니다.

- CRTSQLxxx 명령의 RDB 매개변수를 사용하여 분산 SQL 프로그램이 정상적으로 작성될 경우
- CRTSQLPKG(SQL 패키지 작성) 명령이 수행될 경우

분산 SQL 프로그램 작성시 SQL 패키지명과 내부 일관성 토큰이 이 프로그램에 저장됩니다. 이들은 SQL 패키지를 찾고 SQL 패키지가 이 프로그램에 대해 올바른지 확인하기 위해 런타임에 사용됩니다. 분산 SQL 프로그램 수행중 SQL 패키지명은 매우 중요하므로 SQL 패키지에 다음과 같은 조작이 수행될 수 없습니다.

- 이동
- 이름 변경
- 복제
- 다른 라이브러리에 복원

모듈

모듈은 CRTxxxMOD 명령(또는 CRTBNDxxx 명령, 여기서 xxx는 C, CBL 또는 RPG)을 사용하여 소스 코드를 컴파일함으로써 작성된 ILE(Integrated Language Environment) 오브젝트입니다. CRTPGM(프로그램 작성) 명령을 사용하여 프로그램과 바인드하는 경우에만 모듈을 실행할 수 있습니다. 보통은 몇 개의 모듈을 서로 바인드하지만 여기서는 모듈 하나만을 바인드할 수 있습니다. 모듈은 SQL문에 관한 정보를 포함합니다. 그러나 SQL 액세스 계획은 모듈이 프로그램이나 서비스 프로그램으로 결합될 때까지는 작성되지 않습니다. CRTPGM(프로그램 작성)에 대한 자세한 정보는 명령 언어 주제의 CRTPGM(프로그램 작성) 부분을 참조하십시오.

서비스 프로그램

서비스 프로그램은 외부 자원 호출 가능 루틴(기능 또는 프로시듀어)을 별개의 오브젝트로 패키지하는 수단을 제공하는 통합 언어 환경(ILE) 오브젝트입니다. 결합된 프로그램 및 다른 서비스 프로그램은 서비스 프로그램에 의해 제공된 가져오기를 내보내기로 바꿈으로써 이 루틴에 액세스할 수 있습니다. 이와 같은 서비스로의 연결은 호출 프로그램이 작성되었을 때 이루어집니다. 이는 호출 프로그램 내에 코드를 첨가하지 않으면서도 이 루틴에 대한 호출 성능을 향상시킵니다.

제 2 장 SQL 시작

이 장에서는 대화식 SQL의 SQL문을 사용하여 스키마, 표 및 보기를 작성하고 이에 대해 작업하는 방법을 설명합니다.

이 장에서 사용된 SQL문 각각에 대한 구문은 SQL 참조서에 상세하게 설명되어 있습니다. 보다 복잡한 상태에서 SQL문과 절을 사용하는 방법에 대한 설명은 53 페이지의 제 4 장『DDL(Data Definition Language)』, 제 5 장『SELECT문을 사용하여 자료 검색』 및 제 6 장『SQL 삽입, 갱신 및 삭제』에서 제공합니다.

이 장에 나오는 예는 대화식 SQL 인터페이스를 사용하여 SQL문을 실행하는 것을 보여줍니다. 각각의 SQL 인터페이스에서는 표, 보기 및 기타 오브젝트를 정의하기 위해 SQL문을 사용하는 방법과 오브젝트로부터 자료를 읽는 방법을 제공합니다.

세부사항은 다음 주제를 참조하십시오.

- 『대화식 SQL 시작』
- 18 페이지의『스키마 작성』
- 18 페이지의『표 작성 및 사용』
- 21 페이지의『LABEL ON문 사용』
- 22 페이지의『표에 정보 삽입』
- 25 페이지의『단일 표에서 정보 가져오기』
- 28 페이지의『하나 이상의 표에서 정보 가져오기』
- 30 페이지의『표의 정보 변경』
- 33 페이지의『표에서 정보 삭제』
- 33 페이지의『보기 작성 및 사용』

주: 코드 예에 대한 정보는 x 페이지의『코드 면책사항 정보』를 참조하십시오.

대화식 SQL 시작

다음 예에 대해 대화식 SQL을 시작하려면 다음을 입력하십시오.

```
STRSQL NAMING(*SQL)
```

Enter 키를 누르십시오. SQL 입력 화면이 나타나면 SQL문을 입력할 준비가 된 것입니다. 대화식 SQL 및 STRSQL 명령에 대한 자세한 내용은 제 17 장『대화식 SQL 사용』을 참조하십시오.

기존의 대화식 SQL 세션을 재사용하는 경우에는 명명 모드를 반드시 *SQL 명명*으로 설정해야 합니다. 이것은 F13(서비스) 패널, 옵션 1(세션 속성 변경)에 지정될 수 있습니다.

스키마 작성

스키마는 표, 보기, 색인 및 패키지가 배치되는 기본 오브젝트입니다. 스키마 작성에 대한 자세한 내용은 SQL CREATE SCHEMA 명령문을 참조하십시오.

주: 용어 *콜렉션*을 *스키마*와 동의어로 사용할 수 있습니다.

대화식 SQL을 사용한 스키마 작성의 예는 『예: 스키마 작성(SAMPLECOLL)』을 참조하십시오.

예: 스키마 작성(SAMPLECOLL)

SQL문 입력 화면에서 다음의 SQL문을 입력하고 Enter 키를 눌러 이름이 SAMPLECOLL인 샘플 스키마를 작성할 수 있습니다.

```
                                SQL문 입력

SQL문을 입력한 후 Enter 키를 누르십시오.
     현재 연결은 관계형 데이터베이스 SYSTEM1에 연결되어 있습니다.
===> CREATE SCHEMA SAMPLECOLL_____________________________________________
     ____________________________________________________________________
     ____________________________________________________________________
     ____________________________________________________________________
                                                                      맨 아래
F3=나감    F4=프롬트    F6=행 삽입       F9=검색       F10=행 복사
F12=취소               F13=서비스       F24=추가 키
```

주: 이 명령문을 수행하면 여러 오브젝트가 작성되며 이 작업에는 약간의 시간이 소요됩니다.

스키마를 성공적으로 작성한 후에 그 안에 표, 보기 및 색인을 작성할 수 있습니다. 표, 보기 및 색인은 스키마 대신 라이브러리 내에 작성할 수도 있습니다.

표 작성 및 사용

SQL CREATE TABLE 명령문을 사용하여 표를 작성할 수 있습니다. CREATE TABLE문은 표를 작성하고 표에 있는 열의 실제 속성을 정의하며 표에서 허용되는 값을 제한하는 제한조건을 정의할 수 있게 합니다.

대화식 SQL를 사용하여 콜렉션을 작성하는 예에 대해서는 19 페이지의 『예: 표 작성(INVENTORY_LIST)』을 참조하십시오.

표를 작성할 때, 널 값과 디폴트 값의 개념을 알아야 합니다. 널 값은 그 행에 열값이 없다는 것을 나타냅니다. 그것은 제로 또는 모두 공백의 값이 아니고 "알 수 없음"을

의미합니다. 어느 값이나 다른 널 값과도 다릅니다. 어떤 열이 널 값을 허용하지 않으면 그 열에는 디폴트 값이나 사용자 제공값이 할당되어야 합니다.

디폴트 값은 표에 한 행이 추가되고 해당 열에 지정된 값이 없을 때 그 열에 할당됩니다. 어떤 열에 특정한 디폴트 값이 정의되지 않으면 시스템 디폴트 값이 사용됩니다. INSERT에 사용되는 디폴트 값에 대한 자세한 내용은 105 페이지의 『INSERT 명령문을 사용하여 열 삽입』을 참조하십시오.

예: 표 작성(INVENTORY_LIST)

이제 사업장의 현재 재고에 대한 정보를 유지보수하기 위한 표를 작성할 것입니다. 표에는 재고 품목, 원가, 현재 보유량, 최종 주문일 및 최종 주문 번호에 대한 정보가 포함됩니다. 품번은 필수값입니다. 이것은 널(null)이 될 수 없습니다. 품목명, 보유량 및 주문량은 사용자 제공 디폴트 값을 가집니다. 최종 주문일과 수량은 널 값을 허용합니다.

SQL 입력 화면에서 CREATE TABLE을 입력한 후 F4(프롬트) 키를 누르십시오. 다음 화면이 표시됩니다(입력 영역이 아직 비어있는 상태로 보여집니다).

```
CREATE TABLE문 지정

정보를 입력한 후 Enter 키를 누르십시오.

표 . . . . . .  INVENTORY_LIST_____     이름
   콜렉션    . . . . . .     SAMPLECOLL__          이름, 리스트의 경우 F4

널:  1=NULL, 2=NOT NULL, 3=NOT NULL WITH DEFAULT

열                  FOR열        유형             길이   스케일  널
ITEM_NUMBER_____    ___________  CHAR___________   6____  __     2
ITEM_NAME_______    ___________  VARCHAR________  20___  __     3
UNIT_COST_______    ___________  DECIMAL________  8____  2_     3
QUANTITY_ON_HAND__  ___________  SMALLINT_______  _____  __     1
LAST_ORDER_DATE___  ___________  DATE___________  _____  __     1
ORDER_QUANTITY____  ___________  SMALLINT_______  _____  __     1
__________________  ___________  _______________  _____  __     3
                                                                 맨 아래
  표 제한  . . . . . .  N     Y=예, N=아니오
  분산표 . . . . . .  N     Y=예, N=아니오

F3=나감   F4=프롬트   F5=화면정리  F6=행 삽입        F10=행 복사
F11=추가 속성 표시          F12=취소     F14=행 삭제       F24=추가 키
```

*표*와 *콜렉션* 프롬트에 대해 작성중인 표의 표 이름과 *스키마*명, INVENTORY_LIST와 SAMPLECOLL을 입력하십시오. 표에 대해 정의하려는 각각의 열은 화면 아래 리스트에 있는 항목에 의해 표현됩니다. 각각의 열에 대해 열 이름, 열의 자료 유형, 길이와 스케일 및 널 속성을 입력하십시오.

그 열에 지정될 수 있는 추가 속성을 보려면 F11 키를 누르십시오. 여기서는 디폴트 값이 지정될 수 있습니다.

```
CREATE TABLE문 지정

정보를 입력한 후 Enter 키를 누르십시오.

표 . . . . . .   INVENTORY_LIST_____     이름
  Collection . . . . . .    SAMPLECOLL__          Name, F4 for list

자료:  1=BIT, 2=SBCS, 3=MIXED, 4=CCSID

열                  자료   할당     CCSID  CONSTRAINT  디폴트
ITEM NUMBER_______    _    _____    _____      N       _________________
ITEM NAME_________    _    _____    _____      N       '***알 수 없음***'__
UNIT_COST_________    _    _____    _____      N       _________________
QUANTITY_ON_HAND__    _    _____    _____      N       NULL_____________
LAST_ORDER_DATE___    _    _____    _____      N       _________________
ORDER_QUANTITY____    _    _____    _____      N       20_______________
__________________    _    _____    _____      _       _________________
                                                                  맨 아래
  표 제한  . . . . . .  N      Y=예, N=아니오
   분산표  . . . . . .  N      Y=예, N=아니오

F3=나감   F4=프롬트   F5=화면정리  F6=행 삽입        F10=행 복사
F11=추가 속성 표시          F12=취소     F14=행 삭제       F24=추가 키
```

주: 열 정의를 입력하는 다른 방법은 커서를 리스트의 열 항목 중 하나에 놓고 F4(프롬트) 키를 누르는 것입니다. 이는 단일 열을 정의하기 위한 모든 속성을 보여주는 화면을 불러옵니다.

모든 값이 입력되고 나면 Enter 키를 눌러 표를 작성하십시오. SQL 입력 화면이 표가 작성되었음을 나타내는 메세지와 함께 다시 나타납니다.

다음과 같이 SQL 입력 화면에서 이 CREATE TABLE문으로 직접 입력할 수도 있습니다.

```
CREATE TABLE SAMPLECOLL.INVENTORY_LIST
     (ITEM_NUMBER CHAR(6) NOT NULL,
      ITEM_NAME VARCHAR(20) NOT NULL WITH DEFAULT '***UNKNOWN***',
      UNIT_COST DECIMAL(8,2) NOT NULL WITH DEFAULT,
      QUANTITY_ON_HAND SMALLINT DEFAULT NULL,
      LAST_ORDER_DATE DATE,
      ORDER_QUANTITY SMALLINT DEFAULT 20)
```

공급자 표(SUPPLIERS) 작성

이 예를 계속 진행하다 보면 2차 표도 필요하게 됩니다. 이 표는 재고 품목의 공급자, 공급 품목, 그리고 품목 원가에 관한 정보가 포함됩니다. 표를 작성하려면 SQL 입력 화면에서 직접 입력하거나 F4(프롬트) 키를 눌러 정의 작성을 위한 대화식 SQL 화면을 사용하십시오.

```
CREATE TABLE SAMPLECOLL.SUPPLIERS
     (SUPPLIER_NUMBER CHAR(4) NOT NULL,
     ITEM_NUMBER CHAR(6) NOT NULL,
     SUPPLIER_COST DECIMAL(8,2))
```

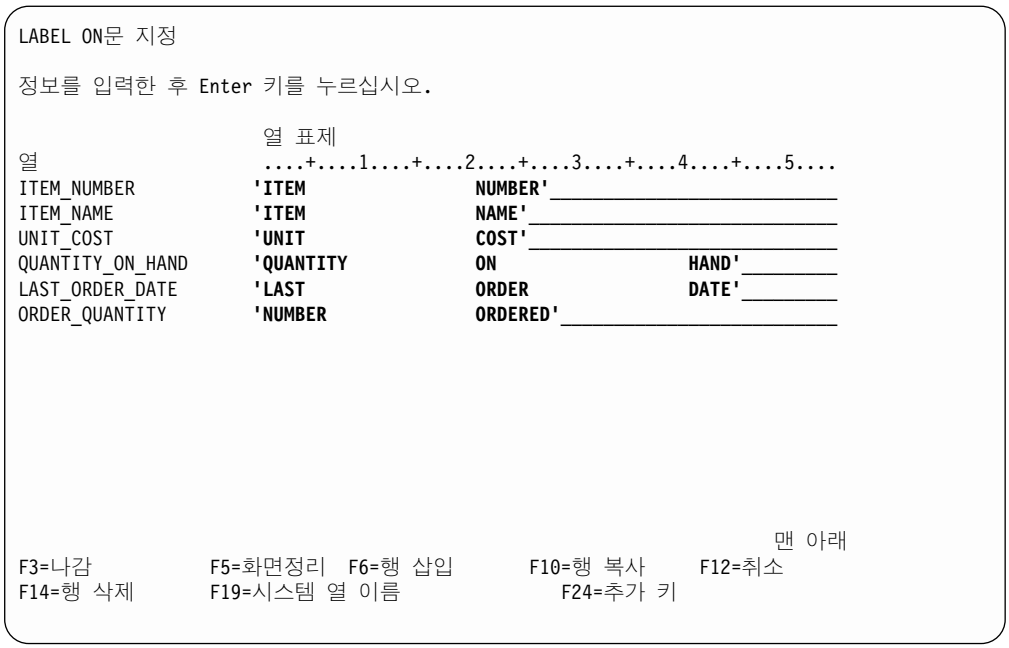
LABEL ON문 사용

일반적으로 열 이름은 대화식 SQL에서 SELECT문의 출력을 표시할 때 열 표제로 사용됩니다. LABEL ON문을 사용함으로써 표 이름, 열 이름, 보기명 또는 SQL 패키지 명에 대한 보다 자세한 레이블을 작성할 수 있습니다. 대화식 SQL에서 예제를 수행할 것이므로 열 표제를 변경하기 위해 LABEL ON문을 사용합니다. 열 이름이 서술적이라고 해도 열 표제가 단일 행에 이름의 각 부분을 표시한다면 읽기가 보다 쉬워집니다. 또한 그렇게 하면 하나의 화면에서 자료의 보다 많은 열을 볼 수 있습니다.

열에 대한 레이블을 변경하려면 SQL 입력 화면에서 LABEL ON COLUMN을 입력한 후 F4(프롬트) 키를 누르십시오. 다음 화면이 나타납니다.

```
LABEL ON문 지정

선택사항을 입력한 후 Enter 키를 누르십시오.

  Label on . . . .   2                        1=표 또는 보기
                                              2=열
                                              3=패키지
                                              4=별명

  표 또는 보기         INVENTORY_LIST_____      이름, 리스트의 경우 F4
    콜렉션 . . . .     SAMPLECOLL__           이름, 리스트의 경우 F4

  옵션 . . . . . .   1                        1=열 표제
                                              2=텍스트

F3=나감   F4=프롬트   F5=화면정리  F12=취소     F20=완전명 표시
F21=명령문 표시
```

레이블을 추가하려는 열이 들어 있는 표와 스키마의 이름을 입력한 후 Enter 키를 누르십시오. 다음 화면이 표시되며 사용자에게 표에 있는 각각의 열을 프롬트합니다.



각 열의 열 표제를 입력하십시오. 열 표제는 20개의 문자 섹션으로 정의됩니다. 각 섹션은 SELECT문의 출력이 표시될 때 서로 다른 행에 표시됩니다. 열 표제 입력 영역의 맨 위에 가로로 그려진 눈금자는 표제의 간격을 정확하고 쉽게 나누는데 사용될 수 있습니다. 표제를 입력한 후 Enter 키를 누르십시오.

다음의 메세지는 LABEL ON문이 성공적으로 수행되었음을 나타냅니다.

```
SAMPLECOLL의 INVEN00001에 대한 LABEL ON이 완료됨.
```

메세지의 표 이름은 실제로 명령문에 지정된 이름이 아니라 이 표의 시스템표 명입니다. iSeries용 DB2 UDB는 10자보다 긴 이름을 사용하여 표에 대한 두 개의 이름을 유지보수합니다. 시스템 표 이름에 대한 자세한 정보는 SQL 참조서의 CREATE TABLE문을 참조하십시오.

LABEL ON문은 또한 다음과 같이 SQL 입력 화면에서 직접 입력할 수도 있습니다.

```
LABEL ON SAMPLECOLL.INVENTORY_LIST
(ITEM_NUMBER      IS 'ITEM              NUMBER',
ITEM_NAME         IS 'ITEM              NAME',
UNIT_COST         IS 'UNIT              COST',
QUANTITY_ON_HAND  IS 'QUANTITY          ON                HAND',
LAST_ORDER_DATE   IS 'LAST              ORDER             DATE',
ORDER_QUANTITY    IS 'NUMBER            ORDERED')
```

표에 정보 삽입

표를 작성한 후에는 SQL INSERT 명령문을 사용하여 정보(자료)를 표에 삽입 또는 추가할 수 있습니다.

대화식 SQL를 사용하여 표에 자료를 삽입하는 예에 대해서는 『예: 표에 정보 삽입 (INVENTORY_LIST)』을 참조하십시오.

예: 표에 정보 삽입(INVENTORY_LIST)

SQL 입력 화면에서 대화식 SQL에 대해 작업하려면, INSERT를 입력한 후 F4(프롬트) 키를 누르십시오. INSERT문 지정 화면이 나타납니다.

```
INSERT문 지정

선택사항을 입력한 후 Enter 키를 누르십시오.

  INTO 표  . . . . . .    INVENTORY_LIST_____     이름, 리스트의 경우 F4
    콜렉션     . . . . . .     SAMPLECOLL__        이름, 리스트의 경우 F4

  INTO를 삽입할 열
    선택  . . . . . .  Y                        Y=예, N=아니오
  삽입 방식 . . . . . .  1                      1=VALUES 입력
                                                2=부속 선택

선택사항을 입력한 후 Enter 키를 누르십시오.

  WITH 분리 레벨  . . . . .  1              1=현재 레벨, 2=NC (NONE)
                                            3=UR (CHG), 4=CS, 5=RS (ALL)
                                            6=RR



F3=나감   F4=프롬트   F5=화면정리  F12=취소     F20=완전명 표시
F21=명령문 표시
```

표시된 바와 같이 입력 필드에 표 이름과 스키마명을 입력하십시오. *INTO를 삽입할 열 선택* 프롬트를 예로 변경하십시오. 값을 삽입할 열을 선택할 수 있는 화면을 보려면 Enter 키를 누르십시오.

```
INSERT문 지정

선택하기 위한 순서 번호(1-999)를 입력한 후 Enter 키를 누르십시오.

Seq  열                  유형                길이  소수 자릿수
1__  ITEM_NUMBER         CHARACTER             6
2__  ITEM_NAME           VARCHAR              20
3__  UNIT_COST           DECIMAL               8    2
4__  QUANTITY_ON_HAND    SMALLINT              4
___  LAST_ORDER_DATE     DATE
___  ORDER_QUANTITY      SMALLINT      4

                                                                  맨 아래
F3=나감   F5=화면정리     F12=취소     F19=시스템 열 이름 표시
F20=완전명 표시           F21=명령문 표시
```

이 예에서는 4개의 열에만 삽입할 것입니다. 다른 열은 디폴트 값이 삽입되도록 할 것입니다. 이 화면의 순서 번호는 열과 그 값이 INSERT문에서 나열되는 순서를 나타냅니다. 선택된 열 값을 입력할 수 있는 화면을 표시하려면 Enter 키를 누르십시오.

```
INSERT문 지정

삽입할 값을 입력한 후 Enter 키를 누르십시오.

열                    값
ITEM_NUMBER           '153047'_____________________________________________
ITEM_NAME             'Pencils, red'_______________________________________
UNIT_COST             10.00________________________________________________
QUANTITY_ON_HAND      25___________________________________________________

                                                                  맨 아래
F3=나감      F5=화면정리  F6=행 삽입       F10=행 복사     F11=유형 표시
F12=취소     F14=행 삭제  F15=행 분할      F24=추가 키
```

주: 삽입 리스트에 있는 각 열의 자료 유형과 길이를 보려면 F11(유형 표시) 키를 누르십시오. 열 정의에 관한 정보를 제공하는 값을 삽입하는 화면의 다른 보기가 표시됩니다.

모든 열에 삽입될 값을 입력한 후 Enter 키를 누르십시오. 이 값을 포함하는 행이 표에 추가됩니다. 지정되지 않은 열 값에는 디폴트 값이 삽입됩니다.
LAST_ORDER_DATE의 경우, 디폴트가 제공되지 않고 그 열이 널 값을 허용하므로 널 값이 삽입됩니다. ORDER_QUANTITY의 경우, CREATE TABLE문의 디폴트 값으로 지정된 값 20이 삽입됩니다.

INSERT문은 다음과 같이 SQL 입력 화면에서 입력될 수도 있습니다.

```
INSERT INTO SAMPLECOLL.INVENTORY_LIST
                  (ITEM_NUMBER,
                   ITEM_NAME,
                   UNIT_COST,
                   QUANTITY_ON_HAND)
           VALUES('153047',
                  'Pencils, red',
                   10.00,
                   25)
```

표에 다음 행을 추가하려면 SQL 입력 화면에서 F9(검색) 키를 누르십시오. 그러면 이전의 INSERT문이 입력 영역에 복사됩니다. 이전 INSERT문의 값에 겹쳐 입력하거나 F4(프롬트) 키를 눌러 대화식 SQL 화면을 사용하여 자료를 입력할 수 있습니다.

INSERT문의 사용을 계속하여 다음 행을 표에 추가하십시오. 아래의 도표에 표시되지 않은 값은 삽입될 수 없으므로 디폴트가 사용됩니다. INSERT문 열 리스트에 값을 삽입하려는 열 이름만을 지정하십시오. 예를 들어 세 번째 행을 삽입하려면 열 이름에 대해 ITEM_NUMBER와 UNIT_COST만을 지정하고 이들 열에 대한 두 개의 값만을

VALUES 리스트에 지정합니다.

ITEM_NUMBER	ITEM_NAME	UNIT_COST	QUANTITY_ON_HAND
153047	Pencils, red	10.00	25
229740	Lined tablets	1.50	120
544931		5.00	
303476	Paper clips	2.00	100
559343	Envelopes, legal	3.00	500
291124	Envelopes, standard		
775298	Chairs, secretary	225.00	6
073956	Pens, black	20.00	25

다음 행을 SAMPLECOLL.SUPPLIERS 표에 추가하십시오.

SUPPLIER_NUMBER	ITEM_NUMBER	SUPPLIER_COST
1234	153047	10.00
1234	229740	1.00
1234	303476	3.00
9988	153047	8.00
9988	559343	3.00
2424	153047	9.00
2424	303476	2.50
5546	775298	225.00
3366	303476	1.50
3366	073956	17.00

이제 샘플 스키마에는 각각 여러 개의 자료 행이 있는 두 개의 표가 들어 있습니다.

단일 표에서 정보 가져오기

이제 표에 정보를 모두 삽입했으므로 다시 검토해야 합니다. SQL에서는 SELECT문으로 이를 수행합니다. SELECT문은 모든 SQL문 중 가장 복합적입니다. 이 명령문은 세 개의 기본 절로 구성됩니다.

1. 요구된 자료를 포함하기 위해 열들을 지정하는 SELECT절
2. 요구된 자료를 가진 열을 포함하기 위해 표를 지정하는 FROM절
3. 검색되는 자료의 행을 결정하기 위해 조건을 제공하는 WHERE절

세 개의 기본 절 외에도 리턴 자료의 최종 양식에 영향을 주는 다른 여러 절이 69 페이지의 제 5 장『SELECT문을 사용하여 자료 검색』및 SQL 참조서에 설명되어 있습니다.

INVENTORY_LIST 표에 삽입할 값을 보려면 SELECT를 입력한 후 F4(프롬트) 키를 누르십시오. 다음 화면이 표시됩니다.

```
                              SELECT문 지정

SELECT문 정보를 입력한 후 리스트를 보려면 F4를 누르십시오.

  FROM 표  . . . . . .  SAMPLECOLL.INVENTORY_LIST_____________________
  SELECT 열    . . . . . .  *_____________________________________________
  WHERE 조건 . . . . . .  _____________________________________________
  GROUP BY열 . . . . . .  _____________________________________________
  HAVING 조건  . . . . . .  _____________________________________________
  ORDER BY 열  . . . . . .  _____________________________________________
  FOR UPDATE OF 열   . . . . .  _____________________________________________

                                                                  맨 아래
선택사항을 입력한 후 Enter 키를 누르십시오.

  결과 표의 DISTINCT 행  . . . . . .  N    Y=예, N=아니오
  다른 SELECT와 UNION  . . . . . .  N    Y=예, N=아니오
  추가 옵션 지정 . . . . . .  N    Y=예, N=아니오



F3=나감         F4=프롬트   F5=화면정리    F6=행 삽입     F9=부속 조회 지정
F10=행 복사     F12=취소    F14=행 삭제    F15=행 분할    F24=추가 키
```

표 이름을 화면의 *FROM 표* 필드에 입력하십시오. 표에서 모든 열을 선택하려면, *를 화면의 *SELECT 열* 필드에 입력하십시오. Enter 키를 누르면 표에 있는 열에 대한 모든 자료를 선택하는 명령문이 수행됩니다. 다음 출력이 표시됩니다.

```
                              자료 표시
                                          자료 폭    . . . . . :     71
위치 행 . . . . . .                 열로 이동  . . . . . .
....+....1....+....2....+....3....+....4....+....5....+....6....+....7.
ITEM    ITEM                      UNIT   QUANTITY  LAST      NUMBER
NUMBER  NAME                      COST   ON        ORDER     ORDERED
                                         HAND      DATE
153047  Pencils, red             10.00       25   -              20
229740  Lined tablets             1.50      120   -              20
544931  ***UNKNOWN***             5.00        -   -              20
303476  Paper clips               2.00      100   -              20
559343  Envelopes, legal          3.00      500   -              20
291124  Envelopes, standard        .00        -   -              20
775298  Chairs, secretary        225.00        6   -              20
073956  Pens, black               20.00       25   -              20
********   자료의 끝   ********

F3=나감      F12=취소        F19=왼쪽      F20=오른쪽     F21=분할
```

LABEL ON문을 사용하여 정의된 열 표제가 표시됩니다. 세 번째 항목의 ITEM_NAME은 CREATE TABLE문에 지정된 디폴트 값을 가집니다. QUANTITY_ON_HAND 열은 값이 삽입되지 않은 행에 대한 널 값을 가집니다. *LAST_ORDER_DATE* 열에는 이 열이 INSERT문 중 어느 곳에도 존재하지 않으며 디폴트 값을 가지도록 정의되지 않았기 때문에 모두 널 값이 포함됩니다. 마찬가지로 ORDER_QUANTITY 열에는 모든 행에 디폴트 값이 포함됩니다.

이 명령문은 다음과 같이 SQL 입력 화면에서 입력될 수도 있습니다.

```
SELECT *
       FROM SAMPLECOLL.INVENTORY_LIST
```

SELECT문에 의해 리턴되는 열의 수를 제한하려면 사용자가 보려는 열이 지정되어야 합니다. 리턴되는 출력 행의 수를 제한하는데에는 WHERE절이 사용됩니다. 원가가 10 달러를 초과하는 품목만을 보고 ITEM_NUMBER, UNIT_COST 및 ITEM_NAME 열의 값만을 리턴시키려면 SELECT를 입력한 후 F4(프롬트) 키를 누르십시오. SELECT 문 지정 화면이 나타납니다.

```
                              SELECT문 지정

SELECT문 정보를 입력한 후 리스트를 보려면 F4를 누르십시오.

  FROM 표  . . . . . .   SAMPLECOLL.INVENTORY_LIST_____________________
  SELECT 열   . . . . .  ITEM_NUMBER, UNIT_COST, ITEM_NAME_____________
  WHERE 조건 . . . . . .  UNIT_COST > 10.00_____________________________
  GROUP BY열 . . . . . .  ______________________________________________
  HAVING 조건  . . . . .  ______________________________________________
  ORDER BY 열  . . . . .  ______________________________________________
  FOR UPDATE OF 열  . . . .  ___________________________________________

                                                                  맨 아래
선택사항을 입력한 후 Enter 키를 누르십시오.

  결과 표의 DISTINCT 행  . . . . . .  N    Y=예, N=아니오
  다른 SELECT와 UNION  . . . . . .  N    Y=예, N=아니오
  추가 옵션 지정 . . . . . .  N    Y=예, N=아니오



F3=나감        F4=프롬트   F5=화면정리   F6=행 삽입    F9=부속 조회 지정
F10=행 복사    F12=취소    F14=행 삭제   F15=행 분할   F24=추가 키
```

SELECT문 지정 화면의 각 프롬트에 대해 초기에는 한 행만 표시되지만 화면 맨 윗부분의 입력 영역에 행을 추가하기 위해서는 F6(행 삽입) 키를 사용할 수 있습니다. 이는 여러 열이 *SELECT* 열 리스트에 입력되거나 더 길고 보다 복잡한 WHERE 조건이 요구될 경우 사용될 수 있습니다.

위에 표시된 대로 화면을 채워 넣으십시오. Enter 키를 누르면 SELECT문이 수행됩니다. 다음 출력이 표시됩니다.

```
                              자료 표시
                                        자료 폭 . . . . . :     41
위치 행 . . . . . .            열로 이동  . . . . . .
....+....1....+....2....+....3....+....4.
ITEM          UNIT   ITEM
NUMBER        COST   NAME
775298      225.00   Chairs, secretary
073956       20.00   Pens, black
********   자료의 끝   ********

F3=나감      F12=취소        F19=왼쪽      F20=오른쪽     F21=분할
```

리턴되는 행만이 그 자료의 값이 WHERE절에 지정된 조건과 비교되는 행입니다. 또한 리턴된 자료값은 SELECT절에서 명시적으로 지정한 열로부터 리턴됩니다. 이와 같이 명시적으로 식별된 것 이외의 열 자료값은 리턴되지 않습니다.

이 명령문은 다음과 같이 SQL 입력 화면에서 입력될 수도 있습니다.

```
SELECT ITEM_NUMBER, UNIT_COST, ITEM_NAME
         FROM SAMPLECOLL.INVENTORY_LIST
  WHERE UNIT_COST > 10.00
```

하나 이상의 표에서 정보 가져오기

SQL에서는 하나 이상의 표에 포함된 열로부터 정보를 얻을 수 있습니다. 이 조작을 *결합*이라 합니다(결합 조작에 대해 자세히 알려면, 88 페이지의 『하나 이상의 표로부터 자료 결합』 참조). SQL에서 결합 조작은 사용자가 SELECT문의 동일한 FROM절에 함께 결합시키려는 표 이름을 배치함으로써 지정됩니다.

사용자가 모든 공급자, 품목 번호 및 그들이 공급하는 품목명의 리스트를 보고자 한다고 가정합니다. 그 품목명은 SUPPLIERS 표에 없습니다. 그것은 INVENTORY_LIST 표에 있습니다. 일반 열, ITEM_NUMBER를 사용하여 단일 표에 있었던 것처럼 세 개의 열 모두를 볼 수 있습니다.

결합된 둘 이상의 표에 동일한 열 이름이 존재할 때마다 열 이름은 실제로 참조될 열을 지정하는 표 이름에 의해 규정되어야 합니다. 이 SELECT문에서 열 이름이 표 이름에 의해 규정될 표 모두에 열 이름 ITEM_NUMBER가 정의됩니다. 열이 다른 이름을 가질 경우에는 혼동할 염려가 없으므로 규정화 조작이 필요없습니다.

이 결합을 수행하기 위해 다음의 SELECT문이 사용될 수 있습니다. SQL 입력 화면에서 직접 입력하거나 프롬트하여 입력하십시오. 프롬트를 사용하는 경우에는 두 표 이름 모두가 *FROM 표* 입력 행에 입력되어야 합니다.

```
SELECT SUPPLIER_NUMBER, SAMPLECOLL.INVENTORY_LIST.ITEM_NUMBER, ITEM_NAME
  FROM SAMPLECOLL.SUPPLIERS, SAMPLECOLL.INVENTORY_LIST
  WHERE SAMPLECOLL.SUPPLIERS.ITEM_NUMBER
                    = SAMPLECOLL.INVENTORY_LIST.ITEM_NUMBER
```

동일한 명령문을 입력하는 또 하나의 방법은 상관명을 사용하는 것입니다. 상관명은 명령문에 사용할 표 이름의 다른 이름을 제공합니다. 상관명은 표 이름이 동일할 때 사용해야 합니다. 그것은 FROM 리스트에 있는 각각의 표 이름 다음에 지정될 수 있습니다. 이전 명령문을 다음과 같이 다시 쓸 수 있습니다.

```
SELECT SUPPLIER_NUMBER, Y.ITEM_NUMBER, ITEM_NAME
  FROM SAMPLECOLL.SUPPLIERS X, SAMPLECOLL.INVENTORY_LIST Y
  WHERE X.ITEM_NUMBER = Y.ITEM_NUMBER
```

이 예에서는 SAMPLECOLL.SUPPLIERS가 X의 상관명으로 제공되고 SAMPLECOLL.INVENTORY_LIST가 Y의 상관명으로 제공됩니다. X와 Y는 ITEM_NUMBER 열 이름을 규정하는데 사용됩니다.

열 및 상관명에 대한 자세한 내용은 iSeries Information Center에서 *SQL 참조서*의 상관명을 참조하십시오.

이 예를 수행하면 다음 출력이 리턴됩니다.

```
                              자료 표시
                                                 자료 폭  . . . . . :     45
위치 행 . . . . . .                            열로 이동  . . . . . . . .
....+....1....+....2....+....3....+....4....+
SUPPLIER_NUMBER  ITEM    ITEM
                 NUMBER  NAME
     1234        153047  Pencils, red
     1234        229740  Lined tablets
     1234        303476  Paper clips
     9988        153047  Pencils, red
     9988        559343  Envelopes, legal
     2424        153047  Pencils, red
     2424        303476  Paper clips
     5546        775298  Chairs, secretary
     3366        303476  Paper clips
     3366        073956  Pens, black
********   자료의 끝   ********

F3=나감      F12=취소        F19=왼쪽      F20=오른쪽     F21=분할
```

주: 이 조회에 대하여 ORDER BY 절이 지정되지 않았으므로 조회에 의해 리턴된 행 순서는 다를 수 있습니다.

결과 표의 자료값은 두 개의 표 INVENTORY_LIST와 SUPPLIERS에 포함된 자료값의 조합을 표시합니다. 이 결과 표는 SUPPLIER 표로부터의 공급자 번호와 INVENTORY_LIST 표로부터의 품번과 품목명을 포함합니다. SUPPLIER 표에 나타나지 않는 품번과 이 결과 표에 표시되지 않습니다. 결과는 SELECT문에 대해 ORDER BY절이 지정되지 않는 한 어떤 순서로의 배열을 보장하지 않습니다. SUPPLIER 표에 대한 열 표제를 변경하지 않았으므로 SUPPLIER_NUMBER 열 이름은 열 표제로 사용됩니다.

다음은 행의 순서를 보장하기 위해 ORDER BY를 사용하는 예입니다. 명령문은 먼저 SUPPLIER_NUMBER 열에 의해 결과 표를 순서화합니다. SUPPLIER_NUMBER에 대해 동일한 값을 갖는 행은 그것의 ITEM_NUMBER에 의해 순서화됩니다.

```
SELECT SUPPLIER_NUMBER, Y.ITEM_NUMBER, ITEM_NAME
  FROM SAMPLECOLL.SUPPLIERS X, SAMPLECOLL.INVENTORY_LIST Y
  WHERE X.ITEM_NUMBER = Y.ITEM_NUMBER
  ORDER BY SUPPLIER_NUMBER, Y.ITEM_NUMBER
```

이전의 명령문을 수행하면 다음의 출력이 생성됩니다.

```
                              자료 표시
                                                 자료 폭    . . . . . :     45
위치 행 . . . . . .                          열로 이동  . . . . . .
....+....1....+....2....+....3....+....4....+
SUPPLIER_NUMBER  ITEM    ITEM
                 NUMBER  NAME
     1234        153047  Pencils, red
     1234        229740  Lined tablets
     1234        303476  Paper clips
     2424        153047  Pencils, red
     2424        303476  Paper clips
     3366        073956  Pens, black
     3366        303476  Paper clips
     5546        775298  Chairs, secretary
     9988        153047  Pencils, red
     9988        559343  Envelopes, legal
********   자료의 끝   ********

F3=나감       F12=취소        F19=왼쪽       F20=오른쪽      F21=분할
```

표의 정보 변경

UPDATE문을 사용하여 표의 열 중 일부 또는 모두에 있는 자료 값을 변경할 수 있습니다.

대화식 SQL를 사용하여 표에 있는 정보를 변경하는 예에 대해서는 『예: 표의 정보 변경』을 참조하십시오.

단일 명령문이 실행되는 동안 변경될 행의 수를 제한하려면, UPDATE문과 함께 WHERE절을 사용하십시오. 추가 정보는 110 페이지의 『UPDATE 명령문을 사용하여 표의 자료 변경』을 참조하십시오. WHERE절을 지정하지 않으면 지정된 표에 있는 모든 행이 변경됩니다. 그러나, WHERE절을 사용하면, 시스템은 사용자가 지정하는 조건을 만족하는 행만 변경합니다. 추가 정보는 71 페이지의 『WHERE 절을 사용하는 탐색 조건 지정』을 참조하십시오.

예: 표의 정보 변경

대화식 SQL을 사용하고 오늘 추가로 종이 클립 주문을 한다고 가정할 경우, 항목 번호 303476에 대한 LAST_ORDER_DATE 및 ORDER_QUANTITY를 갱신하려면 UPDATE를 입력하고 F4(프롬트)를 입력하십시오. UPDATE문 지정 화면이 나타납니다.

```
UPDATE문 지정

선택사항을 입력한 후 Enter 키를 누르십시오.

  표 . . . . . .   INVENTORY_LIST_____     이름, 리스트의 경우 F4
    콜렉션 . . . . . .    SAMPLECOLL__     이름, 리스트의 경우 F4

  상관 . . . . . .   ____________________     이름

F3=나감    F4=프롬트    F5=화면정리  F12=취소      F20=완전명 표시
F21=명령문 표시
```

표 이름과 스키마명을 입력한 후에 Enter 키를 누르십시오. 표에 있는 열의 리스트가 표시된 화면이 다시 나타납니다.

```
UPDATE문 지정

선택사항을 입력한 후 Enter 키를 누르십시오.

  표   . . . . . .   INVENTORY_LIST_____     이름, F4=리스트
    콜렉션   . . . . . .    SAMPLECOLL__     이름, F4=리스트

  상관 . . . . . .   ____________________     이름


정보를 입력한 후 Enter 키를 누르십시오.

열                  값
ITEM_NUMBER         ____________________________________________________
ITEM_NAME           ____________________________________________________
UNIT_COST           ____________________________________________________
QUANTITY_ON_HAND    ____________________________________________________
LAST_ORDER_DATE     CURRENT DATE_______________________________________
ORDER_QUANTITY      50__________________________________________________

                                                                  맨 아래
F3=나감    F4=프롬트    F5=화면정리  F6=행 삽입       F10=행 복사
F11=유형 표시        F12=취소     F14=행 삭제      F24=추가 키
```

값에 대해 CURRENT DATE를 지정하면 선택된 모든 행의 날짜가 오늘 날짜로 변경됩니다.

표에 대해 갱신될 값을 입력한 후 WHERE 조건이 지정될 수 있는 화면을 보려면 Enter 키를 누르십시오. WHERE 조건이 지정되지 않으면 표에 있는 모든 행이 이전 화면의 값을 사용하여 갱신됩니다.

```
UPDATE문 지정

WHERE 조건을 입력한 후 Enter 키를 누르십시오. 리스트를 보려면 F4 키를 누르십시오.
  ITEM_NUMBER = '303476'_____________________________________________
  ___________________________________________________________________



                                                                  맨 아래
선택사항을 입력한 후 Enter 키를 누르십시오.

  WITH 분리 레벨   . . . . .  1              1=현재 레벨, 2=NC (NONE)
                                             3=UR (CHG), 4=CS, 5=RS (ALL)
                                             6=RR




F3=나감         F4=프롬트   F5=화면정리    F6=행 삽입     F9=부속 조회 지정
F10=행 복사     F12=취소    F14=행 삭제    F15=행 분할    F24=추가 키
```

조건을 입력한 후 표에서 갱신을 수행하려면 Enter 키를 누르십시오. 기능이 완료되었음을 나타내는 메세지가 표시됩니다.

이 명령문은 다음과 같이 SQL 입력 화면에서 입력될 수도 있습니다.

```
UPDATE SAMPLECOLL.INVENTORY_LIST
  SET LAST_ORDER_DATE = CURRENT DATE,
      ORDER_QUANTITY = 50
  WHERE ITEM_NUMBER = '303476'
```

표로부터 모든 행을 얻기 위해 SELECT문을 수행하면(SELECT * FROM SAMPLECOLL.INVENTORY_LIST) 다음의 결과가 리턴됩니다.

```
                          자료 표시
                                        자료 폭    . . . . . :     71
위치 행 . . . . . .                   열로 이동  . . . . . .
....+....1....+....2....+....3....+....4....+....5....+....6....+....7.
ITEM    ITEM                    UNIT   QUANTITY  LAST      NUMBER
NUMBER  NAME                    COST   ON        ORDER     ORDERED
                                       HAND      DATE
153047  Pencils, red           10.00        25   -              20
229740  Lined tablets           1.50       120   -              20
544931  ***UNKNOWN***           5.00         -   -              20
303476  Paper clips             2.00       100   05/30/94       50
559343  Envelopes, legal        3.00       500   -              20
291124  Envelopes, standard      .00         -   -              20
775298  Chairs, secretary      225.00         6   -              20
073956  Pens, black             20.00        25   -              20
********   자료의 끝   ********
                                                              맨 아래
F3=나감      F12=취소        F19=왼쪽      F20=오른쪽     F21=분할
```

*종이 클립*에 대한 항목만 변경되었습니다. LAST_ORDER_DATE는 현재 날짜로 변경되었습니다. 이 날짜는 항상 갱신되는 날짜입니다. NUMBER_ORDERED는 갱신된 값을 표시합니다.

표에서 정보 삭제

SQL DELETE문을 사용하여 표에서 자료를 삭제할 수 있습니다. 더 이상 필요한 정보가 들어 있지 않으면 표에서 전체 행을 삭제하거나, DELETE문이 있는 WHERE절을 사용하여 단일 명령문 실행하는 동안 작제할 행을 식별할 수 있습니다. 추가 정보는 115 페이지의 『DELETE 명령문을 사용하여 표에서 행 제거』를 참조하십시오.

대화식 SQL를 사용하여 표에서 정보를 삭제하는 예에 대해서는 『예: 표에서 정보 삭제(INVENTORY_LIST)』를 참조하십시오.

예: 표에서 정보 삭제(INVENTORY_LIST)

QUANTITY_ON_HAND 열에 대해 널 값을 갖는 모든 행을 표에서 제거하려는 경우, SQL 입력 화면에 다음 명령문을 입력하면 됩니다.

```
DELETE
       FROM SAMPLECOLL.INVENTORY_LIST
  WHERE QUANTITY_ON_HAND IS NULL
```

널 값에 대해 열을 체크하기 위해서는 IS NULL 비교가 사용됩니다. 삭제가 완료된 후 다른 SELECT문을 수행하면 다음의 결과 표가 리턴됩니다.

```
                              자료 표시
                                          자료 폭    . . . . . :     71
위치 행 . . . . . .           열로 이동  . . . . . .
....+....1....+....2....+....3....+....4....+....5....+....6....+....7.
ITEM    ITEM                    UNIT   QUANTITY  LAST      NUMBER
NUMBER  NAME                    COST   ON        ORDER     ORDERED
                                       HAND      DATE
153047  Pencils, red           10.00       25   -              20
229740  Lined tablets           1.50      120   -              20
303476  Paper clips             2.00      100   05/30/94       50
559343  Envelopes, legal        3.00      500   -              20
775298  Chairs, secretary     225.00        6   -              20
073956  Pens, black            20.00       25   -              20
********   자료의 끝   ********
                                                          맨 아래
F3=나감      F12=취소       F19=왼쪽      F20=오른쪽     F21=분할
```

QUANTITY_ON_HAND에 대해 널 값을 가진 행이 삭제되었습니다.

보기 작성 및 사용

사용자가 필요로 하는 모든 정보가 있는 단일 표가 없을 수도 있습니다. 또한 사용자는 표의 일부 자료에 대해서만 사용자 액세스를 부여할 수도 있습니다. 보기를 사용하면 필요한 자료만을 처리할 수 있도록 표를 서브세트로 나눌 수 있습니다. 보기는 복잡성을 줄이고 동시에 액세스를 제한합니다.

SQL CREATE VIEW문을 사용하여 보기를 작성할 수 있습니다. CREATE VIEW문을 사용할 경우, 표에서 보기를 정의하는 것은 사용자가 원하는 열과 행만을 포함하는 새로운 표를 작성하는 것과 같습니다. 사용자의 어플리케이션이 보기를 사용할 경우, 보

기에 없는 표의 행이나 열에는 액세스할 수 없습니다. 그러나, SQL WITH CHECK OPTION이 사용되지 않을 경우, 여전히 선택 기준에 일치하지 않는 행이 보기를 통해 삽입될 수 있습니다. WITH CHECK OPTION 사용에 대한 자세한 내용은 제 10 장 『자료 무결성』을 참조하십시오.

대화식 SQL를 사용하여 보기를 작성하는 예에 대해서는 다음을 참조하십시오.

- 『예: 단일 표에서 보기 작성』
- 35 페이지의 『예: 두 개 이상의 표로부터 자료를 결합하는 보기 작성』

보기를 작성하기 위해서는 보기가 기초를 두고 있는 표와 실제 파일에 대해 적합한 권한을 갖고 있어야 합니다. 필요한 권한 리스트는 *SQL 참조서*의 CREATE VIEW문을 참조하십시오.

보기에 대해 열 이름을 지정하지 않으면 열 이름은 보기가 기초로 하는 표에 대한 열 이름과 동일하게 됩니다.

보기에 표와 다른 수의 열이나 행이 있더라도 보기를 통해 표를 변경시킬 수 있습니다. INSERT의 경우, 보기에 없는 표의 열은 디폴트 값을 가져야 합니다.

보기는 자료에 대한 하나 이상의 표에 전적으로 의존하지만 사용자는 보기를 표처럼 사용할 수 있습니다. 보기에는 자체 자료가 없으므로 자료에 대한 기억영역이 필요하지 않습니다. 보기는 기억영역에 존재하는 표로부터 파생되므로 보기 자료를 갱신할 때 실제로 표 내의 자료가 갱신됩니다. 그러므로 보기는 보기가 종속되는 표가 갱신됨에 따라 자동적으로 최신 표로 유지됩니다.

추가 정보는 63 페이지의 『보기 작성 및 사용』을 참조하십시오.

예: 단일 표에서 보기 작성

다음 예제에서는 단일 표에서 보기를 작성하는 방법을 보여줍니다. 보기는 INVENTORY_LIST 표에서 작성됩니다. 표에는 6개의 열이 있지만 보기는 세 개의 열 즉, ITEM_NUMBER, LAST_ORDER_DATE 및 QUANTITY_ON_HAND만을 사용합니다. SELECT절에서 열 순서는 보기에서 그 열이 나타나는 순서입니다. 보기는 마지막 2주 내에 주문된 품목에 대한 행만을 포함합니다. CREATE VIEW문은 다음과 같이 표시될 것입니다.

```
CREATE VIEW SAMPLECOLL.RECENT_ORDERS AS
    SELECT ITEM_NUMBER, LAST_ORDER_DATE, QUANTITY_ON_HAND
         FROM SAMPLECOLL.INVENTORY_LIST
         WHERE LAST_ORDER_DATE > CURRENT DATE - 14 DAYS
```

위의 예에서는 보기명 다음에 오는 열 리스트가 없으므로 보기의 열이 표 열과 동일한 이름을 가집니다. 보기가 작성되는 스키마는 그 보기가 작성된 표와 동일한 스키마일 필요는 없습니다. 모든 스키마 또는 라이브러리를 사용할 수 있습니다. 다음 화면은 SQL 문의 실행 결과입니다.

```
SELECT * FROM SAMPLECOLL.RECENT_ORDERS
```

```
                              자료 표시
                                          자료 폭  . . . . . :     26
위치 행 . . . . . .           열로 이동  . . . . . .
....+....1....+....2....+.
ITEM    LAST      QUANTITY
NUMBER  ORDER     ON
        DATE      HAND
303476  05/30/94      100
********   자료의 끝   ********
                                                              맨 아래
F3=나감      F12=취소       F19=왼쪽      F20=오른쪽     F21=분할
```

보기에 의해 선택된 유일한 행은 현재 날짜를 갖도록 갱신한 행입니다. 표의 다른 날짜들은 모두 여전히 널 값을 갖고 있으므로 리턴되지 않습니다.

예: 두 개 이상의 표로부터 자료를 결합하는 보기 작성

FROM절에서는 두 개 이상의 표를 명명하여 두 개 이상의 표에서 자료를 결합하는 보기를 작성할 수 있습니다. 다음 예에서 INVENTORY_LIST 표는 항목 번호 열 *ITEM_NUMBER*와 항목 비용 열 *UNIT_COST*를 포함합니다. 이들은 SUPPLIERS 표의 *ITEM_NUMBER* 열과 *SUPPLIER_COST* 열으로 결합됩니다. WHERE절은 리턴되는 행의 수를 제한하는데 사용됩니다. 보기는 현재 단위 원가보다 낮은 원가로 품목을 공급할 수 있는 공급자들에 대한 항목 번호만을 포함합니다.

CREATE VIEW문은 다음과 같이 표시될 것입니다.

```
CREATE VIEW SAMPLECOLL.LOWER_COST AS
  SELECT SUPPLIER_NUMBER, A.ITEM_NUMBER, UNIT_COST, SUPPLIER_COST
    FROM SAMPLECOLL.INVENTORY_LIST A, SAMPLECOLL.SUPPLIERS B
    WHERE A.ITEM_NUMBER = B.ITEM_NUMBER
    AND UNIT_COST > SUPPLIER_COST
```

다음 표는 SQL문의 수행 결과입니다.

```
SELECT * FROM SAMPLECOLL.LOWER_COST
```

```
                              자료 표시
                                          자료 폭  . . . . . :     51
위치 행 . . . . . .           열로 이동  . . . . . .
....+....1....+....2....+....3....+....4....+....5.
SUPPLIER_NUMBER  ITEM          UNIT   SUPPLIER_COST
                 NUMBER        COST
     1234        229740         1.50          1.00
     9988        153047        10.00          8.00
     2424        153047        10.00          9.00
     3366        303476         2.00          1.50
     3366        073956        20.00         17.00
********   자료의 끝   ********
                                                              맨 아래
F3=나감      F12=취소       F19=왼쪽      F20=오른쪽     F21=분할
```

주: 이 조회에 대하여 ORDER BY 절이 지정되지 않았으므로 조회에 의해 리턴된 행 순서는 다를 수 있습니다.

이 보기를 통해 표시될 수 있는 행은 단위 원가보다 낮은 공급자 원가를 가진 행뿐입니다.

대화식 SQL 사용에 대한 자세한 정보는 323 페이지의 제 17 장『대화식 SQL 사용』을 참조하십시오.

제 3 장 iSeries Navigator 데이터베이스 시작

이 장에서는 iSeries Navigator를 사용하여 라이브러리(스키마 또는 SQL 콜렉션), 표 및 보기를 작성하고 이에 대해 작업하는 방법을 설명합니다. iSeries Navigator 데이터베이스는 여러 가지 공통 관리 데이터베이스 작업을 수행하는 데 사용할 수 있는 그래픽 인터페이스입니다. 대부분의 iSeries Navigator 작업은 SQL(구조화 조회 언어)에 기초하지만 이를 수행하기 위해 SQL을 완전히 이해할 필요는 없습니다. 305 페이지의 제 16 장『iSeries Navigator를 사용한 확장 데이터베이스 기능』에서는 iSeries Navigator를 사용하는 일부 확장 데이터베이스 기능을 설명합니다. 이 장에 나오는 예는 iSeries Navigator를 사용하여 공통 데이터베이스 타스크의 실행을 보여줍니다. 작성된 오브젝트들은 17 페이지의 제 2 장『SQL 시작』에서 대화식 SQL을 사용하여 예에 작성되는 오브젝트와 동일한 오브젝트입니다.

세부사항은 다음 주제를 참조하십시오.

- 『iSeries Navigator 시작』
- 38 페이지의『iSeries Navigator를 사용한 라이브러리 작성』
- 39 페이지의『iSeries Navigator에 표시되는 라이브러리 리스트 편집』
- 40 페이지의『iSeries Navigator를 사용한 표 작성 및 사용』
- 41 페이지의『iSeries Navigator를 사용하여 표에서 열 정의』
- 43 페이지의『iSeries Navigator를 사용한 열 정의 복사』
- 43 페이지의『iSeries Navigator를 사용하여 표에 정보 삽입』
- 44 페이지의『iSeries Navigator를 사용하여 표 내용 보기』
- 46 페이지의『iSeries Navigator를 사용하여 표 복사 및 이동』
- 47 페이지의『iSeries Navigator를 사용한 보기 작성 및 사용』
- 51 페이지의『iSeries Navigator를 사용한 데이터베이스 오브젝트 삭제』

주: 코드 예에 대한 정보는 x 페이지의『코드 면책사항 정보』를 참조하십시오.

iSeries Navigator 시작

다음의 예에 대해 iSeries Navigator를 시작하려면, 다음과 같이 하십시오.

1. **iSeries Navigator** 아이콘을 두 번 클릭하십시오.
2. 사용하려는 시스템을 펼치십시오.

iSeries Navigator 설정에 대한 자세한 내용은 iSeries Navigator 알기를 참조하십시오.

iSeries Navigator를 사용한 라이브러리 작성

라이브러리는 표, 보기 및 기타 오브젝트 유형이 포함된 데이터베이스 구조입니다. 라이브러리를 사용하여 관련된 오브젝트들을 그룹화하고 오브젝트를 이름으로 찾을 수 있습니다. 또한 스키마로 라이브러리를 작성하고 자료 사전을 지정할 수도 있습니다.

또한 스키마(SQL 콜렉션)에는 라이브러리에 작성된 모든 표, 보기, 색인, 파일, 패키지 및 제한사항에 대한 설명과 정보가 들어 있는 카탈로그 보기도 포함됩니다. 스키마에서 작성된 모든 표에는 저널링이 자동으로 수행됩니다.

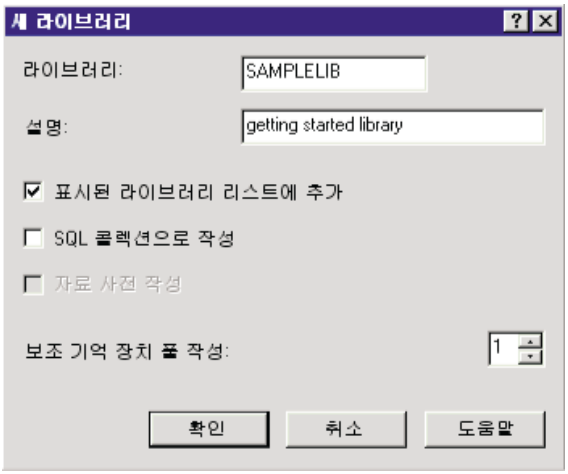
또한 복수 데이터베이스로 작업할 수도 있습니다. 세부사항은 복수 데이터베이스로 작업을 참조하십시오.

iSeries Navigator를 사용하여 라이브러리(스키마)를 작성하는 방법에 대한 예는 『예: iSeries Navigator를 사용한 라이브러리 작성(SAMPLELIB)』을 참조하십시오.

예: iSeries Navigator를 사용한 라이브러리 작성(SAMPLELIB)

다음과 같이 수행하여 이름이 SAMPLELIB인 샘플 라이브러리를 작성할 수 있습니다.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스를 펼치십시오.
2. **라이브러리**에서 오른쪽 마우스 버튼을 클릭하고 **새 라이브러리**를 선택하십시오.
3. **새 라이브러리** 대화 상자에서 이름 필드에 SAMPLELIB를 입력하십시오.
4. 설명을 지정하십시오(선택적).
5. 표시할 라이브러리 리스트에 추가하려면, **라이브러리 리스트에 추가**를 표시하도록 선택하십시오.
6. **스키마로 작성**을 선택하여 스키마로 라이브러리를 작성하고 **자료 사전 작성**을 선택하여 자료 사전을 작성할 수 있습니다. 그러나, 이 샘플 라이브러리를 기본 라이브러리로만 작성하십시오.
7. 라이브러리를 포함할 디스크 풀을 지정하십시오. 라이브러리가 시스템 디스크 풀에 작성되도록 1을 선택하십시오.
8. 확인을 클릭하십시오.



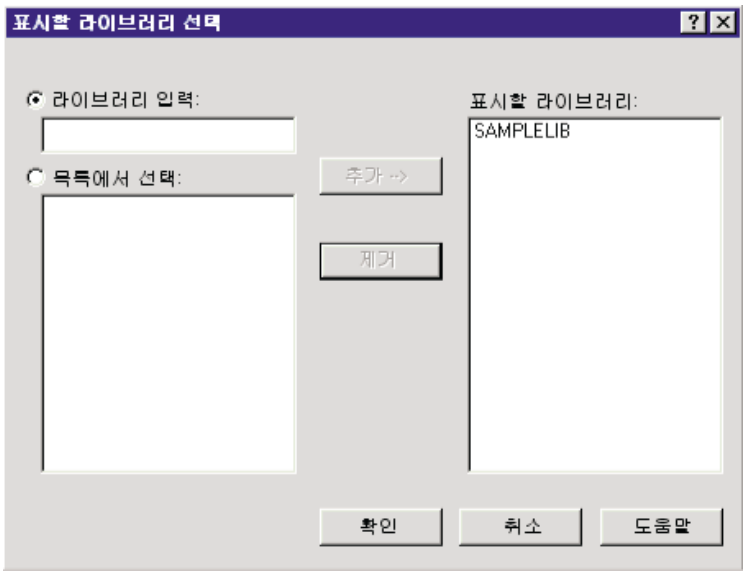
주:

1. 스키마로 SAMPLELIB를 작성했다면, 여러 개의 오브젝트가 작성되며 이 프로세스에 몇 초가 소요될 수 있습니다.
2. 사용자 디스크 풀의 라이브러리 작성에 대한 자세한 정보는 복수 데이터베이스로 작업을 참조하십시오.

iSeries Navigator에 표시되는 라이브러리 리스트 편집

라이브러리를 성공적으로 작성했으면, 그 안에 표, 보기, 색인, 저장 프로시듀어, 사용자 정의 기능 및 사용자 정의 유형을 작성할 수 있습니다. 라이브러리를 클릭하여 표시되는 라이브러리 리스트를 편집하려면, 다음과 같이 하십시오.

1. **라이브러리**에서 오른쪽 마우스 버튼을 클릭하고 **표시할 라이브러리 선택**을 선택하십시오.
2. **표시할 라이브러리 선택** 대화 상자에서 라이브러리명을 선택하고 추가를 클릭하여 리스트를 편집할 수 있습니다.
3. 표시할 라이브러리 리스트에서 해당 라이브러리를 선택하고 **제거**를 클릭하여 표시할 라이브러리 리스트에서 라이브러리를 제거할 수 있습니다.



라이브러리가 표시되면 SAMPLELIB를 즉시 종료하십시오.

iSeries Navigator를 사용한 표 작성 및 사용

표는 정보를 저장하는 데 사용되는 기본 데이터베이스 오브젝트입니다. 표를 작성했으면, 표 등록정보 대화 상자를 사용하여 열을 정의하고, 색인을 작성하고, 트리거 및 제한사항을 추가할 수 있습니다.

iSeries Navigator를 사용한 표 작성 예는 『예: iSeries Navigator를 사용한 표 작성 (INVENTORY_LIST)』을 참조하십시오.

표를 작성할 때, 널 값 및 디폴트 값의 개념을 이해해야 합니다. 널 값은 그 행에 열값이 없다는 것을 나타냅니다. 그것은 제로 또는 모두 공백의 값이 아니고 "알 수 없음"을 의미합니다. 어느 값이나 다른 널 값과도 다릅니다. 열이 널 값을 허용하지 않으면, 열에 값을 지정해야 합니다. 이 값은 디폴트 값 또는 사용자가 제공하는 값입니다.

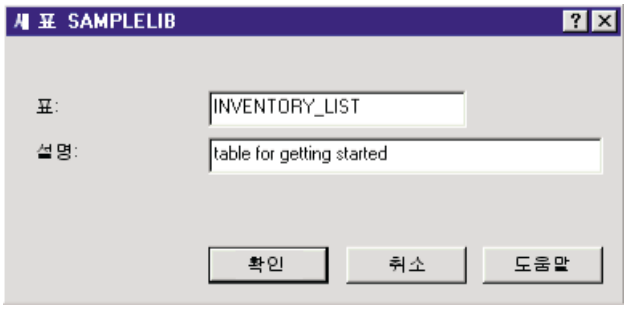
표에 행이 추가될 때 열에 값이 지정되지 않은 경우, 이 행에는 디폴트 값이 지정됩니다. 열에 특정 디폴트 값이 지정되지 않은 경우, 열은 시스템 디폴트 값을 사용합니다. INSERT에 사용되는 디폴트 값에 대한 자세한 내용은 105 페이지의 『INSERT 명령문을 사용하여 열 삽입』을 참조하십시오.

예: iSeries Navigator를 사용한 표 작성(INVENTORY_LIST)

이제 사업장의 현재 재고에 대한 정보를 유지보수하기 위한 표를 작성할 것입니다. 표에는 재고 품목, 원가, 현재 보유량, 최종 주문일 및 최종 주문 번호에 대한 정보가 포함됩니다. 품번은 필수값입니다. 이것은 널(null)이 될 수 없습니다. 품목명, 보유량 및 주문량은 사용자 제공 디폴트 값입니다. 최종 주문일과 수량은 널 값을 허용합니다.

표를 작성하려면, 다음과 같이 하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. SAMPLELIB를 마우스 오른쪽 버튼으로 클릭하고 **신규**를 선택하십시오.
3. 표를 선택하십시오.
4. **새 표** 대화 상자에서 표 이름으로 INVENTORY_LIST를 입력하십시오.
5. 설명을 지정하십시오(선택적).
6. **확인**을 클릭하십시오.



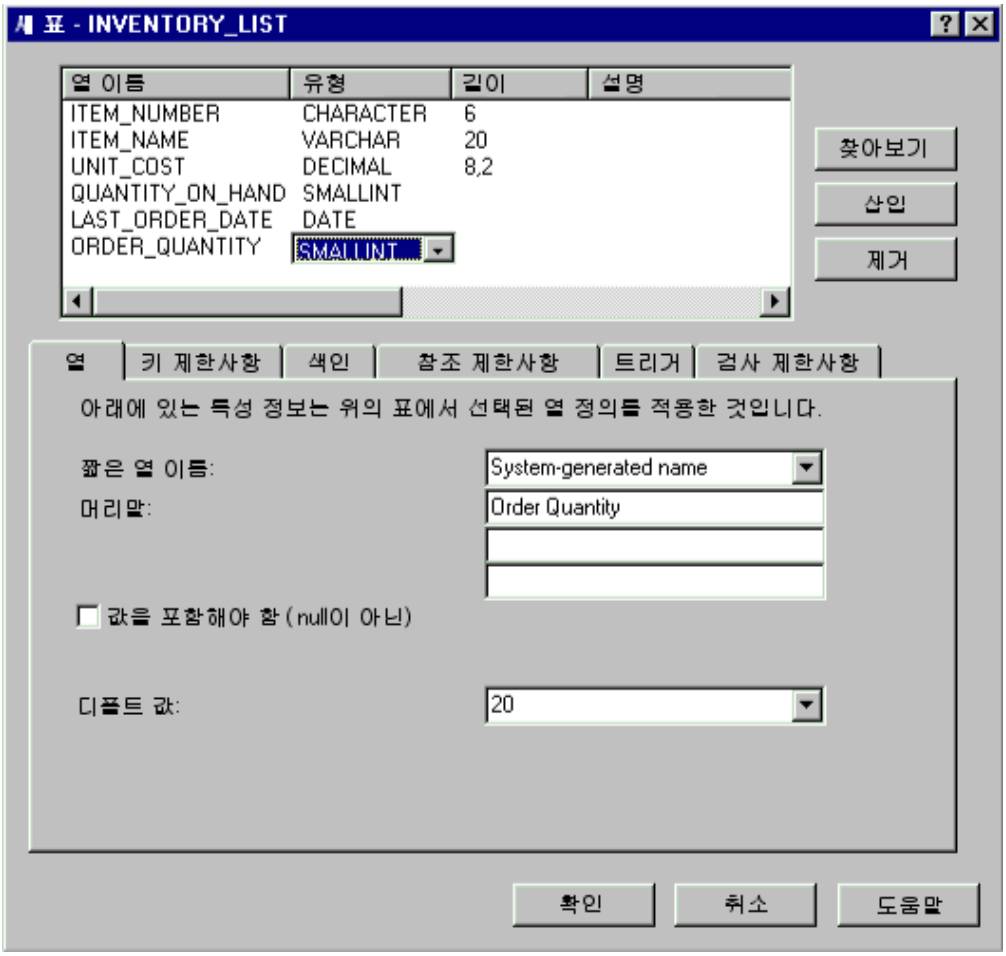
새 표 - INVENTORY_LIST가 나타납니다. 이 대화 상자를 닫지 마십시오. 다음 단계에서 필요합니다.

iSeries Navigator를 사용하여 표에서 열 정의

새로운 표나 기존의 표에서 열을 정의할 수 있습니다. 기존 표에 대해 열을 정의하는 경우, **데이터베이스** → **라이브러리** → *SAMPLELIB*(또는 적합한 라이브러리명)를 펼쳐서 표를 탐색하십시오. 세부사항 분할 창에서 표 *INVENTORY_LIST*를 오른쪽 마우스 버튼으로 클릭하고 **등록정보**를 선택하십시오.

1. **표 등록정보** 또는 **새 표** 대화 상자에서 열을 정의하려면, **신규** 또는 **삽입**을 클릭하십시오. **열 정의 격자**에 새로운 열이 나타납니다.
2. **열 정의 격자**에 이름 *ITEM_NUMBER*를 입력하십시오.
3. 유형이 CHARACTER인지 확인하십시오. 현재 나열된 유형을 클릭하고 아래 방향 화살표를 클릭하고 제공된 리스트에서 새로운 유형을 선택하여 유형을 변경할 수 있습니다.
4. 이 열에 길이 6을 지정하십시오. *크기*가 사전에 결정된 자료 유형의 경우, *크기*가 채워지며 이 값을 변경할 수 없습니다.
5. 열에 대한 설명을 지정할 수 있습니다. 이 단계는 선택적입니다.
6. **열** 탭의 아래에서 **짧은 열 이름** 텍스트 상자에 짧은 이름을 지정할 수 있습니다. 짧은 이름을 지정하지 않으면, 시스템이 자동으로 이름을 생성합니다. 열 이름이 10자 이내인 경우, 짧은 이름은 열 이름과 동일합니다. 어느 쪽 열 이름을 사용하든지 조회를 수행할 수 있습니다. 지금은 이 공간을 공백으로 두십시오.

7. 각 열에 대해 열 머리말을 입력하십시오.
8. **값(널이 아님)이 있어야 함**을 선택하십시오. 그러면, 행 삽입이 성공하기 위해 이 열에 값이 반드시 입력됩니다.
9. **디폴트 없음**에 반드시 디폴트 값을 설정하십시오. 이것은 값을 입력해야 하는 열입니다.



이제 열 ITEM_NUMBER를 정의했습니다. 표 INVENTORY_LIST에 다음의 열을 추가하십시오.

열 이름	유형	길이	스케일	널	디폴트 값
ITEM_NAME	VARCHAR	20		널이 아님	UNKNOWN
UNIT_COST	DECIMAL	8	2	널이 아님	(열 자료 유형 디폴트로 설정)
QUANTITY_ON_HAND	SMALLINT			NULL	NULL
LAST_ORDER_DATE	DATE			NULL	
ORDER_QUANTITY	SMALLINT			NULL	20

이러한 열 정의를 완료했으면, **확인**을 클릭하여 표를 작성하십시오.

iSeries Navigator를 사용한 공급업체 표(SUPPLIERS) 작성

이 예를 계속 진행하다 보면 2차 표도 필요하게 됩니다. 이 표는 재고 품목의 공급자, 공급 품목, 그리고 품목 원가에 관한 정보가 포함됩니다. SAMPLELIB에 SUPPLIERS라고 하는 표를 작성하십시오. 이 표에는 SUPPLIER_NUMBER, ITEM_NUMBER 및 SUPPLIER_COST의 3개의 열이 있습니다. 이 표에는 표 INVENTORY_LIST와 공통되는 열인 ITEM_NUMBER가 있다는 점을 유의하십시오. 새로운 ITEM_NUMBER 열을 작성하기보다는 INVENTORY_LIST 표에 ITEM_NUMBER에 대해 사용되는 열 정의를 복사할 수 있습니다.

iSeries Navigator를 사용한 열 정의 복사

열 정의를 복사하려면, 다음과 같이 하십시오.

1. SUPPLIER 표 **등록정보** 또는 **새 표** 대화 상자에서 **찾아보기**를 클릭하십시오.
2. **표 찾아보기** 대화 상자에서 SAMPLELIB를 펼치십시오.
3. INVENTORY_LIST를 클릭하십시오. 이 표의 열들이 자료 유형, 크기 및 설명과 함께 나열됩니다.
4. ITEM_NUMBER를 선택하십시오.
5. 이 열 정의를 SUPPLIERS 표에 복사하도록 **확인**을 클릭하십시오.

SUPPLIER_NUMBER, CHAR(4), NOT NULL 및 SUPPLIER_COST, DECIMAL(8,2)의 값으로 표 SUPPLIERS에 대해 마지막 두 개의 열을 추가하십시오.

iSeries Navigator를 사용하여 표에 정보 삽입

표에서 자료를 삽입, 편집 또는 삭제하려면, 이 표에 대한 권한이 있어야 합니다. 표 INVENTORY_LIST에 자료를 추가하려면, 다음과 같이 하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. SAMPLELIB를 클릭하십시오.
3. INVENTORY_LIST에서 마우스 오른쪽 버튼을 클릭하고 **열기**를 선택하십시오.
4. 행 메뉴에서 **삽입**을 선택하십시오. 새로운 행이 나타납니다.
5. 적합한 표제 아래에 다음의 정보를 입력하십시오.

SAMPLELIB.INVENTORY_LIST

파일(F) 편집(E) 보기(V) 행(R) 도움말(H)

ITEM_NUMBER	ITEM_NAME	UNIT_COST	QUANTITY_ON_HAND	LAST_ORDER_DATE	ORDER_QUANTITY
153047	Pencils, red	10.00	25		20
229740	Lined tablets	1.50	120		20
544931	***UNKNOWN***	5.00			20
303476	Paper clips	2.00	100		20
559343	Envelopes, legal	3.00	500		20
291124	Envelopes, stand...	0			20
775298	Chairs, secretary	225.00	6		20
073956	Pens, black	20.00	25		20

주: 입력한 값은 모든 제한사항을 충족시키고 각 열의 유형을 충족시켜야 합니다. 표에 대해 고유 제한사항이나 색인이 있는 경우, 입력한 값은 고유 키 값을 정의해야 합니다. 열에 값을 입력하지 않은 경우, 허용되면 디폴트 값이 입력됩니다. 이 연습에서는 디폴트 값을 사용하도록 아래의 도표에 표시되지 않은 값을 삽입하지 마십시오.

ITEM_NUMBER	ITEM_NAME	UNIT_COST	QUANTITY_ON_HAND
153047	Pencils, red	10.00	25
229740	Lined tablets	1.50	120
544931		5.00	
303476	Paper clips	2.00	100
559343	Envelopes, legal	3.00	500
291124	Envelopes, standard		
775298	Chairs, secretary	225.00	6
073956	Pens, black	20.00	25

파일 메뉴에서 **저장**을 선택하십시오.

다음 행을 SAMPLELIB.SUPPLIERS 표에 추가하십시오.

SUPPLIER_NUMBER	ITEM_NUMBER	SUPPLIER_COST
1234	153047	10.00
1234	229740	1.00
1234	303476	3.00
9988	153047	8.00
9988	559343	3.00
2424	153047	9.00
2424	303476	2.50
5546	775298	225.00
3366	303476	1.50
3366	073956	17.00

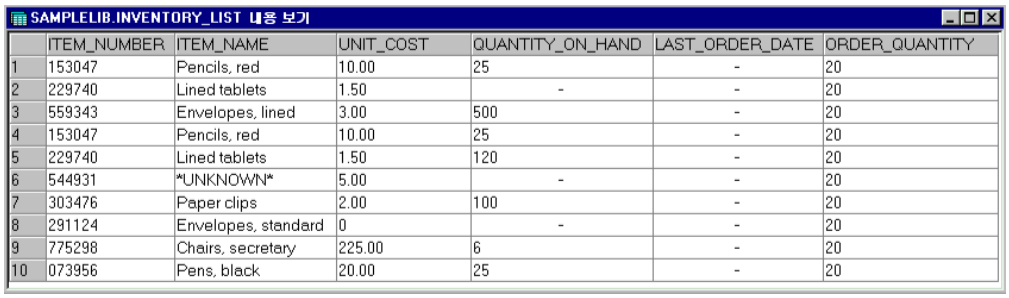
파일 메뉴에서 **저장**을 선택하십시오. 샘플 라이브러리에는 이제 각각의 자료에 몇 개의 행이 있는 두 개의 표가 포함됩니다.

iSeries Navigator를 사용하여 표 내용 보기

빨리 보기를 사용하여 표와 보기의 내용을 표시할 수 있습니다. 내용을 볼 수만 있으며 표를 변경하려면 표를 열어야 합니다. INVENTORY_LIST의 내용을 보려면, 다음과 같이 하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. SAMPLELIB를 클릭하십시오.

3. INVENTORY_LIST에서 마우스 오른쪽 버튼을 클릭하고 **빨리 보기**를 선택하십시오.


SAMPLELIB.INVENTORY_LIST 내용 보기

	ITEM_NUMBER	ITEM_NAME	UNIT_COST	QUANTITY_ON_HAND	LAST_ORDER_DATE	ORDER_QUANTITY
1	153047	Pencils, red	10.00	25	-	20
2	229740	Lined tablets	1.50	-	-	20
3	559343	Envelopes, lined	3.00	500	-	20
4	153047	Pencils, red	10.00	25	-	20
5	229740	Lined tablets	1.50	120	-	20
6	544931	*UNKNOWN*	5.00	-	-	20
7	303476	Paper clips	2.00	100	-	20
8	291124	Envelopes, standard	0	-	-	20
9	775298	Chairs, secretary	225.00	6	-	20
10	073956	Pens, black	20.00	25	-	20

주: 빨리 보기는 자료 링크 열 및 연관된 URL에 액세스할 수 있도록 허용하며 선택하면 브라우저를 시작합니다.

iSeries Navigator를 사용하여 표의 정보 변경

iSeries Navigator를 사용하여 표의 열에서 자료 값을 변경할 수 있습니다. 오늘 추가 클립 주문을 받았음을 나타내기 위해 iSeries Navigator를 사용하여 열을 갱신하려고 한다고 가정합시다. 입력한 값이 해당 열에 대해 유효해야 함을 기억하십시오.

1. 표 INVENTORY_LIST를 두 번 클릭하여 여십시오.
2. 클립 행에 대해 LAST_ORDER_DATE 열에 현재 날짜를 입력하십시오. 반드시 시스템에 올바른 날짜 형식을 사용하십시오.
3. ORDER_QUANTITY를 50으로 변경하십시오.
4. 변경사항을 저장하고 **빨리 보기**를 사용하여 표 내용을 보십시오.

클립 행이 변경된 사항을 반영합니다.

iSeries Navigator를 사용하여 표에서 정보 삭제

iSeries Navigator를 사용하여 표에서 자료를 삭제할 수 있습니다. 한 행의 단일 열에서 정보를 삭제하거나 행을 전부 삭제할 수 있습니다. 열에 값이 필요한 경우, 전체 행을 삭제하지 않고는 이를 삭제할 수 없다는 점을 기억하십시오.

1. 표 INVENTORY_LIST를 두 번 클릭하여 여십시오.
2. Envelopes, standard 행의 ORDER_QUANTITY에 대한 열 값을 삭제하십시오. 널 값을 허용하는 열이므로 값을 삭제할 수 있습니다.
3. Lined tablets 행의 UNIT_COST에 대한 열 값을 삭제하십시오. 열은 널 값을 허용하지 않으므로 삭제는 허용되지 않습니다.

열 값을 한 번에 하나씩 모두 제거하지 않고 전체 행을 삭제할 수도 있습니다.

1. 표 INVENTORY_LIST를 두 번 클릭하여 여십시오.
2. *UNKNOWN* 행의 왼쪽에 있는 회색 셀을 클릭하십시오. 그러면, 전체 행이 강조표시됩니다.

3. Rows 메뉴에서 **삭제**를 선택하거나 키보드에서 Delete 키를 누르십시오. *UNKNOWN* 행이 삭제됩니다.
4. 표 INVENTORY_LIST에서 QUANTITY_ON_HAND 열에 값이 없는 모든 행을 삭제하십시오.
5. 변경사항을 저장하고 빨리 보기를 사용하여 내용을 보십시오. 다음의 자료가 들어 있는 표가 있어야 합니다.

ITEM_ NUMBER	ITEM_ NAME	UNIT_ COST	QUANTITY_ ON_ HAND	LAST_ ORDER_ DATE	ORDER_ QUANTITY
153047	Pencils, red	10.00	25		20
229740	Lined tablets	1.50	120		20
303476	Paper clips	2.00	100	2000-10-02	50
559343	Envelopes, legal	3.00	500		20
775298	Chairs, secretary	225.00	6		20
073956	Pens, black	20.00	25		20

iSeries Navigator를 사용하여 표 복사 및 이동

iSeries Navigator를 통해 한 라이브러리나 시스템에서 다른 라이브러리나 시스템으로 표를 복사하거나 이동시킬 수 있습니다. 표를 복사하면 둘 이상의 표 인스턴스가 작성되며 이동시키면 새로운 위치로 표가 전송되고 이전 위치에서 인스턴스가 제거됩니다.

LIBRARY1이라고 하는 새로운 라이브러리를 작성하고 이를 표시되는 라이브러리 리스트에 추가하십시오. 이 새로운 라이브러리를 작성했으면, INVENTORY_LIST를 LIBRARY1로 복사하십시오. 표를 복사하려면, 다음과 같이 하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. SAMPLELIB를 클릭하십시오.
3. INVENTORY_LIST에서 마우스 오른쪽 버튼을 클릭하고 **복사**를 선택하십시오.
4. LIBRARY에서 마우스 오른쪽 버튼을 클릭하고 **붙여넣기**를 선택하십시오.

이제 표 INVENTORY_LIST를 LIBRARY1로 복사했으므로 표 SUPPLIERS를 LIBRARY1로 이동시키십시오. 표를 이동시키려면, 다음과 같이 하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. SAMPLELIB를 클릭하십시오.
3. SUPPIERS에서 마우스 오른쪽 버튼을 클릭하고 **잘라내기**를 선택하십시오.
4. LIBRARY1에서 마우스 오른쪽 버튼을 클릭하고 **붙여넣기**를 선택하십시오.

주: 새로운 라이브러리에 표를 끌어 놓아서 표를 이동시킬 수 있습니다. 표를 새로운 위치로 이동시킨다고 소스 시스템에서 이 표가 항상 제거되는 것은 아닙니다. 예를 들어, 소스 표에 대해 읽기 권한이 있지만 삭제 권한이 없는 경우, 표를 목표 시스템에 이동시킬 수 있습니다. 그러나, 두 개의 표 인스턴스가 존재하게 되므로 소스 시스템에서 표를 삭제할 수 없습니다.

iSeries Navigator를 사용한 보기 작성 및 사용

사용자가 필요로 하는 모든 정보가 있는 단일 표가 없을 수도 있습니다. 또한 사용자는 표의 일부 자료에 대해서만 사용자 액세스를 부여할 수도 있습니다. 보기를 사용하면 필요한 자료만을 처리할 수 있도록 표를 서브세트로 나눌 수 있습니다. 보기는 복잡성을 줄이고 동시에 액세스를 제한합니다.

보기를 작성하기 위해서는 보기가 기초를 두고 있는 표와 실제 파일에 대해 적합한 권한을 갖고 있어야 합니다. 필요한 권한 리스트를 보려면 SQL 참조서에서 CREATE VIEW문을 참조하십시오.

보기에 대해 열 이름을 지정하지 않으면 열 이름은 보기가 기초로 하는 표에 대한 열 이름과 동일하게 됩니다.

보기에 표와 다른 수의 열이나 행이 있더라도 보기를 통해 표를 변경시킬 수 있습니다. INSERT의 경우, 보기에 없는 표의 열은 디폴트 값을 가져야 합니다.

보기는 자료에 대한 하나 이상의 표에 전적으로 의존하지만 사용자는 보기를 표처럼 사용할 수 있습니다. 보기에는 자체 자료가 없으므로 자료에 대한 기억영역이 필요하지 않습니다. 보기는 기억영역에 존재하는 표로부터 파생되므로 보기 자료를 갱신할 때 실제로 표 내의 자료가 갱신됩니다. 그러므로 보기는 보기가 종속되는 표가 갱신됨에 따라 자동적으로 최신 표로 유지됩니다.

추가 정보는 63 페이지의 『보기 작성 및 사용』을 참조하십시오.

iSeries Navigator를 사용한 보기 작성 예는 다음의 예를 참조하십시오.

- 『iSeries Navigator를 사용하여 단일 표에 대한 보기 작성』
- 50 페이지의 『iSeries Navigator를 사용하여 둘 이상의 표에서 자료를 결합하는 보기 작성』

iSeries Navigator를 사용하여 단일 표에 대한 보기 작성

다음 예제에서는 단일 표에서 보기를 작성하는 방법을 보여줍니다. 보기는 INVENTORY_LIST 표에서 작성됩니다. 표에는 6개의 열이 있지만 보기는 세 개의 열 즉, ITEM_NUMBER, LAST_ORDER_DATE 및 QUANTITY_ON_HAND만을 사용합니다.

단일 표에 대해 보기를 작성하려면, 다음과 같이 하십시오.

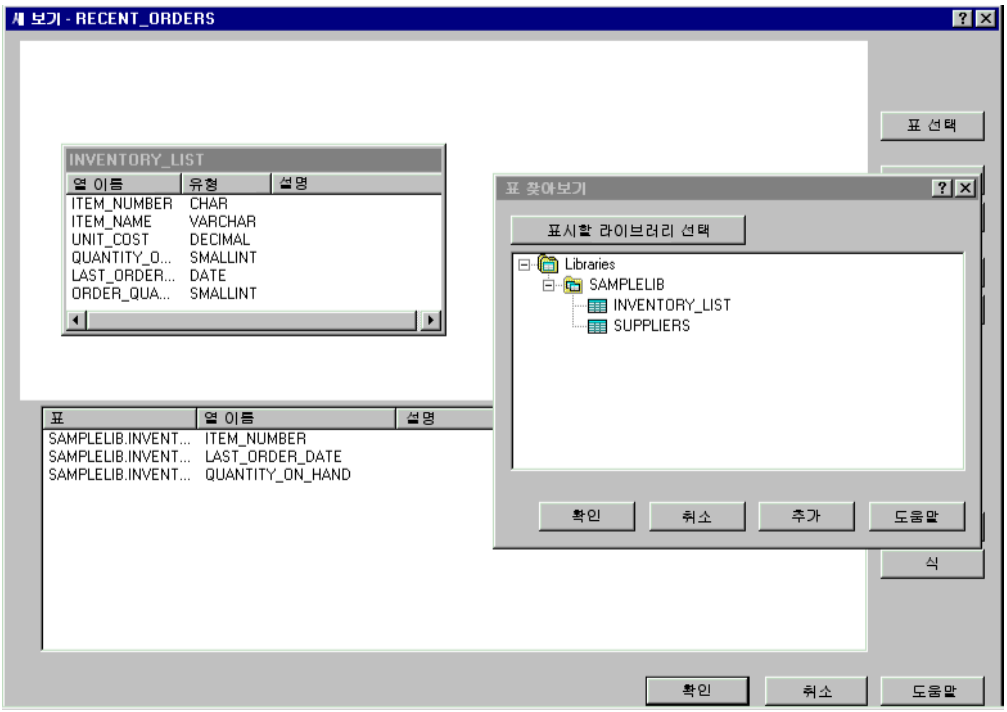
1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. SAMPLELIB에서 마우스 오른쪽 버튼을 클릭하고 **신규**를 선택한 다음, **보기**를 선택하십시오.
3. **새 보기** 대화 상자에서 **이름** 필드에 RECENT_ORDERS를 입력하십시오.
4. 선택적으로 설명을 지정할 수 있습니다.
5. 추가로, 이 대화 상자에서 검사 옵션을 선택하십시오. 보기에서의 검사 옵션은 행에 삽입되었거나 갱신된 값이 보기의 조건을 만족시켜야 한다는 점을 지정합니다. 검사 옵션에 대한 자세한 내용은 162 페이지의 『보기에서 WITH CHECK OPTION』을 참조하십시오. 이 보기의 경우, **없음**을 선택하십시오.
6. **확인**을 클릭하십시오.

새 보기 대화 상자에서 다음과 같이 수행하십시오.

1. **표 선택**을 클릭하십시오.
2. **표 찾아보기** 대화 상자에서 SAMPLELIB를 펼친 다음, INVENTORY_LIST를 선택하십시오.
3. **추가**를 클릭하십시오.
4. **확인**을 클릭하십시오. 이제 INVENTORY_LIST가 새 보기 대화 상자의 작업 영역에 있어야 합니다.
5. 새로운 보기에서 원하는 열을 선택하려면, 선택된 표에서 이를 클릭하고 대화 상자의 아랫쪽 절반에 있는 선택 격자에 이를 끌어다 놓으십시오. ITEM_NUMBER, LAST_ORDER_DATE 및 QUANTITY_ON_HAND를 선택하십시오.
6. 선택 격자에 표시되는 열의 순서는 보기에 나타나는 순서입니다. 이 순서를 변경하려면, 열을 선택하고 새로운 위치로 끌어오십시오. ITEM_NUMBER LAST_ORDER_DATE QUANTITY_ON_HAND의 순서로 열을 배치하십시오.

이제 보기가 완료되기 마련이지만 이전 14일 동안에 주문된 품목만을 보려고 합니다. 이 정보를 지정하기 위해 WHERE 절을 작성해야 합니다.

1. **행 선택**을 클릭하십시오.
2. **행 선택** 대화 상자에 WHERE LAST_ORDER_DATE > CURRENT DATE - 14 DAYS를 입력하십시오. 표시된 옵션에서 선택하여 이 WHERE 절을 구성하는 요소를 선택할 수 있습니다.
3. **확인**을 클릭하십시오.
4. 이 보기를 생성하는 데 사용된 SQL을 보려면, **SQL 표시**를 클릭하십시오.
5. **확인**을 클릭하여 보기를 작성하십시오.



RECENT_ORDERS의 내용을 표시하려면, RECENT_ORDERS에서 마우스 오른쪽 버튼을 클릭하고 **빨리 보기**를 선택하십시오. 다음의 정보가 표시되는 것을 볼 수 있습니다.

ITEM_NUMBER	LAST_ORDER_DATE	QUANTITY_ON_HAND
303476	2000-10-02	100

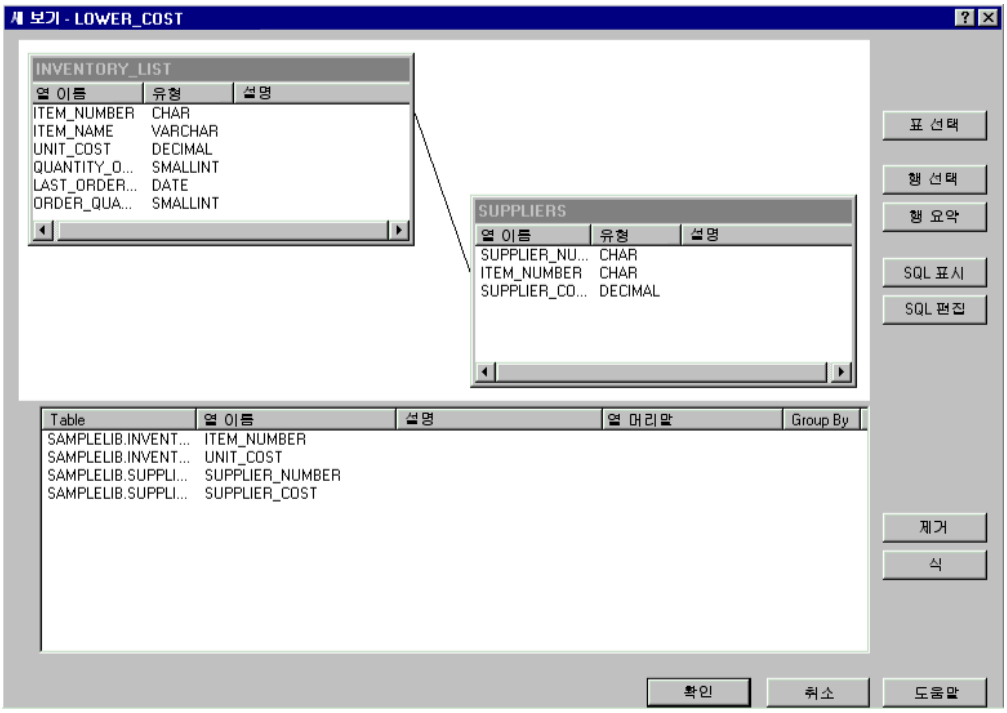
위의 예에서, 새로운 이름을 지정하지 않았으므로 보기의 열은 표의 열과 이름이 동일합니다. 보기가 작성되는 스키마는 그 보기가 작성된 표와 동일한 스키마일 필요는 없습니다. 스키마 또는 라이브러리를 사용할 수 있습니다.

iSeries Navigator를 사용하여 둘 이상의 표에서 자료를 결합하는 보기 작성

새 보기 대화 상자의 작업 영역에서 둘 이상의 표를 선택하여 둘 이상의 표에서 정보를 결합하는 보기를 작성할 수 있습니다. 여러 개의 다른 표에서 포함시키려는 열을 선택하고 확인을 클릭하여 둘 이상의 표에서 간단한 보기를 작성할 수 있습니다. 그러나, 이 예는 두 개의 다른 표에서 정보를 결합하고 보려는 행만을 리턴하는 보기를 작성하는 방법을 보여주며 WHERE 절을 사용하는 것과 매우 유사합니다.

현재 단위 원가보다 낮은 원가로 품목을 공급할 수 있는 공급업체들에 대한 품목 번호만이 포함된 보기를 작성하십시오. 여기에는 INVENTORY_LIST 표에서 ITEM_NUMBER 및 UNIT_COST를 선택하고 이를 SUPPLIERS 표의 SUPPLIER_NUMBER 및 SUPPLIER_COST와 결합시키는 작업이 필요합니다. WHERE절은 리턴되는 행의 수를 제한하는데 사용됩니다.

1. LOWER_COST라고 하는 보기를 작성하십시오.
2. **새 보기** 대화 상자에서 **표 선택**을 클릭하십시오.
3. SAMPLELIB에서 INVENTORY_LIST를 선택하고 LIBRARY1에서 SUPPLIERS를 선택하십시오.
4. **확인**을 클릭하십시오. 두 표 모두 대화 상자의 작업 영역에 나타나야 합니다.
5. INVENTORY_LIST에서 ITEM_NUMBER 및 UNIT_COST를 선택하십시오.
6. SUPPLIERS에서 SUPPLIER_NUMBER 및 SUPPLIER_COST를 선택하십시오.
7. 결합을 정의하려면, INVENTORY_LIST에서 ITEM_NUMBER를 선택하고 이를 SUPPLIERS의 ITEM_NUMBER로 끌어오십시오. 한 열에서 다른 열로 선이 그려지며 **결합** 대화 상자가 열립니다.
8. **결합** 대화 상자에서 **내부 결합**을 선택하십시오. 결합에 대한 자세한 내용은 88 페이지의 『하나 이상의 표로부터 자료 결합』을 참조하십시오.
9. **확인**을 클릭하십시오.
10. 다시 한 번, **SQL 표시**를 선택하여 이 보기를 작성하는 데 사용된 SQL을 볼 수 있습니다. **SQL 편집**을 선택하여 SQL을 편집할 수도 있습니다. **SQL 편집**은 SQL 문을 편집할 수 있는 **SQL 스크립트 실행**을 시작합니다. 그러나, SQL을 변경할 경우, **새 보기** 대화 상자에 리턴하기 보다는 **SQL 스크립트 실행**에서 명령문을 실행해야 합니다. **새 보기** 대화 상자로 되돌아가면, 변경사항이 저장되지 않습니다.
11. **행 선택**을 클릭하여 보기에 대한 WHERE절을 작성하십시오. SUPPLIER_COST를 더블 클릭한 다음, < 오퍼레이터를 더블 클릭하고, 마지막으로 UNIT_COST를 더블 클릭하십시오. 항목을 클릭하면, 대화 상자가 나타납니다. 이를 간접적으로 입력했을 수도 있습니다.
12. **확인**을 클릭하여 보기 LOWER_COST를 작성하십시오.



이 새로운 보기의 내용을 표시하려면, LOWER_COST에서 마우스 오른쪽 버튼을 클릭하고 **빨리 보기**를 선택하십시오. 이 보기를 통해서는 원가가 단위 원가보다 낮은 행들만 표시됩니다.

SUPPLIER_NUMBER	ITEM_NUMBER	UNIT_COST	SUPPLIER_COST
9988	153047	10.00	8.00
2424	153047	10.00	9.00
1234	229740	1.50	1.00
3366	303476	2.00	1.50
3366	073956	20.00	17.00

iSeries Navigator를 사용한 데이터베이스 오브젝트 삭제

시스템에서 이러한 오브젝트를 작성했으면, 시스템 자원에 저장하기 위해 드롭시킬 수 있습니다. 이러한 타스크를 수행하려면 삭제 권한이 필요합니다.

주: 이러한 표에 정보를 보유하려면, 세 번째 라이브러리를 작성하고 표와 보기를 여기에 복사하십시오. 먼저, LIBRARY1에서 INVENTORY_LIST 표를 드롭하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. LIBRARY1을 펼치십시오.
3. INVENTORY_LIST에서 마우스 오른쪽 버튼을 클릭하고 **삭제**를 선택하거나 Delete 키를 누르십시오.

4. **오브젝트 삭제 확정** 대화 상자에서, **삭제**를 선택하십시오. INVENTORY_LIST 표가 드롭됩니다.

다음으로 LIBRARY1에서 SUPPLIERS를 삭제하십시오.

1. SUPPLIERS에서 마우스 오른쪽 버튼을 클릭하고 **삭제**를 선택하거나 Delete 키를 누르십시오.
2. 오브젝트 삭제 확정 대화 상자에서 **예**를 선택하십시오.
3. 보기 LOWER_COST가 SUPPLIERS에 종속되어 있으며 이 역시 삭제해야 하는지의 여부를 나타내는 새로운 대화 상자가 열립니다. **삭제**를 클릭하십시오.

SUPPLIERS 및 LOWER_COST가 삭제됩니다. 이제 LIBRARY1이 비어 있으며 이를 마우스 오른쪽 버튼으로 클릭하고 삭제를 선택하여 삭제하십시오. 오브젝트 삭제 확정 대화 상자에서 예를 선택하십시오. LIBRARY1이 삭제됩니다.

마지막으로 SAMPLELIB를 삭제하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. SAMPLELIB에서 마우스 오른쪽 버튼을 클릭하고 **삭제**를 선택하십시오.
3. 오브젝트 삭제 확정 대화 상자에서, **삭제**를 선택하십시오.
4. 표 INVENTORY_LIST와 보기 RECENT_ORDERS가 INVENTORY_LIST에 종속되어 있으며 이 역시 삭제해야 하는지의 여부를 나타내는 새로운 대화 상자가 열립니다. **예**를 클릭하십시오.

SAMPLELIB, INVENTORY_LIST 및 RECENT_ORDERS가 삭제됩니다.

iSeries Navigator 사용에 대한 자세한 정보는 305 페이지의 제 16 장『iSeries Navigator를 사용한 확장 데이터베이스 기능』을 참조하십시오.

제 4 장 DDL(Data Definition Language)

자료 정의문(DDL)은 데이터베이스 오브젝트를 작성, 변경 및 삭제할 수 있도록 하는 SQL 부분을 설명합니다. 이러한 데이터베이스 오브젝트에는 스키마, 표, 보기, 카탈로그, 색인 및 별명이 포함됩니다.

세부사항은 다음의 절을 참조하십시오.

- 『스키마 작성』
- 54 페이지의 『표 작성』
- 55 페이지의 『LIKE를 사용하여 표 작성』
- 55 페이지의 『AS를 사용하여 표 작성』
- 56 페이지의 『글로벌 임시 표 선언』
- 56 페이지의 『식별 열 작성 및 변경』
- 57 페이지의 『ROWID』
- 58 페이지의 『LABEL ON문을 사용한 설명 레이블 작성』
- 59 페이지의 『COMMENT ON을 사용하는 SQL 오브젝트 설명』
- 59 페이지의 『표 정의 변경』
- 62 페이지의 『ALIAS 이름 작성 및 사용』
- 63 페이지의 『보기 작성 및 사용』
- 65 페이지의 『색인 추가』
- 66 페이지의 『데이터베이스 설계의 카탈로그』
- 67 페이지의 『데이터베이스 오브젝트 드롭』

스키마 작성

스키마는 SQL 오브젝트의 논리 그룹을 제공합니다. 스키마는 라이브러리, 저널, 저널 리시버, 카탈로그로 구성되며 자료 사전도 선택할 수 있습니다. 표, 보기 및 시스템 오브젝트(예: 프로그램)를 시스템 라이브러리로 작성, 이동 또는 복원할 수 있습니다. SQL 스키마에 자료 사전이 포함되지 않은 경우, 모든 시스템 파일이 SQL 스키마로 작성되거나 이동될 수 있습니다. SQL 스키마에 자료 사전이 포함된 경우, 다음과 같은 사항이 가능합니다.

- 멤버가 하나인 소스 실제 파일이나 비소스 실제 파일이 SQL 스키마로 작성, 이동 또는 복원될 수 있습니다.
- 논리 파일은 자료 사전에서 설명될 수 없으므로 SQL 스키마에 배치할 수 없습니다.

여러개의 스키마를 작성하고 소유할 수 있습니다.

스키마는 CREATE SCHEMA문을 사용하여 작성됩니다. 예를 들면 다음과 같습니다.

DBTEMP라고 하는 스키마를 작성하십시오.

```
CREATE SCHEMA DBTEMP
```

용어 *콜렉션*을 *스키마*와 동의어로 사용할 수 있습니다. CREATE SCHEMA문에 대한 자세한 정보는 *SQL 참조서*의 CREATE SCHEMA를 참조하십시오.

표 작성

표는 행과 열로 구성된 자료의 2차원 배열로 시각화될 수 있습니다. 행은 하나 이상의 열이 들어 있는 수평 부분입니다. 열은 한 가지 유형의 자료를 한 행 이상 포함하고 있는 수직 부분입니다. 한 열의 모든 자료는 동일한 유형이어야 합니다. SQL에서 표는 키가 있거나 키가 없는 실제 파일입니다. 자료 유형에 대한 설명은 SQL 참조서의 자료 유형 주제를 참조하십시오.

표는 CREATE TABLE문을 사용하여 작성됩니다. 이 정의에는 이름 및 컬럼의 이름과 속성이 포함되어야 합니다. 정의에는 표의 다른 속성(예: 1차 키)이 포함될 수도 있습니다.

예 1: 사용자에게 관리 권한이 있으며 다음과 같은 열이 포함된 'INVENTORY'라는 이름의 표를 작성하십시오.

- 부품 번호: 1 - 9 999 사이의 정수로 널(null)은 사용할 수 없습니다.
- 설명: 0 - 24의 문자 길이
- 사용 가능 수량: 0 - 100000 사이의 정수

1차 키는 PARTNO입니다.

```
CREATE TABLE INVENTORY
                (PARTNO         SMALLINT     NOT NULL,
                 DESCR          VARCHAR(24 ),
                 QONHAND        INT,
                 PRIMARY KEY(PARTNO))
```

표에 제한사항 추가

새 표나 기존 표에 제한사항을 추가할 수 있습니다. CREATE TABLE 또는 ALTER TABLE문에 *ADD constraint* 절을 사용하여 고유 키 또는 1차 키, 참조 제한사항 또는 검사 제한사항 등을 추가할 수 있습니다. 예를 들어, 새로운 표 또는 기존의 표에 1차 키를 추가합니다. 다음 예는 ALTER TABLE문을 사용하여 기존의 표에 1차 키를 추가하는 설명입니다.

```
ALTER TABLE CORPDATA.DEPARTMENT
  ADD PRIMARY KEY (DEPTNO)
```

이 키를 고유 키로 하려면 PRIMARY 키워드를 UNIQUE로 바꾸십시오. 자세한 내용은 151 페이지의 제 10 장 『자료 무결성』을 참조하십시오.

LIKE를 사용하여 표 작성

또다른 표와 유사한 표를 작성할 수 있습니다. 즉, 기존의 표에서 모든 열 정의를 포함하는 표를 작성할 수 있습니다. 복사되는 정의는 다음과 같습니다.

- 열 이름(및 시스템 열 이름)
- 자료 유형, 정밀도 및 스케일
- CCSID
- 열 텍스트(LABEL ON)
- 열 머리말(LABEL ON)

LIKE 절이 표 이름 바로 다음에 나오고 괄호에 들어가 있지 않은 경우 다음과 같은 속성도 포함됩니다.

- 디폴트 값
- 널 사용 가능

지정된 표나 보기에 식별 열이 있고 새 표에 식별 열이 존재하도록 하려면 CREATE TABLE문에 INCLUDING IDENTITY를 지정해야 합니다. CREATE TABLE에 대한 디폴트 작동은 EXCLUDING IDENTITY입니다. 지정된 표 또는 보기가 SQL로 작성된 것이 아닌 실제 파일이거나 논리 파일인 경우 비 SQL 속성은 제거됩니다.

EMPLOYEE의 모든 열을 포함하는 표 EMPLOYEE2를 작성하십시오.

```
CREATE TABLE EMPLOYEE2 LIKE EMPLOYEE
```

CREATE TABLE LIKE에 대한 완벽한 내용은 *SQL 참조서*의 CREATE TABLE 주제를 참조하십시오.

AS를 사용하여 표 작성

CREATE TABLE AS는 SELECT문의 결과로부터 표를 작성합니다. SELECT문에 사용할 수 있는 모든 표현식은 CREATE TABLE AS문에 사용할 수 있습니다. 또한 선택한 하나 이상의 표로부터 데이터를 모두 포함시킬 수 있습니다.

예를 들어, DEPTNO = D11인 EMPLOYEE로부터 모든 열 정의를 포함하는 EMPLOYEE3이라는 표를 작성합니다.

```
CREATE TABLE EMPLOYEE3 AS
   (SELECT PROJNO, PROJNAME, DEPTNO
    FROM EMPLOYEE
    WHERE DEPTNO = 'D11') WITH NO DATA
```

지정된 표나 보기에 식별 열이 있고 새 표에 식별 열이 존재하도록 하려면 CREATE TABLE문에 INCLUDING IDENTITY를 지정해야 합니다. CREATE TABLE에 대한 디폴트 작동은 EXCLUDING IDENTITY입니다. WITH NO DATA 절은 데이터 없이 열 정의를 복사해야 함을 의미합니다. 새 표 EMPLOYEE3에 데이터를 포함시키려면 WITH DATA를 사용해야 합니다. SELECT 사용에 대한 자세한 정보는 69 페이지의 제 5 장 『SELECT문을 사용하여 자료 검색』을 참조하십시오. 지정된 조회에 SQL로 작성된 것이 아닌 실제 파일이나 논리 파일이 포함되어 있는 경우, 비SQL 결과 속성은 제거됩니다. CREATE TABLE AS에 대한 완벽한 내용은 *SQL 참조서*의 CREATE TABLE 주제를 참조하십시오.

글로벌 임시 표 선언

DECLARE GLOBAL TEMPORARY TABLE 문을 사용하여 현재 세션에 사용할 임시 표를 작성할 수 있습니다. 이 임시 테이브른 시스템 카탈로그에는 나타나지 않으므로 다른 세션과 공유할 수 없습니다. 세션을 종료할 때 표 행이 삭제되고 표는 드롭(drop)됩니다.

이 명령문의 구문은 CREATE TABLE과 유사합니다(LIKE 및 AS절 포함).

예를 들어, 임시 표 ORDERS를 작성합니다.

```
DECLARE GLOBAL TEMPORARY TABLE ORDERS
               (PARTNO   SMALLINT NOT NULL,
                DESCR    VARCHAR(24),
                QONHAND  INT)
       ON COMMIT DELETE ROWS
```

이 표는 QTEMP에 작성됩니다. 스키마 이름을 사용하여 표를 참조하려면 SESSION 또는 QTEMP 중 하나를 사용하십시오. 다른 표와 마찬가지로 이 표에 대해서도 SELECT, INSERT, UPDATE 및 DELETE 문을 발행할 수 있습니다. DROP TABLE 문을 발행하여 이 표를 드롭(drop)할 수 있습니다.

```
DROP TABLE ORDERS
```

완벽한 내용은 *SQL 참조서*의 DECLARE GLOBAL TEMPORARY TABLE 주제를 참조하십시오.

식별 열 작성 및 변경

새 행이 식별 열이 포함된 표에 추가될 때마다 새 행의 식별 열 값은 시스템에 의해 증가(또는 감소)됩니다. SMALLINT, INTEGER, BIGINT, DECIMAL 또는 NUMERIC 유형의 열만 식별 열로 작성할 수 있습니다. 표당 하나의 식별 열만 허용됩니다. 표 정의를 변경할 때 추가하려는 단 하나의 열만 식별 열로 지정할 수 있으며 기존 열은 식별 열이 될 수 없습니다.

표를 작성할 때 표의 열 하나를 식별 열로 정의할 수 있습니다. 예를 들어, ORDERNO, SHIPPED_TO 및 ORDER_DATEF라는 세 개의 열로 표 ORDERS를 작성합니다. 식별 열로 ORDERNO를 정의하십시오.

```
CREATE TABLE ORDERS
   (ORDERNO SMALLINT NOT NULL
   GENERATED ALWAYS AS IDENTITY
   (START WITH 500
   INCREMENT BY 1
   CYCLE),
   SHIPPED_TO VARCHAR (36) ,
   ORDER_DATE DATE)
```

이 열은 시작 값 500으로 정의하고 새 행이 삽입될 때마다 1씩 증가하여 최대값에 도달하면 다시 처음으로 돌아갑니다. 이 예에서, 식별 열에 대한 최대값은 자료 유형에 대한 최대값입니다. 자료 유형이 SMALLINT로 정의되어 있기 때문에, ORDERNO에 할당할 수 있는 값의 범위는 500 - 32767입니다. 열 값이 32767에 도달하면 500에서 다시 시작합니다. 임의의 열에 500이 지정되고 고유 키가 식별 열에 지정되면 중복 키 오류가 리턴됩니다. 다음 삽입 시 501을 사용하려 할 것입니다. 해당 식별 열에 대하여 지정된 고유 키가 없는 경우 표가 몇 번 표시되든 500이 다시 사용됩니다.

더 큰 범위의 값의 경우 열을 INTEGER 또는 BIGINT로 지정할 수 있습니다. 식별 열 값을 감소시키려면 INCREMENT 옵션에 음수 값을 지정합니다. 또한 MINVALUE 및 MAXVALUE를 사용하여 정확한 숫자 범위를 지정할 수 있습니다.

ALTER TABLE문을 사용하여 기존 식별 열의 속성을 수정할 수 있습니다. 예를 들어 새 값으로 식별 열을 다시 시작하려면 다음과 같이 하십시오.

```
ALTER TABLE ORDER
  ALTER COLUMN ORDERNO
  RESTART WITH 1
```

또한 열에서 식별 속성을 드롭(drop)할 수도 있습니다.

```
ALTER TABLE ORDER
  ALTER COLUMN ORDERNO
  DROP IDENTITY
```

ORDERNO 열은 SMALLINT로서 남아 있지만, 식별 속성이 드롭(drop)됩니다. 시스템은 더 이상 이 열에 값을 생성하지 않습니다.

ROWID

ROWID를 사용하는 것도 시스템이 표의 열에 고유한 값을 지정하도록 하는 또 다른 방법입니다. ROWID는 식별 열과 유사하지만 숫자 열의 속성이라기 보다는 별도의 데이터 유형입니다. 식별 열 예와 유사한 표를 작성하려면 다음을 수행하십시오.

```
CREATE TABLE ORDERS
   (ORDERNO ROWID
   GENERATED ALWAYS,
   SHIPPED_TO VARCHAR (36) ,
   ORDER_DATE DATE)
```

LABEL ON문을 사용한 설명 레이블 작성

때때로 표 이름, 열 이름, 보기명, 별명 또는 SQL 패키지명이 표의 대화식 화면상에 나타난 자료를 명확하게 정의하지 않는 경우가 있습니다. LABEL ON문을 사용하여 표 이름, 열 이름, 보기명, 별명 또는 SQL 패키지명에 대한 보다 자세한 레이블을 작성할 수 있습니다. 이들 레이블은 LABEL 열의 SQL 카탈로그에서 볼 수 있습니다.

LABEL ON문은 다음과 같습니다.

```
 LABEL ON
TABLE CORPDATA.DEPARTMENT IS 'Department Structure Table'

LABEL ON
  COLUMN CORPDATA.DEPARTMENT.ADMRDEPT IS 'Reports to Dept.'
```

이러한 명령문 수행 후 표 DEPARTMENT는 텍스트 설명을 *Department Structure Table*로 표시하고 열 *ADMRDEPT*는 표제 *Reports to Dept*를 표시합니다. 표, 보기, SQL 패키지 및 열 텍스트에 대한 레이블은 50문자 이상이 될 수 없으며 열 표제에 대한 레이블은 60문자 이상이 될 수 없습니다. 다음은 열 머리말에 대한 LABEL ON 문의 예입니다.

LABEL ON문은 열 표제 1과 열 표제 2를 제공합니다.

```
*...+....1....+....2....+....3....+....4....+....5....+....6..*
LABEL ON COLUMN CORPDATA.EMPLOYEE.EMPNO IS
          'Employee            Number'
```

LABEL ON문은 SALARY열에 대해 3 레벨의 열 표제를 제공합니다.

```
*...+....1....+....2....+....3....+....4....+....5....+....6..*
LABEL ON COLUMN CORPDATA.EMPLOYEE.SALARY IS
           'Yearly              Salary              (in dollars)'
```

이 LABEL ON문은 SALARY에 대한 열 표제를 제거합니다.

```
*...+....1....+....2....+....3....+....4....+....5....+....6..*
LABEL ON COLUMN CORPDATA.EMPLOYEE.SALARY IS ''
```

두 레이블이 지정된 DBCS 열 표제의 예제입니다.

```
*...+....1....+....2....+....3....+....4....+....5....+....6..*
LABEL ON COLUMN CORPDATA.EMPLOYEE.SALARY IS
           '<AABBCCDD>          <EEFFGG>'
```

LAVEL ON문은 EDLEVEL 열에 대해 열 텍스트를 제공합니다.

```
*...+....1....+....2....+....3....+....4....+....5....+....6..*
LABEL ON COLUMN CORPDATA.EMPLOYEE.EDLEVEL TEXT IS
   'Number of years of formal education'
```

LABEL ON문에 대한 자세한 정보는 SQL 참조서의 LABEL ON문을 참조하십시오.

COMMENT ON을 사용하는 SQL 오브젝트 설명

표, 보기, 색인, 패키지, 프로시듀어, 매개변수, 사용자 정의 유형, 기능 또는 트리거와 같은 SQL 오브젝트를 작성한 후에 오브젝트의 목적, 사용자, 일반적이지 않거나 특별한 경우 등과 같이 추후 참조를 위해 이에 대한 정보를 제공할 수 있습니다. 또한 표나 보기의 각 열에 대한 유사한 정보를 포함할 수 있습니다. 사용자 주석은 2000바이트를 초과할 수 없습니다. COMMENT ON문에 대한 자세한 내용은 *SQL 참조서* 책에서 COMMENT ON을 참조하십시오.

사용자명이 열이나 오브젝트의 내용을 명확하게 나타내지 않는 경우, 특히 주석이 유용합니다. 그러한 경우 열이나 오브젝트의 특정한 내용을 설명하기 위해 주석을 사용합니다.

COMMENT ON을 사용하는 예는 다음과 같습니다.

```
COMMENT ON TABLE CORPDATA.EMPLOYEE IS
   'Employee table.  Each row in this table represents
    one employee of the company.'
```

COMMENT ON문을 실행한 후 주석 작성

표에 대해 COMMENT ON문을 수행하고 나면, 사용자의 주석은 SYSTABLES의 *LONG_COMMENT* 열에 저장됩니다. 다른 오브젝트에 대한 주석은 해당되는 카탈로그 표의 LONG_COMMENT 열에 저장됩니다(표시된 행이 이미 주석에 들어 있으면 기존 주석이 새로운 주석으로 대체됩니다). 다음 예제는 이전 예제에서 COMMENT ON문에 의해 추가된 주석입니다.

```
SELECT LONG_COMMENT
 FROM CORPDATA.SYSTABLES
  WHERE NAME = 'EMPLOYEE'
```

표 정의 변경

표 정의를 변경하면 새로운 열을 추가할 수 있고 기존 열 정의(예: 길이, 디폴트 값)를 변경하며 기존 열을 제거하고 제한조건을 추가 및 제거할 수 있습니다. 표 정의는 SQL ALTER TABLE문을 사용하여 변경됩니다.

하나의 ALTER TABLE문이 있는 모든 제한 조건을 제거하거나 열을 추가, 변경 또는 제거할 수 있습니다. 그러나, 하나의 열이 ADD COLUMN, ALTER COLUMN 및 DROP COLUMN절에 단 한번만 참조됩니다. 즉, 열을 추가한 후에 그 열을 동일한 ALTER TABLE문에서 변경하는 조치는 허용되지 않습니다.

자세한 내용은 다음의 주제를 참조하십시오.

- 60 페이지의『열 추가』

- 『열 변경』
- 『허용되는 변환』
- 61 페이지의『열 삭제』
- 62 페이지의『ALTER TABLE문의 조작 순서』

열 추가

SQL ALTER TABLE문의 ADD COLUMN절을 사용하여 표에 열을 추가할 수 있습니다.

새로운 열이 표에 추가될 때 그 열은 기존의 모든 행에 대한 디폴트 값으로 초기설정됩니다. NOT NULL이 지정되는 경우에도 디폴트가 지정되어야 합니다.

변경된 표는 최대 8000개의 열로 구성될 수 있습니다. 열의 바이트 수 합은 32,766 이하여야 하며 VARCHAR이나 VARGRAPHIC 열이 지정되는 경우에는 32,740 이하여야 합니다. LOB 열을 지정한 경우, 열의 레코드 자료 바이트 수 합은 328 640 이하여야 합니다.

열 변경

ALTER TABLE문의 ALTER COLUMN절을 사용하여 표에서 열 정의를 변경할 수 있습니다. 기존 열의 자료 유형을 변경할 경우, 기존의 속성과 새로운 속성이 서로 호환되어야 합니다.『허용되는 변환』은 호환될 수 있는 속성과의 변환을 보여줍니다.

더 긴 길이의 자료 유형으로 변환될 때는 자료에 해당되는 채움 문자가 채워집니다. 더 짧은 길이의 자료 유형으로 변환될 때는 자료가 잘려서 손실될 수도 있습니다. 사용자가 요구를 확정하도록 조회 메세지가 프롬트됩니다.

널 값을 허용하지 않은 열이 있을 때 널 값을 허용하도록 그 열을 변경하려면 DROP NOT NULL절을 사용하십시오. 널 값을 허용하는 열이 있을 때 그 열에 널 값을 사용하지 못하도록 금지하려면 SET NOT NULL절을 사용하십시오. 해당 열의 기존값이 널이면, ALTER TABLE이 실행되지 않으므로 결과는 SQLCODE -190입니다.

허용되는 변환

표 2. 허용되는 변환

FROM 자료 유형	**TO 자료 유형**
10진	숫자
10진	Bigint, 정수, Smallint
10진	부동 소수
숫자	10진
숫자	Bigint, 정수, Smallint
숫자	부동 소수
Bigint, 정수, Smallint	10진

표 2. 허용되는 변환 (계속)

FROM 자료 유형	**TO 자료 유형**
Bigint, 정수, Smallint	숫자
Bigint, 정수, Smallint	부동 소수
부동 소수	숫자
부동 소수	Bigint, 정수, Smallint
문자	DBCS 개방
문자	UCS-2 그래픽
DBCS 개방	문자
DBCS 개방	UCS-2 그래픽
DBCS 선택	문자
DBCS 선택	DBCS 개방
DBCS 선택	UCS-2 그래픽
DBCS 전용	DBCS 개방
DBCS 전용	DBCS 그래픽
DBCS 전용	UCS-2 그래픽
DBCS 그래픽	UCS-2 그래픽
UCS-2 그래픽	문자
UCS-2 그래픽	DBCS 개방
UCS-2 그래픽	DBCS 그래픽
고유한 유형	소스 유형
소스 유형	고유한 유형

기존의 열을 수정할 때는 사용자가 지정하는 속성만 변경됩니다. 기타 다른 속성들은 변경되지 않고 남아 있습니다. 예를 들어 다음과 같은 표 정의가 있다고 가정합니다.

```
CREATE TABLE EX1 (COL1 CHAR(10) DEFAULT 'COL1',
                  COL2 VARCHAR(20) ALLOCATE(10) CCSID 937,
                  COL3 VARGRAPHIC(20) ALLOCATE(10)
                       NOT NULL WITH DEFAULT)
```

여기에 다음과 같은 ALTER TABLE문을 수행하면

```
ALTER TABLE EX1 ALTER COLUMN COL2 SET DATA TYPE VARCHAR(30)
                ALTER COLUMN COL3 DROP NOT NULL
```

여전히 COL2의 할당 길이는 10이고 CCSID는 937이며 COL3 역시 할당 길이는 변함없이 10입니다.

열 삭제

ALTER TABLE문의 DROP COLUMN절을 사용하여 열을 삭제할 수 있습니다.

열을 드롭하면 표 정의에서 해당 열이 삭제됩니다. CASCADE가 지정된 경우에는 해당 열에 종속적인 보기, 색인 및 제한조건들도 함께 제거됩니다. RESTRICT가 지정되고, 보기, 색인 및 제한조건이 열에 종속적인 경우, 열은 제거되지 않고 SQLCODE -196이 발행됩니다.

```
ALTER TABLE DEPT
  DROP COLUMN NUMDEPT
```

ALTER TABLE문의 조작 순서

ALTER TABLE문은 다음과 같이 일련의 단계로 수행됩니다.

1. 드롭 제한조건
2. RESTRICT 옵션이 지정된 열들을 드롭
3. 열 정의 변경(CASCADE 옵션이 지정된 열의 추가 및 드롭 포함)
4. 제한조건의 추가

각 단계 내에서, 사용자가 절을 지정하는 순서는 한가지 예외를 제외하고는 수행되는 순서입니다. 열이 드롭될 경우에는 열 정의가 추가되거나 변경되기 전에 해당 조작이 논리적으로 수행됩니다. 이러한 경우에는 ALTER TABLE문의 결과로 레코드 길이가 증가합니다.

ALIAS 이름 작성 및 사용

여러 멤버로 구성된 실제 파일에 대해 기존의 표나 보기를 참조할 경우, 별명을 작성하여 파일 대체를 사용하지 않을 수 있습니다. SQL CREATE ALIAS문을 사용하여 이를 수행할 수 있습니다.

다음에 대한 별명을 작성할 수 있습니다.

- 표 또는 보기
- 표의 *멤버*

표 별명은 특정 멤버의 이름을 포함하는 파일에 대해 이름을 정의합니다. 표 이름을 사용하는 것과 같은 방법으로 SQL문에 이 별명을 사용할 수 있습니다. 대체와 달리 별명은 드롭(drop)될 때까지 존재하는 오브젝트입니다.

예를 들어 멤버 MBR1 및 MBR2와 함께 다중 멤버 파일 MYLIB.MYFILE이 있는 경우, 별명은 두 번째 멤버에 대한 별명을 작성할 수 있고 SQL에 의해 쉽게 조회될 수 있습니다.

```
CREATE ALIAS MYLIB.MYMBR2_ALIAS FOR MYLIB.MYFILE (MBR2)
```

별명 MYLIB.MYMBR2_ALIAS가 다음의 INSERT문에 지정되는 경우, 값은 MYLIB.MYFILE의 멤버 MBR2에 삽입됩니다.

```
INSERT INTO MYLIB.MYMBR2_ALIAS VALUES('ABC', 6)
```

또한 별명을 DDL문에 지정할 수도 있습니다. 별명 MYLIB.MYALIAS가 존재하고 이것이 표 MYLIB.MYTABLE에 대한 별명이라고 가정합니다. 다음의 DROP문은 표 MYLIB.MYTABLE을 드롭할 것입니다.

```
DROP TABLE MYLIB.MYALIAS
```

대신 별명을 드롭하려면 ALIAS 키워드를 DROP문에 지정하십시오.

```
DROP ALIAS MYLIB.MYALIAS
```

보기 작성 및 사용

보기는 하나 이상의 표 또는 보기에 있는 자료에 액세스하는 데 사용될 수 있습니다. 이는 SELECT문을 사용하여 수행됩니다. SELECT문 사용에 대한 세부사항은 69 페이지의 제 5 장『SELECT문을 사용하여 자료 검색』을 참조하십시오. 보기의 경우, ORDER BY절을 사용할 수 없습니다.

예를 들어, 단지 모든 관리자의 성과 부서를 선택하는 보기를 작성하려면 다음을 지정하십시오.

```
CREATE VIEW CORPDATA.EMP_MANAGERS AS
   SELECT LASTNAME, WORKDEPT FROM  CORPDATA.EMPLOYEE
   WHERE JOB = 'MANAGER'
```

선택 리스트가 표현식, 기능, 상수 또는 레지스터와 같이 열 이외의 요소를 포함하며 AS절이 열을 명명하는데 사용되지 않았을 경우 보기에 대해 열 리스트가 지정되어야 합니다. 다음 예에서 보기의 열은 LASTNAME과 YEARSOFSERVICE입니다.

```
CREATE VIEW CORPDATA.EMP_YEARSOFSERVICE
    (LASTNAME, YEARSOFSERVICE) AS
   SELECT LASTNAME, YEARS (CURRENT DATE - HIREDATE)
   FROM CORPDATA.EMPLOYEE
```

이전 보기는 보기의 열을 명명하기 위해 선택 리스트에서 AS절을 사용하여 정의될 수도 있습니다. 예를 들면 다음과 같습니다.

```
CREATE VIEW CORPDATA.EMP_YEARSOFSERVICE AS
   SELECT LASTNAME,
          YEARS (CURRENT_DATE - HIREDATE) AS YEARSOFSERVICE
   FROM CORPDATA.EMPLOYEE
```

일단 보기를 작성하면, 표 이름과 유사한 SQL문에서만 사용할 수 있습니다. 또한 기본 표에서도 변경할 수 있습니다.

보기를 작성할 때에는 다음 제한 조건을 반드시 고려해야 합니다.

- 읽기 전용 보기를 사용하여 자료를 변경, 삽입 또는 삭제할 수 없습니다. 보기는 다음 가운데 하나가 포함되는 경우 읽기 전용입니다.
  - 첫 번째 FROM절이 두 개 이상의 표(결합)를 식별합니다.
  - 첫 번째 FROM절이 읽기 전용 보기를 식별합니다.

  - 첫 번째 FROM절이 사용자 정의 표 함수를 식별합니다.
  - 첫 번째 SELECT절에 SQL 열 함수(SUM, MAX, MIN, AVG, COUNT, STDDEV, COUNT_BIG 또는 COUNT) 중 하나가 포함됩니다.
  - 첫 번째 SELECT절이 키워드 DISTINCT를 지정합니다.
  - 외부 부속 선택에 GROUP BY나 HAVING절이 포함됩니다.
  - 외부 부속 선택에 UNION절이 포함됩니다.
  - 가장 외부에 있는 부속 선택의 기본 오브젝트가 있는 부속 조회와 부속 조회의 표는 동일한 표입니다.

    위의 경우에는 SQL SELECT문에 의해 보기에서 자료를 얻을 수 있으나 INSERT, UPDATE 또는 DELETE와 같은 명령문은 사용할 수 없습니다.
- 다음의 경우에는 보기를 삽입할 수 없습니다.
  - 보기가 기초로 하는 표가 디폴트 값이 없고 널(null)을 허용하지 않으며 보기에 없는 열을 갖는 경우
  - 보기가 표현식, 상수, 기능 또는 특수 레지스터로부터 만들어진 열을 가지며 그 열이 INSERT 열 리스트에 지정된 경우
  - 보기 작성시 WITH CHECK OPTION이 지정되었으며 행이 선택 기준에 일치하지 않은 경우
- 사용자는 표현식, 상수, 기능 또는 특수 레지스터로부터 만들어진 보기의 열을 갱신할 수 없습니다.
- FOR UPDATE OF, FOR READ ONLY, FETCH FIRST *n* ROWS, ORDER BY, OPTIMIZE FOR n ROWS 또는 보기 정의에 있는 분리 절을 사용할 수 없습니다.

보기는 CREATE VIEW문 수행시 효과적인 정렬 순서로 작성됩니다. 정렬 순서는 CREATE VIEW문 부속 선택에서 모든 문자와 UCS-2 그래픽 비교에 적용됩니다. 정렬 순서에 대한 자세한 내용은 129 페이지의 제 8 장 『SQL 정렬 순서』를 참조하십시오.

보기는 UNION 연산자를 사용하여 작성할 수 없습니다. 자세한 내용은 『UNION을 사용하여 보기 작성』을 참조하십시오.

보기는 또한 보기를 통해 자료가 삽입되거나 갱신될 때 수행되어야 하는 검사의 레벨을 지정하기 위해 WITH CHECK OPTION을 사용하여 작성될 수도 있습니다. 자세한 내용은 162 페이지의 『보기에서 WITH CHECK OPTION』을 참조하십시오.

UNION을 사용하여 보기 작성

UNION 키워드를 사용하면, 두 개 이상의 부속 선택을 결합하여 단일 보기를 형성할 수 있습니다. 예를 들면 다음과 같습니다.

```
CREATE VIEW D11_EMPS_PROJECTS AS
  (SELECT EMPNO
        FROM CORPDATA.EMPLOYEE
       WHERE WORKDEPT = 'D11'
 UNION
 SELECT EMPNO
           FROM CORPDATA.EMPPROJACT
   WHERE PROJNO = 'MA2112' OR
      PROJNO = 'MA2113' OR
        PROJNO = 'AD3111')
```

결과는 다음과 같은 데이터와 함께 보기에 표시됩니다.

표 3. UNION 결과로 보기 작성

EMPNO
000060
000150
000160
000170
000180
000190
000200
000210
000220
000230
000240
200170
200220

UNION에 대한 자세한 내용은은 97 페이지의 『UNION 키워드를 사용하는 부속 선택 결합』을 참조하십시오.

색인 추가

색인을 사용하여 자료를 정렬하고 선택할 수 있습니다. 또한, 색인을 사용하면 자료를 빨리 검색할 수 있어서 조회 성능이 더 좋아집니다.

SQL CREATE INDEX문을 사용하여 색인을 작성하십시오. 다음의 예는 CORPDATA.EMPLOYEE 표의 열 *LASTNAME*에 색인을 작성합니다.

```
CREATE INDEX CORPDATA.INX1 ON CORPDATA.EMPLOYEE (LASTNAME)
```

색인을 임의 수 만큼 작성할 수 있습니다. 그러나 색인이 시스템으로 관리되기 때문에 많은 색인은 시스템 성능 측면에 있어서 부적절한 영향을 미칠 수 있습니다. 색인 및 조회 성능에 대한 자세한 내용은 *데이터베이스 성능 및 조회 최적화* 정보에서 SQL 색인의 효과적인 사용을 참조하십시오.

한 색인 유형, 코드화된 벡터 색인은 사용하면 병렬로 더 쉽게 프로세스할 수 있는 빠른 스캔이 가능하게 됩니다. SQL CREATE INDX문을 사용하여 코드화된 벡터 색인을 작성합니다. accelerating your queries with encoded vector indexes  에 대한 자세한 내용은 iSeries용 DB2 웹페이지로 가십시오.

기존의 색인과 똑같은 속성을 가진 색인이 작성되면 새로운 색인은 기존 색인의 2진 트리를 공유합니다. 그렇지 않은 경우에는 다른 2진 트리가 작성됩니다. 새로운 색인의 행 수가 1행 이상 줄어든 것을 제외하고는 새로운 색인의 속성이 다른 색인과 똑같은 경우에도 마찬가지로 다른 2진 트리가 작성됩니다. 그것이 여전히 작성되는 이유는 추가 열로 인해 색인이 커서 또는 그 추가 열을 갱신하는 UPDATE문에 의해 사용되는 것을 방해받을 수 있기 때문입니다.

색인은 CREATE VIEW문 수행시 효과적인 정렬 순서로 작성됩니다. 정렬 순서는 해당 색인의 모든 SBCS 문자 필드와 UCS-2 그래픽 필드에 적용됩니다. 정렬 순서에 대한 자세한 내용은 129 페이지의 제 8 장『SQL 정렬 순서』를 참조하십시오.

데이터베이스 설계의 카탈로그

카탈로그는 스키마를 작성할 때 자동으로 작성됩니다. 또한 언제나 QSYS2 라이브러리에 상주하는 시스템 전반적인 카탈로그가 있습니다. SQL 오브젝트가 스키마에 작성되면 시스템 카탈로그 표와 스키마의 카탈로그 표에 모두 정보가 추가됩니다. SQL 오브젝트가 라이브러리에 작성되면 QSYS2 카탈로그만 갱신됩니다. DECLARE GLOBAL TEMPORARY TABLE을 사용하여 작성된 표는 카탈로그에 추가되지 않습니다. 카탈로그에 대한 자세한 내용은 SQL 참조서를 참조하십시오.

다음의 예에서 보여주는 바와 같이 사용자는 카탈로그 정보를 표시할 수 있습니다. 카탈로그 정보를 INSERT, DELETE 또는 UPDATE할 수 없습니다. 다음 예제를 수행하려면 카탈로그 보기에 SELECT 권한이 있어야 합니다.

표에 관한 카탈로그 정보 확보

SYSTABLES에는 SQL 스키마의 모든 표와 보기에 대한 행이 들어 있습니다. 이 행은 오브젝트가 표 또는 보기인지의 여부, 오브젝트명, 오브젝트의 소유자, 오브젝트가 들어 있는 SQL 스키마 등을 알려줍니다.

다음 샘플 명령문은 CORPDATA.DEPARTMENT 표에 대한 정보를 표시합니다.

```
SELECT *
  FROM CORPDATA.SYSTABLES
  WHERE NAME = 'DEPARTMENT'
```

열에 관한 카탈로그 정보 확보

SYSCOLUMNS에는 스키마의 모든 표와 보기의 각 열에 대한 행이 들어 있습니다.

다음 샘플 명령문은 CORPDATA.DEPARTMENT 표의 모든 열 이름을 표시합니다.

```
SELECT *
  FROM CORPDATA.SYSCOLUMNS
  WHERE TBNAME = 'DEPARTMENT'
```

이전 샘플 명령문의 결과는 표에 있는 각 열에 대한 정보의 행으로 나타납니다. 정보의 폭이 표시 화면을 초과하기 때문에 일부 정보는 표시되지 않을 수도 있습니다.

각 열에 대한 자세한 정보를 보려면 다음과 같이 SELECT문을 지정하십시오.

```
SELECT NAME, TBNAME, COLTYPE, LENGTH, DEFAULT
  FROM CORPDATA.SYSCOLUMNS
  WHERE TBNAME = 'DEPARTMENT'
```

각 열의 열 이름 외에 SELECT문은 다음을 표시합니다.

- 열이 있는 표 이름
- 열의 자료 유형
- 열의 길이 속성
- 열의 디폴트 값 허용 여부

결과는 다음과 같습니다.

NAME	TBNAME	COLTYPE	LENGTH	DEFAULT
DEPTNO	DEPARTMENT	CHAR	3	N
DEPTNAME	DEPARTMENT	VARCHAR	29	N
MGRNO	DEPARTMENT	CHAR	6	Y
ADMRDEPT	DEPARTMENT	CHAR	3	N

데이터베이스 오브젝트 드롭

DROP문은 오브젝트를 삭제합니다. 요구되는 조치에 따라 해당 오브젝트에 직접 또는 간접적으로 종속되는 모든 오브젝트는 삭제되거나 드롭(drop)되는 것을 방지할 수도 있습니다. 예를 들어 표를 드롭(drop)하는 경우 그 표와 연관된 별명, 제한사항, 트리거, 보기 또는 색인도 드롭됩니다. 오브젝트가 삭제될 때마다 카탈로그에서 설명이 삭제됩니다.

예를 들어, 표 EMPLOYEE를 드롭(drop)하여 다음 명령문을 발행하십시오.

```
DROP TABLE EMPLOYEE RESTRICT
```

자세한 내용은 SQL 참조서의 DROP문을 참조하십시오.

제 5 장 SELECT문을 사용하여 자료 검색

SELECT문을 사용하여 데이터베이스에서 자료를 검색할 수 있습니다. 이 절에서는 다음을 설명합니다.

- 『SELECT문을 사용하여 자료 조회』

SQL문은 하나의 행 또는 여러 행으로 작성할 수 있습니다. 컴파일된 프로그램의 SQL문의 경우, 행 계속에 대한 규칙은 호스트 언어(프로그램이 작성된 언어)의 규칙과 동일합니다.

주:

1. 여기에서 설명되는 SQL문은 SQL 표, 보기, 데이터베이스 실제 및 논리 파일에서 실행시킬 수 있습니다. 표, 보기 및 파일은 스키마 또는 라이브러리 중 하나에 있을 수 있습니다.
2. SQL문에 지정된 문자 스트링(WHERE 또는 VALUES절에 사용된 것과 같은)은 대소문자가 구분됩니다. 즉, 대문자는 대문자로 입력되어야 하고 소문자는 소문자로 입력되어야 합니다.

   ```
   WHERE ADMRDEPT='a00'      (결과를 리턴하지 않음)

   WHERE ADMRDEPT='A00'      (유효한 부서 번호 리턴)
   ```

   대문자 및 소문자가 동일한 문자로 취급되는 곳에서 공유 가중치 정렬 순서가 사용된다면 비교시에 대소문자를 구분할 수 없습니다.

SELECT문을 사용하여 자료 조회

다양한 명령문과 절을 사용하여 자료를 조회할 수 있습니다. 이를 수행하는 한 가지 방법은 프로그램에서 SELECT문을 사용하여 특정 행(예: 사원에 대한 행)을 검색하는 것입니다. 또한, 이 예에서 특정 방법으로 자료를 모으기 위해 다양한 절이 사용됩니다. SQL은 특정 방식으로 자료를 모으기 위해 조회를 조정하는 몇 가지 방법을 제공합니다. 이러한 방법은 다음과 같습니다.

- 71 페이지의『WHERE 절을 사용하는 탐색 조건 지정』
- 74 페이지의『GROUP BY절』
- 76 페이지의『HAVING절』
- 77 페이지의『ORDER BY절』

여기에 표시된 형식과 구문은 아주 기본이 되는 내용입니다. SELECT문은 이 장에 표시된 예보다 더 다양할 수 있습니다. SELECT문에는 다음이 포함될 수 있습니다.

1. 포함하고자 하는 각 열 이름

2. 자료가 들어 있는 표 또는 보기명
3. 원하는 정보가 들어 있는 행을 고유하게 식별하는 탐색 조건
4. 자료 그룹화에 사용되는 각 열 이름
5. 원하는 정보가 들어 있는 그룹을 고유하게 식별하는 탐색 조건
6. 복제된 행 중에서 특정 행이 리턴될 수 있도록 하는 결과의 순서

SELECT문은 다음과 같습니다.

```
SELECT column names
  FROM table or view name
  WHERE search condition
  GROUP BY column names
  HAVING search condition
  ORDER BY column-name
```

SELECT 및 FROM절이 지정되어야 합니다. 다른 절은 생략 가능합니다.

SELECT절의 경우, 검색하고자 하는 각 열의 이름을 지정합니다. 예를 들면 다음과 같습니다.

```
SELECT EMPNO, LASTNAME, WORKDEPT
  ⋮
```

하나의 열 또는 8000 열이 검색되도록 지정할 수 있습니다. 명명한 각 열값은 SELECT 절에서 지정된 순서로 검색됩니다.

모든 열을 검색하려면(표의 정의에 표시되는 것과 동일한 순서로) 열을 명명하는 대신 다음과 같이 별표(*)를 사용하십시오.

```
SELECT *
  ⋮
```

FROM 절은 데이터를 선택할 *대상*이 되는 표를 지정합니다. 둘 이상의 표에서 열을 선택할 수 있습니다. SELECT를 발행하면, FROM절을 지정해야 합니다. 다음 명령문을 발행하십시오.

```
SELECT *
       FROM EMPLOYEE
```

결과는 EMPLOYEE 표의 열과 행이 모두 표시됩니다.

또한 SELECT 리스트는 표현식(제한조건, 특수 레지스터 및 스칼라 부속 선택)을 포함할 수도 있습니다. 결과 열에 이름을 부여하려면 AS 절을 사용할 수 있습니다. 예를 들어, 다음 명령문을 발행합니다.

```
SELECT LASTNAME, SALARY * .05 AS RAISE
       FROM EMPLOYEE
       WHERE EMPNO = '200140'
```

이 명령문의 결과는 다음과 같습니다.

표 4. 조회에 대한 결과

LASTNAME	RAISE
NATZ	1421

SQL이 탐색 조건을 만족시키는 한 행을 찾을 수 없으면 +100인 SQLCODE가 리턴됩니다.

사용자의 선택문 수행시 SQL이 오류를 발견하면 음의 SQLCODE가 리턴됩니다. SQL이 결과보다 더 많은 호스트 변수를 발견하면 +326이 리턴됩니다.

리턴된 자료 규정에 대한 정보는 『WHERE 절을 사용하는 탐색 조건 지정』을 참조하십시오.

WHERE 절을 사용하는 탐색 조건 지정

WHERE절은 검색, 갱신 또는 삭제하려는 행을 식별하기 위한 탐색 조건을 지정합니다. SQL문으로 처리하는 행의 수는 WHERE절 **탐색 조건**을 만족하는 행의 수에 따라 달라집니다. 탐색 조건은 하나 이상의 **술부**로 이루어 집니다. 술부는 표의 지정된 열에 SQL 적용 여부 테스트를 정의합니다. 술부에 대한 자세한 정보는 84 페이지의 『복합 탐색 조건 수행』을 참조하십시오.

다음 예제에서 WORKDEPT = 'C01'은 술부이고 WORKDEPT와 'C01'은 표현식이며 등호(=)는 비교 연산자입니다. 문자값은 어포스트로피(')로 묶여지고 숫자값은 묶이지 않는다는 점에 유의하십시오. 이는 SQL문 내에서 상수값이 코딩되는 곳마다 모든 상수값에 적용됩니다. 예를 들어 부서 번호가 C01인 행을 지정하려면 다음과 같이 입력해야 합니다.

```
... WHERE WORKDEPT = 'C01'
```

이 경우, 탐색 조건은 하나의 술부로 구성됩니다. 즉 WORKDEPT = 'C01'과 같습니다.

WHERE를 좀 더 알아보기 위해 SELECT 문에 넣어 봅시다. CORPDATA.DEPARTMENT 표에 나열된 각 부서에 고유 부서 번호가 있다고 가정합니다. 부서 C01에 대해 CORPDATA.DEPARTMENT 표에서 부서명과 관리자 번호를 검색할 수 있습니다. 다음 명령문을 발행하십시오.

```
SELECT DEPTNAME, MGRNO
  FROM CORPDATA.DEPARTMENT
       WHERE DEPTNO = 'C01'
```

이 명령문이 수행되면 결과가 한 행으로 나타납니다.

표 5. 결과표

DEPTNAME	MGRNO
INFORMATION CENTER	000030

탐색 조건에 문자 또는 UCS-2 그래픽 열 술부가 있는 경우, 조회 수행시 유효한 정렬 순서가 이러한 술부에 적용됩니다. 정렬 순서 및 선택에 대한 자세한 내용은 129 페이지의 제 8 장『SQL 정렬 순서』를 참조하십시오. 정렬 순서가 사용되지 않는 경우 문자 상수를 비교될 열이나 표현식과 일치되도록 대문자 또는 소문자로 지정해야 합니다.

WHERE절에 대한 자세한 내용은 다음 절을 참조하십시오.

- 『WHERE 절에서 표현식』
- 74 페이지의『비교 연산자』
- 74 페이지의『NOT 키워드』

WHERE 절에서 표현식

WHERE절 내의 표현식은 다른 어떤 것과 비교하려는 것을 명명하거나 지정합니다. 각 표현식은 SQL에 의해 평가될 때 문자 스트링, 날짜/시간/시간소인 또는 숫자값이 됩니다. 다음과 같은 표현식을 지정할 수 있습니다.

- **열 이름**은 열을 명명합니다. 예를 들면 다음과 같습니다.

  ```
  ... WHERE EMPNO = '000200'
  ```

  *EMPNO*는 6바이트 문자 값으로 정의된 열을 명명합니다. 등호 비교(즉, X = Y 또는 X <> Y)가 문자 자료에 대해 수행될 수 있습니다. 다른 유형의 비교도 문자 자료에 대해 평가될 수 있습니다.

  그러나 문자 스트링을 숫자와 비교할 수는 없습니다. 또한 문자 자료에서는 산술 연산을 수행할 수 없습니다(EMPNO가 숫자로 표시되는 문자 스트링인 경우에도 해당). 문자나 숫자 데이터를 비교할 수 있는 값으로 변환하기 위해 캐스트 기능을 사용할 수 있습니다. 날짜/시간값 및 기간을 더하거나 뺄 수 있습니다.
- **표현식**은 더하기(+), 빼기(-), 곱하기(*), 나누기(/)로 이루어진 두 값을 식별하고 한 값의 결과로 지수화(**)하거나 연결(CONCAT 또는 ||)합니다. 표현식의 피연산자는 다음이 될 수 있습니다.

  상수

  열

  호스트 변수

  함수에서 리턴된 값

  특수 레지스터

부속 조회

다른 표현식

예를 들면 다음과 같습니다.

```
... WHERE INTEGER(PRENDATE - PRSTDATE) > 100
```

평가의 순서가 괄호로 지정되지 않은 경우, 표현식은 다음 순서로 평가됩니다.

1. 전위 연산자
2. 지수
3. 곱셈, 나눗셈, 연결
4. 덧셈, 뺄셈

동일한 선행 레벨에서 연산자는 왼쪽에서 오른쪽으로 적용됩니다.

- **상수**는 표현식에 대한 리터럴 값을 지정합니다. 예를 들면 다음과 같습니다.

  ```
  ... WHERE 40000 < SALARY
  ```

  *SALARY*는 9자리 팩 십진 값(DECIMAL(9,2))으로 정의된 열을 명명합니다. 그것은 숫자 상수 40000에 비교됩니다.
- **호스트 변수**는 어플리케이션 프로그램에서 변수를 식별합니다. 예를 들면 다음과 같습니다.

  ```
  ... WHERE EMPNO = :EMP
  ```
- **특수 레지스터**는 데이터베이스 관리자에 의해 정의된 특수 값을 식별합니다. 예를 들면 다음과 같습니다.

  ```
  ... WHERE LASTNAME = USER
  ```
- **NULL**은 알 수 없는 값을 갖는 조건을 지정합니다.

  ```
  ... WHERE DUE_DATE IS NULL
  ```
- 부속 조회. 부속 조회 사용에 대한 세부사항은 117 페이지의 제 7 장 『부속 조회 사용』을 참조하십시오.

탐색 조건을 두 개의 열 이름이나 산술 연산자 또는 비교 연산자에 의해 분리된 상수로 제한할 필요는 없습니다. AND와 OR, 이름과 상수에 의해 분리된 여러 개의 술부를 지정하는 복합 탐색 조건을 개발할 수 있습니다. 탐색 조건이 아무리 복잡하더라도 행에 대해 평가할 때 참 또는 거짓값을 제공합니다. 거짓의 효과를 나타내는 알 수 없는 참 값이기도 합니다. 즉, 행 값이 널(null)이면 이 널은 탐색 조건에 지정된 값보다 작지도 같지도 크지도 않기 때문에 탐색 결과가 리턴되지 않습니다. 보다 복잡한 탐색 조건과 술부는 84 페이지의 『복합 탐색 조건 수행』에 설명되어 있습니다.

WHERE절을 완전히 이해하려면 SQL이 탐색 조건과 술부를 평가하고 표현식 값을 비교하는 방법을 알아야 합니다. 이 주제는 SQL 참조서 책에서 논의합니다.

비교 연산자

SQL은 다음의 비교 연산자를 지원합니다.

=	같음
<> 또는 ¬= 또는 !=	같지 않음
<	보다 작음
>	보다 큼
<= 또는 ¬> 또는 !>	보다 작거나 같음(또는 보다 크지 않음)
> = 또는 ¬< 또는 !<	보다 크거나 같음(또는 보다 작지 않음)

NOT 키워드

술부 값의 역을 지정하기 위해 NOT 키워드의 술부를 선행시킬 수 있습니다(즉, 술부가 FALSE이면 TRUE이고 그 역도 성립함). 그러나 NOT은 WHERE절 내의 모든 술부가 아닌 NOT이 앞에 표시된 술부에만 적용됩니다. 예를 들어 부서 C01에서 일하는 사원을 제외한 모든 사원에 대해 표시하려면 다음과 같이 작성할 수 있습니다.

```
... WHERE NOT WORKDEPT = 'C01'
```

이는 다음과 같습니다.

```
... WHERE WORKDEPT <> 'C01'
```

GROUP BY절

GROUP BY절 없이 SQL 열 함수의 어플리케이션은 *하나의* 행을 리턴합니다. GROUP BY 사용시 이 함수는 그룹 수 만큼의 많은 행을 리턴하는 *각* 그룹에 적용됩니다.

GROUP BY절을 사용하여 개별 행이 아닌 행 그룹의 특성을 찾을 수 있습니다. GROUP BY절을 지정하면 SQL이 선택된 행을 그룹으로 나누어서, 각 그룹의 행들이 하나 이상의 열이나 표현식의 일치 값을 갖도록 합니다. 그런 후 SQL은 각 그룹을 처리하여 그룹에 대해 단일 행 결과를 생성합니다. GROUP BY절에 하나 이상의 열이나 표현식을 지정하여 행을 그룹화할 수 있습니다. SELECT문에 지정하는 항목은 표나 보기 내의 개별 행에 대한 특성이 아니라 행들의 각 그룹에 대한 특성입니다.

예를 들어 CORPDATA.EMPLOYEE 표에는 여러 행 세트가 있고 각 세트는 특정 부서의 구성원을 설명하는 행으로 구성됩니다. 각 부서의 평균 급여를 알기 위해서는 다음을 입력할 수 있습니다.

```
SELECT WORKDEPT, DECIMAL (AVG(SALARY),5,0)
        FROM CORPDATA.EMPLOYEE
      GROUP BY WORKDEPT
```

그 결과로 각 부서에 하나씩, 몇 개의 행이 표시됩니다.

WORKDEPT	AVG-SALARY
A00	40850

WORKDEPT	AVG-SALARY
B01	41250
C01	29722
D11	25147
D21	25668
E01	40175
E11	21020
E21	24086

주:

1. 행의 그룹화가 행의 순서화를 의미하지는 않습니다. 그룹화는 그룹에서 선택된 각 행을 기록한 후 SQL이 처리하여 그룹의 특성을 나타내는 것이며 행의 순서화는 결과 표 내의 모든 행을 오름차순 또는 내림차순의 배열순서로 기록하는 것입니다(77 페이지의 『ORDER BY절』에 이 수행 방법이 설명되어 있습니다). 데이터베이스 관리자에 의해 선택된 구현에 따라 결과 그룹이 정렬된 것처럼 보일 수 있습니다.
2. GROUP BY절에서 지정한 열에 널(null)이 있으면 널이 있는 행의 자료에 단일 행 결과가 생성됩니다.
3. 문자 또는 UCS-2 그래픽 열에 대한 그룹화가 발생하는 경우, 조회 수행시 유효한 정렬 순서가 그룹화에 적용됩니다. 정렬 순서 및 선택에 대한 자세한 내용은 129 페이지의 제 8 장 『SQL 정렬 순서』를 참조하십시오.

GROUP BY 사용시, 사용자는 SQL이 행을 그룹화하기 위해 사용할 열이나 표현식을 나열합니다. 예를 들어 CORPDATA.PROJECT 표에 설명된 각 주요 프로젝트를 담당하는 사원의 리스트를 보기 위해 다음과 같이 입력할 수 있습니다.

```
SELECT SUM(PRSTAFF), MAJPROJ
       FROM CORPDATA.PROJECT
       GROUP BY MAJPROJ
```

결과는 회사의 현재 주요 프로젝트와 각 프로젝트에서 일하는 사원 수의 리스트입니다.

SUM(PRSTAFF)	MAJPROJ
6	AD3100
5	AD3110
10	MA2100
8	MA2110
5	OP1000
4	OP2000
3	OP2010
32.5	?

또한, 행을 두 개 이상의 열이나 표현식으로 그룹화할 수도 있습니다. 예를 들어 CORPDATA.EMPLOYEE 표를 사용하여 각 부서 남녀 평균 급여를 표시하기 위해 SELECT문을 입력할 수 있습니다. 이를 위해 다음을 실행할 수 있습니다.

```
SELECT WORKDEPT, SEX, DECIMAL(AVG(SALARY),5,0) AS AVG_WAGES
        FROM CORPDATA.EMPLOYEE
       GROUP BY WORKDEPT, SEX
```

결과는 다음과 같습니다.

WORKDEPT	SEX	AVG_WAGES
A00	F	49625
A00	M	35000
B01	M	41250
C01	F	29722
D11	F	25817
D11	M	24764
D21	F	26933
D21	M	24720
E01	M	40175
E11	F	22810
E11	M	16545
E21	F	25370
E21	M	23830

이 예제에서는 WHERE절을 포함시키지 않았기 때문에 SQL은 CORPDATA.EMPLOYEE 표 내의 모든 행을 체크하고 처리합니다. SQL이 각 그룹의 평균 급여 값을 추출하기 전에 행은 부서 번호별, (각 부서 내에서) 성별로 차례로 그룹화됩니다.

HAVING절

HAVING절을 사용하여 GROUP BY절에 따라 선택된 그룹에 탐색 조건을 지정할 수 있습니다. HAVING절은 해당 절에서 조건을 만족시키는 그룹만을 원함을 의미합니다. 그러므로 HAVING절에 지정하는 탐색 조건은 그룹 내의 개별 행의 특성보다는 각 그룹의 특성을 테스트해야 합니다.

HAVING절은 GROUP BY절 뒤에 오고 WHERE절에서 지정할 수 있는 같은 종류의 탐색 조건을 포함할 수 있습니다. 또한 HAVING절에 열 함수를 지정할 수 있습니다. 예를 들어 각 부서 내 여성의 평균 급여를 검색하려고 한다고 가정합니다. 이를 수행하려면 AVG 열 함수를 사용하여 WORKDEPT에 의한 결과 행을 그룹화하고 WHERE절에 SEX = 'F'를 지정하십시오.

선택된 부서의 여사원들이 모두 16(대학 졸업자) 이상의 교육 레벨을 갖고 있을 때에만 이 자료를 원하는 것으로 지정하려면 HAVING절을 사용하십시오. HAVING절은 그룹의 등록 정보를 테스트합니다. 이 경우 테스트는 그룹 등록 정보 MIN(EDLEVEL)에 수행됩니다.

```
SELECT WORKDEPT, DECIMAL(AVG(SALARY),5,0) AS AVG_WAGES, MIN(EDLEVEL) AS MIN_EDUC
        FROM CORPDATA.EMPLOYEE
       WHERE SEX='F'
      GROUP BY WORKDEPT
       HAVING MIN(EDLEVEL)>=16
```

결과는 다음과 같습니다.

WORKDEPT	AVG_WAGES	MIN_EDUC
A00	49625	18
C01	29722	16
D11	25817	17

AND 및 OR과 연결하여 HAVING절에 복수 술부를 사용할 수 있으며 탐색 조건의 모든 술부에 NOT을 사용할 수 있습니다.

주: 열을 갱신하거나 행을 삭제하는 경우 DECLARE CURSOR문 내의 SELECT문에 GROUP BY절이나 HAVING절을 포함시킬 수 없습니다(DECLARE CURSOR 문은 135 페이지의 제 9 장『커서 사용』에 설명되어 있습니다). 이 절은 읽기 전용 커서로 만듭니다.

열 함수가 아닌 인수가 있는 술부는 WHERE 또는 HAVING절 중의 하나로 작성될 수 있습니다. 조회 처리에서 초기에 처리되었기 때문에 대개 WHERE절에서 선택 기준을 작성하는 것이 더 효율적입니다. HAVING 선택은 결과 표의 이후 처리에서 수행됩니다.

탐색 조건에 문자 또는 UCS-2 그래픽 열이 포함된 술부가 있는 경우, 조회 수행시 유효한 정렬 순서는 이러한 술부에 적용됩니다. 정렬 순서 및 선택에 대한 자세한 내용은 129 페이지의 제 8 장『SQL 정렬 순서』를 참조하십시오.

ORDER BY절

선택된 행을 특정 순서로 리턴하고 ORDER BY절을 사용하여 열 또는 표현식 값의 오름차순 또는 내림차순으로 정렬하도록 지정할 수 있습니다. GROUP BY절을 사용한 것처럼 ORDER BY절을 사용하여, 배열된 순서로 행을 검색할 때 SQL문이 사용할 열 또는 표현식을 지정할 수 있습니다.

예를 들어, 부서 번호의 영문자순으로 나오는 여자 사원의 이름과 부서 번호를 검색하려는 경우 다음 SELECT문을 사용할 수 있습니다.

```
SELECT LASTNAME,WORKDEPT
        FROM CORPDATA.EMPLOYEE
       WHERE SEX='F'
       ORDER BY WORKDEPT
```

결과는 다음과 같습니다.

LASTNAME	WORKDEPT
HAAS	A00
HEMMINGER	A00
KWAN	C01
QUINTANA	C01
NICHOLLS	C01
NATZ	C01
PIANKA	D11
SCOUTTEN	D11
LUTZ	D11
JOHN	D11
PULASKI	D21
JOHNSON	D21
PEREZ	D21
HENDERSON	E11
SCHNEIDER	E11
SETRIGHT	D11
SCHWARTZ	E11
SPRINGER	E11
WONG	E21

주: 널은 가장 큰 값으로 순서화됩니다.

ORDER BY 절에 지정된 열은 SELECT절에 포함시킬 필요가 없습니다. 예를 들어, 다음 명령은 가장 많은 봉급을 받은 순서대로 정렬된 여성 직원을 리턴합니다.

```
SELECT LASTNAME,FIRSTNME
        FROM CORPDATA.EMPLOYEE
       WHERE SEX='F'
       ORDER BY SALARY DESC
```

AS 절이 선택 목록에 결과 열의 이름을 지정하기 위해 사용되었다면 이 이름은 ORDER BY 절에 지정될 수 있습니다. AS절에 지정된 이름은 선택 리스트에서 고유해야 합니다. 예를 들어 영문자순으로 나열된 사원들의 완전명을 검색하기 위해서는 다음 선택문을 사용할 수 있습니다.

```
  SELECT LASTNAME CONCAT FIRSTNME AS FULLNAME
        FROM CORPDATA.EMPLOYEE
    ORDER BY FULLNAME
```

이 SELECT문은 선택적으로 다음과 같이 작성될 수도 있습니다.

```
SELECT LASTNAME CONCAT FIRSTNME
     FROM CORPDATA.EMPLOYEE
  ORDER BY LASTNAME CONCAT FIRSTNME
```

결과를 순서화하기 위해 열을 명명하는 대신 번호를 사용할 수 있습니다. 예를 들어 ORDER BY 3은 SELECT문에 의해 지정된 대로 결과 표의 *세 번째* 열에 의해 결과가 순서화되도록 지정합니다. 순서값이 명명된 열이 아닐 때 결과 표의 행을 순서화하려면 번호를 사용하십시오.

또한 SQL이 행을 오름차순(ASC) 또는 내림차순(DESC) 순서로 배열시키도록 지정할 수 있습니다. 디폴트는 오름차순 배열 순서입니다. 위의 SELECT문에서, SQL은 (영문자 및 숫자로) 가장 낮은 *FULLNAME* 표현식이 있는 행을 먼저 리턴시키고 상위 값이 있는 행을 그 뒤에 리턴시킵니다. 이 이름에 기초해 내림차순 배열 순서로 행을 순서화하려면 다음을 지정하십시오.

```
... ORDER BY FULLNAME DESC
```

GROUP BY절을 사용하여 1차 순서화 순서 뿐만 아니라 2차 순서화 순서(또는 여러 레벨의 순서화 순서)를 지정할 수 있습니다. 이전 예에서, 먼저 부서 번호별로 그리고 각 부서 내에서 사원명별로 행을 순서화시킬 수 있습니다. 이를 수행하려면 다음과 같이 지정하십시오.

```
... ORDER BY WORKDEPT, FULLNAME
```

문자 열 또는 UCS-2 그래픽 열이 ORDER BY절에 사용되는 경우, 이러한 열에 대한 순서화는 조회 수행시 유효한 정렬 순서에 기초를 둡니다. 정렬 순서와 순서화에 미치는 영향에 대한 자세한 내용은 129 페이지의 제 8 장 『SQL 정렬 순서』를 참조하십시오.

정적 SELECT문

정적 SELECT문(SQL 프로그램에 내장된 것)의 경우 INTO절은 FROM절 앞에 지정되어야 합니다. INTO절은 호스트 변수(검색된 열값을 포함시키는데 사용되는 프로그램 내의 변수)를 명명합니다. SELECT절에 지정된 첫 번째 리턴 열의 값은 INTO절에 명명된 첫 번째 호스트 변수에 배치되며 두 번째 값은 두 번째 호스트 변수에 배치되는 식입니다.

SELECT INTO의 결과 표는 단 하나의 행만을 포함해야 합니다. 예를 들어 CORPDATA.EMPLOYEE 표 내의 각 행에는 고유한 EMPNO(사원 번호)가 있습니다. WHERE절이 EMPNO열에 동등 비교를 포함할 경우, 이 표에 대한 SELECT INTO 문의 결과는 정확히 1행(또는 0행)이 됩니다. 두 개 이상의 행이 표시되면 하나의 행이 리턴되어도 오류가 됩니다. ORDER BY절을 지정하여 오류 조건에서 리턴될 행을 제어할 수 있습니다. ORDER BY절을 사용하면 결과 표 내의 첫 번째 행이 리턴됩니다.

SELECT문의 결과로 둘 이상의 행을 원하는 경우, DECLARE CURSOR문을 사용하여 행을 선택하고 FETCH문을 사용하여 한 번에 하나 이상의 호스트 변수 행으로 열 값을 이동하십시오. 커서 사용에 대한 설명은 135 페이지의 제 9 장 『커서 사용』을 참조하십시오

어플리케이션 프로그램 내에서 SELECT문 사용시 사용자 프로그램에 더 많은 자료 독립성을 제공하려면 열 이름을 나열하십시오. 이에 대해서는 두 가지 이유가 있습니다.

1. 소스 코드문 열람시 SELECT절 내의 열 이름과 INTO절 내에서 명명된 호스트 변수 사이에 1 대 1 맵핑을 쉽게 볼 수 있습니다.
2. 열이 사용자가 액세스하고 『SELECT * ....』를 사용하며 소스에서 다시 프로그램을 작성하는 표 또는 보기에 추가된 경우, INTO절은 새로운 열에 대해 명명된 일치하는 호스트 변수를 가지지 않습니다. 추가 열로 인해 사용자는 SQLCA에서 경고(오류는 아님)를 받게 됩니다(SQLWARN4가 『W』를 포함함).

행의 열 값이 없음을 표시하는 널 값

NULL 값은 한 행에 열이 없다는 것을 나타냅니다. 널 값은 제로나 공백이 아니라 『알 수 없음』과 동일합니다. 널 값은 WHERE 및 HAVING절의 조건으로 또 수학적 인수로 사용될 수 있습니다. 예를 들어 WHERE절은 일부 행에 대해 널이 들어 있는 열을 지정할 수 있습니다. 일반적으로 널 값을 포함하고 있는 열을 사용하는 비교 술부는 그 열에 대해 널 값을 갖고 있는 행을 선택하지 않습니다. 이는 널 값이 조건에 지정된 값보다 작지도 않고 크지도 않으며 동등하기 때문입니다. 관리자 번호에 대해 널 값을 갖고 있는 모든 행의 값을 선택하려면 다음을 지정하십시오.

```
SELECT DEPTNO, DEPTNAME, ADMRDEPT
  FROM CORPDATA.DEPARTMENT
  WHERE MGRNO IS NULL
```

그 결과는 다음과 같습니다.

DEPTNO	DEPTNAME	ADMRDEPT
D01	DEVELOPMENT CENTER	A00
F22	BRANCH OFFICE F2	E01
G22	BRANCH OFFICE G2	E01
H22	BRANCH OFFICE H2	E01
I22	BRANCH OFFICE I2	E01
J22	BRANCH OFFICE J2	E01

관리자 번호에 대해 널 값을 갖고 있지 않은 행을 구하려면 다음과 같이 WHERE절을 변경할 수 있습니다.

```
WHERE MGRNO IS NOT NULL
```

널 값의 사용에 대한 자세한 내용은 SQL 참조서 책을 참조하십시오.

SQL 명령문에서 특수 레지스터

SQL문에 『특수 레지스터』를 지정할 수 있습니다. 로컬로 수행된 SQL문의 경우, 특수 레지스터와 내용이 다음 표에 나옵니다.

특수 레지스터	내용
CURRENT DATE CURRENT_DATE	현재 날짜
CURRENT PATH CURRENT_PATH CURRENT FUNCTION PATH	동적으로 준비된 SQL문의 규정되지 않은 자료 유형명, 프로시듀어명 및 기능명을 분석하는 데 사용되는 SQL 경로
CURRENT SCHEMA	규정되지 않은 데이터베이스 오브젝트를 규정하는 데 사용된 스키마 이름은 동적으로 준비된 SQL문에서 적용 가능한 곳을 참조합니다.
CURRENT SERVER CURRENT_SERVER	현재 사용 중인 관계형 데이터베이스의 이름
CURRENT TIME CURRENT_TIME	현재 시간
CURRENT TIMESTAMP CURRENT_TIMESTAMP	시간소인 형식으로 된 현재 날짜와 시간
CURRENT TIMEZONE CURRENT_TIMEZONE	공식을 사용해 로컬 시간을 UTC(Universal Time Coordinated)에 연결하는 기간 `local time - CURRENT TIMEZONE = UTC` 시스템 값 QUTCOFFSET에서 선택됩니다.
USER	작업의 수행시 권한부여 ID(사용자 프로파일)

한 명령문에 CURRENT DATE, CURRENT TIME, CURRENT TIMESTAMP 특수 레지스터 또는 CURDATE, CURTIME, NOW 스칼라 함수가 2개 이상 들어 있는 경우에는 모든 값이 동일한 시계 읽기 방식(clock reading)을 따라야 합니다.

리모트로 수행된 SQL문의 경우, 특수 레지스터와 내용이 다음 표에 표시됩니다.

특수 레지스터	내용
CURRENT DATE CURRENT_DATE CURRENT TIME CURRENT_TIME CURRENT TIMESTAMP CURRENT_TIMESTAMP	로컬 시스템이 아닌 리모트 시스템의 현재 날짜와 시간
CURRENT TIMEZONE CURRENT_TIMEZONE	리모트 시스템 시간을 UTC에 연결하는 기간
CURRENT SERVER CURRENT_SERVER	현재 사용 중인 관계형 데이터베이스의 이름
CURRENT SCHEMA	리모트 시스템에 있는 현재 스키마 값

특수 레지스터	내용
USER	리모트 시스템에서 서버 작업의 수행시 권한부여 ID
CURRENT PATH CURRENT_PATH CURRENT FUNCTION PATH	리모트 시스템에 있는 현재 경로 값

분산표상의 조회가 특수 레지스터를 참조할 때는 조회를 요구한 시스템에 있는 특수 레지스터 내용이 사용됩니다. 분산표에 대한 자세한 내용은 DB2 Multisystem 책을 참조하십시오.

자료 유형 캐스트

때로 데이터의 *유형*이 다른 데이터 유형이나 다른 길이, 정밀도 또는 스케일을 갖는 같은 데이터 유형으로 캐스트되거나 변경되어야 하는 상황이 있습니다. 예를 들어 문자 및 정수와 같은 서로 다른 유형의 두 열을 비교하려는 경우 문자를 정수로 변경하거나 정수를 문자로 변경하여 비교할 수 있도록 할 수 있습니다. 다른 데이터 유형으로 변경될 수 있는 데이터 유형을 소스 데이터 유형에서 목표 데이터 유형으로 *캐스트 가능*하다라고 합니다.

캐스트 기능을 사용하거나 CAST 스펙을 사용하여 데이터 유형을 다른 데이터 유형으로 명시적으로 캐스트할 수 있습니다. 예를 들어 DATE로 정의된 날짜 열(BIRTHDATE)이 있고 이 열의 데이터 유형을 고정 길이 10의 문자로 캐스트하고자 하는 경우 다음과 같이 입력할 수 있습니다.

```
SELECT CHAR (BIRTHDATE,USA)
        FROM CORPDATA.EMPLOYEE
```

또한 CAST 함수를 자료 유형으로 직접 캐스트할 수도 있습니다.

```
SELECT CAST(BIRTHDATE AS CHAR(10))
        FROM CORPDATA.EMPLOYEE
```

데이터 유형에 대한 자세한 내용은 SQL 참조서의 데이터 유형간 캐스팅을 참조하십시오.

날짜, 시간 및 시간소인 자료 유형

날짜, 시간 및 시간소인은 SQL 사용자는 볼 수 없는 내부 형식으로 표시되는 자료 유형입니다. 날짜, 시간 및 시간소인은 문자 스트링에 의해 표시될 수 있으며 문자열 변수에 할당될 수 있습니다. 데이터베이스 관리자가 날짜, 시간, 시간소인 값으로 다음을 인식합니다.

- DATE, TIME 또는 TIMESTAMP 스칼라 함수에 의해 리턴된 값.

- CURRENT DATE, CURRENT TIME 또는 CURRENT TIMESTAMP 특수 레지스터에 의해 리턴된 값.
- 연산식 또는 비교의 피연산자이고 다른 하나는 날짜, 시간 또는 시간소인인 문자 스트링. 예를 들어 술부에서는 다음과 같습니다.

  ```
  ... WHERE HIREDATE < '1950-01-01'
  ```

  HIREDATE가 날짜 열인 경우, 문자 스트링 '1950-01-01'이 날짜로 해석됩니다.
- 날짜, 시간 또는 시간소인 열을 설정하는데 사용되는 문자 스트링 변수나 상수는 UPDATE문의 SET절이나 INSERT절의 VALUES절에 있습니다.

날짜, 시간 및 시간소인 값의 문자 스트링 형식에 대한 자세한 정보는 SQL 참조서의 날짜 시간 값을 참조하십시오.

또한 다음 주제를 참조하십시오.

- 『현재 날짜와 시간 값 지정』
- 『날짜/시간 연산』

현재 날짜와 시간 값 지정

세 개의 특수 레지스터 CURRENT DATE, CURRENT TIME 또는 CURRENT TIMESTAMP 중 하나를 지정하여 표현식에 현재 날짜, 시간 또는 시간소인을 지정할 수 있습니다. 각 값은 명령문 수행시 확보된 시각 시계 읽기에 기초합니다. 동일한 SQL 문 내에서 CURRENT DATE, CURRENT TIME 또는 CURRENT TIMESTAMP에 대한 복수 참조는 동일한 값을 사용합니다. 다음 명령문은 명령문 수행시 EMPLOYEE 표에 있는 각 사원의 나이(연수로)를 리턴시킵니다.

```
SELECT YEAR(CURRENT DATE - BIRTHDATE)
     FROM CORPDATA.EMPLOYEE
```

CURRENT TIMEZONE 특수 레지스터를 사용하여 로컬 시간을 UTC(Universal Time Coordinated)로 변환할 수 있습니다. 예를 들어 STARTT라는 이름의 시간 열 유형이 들어 있는 DATETIME으로 명명된 표가 있고 STARTT를 UTC로 변환하려면 다음 명령문을 사용할 수 있습니다.

```
SELECT STARTT - CURRENT TIMEZONE
  FROM DATETIME
```

날짜/시간 연산

덧셈과 뺄셈은 날짜, 시간 및 시간소인 값에 적용할 수 있는 유일한 산술 연산자입니다. 사용자는 지속 기간 만큼 날짜, 시간 또는 시간소인을 증감하거나 날짜에서 날짜를, 시간에서 시간을 또는 시간소인에서 시간소인을 뺄 수 있습니다. 날짜 및 시간 산술에 대한 자세한 설명은 SQL 참조서 책의 날짜 시간 산술을 참조하십시오.

복제 행 방지

SQL이 SELECT문을 평가할 때 SELECT문의 탐색 조건을 만족시키는 행 수에 따라 여러 행이 결과 표에 규정될 수 있습니다. 결과 표의 일부 행은 복제될 수 있습니다. 열 이름 리스트가 뒤에 오는 DISTINCT 키워드를 사용하여 복제하지 않게 지정할 수 있습니다.

```
SELECT DISTINCT JOB, SEX
...
```

DISTINCT는 사용자가 고유 행만 선택하고자 함을 의미합니다. 선택된 행이 결과 표 내의 다른 행과 복제되면 복제 행은 무시됩니다(이는 결과 표에 기록되지 않음). 예를 들어 사원 직무 코드 리스트를 원한다고 가정합니다. 이 때 사용자는 어느 사원이 어떤 직무 코드를 가지고 있는지를 알 필요가 없습니다. 부서 내의 몇몇 사람이 동일한 직무 코드를 가질 수 있으므로 결과 표에 고유 값만 표시되도록 하기 위해 DISTINCT를 사용할 수 있습니다.

다음 예는 이를 수행하는 방법을 보여줍니다.

```
SELECT DISTINCT JOB
      FROM CORPDATA.EMPLOYEE
    WHERE WORKDEPT = 'D11'
```

결과는 두 행입니다.

JOB
DESIGNER
MANAGER

DISTINCT를 SELECT절에 포함시키지 않을 경우, SQL이 탐색 조건을 만족시키는 각 행에 대한 *JOB* 열의 값을 리턴하기 때문에 결과에서 복제 행을 찾을 수 있습니다. Null 값은 DISTINCT에 대해 복제 행으로 처리됩니다.

SELECT절에 DISTINCT를 포함하고 또한 공유 가중치 정렬 순서를 포함한다면 보다 적은 값이 리턴됩니다. 정렬 순서는 동일한 문자를 포함한 값이 동일한 가중치를 갖도록 합니다. 'MGR', 'Mgr', 'mgr' 모두가 같은 표에 있으면 이 값 가운데 하나만이 리턴됩니다. 정렬 순서 및 선택에 대한 자세한 내용은 129 페이지의 제 8 장『SQL 정렬 순서』를 참조하십시오.

복합 탐색 조건 수행

기본 비교 술부(=, >, <, etc)와 더불어 탐색 조건에는 BETWEEN, IN, EXISTS, IS NULL 및 LIKE 등의 키워드가 포함될 수 있습니다. 또한 탐색 조건은 부속 조회를 포함할 수도 있습니다. 자세한 정보 및 예는 117 페이지의 제 7 장『부속 조회 사용』을 참조하십시오.

문자 및 UCS-2 그래픽 열 술부의 경우, 정렬 순서는 BETWEEN, IN, EXISTS 및 LIKE절에 대한 술부 평가에 앞서 피연산자에 적용됩니다. 선택을 통한 정렬 순서 사용에 대한 자세한 내용은 129 페이지의 제 8 장 『SQL 정렬 순서』를 참조하십시오.

또한 복수 탐색 조건을 수행할 수도 있습니다. 자세한 내용은 87 페이지의 『WHERE 절 내의 복수 탐색 조건』을 참조하십시오.

- **BETWEEN ... AND ...**는 딱 떨어지거나 사이에 있는 값을 만족시키는 탐색 조건을 지정하는데 사용됩니다. 예를 들어, 1987년에 채용된 모든 사원을 보려는 경우 다음을 사용하면 됩니다.

  ```
  ...  WHERE HIREDATE BETWEEN '1987-01-01' AND '1987-12-31'
  ```

  BETWEEN 키워드는 포괄적입니다. 동일한 결과를 생성하는 다소 복잡하지만 명시적인 탐색 조건은 다음과 같습니다.

  ```
  ... WHERE HIREDATE >= '1987-01-01' AND HIREDATE <= '1987-12-31'
  ```

- **IN**은 지정된 표현식의 나열된 값들 사이에 있는 행에 관심이 있음을 의미합니다. 예를 들어, 부서 A00, C01 및 E21에 있는 모든 사원의 이름을 찾기 위해서는 다음을 지정하면 됩니다.

  ```
  ... WHERE WORKDEPT IN ('A00', 'C01', 'E21')
  ```

- **EXISTS**는 특정 행이 존재하는지 테스트하는 데 관심이 있음을 나타냅니다. 예를 들어, 봉급이 60000을 초과하는 사원을 찾으려면 다음을 지정할 수 있습니다.

  ```
  EXISTS (SELECT * FROM EMPLOYEE WHERE SALARY > 60000)
  ```

- **IS NULL**은 널 값 테스트에 관심이 있음을 나타냅니다. 예를 들어, 전화번호 리스팅이 없는 사원을 찾으려면 다음을 지정할 수 있습니다.

  ```
  ... WHERE EMPLOYEE.PHONE IS NULL
  ```

- **LIKE**는 열 값이 사용자가 제공하는 값과 유사한 행에 관심이 있음을 의미합니다. LIKE를 사용할 경우, SQL은 사용자가 지정한 문자 스트링과 유사한 문자 스트링을 탐색합니다. 유사성의 정도는 탐색 조건에 포함된 문자열에서 사용되고 있는 두 개의 특수 문자에 의해 판별됩니다.

  _ 밑줄은 1개 문자를 나타냅니다.

  % 퍼센트 기호는 0개 이상의 아직 값이 지정되지 않은 스트링을 나타냅니다. 퍼센트 기호가 탐색 스트링을 시작하면 SQL은 열에서 일치하는 값 앞에 0개 이상의 문자를 허용합니다. 그렇지 않으면 탐색 문자 스트링은 열의 첫 번째 위치에서 시작되어야 합니다.

  주: MIXED 자료에 대해 조작할 경우에는 다음의 구분이 적용됩니다. SBCS 밑줄 문자는 하나의 SBCS 문자를 나타냅니다. 이러한 제한조건은 퍼센트 기호에는 적용되지 않습니다. 즉, 퍼센트 기호는 모든 수의 SBCS 또는 DBCS 문자를 나타냅니다. LIKE 술부와 MIXED 자료에 대한 자세한 내용은 SQL 참조서를 참조하십시오.

열값의 모든 문자를 모르거나 신경을 쓰지 않으려면 밑줄 또는 퍼센트 기호를 사용하십시오. 예를 들어, Minneapolis에 사는 사원을 찾으려면 다음과 같이 지정하십시오.

```
... WHERE ADDRESS LIKE '%MINNEAPOLIS%'
```

SQL은 그 스트링이 어디에 있든지 ADDRESS 열에서 MINNEAPOLIS 문자 스트링이 있는 모든 행을 리턴시킵니다.

다른 예로 이름이 'SAN'으로 시작되는 도시를 나열하려면 다음과 같이 지정할 수 있습니다.

```
... WHERE TOWN LIKE 'SAN%'
```

거리 이름이 마스터 거리 이름 목록에 없는 주소를 찾으려면, LIKE 표현식에 표현식을 사용할 수 있습니다. 이 예에서, 표의 STREET 열은 대문자로 가정됩니다.

```
... WHERE UCASE (:address_variable) NOT LIKE '%'||STREET||'%'
```

밑줄이나 퍼센트 문자를 포함하는 문자 스트링을 탐색하려면 ESCAPE절을 사용하여 이탈 문자를 지정하십시오. 예를 들어, 이름에 퍼센트가 있는 모든 업무를 보려는 경우 다음을 지정하면 됩니다.

```
... WHERE BUSINESS_NAME LIKE '%@%%' ESCAPE '@'
```

첫 번째와 마지막 퍼센트 문자는 평소와 같이 해석됩니다. '@%'의 조합은 실제 퍼센트 문자로 취급됩니다. 자세한 내용은 『LIKE에 대한 특수 고려사항』을 참조하십시오.

술부에 대한 전체 리스팅은 SQL 참조서의 술부를 참조하십시오.

LIKE에 대한 특수 고려사항

- 호스트 변수가 탐색 패턴에서 문자 스트링 상수의 위치에 사용된 경우, 가변 길이 호스트 변수 사용을 고려해야 합니다. 이것은 다음을 허용합니다.
  - 이전에 사용된 스트링 상수를 변경없이 호스트 변수에 할당합니다.
  - 스트링 상수가 사용된 것처럼 동일한 선택 기준과 결과를 획득합니다.
- 고정 길이 호스트 변수가 탐색 패턴에서 문자 스트링 상수 위치에 사용된 경우, 호스트 변수에 지정된 값과 스트링 상수에 의해 이전에 사용된 패턴이 일치하는지를 확인해야 합니다. 값이 할당되지 않은 호스트 변수의 모든 문자는 공백으로 초기설정됩니다.

  예를 들어 *가변 길이* 호스트 변수에서 스트링 패턴 'ABC%'를 사용하여 탐색을 한 경우 다음과 같은 값들이 리턴될 수 있습니다.

  ```
  'ABCD      '  'ABCDE'    'ABCxxx'      'ABC '
  ```

예를 들어 *고정 길이*가 10인 호스트 변수에 들어 있는 'ABC%'라는 탐색 패턴을 사용하여 탐색을 수행한 경우, 열 길이가 12라고 가정할 때 리턴될 수 있는 값은 다음과 같습니다.

```
'ABCDE       ' 'ABCD        ' 'ABCxxx      ' 'ABC         '
```

리턴되는 모든 값이 'ABC'로 시작하여 최소한 6개의 공백으로 끝난다는 것을 기억하십시오. 이는 호스트 변수의 최종 6자에 특정한 값이 할당되지 않아 공백이 사용되었기 때문입니다.

최종 7자에 어떤 값이든 올 수 있도록 고정 길이 호스트 변수를 탐색하려면 'ABC%%%%%%%'를 탐색하면 됩니다. 다음은 리턴될 수 있는 일부 값입니다.

```
'ABCDEFGHIJ'   'ABCXXXXXXX'  'ABCDE'      'ABCDD'
```

WHERE절 내의 복수 탐색 조건

71 페이지의『WHERE 절을 사용하는 탐색 조건 지정』절에서 단일 탐색 조건을 사용하는 방법을 보았습니다. 여러 개의 술부가 들어 있는 탐색 조건을 작성하면 보다 많은 요구를 규정할 수 있습니다. 사용자가 지정한 탐색 조건에는 비교 연산자 또는 BETWEEN, IN, LIKE, IS NULL 및 IS NOT NULL 등의 술부 중 무엇이든 포함될 수 있습니다.

커넥터 AND와 OR로 두 개의 술부를 결합할 수 있습니다. 또한 NOT 키워드를 사용하여 원하는 탐색 조건이 지정된 탐색 조건의 부정 값인지를 지정할 수 있습니다. WHERE절에는 사용자가 원하는 만큼의 술부가 포함될 수 있습니다.

- **AND**는 규정할 행이 탐색 조건의 두 술부 모두를 만족해야 함을 의미합니다. 예를 들어 부서 D21의 어느 사원이 1987년 12월 31일 이후에 고용되었는지를 알기 위해 다음을 지정할 수 있습니다.

  ```
  ...
    WHERE WORKDEPT = 'D21' AND HIREDATE > '1987-12-31'
  ```

- **OR**는 규정할 행이 탐색 조건의 한 술부 또는 두 술부 모두에 의해 설정된 조건을 만족해야 함을 의미합니다. 예를 들어 부서 C01 또는 D11에 어떤 직원이 있는지 찾으려면 다음을 지정할 수 있습니다.

  ```
  ...
    WHERE WORKDEPT = 'C01' OR WORKDEPT = 'D11'
  ```

  주: 또한 이 요구를 지정하는 데 IN을 사용할 수도 있습니다.
  WHERE WORKDEPT IN ('C01', 'D11').

- **NOT**은 규정할 행이 NOT 다음에 오는 술부나 탐색 조건에 의해 설정된 기준을 만족해서는 안됨을 의미합니다. 예를 들어 직무 코드가 ANALYST인 사원을 제외한 부서 E11의 모든 사원을 보기 위해서는 다음과 같이 지정할 수 있습니다.

  ```
  ...
    WHERE WORKDEPT = 'E11' AND NOT JOB = 'ANALYST'
  ```

SQL은 이러한 연결자가 들어 있는 탐색 조건을 평가할 때 특정 순서에 따라 이를 수행합니다. SQL은 먼저 NOT절을 평가한 다음 AND절을 평가하고 OR절을 평가합니다.

괄호를 사용하여 평가 순서를 변경할 수 있습니다. 괄호로 묶인 탐색 조건이 먼저 평가됩니다. 예를 들어 교육 등급이 12 이상인 부서 E11과 E21의 모든 사원을 선택하기 위해서는 다음을 지정할 수 있습니다.

```
...
  WHERE EDLEVEL > 12 AND
     (WORKDEPT = 'E11' OR WORKDEPT = 'E21')
```

괄호는 탐색 조건의 의미를 결정합니다. 이 예에서는 다음과 같은 모든 행이 표시됩니다.

WORKDEPT 값이 E11 또는 E21이고

EDLEVEL 값이 12보다 큰 행

괄호를 사용하지 않은 경우,

```
...
  WHERE EDLEVEL > 12 AND WORKDEPT = 'E11'
    OR WORKDEPT = 'E21'
```

결과는 달라집니다. 선택된 행은 다음과 같은 행입니다.

WORKDEPT = E11 및 EDLEVEL > 12, 또는

EDLEVEL 값에 관계없이 WORKDEPT = E21인 행

하나 이상의 표로부터 자료 결합

때로는 원하는 정보가 하나의 표에만 있지 않을 수도 있습니다. 결과 표의 행을 형성하기 위해 하나의 표 내의 열값과 다른 표 내의 열값을 같이 검색하는 경우도 있습니다. 두 개 이상의 표에 있는 열값을 검색하여 하나의 행으로 결합할 수 있습니다.

iSeries용 DB2 UDB에 의해 지원되는 4가지 결합 유형은 내부 결합, 왼쪽 외부 결합, 오른쪽 외부 결합, 왼쪽 예외 결합 및 오른쪽 예외 결합 및 상호 결합이 있습니다.

- 89 페이지의 『내부 결합』은 각 표에서 결합 열의 값과 일치하는 행만을 리턴합니다. 표 사이에서 일치하지 않은 행들은 결과 표에 표시되지 않습니다.
- 90 페이지의 『왼쪽 외부 결합』은 첫 번째 표(왼쪽에 있는 표)의 모든 행 값과 두 번째 표에서 일치하는 행 값을 리턴합니다. 두 번째 표의 행 중 일치하지 않은 행들은 두 번째 표의 모든 열에 대해 널 값을 리턴합니다.
- 91 페이지의 『오른쪽 외부 결합』은 두 번째 표(오른쪽의 표)에서 모든 행에 대한 값을 리턴하고 일치하는 행에 대한 첫 번째 표의 값을 리턴합니다. 첫 번째 표의 행 중 일치하지 않는 행들은 첫 번째 표의 모든 열에 대해 널 값을 리턴합니다.

- 왼쪽 예외 결합은 왼쪽 표에서 오른쪽 표에 일치하는 것이 없는 행만을 리턴합니다. 결과 표의 열 중 오른쪽 표에서 온 열들은 널 값을 가집니다.
- 오른쪽 예외 결합은 오른쪽 표에서 왼쪽 표에 일치하는 것이 없는 행만을 리턴합니다. 왼쪽 표에서 온 결과 표의 열에는 널 값이 있습니다.
- 92 페이지의 『상호 결합』은 결합될 표로부터 각각의 행 조합에 사용될 행을 결과 표에 리턴합니다(Cartesian Product).

왼쪽 외부 결합과 오른쪽 예외 결합을 사용하여 전체 외부 결합을 시뮬레이트할 수 있습니다. 세부사항은 93 페이지의 『전체 외부 결합 시뮬레이트』를 참조하십시오. 또한 단일 명령문에 복수 결합 유형을 사용할 수도 있습니다. 세부사항은 94 페이지의 『한 명령문의 복수 결합 유형』을 참조하십시오.

결합에 관한 주의사항

두 개 이상의 표 결합시 다음을 수행하십시오.

- 공통적인 열 이름이 있을 경우에는 각 열 이름 앞에 표 이름(또는 상관명)을 표시해야 합니다. 고유한 열 이름은 규정하지 않아도 됩니다.
- 원하는 열 이름 열거를 원하지 않지만 그 대신 SELECT *를 사용하는 경우 SQL은 두 번째 표의 모든 열이 뒤에 오는 첫 번째 표의 모든 열로 이루어진 행을 리턴합니다.
- FROM절에 지정된 각 표나 보기에서 행을 선택할 권한이 있어야 합니다.
- 정렬 순서는 결합중인 모든 문자와 UCS-2 그래픽 열에 적용됩니다.

내부 결합

내부 결합을 사용하면 한 행의 자료를 구성하는데 있어서 표의 한 행에서 나온 값이 다른(또는 같은) 표의 다른 행에서 나온 값과 결합됩니다. SQL은 결합용으로 지정된 두 표를 검사하여 결합 탐색 조건을 충족시키는 모든 행으로부터 자료를 검색합니다. 내부 결합을 지정하는 방법에는 JOIN 구문을 사용하는 것과 WHERE절을 사용하는 2가지가 있습니다.

프로젝트에 관여하는 모든 사원의 번호, 이름 및 프로젝트 번호를 검색하는 경우를 생각해 보십시오. 즉 CORPDATA.EMPLOYEE 표에서 *EMPNO* 및 *LASTNAME* 열을 구하려하고 CORPDATA.PROJECT 표에서 *PROJNO* 열을 구하려 합니다. 여기에서는 성이 ‘S’나 그 이후 영문자로 시작하는 사원만을 고려해보겠습니다. 이 정보를 보기 위해서는 두 개의 표를 결합해야 합니다.

JOIN 구문을 사용한 내부 결합

내부 결합 구문을 사용하기 위해서는 표에 적용되는 결합 조건과 함께 결합시키려는 두 표 모두가 FROM절에 나와야 합니다. 결합 조건은 ON 키워드 뒤에 지정되어 결합 결과를 생성하기 위해 두 표가 서로 비교되는 방법을 결정합니다. 조건에 비교 연산자를 사용할 수 있으며 이 때 등호 연산자는 사용할 필요가 없습니다. 복수 결합 조건은 AND

키워드로 구분되어 ON절에 지정될 수 있습니다. 실제 결합에 관련되지 않은 추가 조건은 WHERE절이나, ON절에서 실제 결합의 부분으로 지정됩니다.

```
SELECT EMPNO, LASTNAME, PROJNO
  FROM CORPDATA.EMPLOYEE INNER JOIN CORPDATA.PROJECT
       ON EMPNO = RESPEMP
  WHERE LASTNAME > 'S'
```

이 예에서는 표로부터 EMPNO와 RESPEMP 열을 사용하여 두 표에 대해 결합이 수행됩니다. 최소한 성이 'S'로 시작하는 사원들만 리턴되고 이 추가 조건은 WHERE절로 제공됩니다.

이 조회의 출력 결과는 다음과 같습니다.

EMPNO	LASTNAME	PROJNO
000250	SMITH	AD3112
000060	STERN	MA2110
000100	SPENSER	OP2010
000020	THOMPSON	PL2100

WHERE절을 사용한 내부 결합

WHERE절을 사용하여 이와 동일한 결합을 수행하기 위해서는 결합 조건과 WHERE절의 추가 선택 조건 모두를 사용하여 작성합니다. 결합될 표들은 FROM절에 쉼표로 구분되어 나열됩니다.

```
SELECT EMPNO, LASTNAME, PROJNO
  FROM CORPDATA.EMPLOYEE, CORPDATA.PROJECT
  WHERE EMPNO = RESPEMP
  AND LASTNAME > 'S'
```

이 조회의 출력 결과는 이 앞의 예제 경우와 동일합니다.

왼쪽 외부 결합

왼쪽 외부 결합은 내부 결합에서 리턴시키는 것 외에도 두 번째 표에서 일치하지 않았던 첫 번째 표에 있는 각각의 다른 행에 대해 하나의 행을 리턴시킵니다.

모든 사원과 그 사원들이 현재 참여하고 있는 프로젝트를 찾는다고 가정해 보십시오. 또한 현재 프로젝트에 참여하고 있지 않은 사원들을 찾아 볼 경우도 생각해 보십시오. 다음의 조회는 성이 'S'자보다 큰 모든 사원의 리스트 뿐만 아니라 그에 할당된 프로젝트 번호의 리스트를 리턴합니다.

```
SELECT EMPNO, LASTNAME, PROJNO
  FROM CORPDATA.EMPLOYEE LEFT OUTER JOIN CORPDATA.PROJECT
       ON EMPNO = RESPEMP
  WHERE LASTNAME > 'S'
```

이 조회의 결과는 프로젝트 번호가 없는 일부 사원들을 포함하고 있습니다. 이들은 조회 목록에는 있지만 해당 프로젝트 번호에 대해 널 값을 리턴시킵니다.

EMPNO	LASTNAME	PROJNO
000020	THOMPSON	PL2100
000060	STERN	MA2110
000100	SPENSER	OP2010
000170	YOSHIMURA	-
000180	SCOUTTEN	-
000190	WALKER	-
000250	SMITH	AD3112
000280	SCHNEIDER	-
000300	SMITH	-
000310	SETRIGHT	-
200170	YAMAMOTO	-
200280	SCHWARTZ	-
200310	SPRINGER	-
200330	WONG	-

주

RRR 스칼라 함수를 사용하여 왼쪽 외부 결합이나 예외 결합의 오른쪽 표에 있는 열에 대한 상대 레코드 번호를 리턴하면 일치하지 않는 행에 대해 0값이 리턴됩니다.

오른쪽 외부 결합

오른쪽 외부 결합은 내부 결합에서 리턴하는 모든 행 외에도 두 번째 표에서 첫 번째 표에 일치하는 것이 없는 각각의 다른 행들에 대해 하나의 행을 리턴합니다. 표들이 반대 순서로 지정된다는 점을 제외하고는 왼쪽 외부 결합과 동일합니다.

왼쪽 외부 결합 예로 사용된 조회를 다음과 같이 오른쪽 외부 결합으로 다시 쓸 수 있습니다.

```
SELECT EMPNO, LASTNAME, PROJNO
  FROM CORPDATA.PROJECT RIGHT OUTER JOIN CORPDATA.EMPLOYEE
       ON EMPNO = RESPEMP
  WHERE LASTNAME > 'S'
```

이 조회의 결과는 왼쪽 외부 결합 조회의 결과와 동일합니다.

예외 결합

왼쪽 예외 결합은 첫 번째 표에서 두 번째 표와 일치하지 않는 행만을 리턴합니다. 전과 동일한 표를 사용하여 어떤 프로젝트에도 참여하지 않은 사원들을 리턴시키십시오.

```
SELECT EMPNO, LASTNAME, PROJNO
  FROM CORPDATA.EMPLOYEE EXCEPTION JOIN CORPDATA.PROJECT
       ON EMPNO = RESPEMP
  WHERE LASTNAME > 'S'
```

이 결합의 출력은 다음과 같습니다.

EMPNO	LASTNAME	PROJNO
000170	YOSHIMURA	-
000180	SCOUTTEN	-
000190	WALKER	-
000280	SCHNEIDER	-
000300	SMITH	-
000310	SETRIGHT	-
200170	YAMAMOTO	-
200280	SCHWARTZ	-
200310	SPRINGER	-
200330	WONG	-

예외 결합은 NOT EXISTS 술어를 사용하여 부속 조회로 작성될 수도 있습니다. 이전 조회를 다음과 같은 방법으로 다시 작성할 수 있습니다.

```
SELECT EMPNO, LASTNAME
      FROM CORPDATA.EMPLOYEE
  WHERE LASTNAME > 'S'
    AND NOT EXISTS
       (SELECT * FROM CORPDATA.PROJECT
          WHERE EMPNO = RESPEMP)
```

이 조회의 유일한 차이점은 PROJECT 표에서 값을 리턴할 수 없다는 점입니다.

표들의 순서가 반대인 점을 제외하고는 왼쪽 예외 결합과 정확히 동일하게 작업하는 오른쪽 예외 결합이 있습니다.

상호 결합

상호 결합(또는 Cartesian Product 결합)은 첫 번째 표의 각 행이 두 번째 표의 각 행과 결합되어 만들어진 결과 표를 리턴합니다. 결과 표의 행 수는 각 표에 있는 행 수의 결과입니다. 결합되는 표들의 크기가 클 경우에는 결합 시간이 매우 오래 걸릴 수도 있습니다.

상호 결합은 두 가지 방법으로 지정될 수 있는데 하나는 JOIN 구문을 사용하는 방법이고 다른 하나는 결합 기준을 제공하기 위해 WHERE절을 사용하지 않고 FROM절에 표들을 쉼표로 구분하여 나열하는 방법입니다.

다음과 같은 표들이 존재한다고 가정합니다.

표 6. 표 A

ACOL1	ACOL2
A1	AA1
A2	AA2

표 6. 표 A (계속)

ACOL1	ACOL2
A3	AA3

표 7. 표 B

BCOL1	BCOL2
B1	BB1
B2	BB2

다음과 같은 두 SELECT문의 실행 결과는 서로 동일합니다.

```
SELECT * FROM A CROSS JOIN B

SELECT * FROM A, B
```

이 SELECT문의 결과 표는 다음과 같습니다.

ACOL1	ACOL2	BCOL1	BCOL2
A1	AA1	B1	BB1
A1	AA1	B2	BB2
A2	AA2	B1	BB1
A2	AA2	B2	BB2
A3	AA3	B1	BB1
A3	AA3	B2	BB2

전체 외부 결합 시뮬레이트

왼쪽 및 오른쪽 외부 결합에서와 같이 전체 외부 결합은 양쪽 표에서 일치하는 행을 리턴합니다. 그러나 전체 외부 결합은 왼쪽 및 오른쪽의 두 표에서 일치하지 않는 행도 리턴합니다. iSeries용 DB2 UDB가 전체 외부 결합 구문을 지원하지 않으므로 왼쪽 외부 결합과 오른쪽 예외 결합을 사용하여 전체 외부 결합을 시뮬레이트할 수 있습니다. 모든 직원과 모든 프로젝트를 찾고자 하고 현재 프로젝트에 참여하고 있지 않은 직원들도 찾고자 한다고 가정합니다. 다음의 조회는 성이 'S'자보다 큰 모든 사원의 리스트뿐만 아니라 그에 할당된 프로젝트 번호의 리스트를 리턴합니다.

```
SELECT EMPNO, LASTNAME, PROJNO
    FROM CORPDATA.EMPLOYEE LEFT OUTER JOIN CORPDATA.PROJECT
          ON EMPNO = RESPEMP
    WHERE LASTNAME > 'S'
 UNION
  (SELECT EMPNO, LASTNAME, PROJNO
     FROM CORPDATA.PROJECT EXCEPTION JOIN CORPDATA.EMPLOYEE
          ON EMPNO = RESPEMP
     WHERE LASTNAME > 'S');
```

한 명령문의 복수 결합 유형

원하는 결과를 얻기 위해 3개 이상의 표를 결합해야 할 경우도 있을 것입니다. 전체 사원, 각 사원의 부서명 및 사원이 참여하고 있는 프로젝트명을 리턴할 경우에 정보를 얻으려면 EMPLOYEE 표, DEPARTMENT 표 및 PROJECT 표를 결합해야 합니다. 다음은 조회와 그 실행 결과입니다.

```
SELECT EMPNO, LASTNAME, DEPTNAME, PROJNO
  FROM CORPDATA.EMPLOYEE INNER JOIN CORPDATA.DEPARTMENT
        ON WORKDEPT = DEPTNO
     LEFT OUTER JOIN CORPDATA.PROJECT
        ON EMPNO = RESPEMP
  WHERE LASTNAME > 'S'
```

이 조회의 결과는 다음과 같습니다.

EMPNO	LASTNAME	DEPTNAME	PROJNO
000020	THOMPSON	PLANNING	PL2100
000060	STERN	MANUFACTURING SYSTEMS	MA2110
000100	SPENSER	SOFTWARE SUPPORT	OP2010
000170	YOSHIMURA	MANUFACTURING SYSTEMS	-
000180	SCOUTTEN	MANUFACTURING SYSTEMS	-
000190	WALKER	MANUFACTURING SYSTEMS	-
000250	SMITH	ADMINISTRATION SYSTEMS	AD3112
000280	SCHNEIDER	OPERATIONS	-
000300	SMITH	OPERATIONS	-
000310	SETRIGHT	OPERATIONS	-

결합에 대한 자세한 정보는 SQL 참조서를 참조하십시오.

표 표현식 사용

표 표현식을 사용하여 중간 결과 표를 지정할 수 있습니다. 표 표현식은 보기의 일반적인 사용이 요구될 때 보기를 작성하지 않기 위해 보기 대신 사용할 수 있습니다. 표 표현식은 내포된 표 표현식(파생된 표라고도 함)과 공통 표 표현식으로 구성됩니다.

내포된 표 표현식은 FROM 구에 괄호로 묶어 지정됩니다. 예를 들어, 관리자 번호, 부서 번호 및 각 부서에 대한 최대 급료를 보여주는 결과 표를 원한다고 가정합니다. 관리자 번호는 DEPARTMENT 표에, 부서 번호는 DEPARTMENT 및 EMPLOYEE 표 둘 다에 있으며, 급료는 EMPLOYEE 표에 있습니다. FROM절의 표 표현식을 사용하여 각 부서에 대한 최대 급료를 선택할 수 있습니다. 또한 파생된 표에 이름을 부여하기 위해 내포된 표 표현식 다음에 상관명 T2를 추가할 수도 있습니다. 그러면, 외부 선택에서 T2를 사용하여 파생된 표에서 선택된 열(이 경우, MAXSAL 및 WORKDEPT)

을 규정합니다. 외부 선택에서 참조되도록 하려면 내포된 표 표현식에서 선택된 MAX(SALARY) 열에 이름이 부여되어야 합니다. AS절을 사용하여 그 이름을 부여합니다.

```
SELECT MGRNO, T1.DEPTNO, MAXSAL
FROM CORPDATA.DEPARTMENT T1,
      (SELECT MAX(SALARY) AS MAXSAL, WORKDEPT
       FROM CORPDATA.EMPLOYEE E1
       GROUP BY WORKDEPT) T2
WHERE T1.DEPTNO = T2.WORKDEPT
     ORDER BY DEPTNO
```

조회 결과는 다음과 같습니다.

MGRNO	DEPTNO	MAXSAL
000010	A00	52750.00
000020	B01	41250.00
000030	C01	38250.00
000060	D11	32250.00
000070	D21	36170.00
000050	E01	40175.00
000090	E11	29750.00
000100	E21	26150.00

공통 표 표현식은 SELECT문, INSERT문 또는 CREATE VIEW문에서 전체 선택 이전에 지정할 수 있습니다. 그 표현식들은 같은 결과 표가 전체 선택에서 공유되어야 할 때 사용될 수 있습니다. 공통 표 표현식 앞에는 키워드 WITH가 붙습니다.

예를 들어, 특정 부서 집합의 최소 및 최대 평균 급료를 보여주는 표를 원한다고 가정해 보십시오. 부서 번호의 첫 번째 문자에는 어떤 의미가 있으며 사용자가 'D' 문자로 시작하는 부서들과 'E' 문자로 시작하는 부서들에 대해 최소 및 최대 평균 급료를 보려는 경우, 공통 표 표현식을 사용하여 각 부서에 대한 평균 급료를 선택할 수 있습니다. 이 경우에도, 파생된 표에 이름을 부여해야 합니다. 이 경우, 이름은 DT입니다. 그런 후, WHERE절을 사용하는 SELECT문을 지정하여 선택을 특정 문자로 시작하는 부서로만 제한할 수 있습니다. 파생된 표 DT에서 AVGSAL 열의 최소 및 최대값을 지정하십시오. 문자 'E'에 대한 결과와 문자 'D'에 대한 결과를 얻으려면 UNION을 지정하십시오.

```
WITH DT AS (SELECT E.WORKDEPT AS DEPTNO, AVG(SALARY) AS AVGSAL
            FROM CORPDATA.DEPARTMENT D , CORPDATA.EMPLOYEE E
            WHERE D.DEPTNO = E.WORKDEPT
            GROUP BY E.WORKDEPT)
 SELECT 'E', MAX(AVGSAL), MIN(AVGSAL) FROM DT
 WHERE DEPTNO LIKE 'E%'
 UNION
 SELECT 'D', MAX(AVGSAL), MIN(AVGSAL) FROM DT
 WHERE DEPTNO LIKE 'D%'
```

조회 결과는 다음과 같습니다.

	MAX(AVGSAL)	MIN(AVGSAL)
E	40175.00	21020.00
D	25668.57	25147.27

주문 데이터베이스에 대하여 최근 1000개의 고객 주문에서 상위 5 항목(주문된 전체 수량으로)을 리턴하도록 하는 조회를 작성하고자 하며, 고객은 'XXX' 항목도 주문한 상태라고 가정합니다.

```
WITH X AS (SELECT ORDER_ID, CUST_ID
                  FROM ORDERS
                  ORDER BY ORD_DATE DESC
                  FETCH FIRST 1000 ROWS ONLY),
     Y AS (SELECT CUST_ID, LINE_ID, ORDER_QTY
                  FROM X, ORDERLINE
                  WHERE X.ORDER_ID = ORDERLINE.ORDER_ID)
SELECT LINE_ID
     FROM (SELECT LINE_ID
                  FROM Y
                  WHERE Y.CUST_ID IN (SELECT DISTINCT CUST_ID
                                             FROM Y
                                             WHERE LINE.ID = 'XXX' )
                  GROUP BY LINE_ID
                  ORDER BY SUM(ORDER_QTY) DESC)
   FETCH FIRST 5 ROWS ONLY
```

첫 번째 공통 표 표현식(X)는 가장 최근의 1000개의 주문 번호를 리턴합니다. 결과는 날짜의 내림차순으로 정렬되며 주문된 항목의 첫 1000 행만 결과 표로 리턴됩니다.

두 번째 공통 표 표현식(Y)는 최근 1000개의 주문을 라인 항목 표와 결합하여 (1000개의 주문 각각에 대하여)고객, 라인 항목 및 해당 주문에서의 라인 항목 수량 등을 리턴합니다.

주 선택 명령문의 파생 표는 XXX 항목을 주문한 상위 1000개의 주문에 들어 있는 고객의 라인 항목을 리턴합니다. XXX를 주문한 모든 고객에 대한 결과는 라인 항목별로 그룹화되고 이 그룹은 라인 항목 전체 수량별로 정렬됩니다.

마지막으로 외부 선택에서 파생된 표가 리턴한 주문 목록에서 첫 5 행만을 선택합니다.

UNION 키워드를 사용하는 부속 선택 결합

UNION 키워드를 사용하면, 두 개 이상의 부속 선택을 결합하여 전체 선택을 형성할 수 있습니다. SQL이 UNION 키워드를 발견하는 경우, 이는 결과 표를 형성하기 위해 각각의 부속 선택을 처리한 다음 각 부속 선택의 결과 표를 결합하고 결합된 결과 표를 형성하기 위해 복제 행들을 삭제합니다. UNION 키워드를 사용하여 두 개 이상의 표로부터 값의 리스트를 병합합니다. SELECT문 작성시 지금까지 학습한 절 및 기법을 사용할 수 있습니다.

UNION을 사용하면 여러 표에서 얻은 값들의 리스트 병합시 복제되는 것을 없앨 수 있습니다. 예를 들어 다음의 정보를 포함하는 사원 번호의 결합 리스트를 구할 수 있습니다.

- 부서 D11의 사원
- 프로젝트 MA2112, MA2113 및 AD3111을 맡고 있는 사원

결합 리스트는 두 개의 표에서 작성되며 복제가 포함되지 않습니다. 이를 수행하려면 다음과 같이 지정하십시오.

```
SELECT EMPNO
        FROM CORPDATA.EMPLOYEE
      WHERE WORKDEPT = 'D11'
 UNION
  SELECT EMPNO
           FROM CORPDATA.EMPPROJACT
   WHERE PROJNO = 'MA2112' OR
      PROJNO = 'MA2113' OR
      PROJNO = 'AD3111'
   ORDER BY EMPNO
```

또한 공통 표 표현식, 중첩된 표 표현식 또는 보기를 작성할 때 UNION을 사용할 수도 있습니다. 세부사항은 64 페이지의 『UNION을 사용하여 보기 작성』을 참조하십시오.

이들 SQL문의 결과를 보다 쉽게 이해하기 위해 SQL이 다음과 같은 프로세스로 진행된다고 가정합니다.

단계 1. SQL이 첫 번째 SELECT문을 처리합니다.

```
SELECT EMPNO
        FROM CORPDATA.EMPLOYEE
      WHERE WORKDEPT = 'D11'
```

임시적인 결과 표를 작성합니다.

CORPDATA.EMPLOYEE로부터의 EMPNO
000060
000150

CORPDATA.EMPLOYEE로부터의 EMPNO
000160
000170
000180
000190
000200
000210
000220
200170
200220

단계 2. SQL이 두 번째 SELECT문을 처리합니다.

```
SELECT EMPNO
           FROM CORPDATA.EMPPROJACT
     WHERE PROJNO='MA2112' OR
                 PROJNO= 'MA2113' OR
                 PROJNO= 'AD3111'
```

또 다른 임시적인 결과 표를 작성합니다.

CORPDATA.EMPPROJACT로부터의 EMPNO
000230
000230
000240
000230
000230
000240
000230
000150
000170
000190
000170
000190
000150
000160
000180
000170
000210
000210

단계 3. SQL이 중복 행 및 결과 순서화를 제거하여 두 개의 임시적인 결과표를 조합합니다.

```
SELECT EMPNO
         FROM CORPDATA.EMPLOYEE
      WHERE WORKDEPT = 'D11'
```

```
UNION
SELECT EMPNO
           FROM CORPDATA.EMPPROJACT
     WHERE PROJNO='MA2112' OR
                 PROJNO= 'MA2113' OR
                 PROJNO= 'AD3111'
ORDER BY EMPNO
```

값들을 오름차순으로 배열하여 결합된 결과 표를 작성합니다.

EMPNO
000060
000150
000160
000170
000180
000190
000200
000210
000220
000230
000240
200170
200220

UNION 키워드 사용시

- 모든 ORDER BY절은 UNION 키워드의 일부인 최종 부속 선택 뒤에 표시되어야 합니다. 이 예에서 결과는 첫 번째 선택된 열 *EMPNO*에 기초하여 순서화됩니다. ORDER BY절은 결합된 결과 표가 배열된 순서가 되도록 지정합니다. ORDER BY는 보기에서 허용되지 않습니다.
- 결과 열이 명명되면 ORDER BY절에 이름이 지정됩니다. 결합된 선택문 각각의 상응하는 열이 동일한 이름을 가질 경우 결과 열이 명명됩니다. AS절은 선택 리스트에 있는 열에 이름을 할당하는데 사용될 수 있습니다.

  ```
  SELECT A + B AS X ...
  UNION SELECT X ... ORDER BY X
  ```

  결과 열이 명명되지 않으면 결과를 순서화하십시오. 번호는 부속 선택에 포함시킨 표현식 리스트 안에 있는 표현식의 위치를 말합니다.

  ```
  SELECT A + B ...
  UNION SELECT X ... ORDER BY 1
  ```

각 행이 있는 부속 선택을 식별하려면 UNION 키워드에서 각 부속 선택의 SELECT 리스트 끝에 상수를 포함시킬 수 있습니다. SQL이 결과를 리턴시킬 때 최종 열에는 그 행의 소스인 부속 선택에 대한 상수가 포함됩니다. 예를 들어, 다음을 지정할 수 있습니다.

```
SELECT A, B, 'A1' ...
UNION
SELECT X, Y, 'B2'...
```

행이 리턴되면, 이에는 행 값의 소스인 표를 표시하기 위한 값(A1 또는 B2)이 포함됩니다. UNION 키워드에서 열 이름이 다르면 대화식 SQL이 결과를 표시하거나 인쇄할 때 첫 번째 부속 선택에서 지정된 열 이름 세트를 사용하거나 SQL DESCRIBE문 처리시 생성된 SQLDA에서 지정된 열 이름 세트를 사용합니다.

UNION에서 열의 길이 및 자료 유형의 호환성에 대한 정보는 SQL 참조서의 결과 자료 유형의 규칙 주제를 참조하십시오.

주: 정렬 순서는 여러 UNION에 걸쳐 있는 필드가 호환될 수 있으면 적용됩니다. 정렬 순서는 UNION 처리중에 내재적으로 발생하는 고유한 처리에 사용됩니다. 정렬 순서에 대한 자세한 내용은 129 페이지의 제 8 장『SQL 정렬 순서』를 참조하십시오.

UNION ALL 지정

UNION 결과의 복제를 보유하려면 UNION 대신 UNION ALL을 지정하십시오. UNION과 동일한 단계 및 예를 사용합니다.

단계 3. SQL은 두 개의 임시적인 결과 표를 조합합니다.

```
SELECT EMPNO
        FROM CORPDATA.EMPLOYEE
      WHERE WORKDEPT = 'D11'
 UNION ALL
SELECT EMPNO
           FROM CORPDATA.EMPPROJACT
     WHERE PROJNO='MA2112' OR
                 PROJNO= 'MA2113' OR
                 PROJNO= 'AD3111'
 ORDER BY EMPNO
```

중복되는 것이 포함된 결과표가 작성됩니다.

EMPNO
000060
000150
000150
000150
000160
000160

EMPNO
000170
000170
000170
000170
000180
000180
000190
000190
000190
000200
000210
000210
000210
000220
000230
000230
000230
000230
000230
000240
000240
200170
200220

UNION ALL 조작은 결합적입니다. 예를 들어 다음을 보십시오.

```
(SELECT PROJNO FROM CORPDATA.PROJECT
UNION ALL
SELECT PROJNO FROM CORPDATA.PROJECT)
UNION ALL
SELECT PROJNO FROM CORPDATA.EMPPROJACT
```

또한 이 명령문은 다음과 같이 기록될 수도 있습니다.

```
SELECT PROJNO FROM CORPDATA.PROJECT
UNION ALL
(SELECT PROJNO FROM CORPDATA.PROJECT
UNION ALL
SELECT PROJNO FROM CORPDATA.EMPPROJACT)
```

그러나 UNION 연산자로서 동일한 SQL문에 UNION ALL을 포함시키면 조작 결과는 평가 순서에 따라 달라집니다. 괄호가 없으면 평가는 왼쪽에서 오른쪽으로 수행됩니다. 괄호가 포함되었으면 괄호 내의 부속 선택이 먼저 평가되고 다음에는 명령문의 다른 부분이 왼쪽에서 오른쪽으로 평가됩니다.

자료 검색 오류

SQL이 호스트 변수로 검색된 문자나 그래픽 열이 너무 길면 다음을 수행합니다.

- 호스트 변수에 값을 할당하면서 자료를 절단합니다.
- SQLCA의 SQLWARN0과 SQLWARN1을 값 'W'로 설정합니다.
- 인디케이터 변수가 제공될 경우, 절단 이전의 길이 값으로 설정합니다.

SQL이 명령문이 실행되는 동안 자료 맵핑 오류를 발견하면 다음 중 하나가 발생합니다.

- SELECT 리스트 내의 표현식에 오류가 있고 오류가 발생한 표현식에 인디케이터 변수가 제공되는 경우에 다음 상황이 발생합니다.
  - SQL이 인디케이터 변수에 대해 오류를 가진 표현식에 해당하는 -2를 리턴시킵니다.
  - SQL이 해당 행에 대해 유효한 모든 자료를 리턴시킵니다.
  - SQL이 양의 SQLCODE를 리턴시킵니다.
- 인디케이터 변수가 제공되는 경우에는, SQL이 SQLCA내 해당 음의 SQLCODE를 리턴시킵니다.

자료 맵핑 오류는 다음과 같습니다.

- +138 - 서브스트링 함수의 인수가 유효하지 않습니다.
- +180 - 날짜, 시간 또는 시간소인의 스트링 표시에 대한 구문이 유효하지 않습니다.
- +181 - 날짜, 시간 또는 시간소인의 스트링 표시가 유효한 값이 아닙니다.
- +183 - 날짜/시간 표현식에서 결과가 유효하지 않습니다. 결과 날짜나 시간소인이 유효한 범위에 있지 않습니다.
- +191 - MIXED 자료가 적절하게 구성되지 않습니다.
- +304 - 숫자 변환 오류(예를 들면, 넘침, 부족 또는 0으로 나눔)입니다.
- +331 - 문자가 변환될 수 없습니다.
- +420 - CAST 인수의 문자가 유효하지 않습니다.
- +802 - 자료 변환 또는 자료 맵핑 오류입니다.

자료 맵핑 오류의 경우, SQLCA는 마지막으로 검출된 오류만을 보고합니다. 오류를 가진 각 결과 열에 해당하는 인디케이터 변수가 -2로 설정됩니다.

전체 SELECT에 SELECT 리스트 내의 DISTINCT가 들어 있고 SELECT 리스트 내의 열에 유효하지 않은 숫자 자료가 들어 있으면 조회가 정렬로 완료된 경우, 자료는 널(null)과 동등한 값으로 간주됩니다. 기존 색인이 사용되면 자료는 널과 동등한 값으로 간주되지 않습니다.

자료 맵핑 오류가 ORDER BY절에 미치는 영향은 상황에 따라 다릅니다.

- SELECT INTO 또는 FETCH문에서 호스트 변수에 자료가 할당되는 동안 자료 맵핑 오류가 발생하고 ORDER BY절에 동일한 표현식이 사용될 경우, 결과 레코드는 표현식의 값을 토대로 순서가 정해집니다. 그것이 널(다른 모든 값보다 상위인)인 것처럼 배열되지는 않습니다. 이는 호스트 변수에 대한 할당을 시도하기 전에 표현식이 평가되었기 때문입니다.
- 선택 리스트의 표현식이 평가되는 동안 자료 맵핑 오류가 발생하고 ORDER BY절에서 동일한 표현식이 사용될 경우, 결과 열은 마치 그것이 널 값(다른 모든 값보다 상위인)인 것처럼 정상적으로 배열됩니다. ORDER BY절이 정렬을 사용하여 수행되면, 결과 열은 그것이 널 값인 것처럼 배열됩니다. 다음과 같은 경우 ORDER BY 절이 기존 색인을 사용하여 수행되면 결과 열은 색인에 있는 표현식의 실제 값을 기초로 순서가 지정됩니다.
  - 표현식이 *MDY, *DMY, *YMD 또는 *JUL 날짜 형식인 날짜 열이고 날짜가 유효한 범주 내에 있지 않기 때문에 날짜 변환 오류가 발생한 경우
  - 표현식이 문자 열이고 문자가 변환될 수 없는 경우
  - 표현식이 10진 열이고 유효하지 않은 숫자값이 검출된 경우

제 6 장 SQL 삽입, 갱신 및 삭제

이 절에서는 자료를 갱신, 삭제 및 삽입하고 표와 보기에 자료를 삽입하는 기본 SQL 문과 절을 보여줍니다. 사용된 SQL문은 UPDATE, DELETE 및 INSERT입니다. 이러한 SQL문을 사용한 예는 SQL 어플리케이션을 개발하는 데 도움을 주기 위해 제공됩니다. SQL문에 대한 자세한 구문과 매개변수 설명은 SQL 참조서에 나와 있습니다.

주:

1. 여기에서 설명되는 SQL문은 SQL 표, 보기, 데이터베이스 실제 및 논리 파일에서 실행시킬 수 있습니다. 표, 보기 및 파일은 스키마 또는 라이브러리 중 하나에 있을 수 있습니다.
2. SQL문에 지정된 문자 스트링(WHERE 또는 VALUES절에 사용된 것과 같은)은 대소문자가 구분됩니다. 즉, 대문자는 대문자로 입력되어야 하고 소문자는 소문자로 입력되어야 합니다.

   ```
   WHERE ADMRDEPT='a00'      (결과를 리턴하지 않음)

   WHERE ADMRDEPT='A00'      (유효한 부서 번호 리턴)
   ```

   대문자 및 소문자가 동일한 문자로 취급되는 곳에서 공유 가중치 정렬 순서가 사용되는 경우, 문자 스트링에 대소문자를 구분할 수 없습니다. 정렬 순서에 대한 자세한 정보는 129 페이지의 제 8 장『SQL 정렬 순서』를 참조하십시오.

이 장에 포함된 절은 다음과 같습니다.

INSERT 명령문을 사용하여 열 삽입

사용자는 INSERT문을 사용하여 다음 중 하나의 방법으로 표나 보기에 새로운 행을 추가할 수 있습니다.

- 추가될 단일 행의 열에 대해 INSERT문에서 값 지정.
- 다른 표나 보기에 새로운 행에 대해 어떤 자료가 포함되는지를 SQL에 알리기 위해 INSERT문에 SELECT문 포함. 108 페이지의『SELECT문을 사용하여 표에 행 삽입』에는 INSERT문 내에서 SELECT문을 사용하여 표에 행을 추가하는 방법이 설명되어 있습니다.

- 복수 행의 추가를 위해 블록 형식의 INSERT 지정. 109 페이지의 『블록 INSERT 명령문으로 표에 복수 행 삽입』에서는 표에 복수 행을 추가하기 위해 블록 형식의 INSERT문을 사용하는 방법에 대해 설명합니다.

주: 보기는 표 위에 작성되며 실제로 자료를 포함하지 않으므로 보기에 대한 작업이 혼란스러울 수 있습니다. 보기를 사용한 자료 삽입에 대한 자세한 정보 및 제한사항은 63 페이지의 『보기 작성 및 사용』을 참조하십시오.

INSERT의 완벽한 설명은 SQL 참조서의 INSERT문을 참조하십시오.

삽입하는 모든 행에서 사용자는 NOT NULL 속성으로 정의된 열이 디폴트 값을 갖고 있지 않을 경우에 그 각각의 열에 값을 제공해야 합니다. 표나 보기에 행을 추가하기 위한 INSERT문은 다음과 같을 수 있습니다.

```
INSERT INTO table-name
   (column1, column2, ...)
VALUES (value-for-column1, value-for-column2, ...)
```

INTO절은 사용자가 값을 지정하는 열을 명명합니다. VALUES절은 INTO절에서 명명된 각 열에 값을 지정합니다. 사용자가 지정하는 값은 다음을 수행할 수 있습니다.

상수. VALUES 절에 제공된 값을 삽입합니다.

널 값. 키워드 NULL을 사용하여 널 값을 삽입합니다. 열이 널 값 또는 오류 발생을 포함할 수 있다고 정의해야 합니다.

호스트 변수. 호스트 변수의 내용을 삽입합니다.

특수 레지스터. 예를 들어, USER와 같이 특수 레지스터 값을 삽입합니다.

표현식. 표현식의 결과인 값을 삽입합니다.

부속 조회는 select문 실행 결과 값을 삽입합니다.

DEFAULT 키워드. 열의 디폴트 값을 삽입합니다. 이 열은 그에 대해 정의된 디폴트 값을 가지거나 널 값을 허용해야 하며 그렇지 않으면 오류가 발생합니다.

INSERT 명령문의 열 리스트에 명명된 각 열의 VALUES절에는 값을 반드시 제공해야 합니다. 열 이름 리스트는 표 내의 모든 열에 VALUES절에서 제공된 값이 있다면 생략될 수 있습니다. 열이 디폴트 값을 가지는 경우, 키워드 DEFAULT는 VALUES 절의 값으로 사용될 수 있습니다. 이로 인해 열의 기본 값이 열에 위치하게 됩니다.

삽입중인 모든 열을 명명하는 것이 좋은 이유는 다음과 같습니다.

- INSERT문이 보다 설명적인 성격을 갖게 됩니다.
- 열 이름을 기초로 하여 올바른 순서로 값을 제공함을 확인할 수 있습니다.
- 자료 독립성이 향상됩니다. 열이 표에 정의된 순서는 INSERT문에 영향을 주지 않습니다.

열이 널 값을 허용하거나 디폴트를 갖도록 정의되면, 그 열을 열 이름 리스트에 명명하거나 열에 값을 지정할 필요가 없습니다. 디폴트 값이 사용됩니다. 열이 디폴트 값을 갖도록 정의되면 열에 디폴트 값이 배치됩니다. 명시적인 디폴트 값 없이 열 정의에 대해 DEFAULT가 지정되었다면 SQL이 열의 자료 유형에 대해 디폴트 값을 놓습니다. 널 값을 허용하는 것으로 정의된(열 정의에 NOT NULL이 지정되지 않은) 경우, SQL은 열에 널 값을 배치합니다.

- 숫자 열의 경우, 디폴트 값은 0입니다.
- 고정 길이의 문자 또는 그래픽 열의 경우, 디폴트는 공백입니다.
- 가변 길이의 문자나 그래픽 열 또는 LOB 열의 경우, 디폴트는 길이가 0인 스트링입니다.
- 날짜, 시간, 시간소인 열의 경우, 디폴트 값은 현재 날짜, 시간, 시간소인입니다. 레코드 블록 삽입시, 디폴트 날짜/시간 값은 블록 작성시의 시스템에서 발췌됩니다. 이것은 그 열이 블록의 각 열에 대해 같은 디폴트 값을 할당할 것임을 의미합니다.
- 자료 링크 열의 경우, 디폴트 값은 DLVALUE(’’,’URL’,’’)에 해당됩니다.
- 고유 유형 열의 경우, 디폴트 값은 해당되는 소스 유형의 디폴트 값입니다.
- ROWID 열 또는 AS IDENTITY로 정의된 열의 경우 데이터베이스 관리자는 기본 값을 생성합니다. 109 페이지의 『식별 열에 삽입』을 참조하십시오.

프로그램이 이미 표에 있는 다른 행과 복제되는 행을 삽입하고자 할 때에는 오류가 발생합니다. 색인 작성시 사용된 옵션에 따라 복수의 널 값이 복제값을 고려할 수도 고려하지 않을 수도 있습니다.

- 표가 1차 키, 고유 키 또는 고유 색인을 가질 경우 행은 삽입되지 않습니다. 대신, SQL이 -803의 SQLCODE를 리턴합니다.
- 표가 1차 키, 고유 키 또는 고유 색인을 갖지 않을 경우 행은 오류 없이 삽입될 수 있습니다.

INSERT문 수행시 SQL이 오류를 발견하면 자료 삽입이 중단됩니다. COMMIT(*ALL), COMMIT(*CS), COMMIT(*CHG), COMMIT(*RR)을 지정하는 경우에는 어느 행도 삽입되지 않습니다. SELECT문이 있는 INSERT 또는 블록 삽입의 경우, 이 명령문에 의해 이미 삽입된 행은 삭제됩니다. COMMIT(*NONE)을 지정하는 경우, 이미 삽입된 행은 *삭제되지 않습니다*.

SQL로 작성된 표가 레코드 삭제 매개변수 *YES로 작성됩니다. 데이터베이스 관리자는 이를 사용하여 삭제된 것으로 표시된 표 내의 행을 다시 사용할 수 있습니다. CHGPF 명령을 사용하여 *NO로 속성을 변경할 수 있으며 이것은 INSERT가 항상 표의 끝에 행을 삽입하도록 만듭니다.

행이 삽입된 순서가 검색될 순서를 보장하지는 않습니다.

오류 없이 행이 삽입되면 SQLCA의 SQLERRD(3) 필드가 1의 값을 가집니다.

주: 블록 INSERT 또는 SELECT문이 있는 INSERT에는 두 개 이상의 행이 삽입될 수 있습니다. 삽입된 행의 수는 SQLERRD(3)에 반영됩니다.

SELECT문을 사용하여 표에 행 삽입

INSERT문 내의 SELECT문을 사용하여 지정한 표나 보기에서 선택된 0, 1 또는 그 이상의 행을 다른 표에 삽입할 수 있습니다. 행을 선택해오는 표는 삽입하려는 표와 같은 것일 수 있습니다. 표가 동일할 경우에 SQL은 선택된 행이 들어 있는 임시 결과 표를 작성한 후에 임시표에서 목표 표로 삽입하게 됩니다.

이러한 종류의 INSERT문을 사용하는 한가지 방법은 요약 자료에 대해 작성한 표에 자료를 이동하는 것입니다. 예를 들어, 프로젝트에 대한 각 사원 완료 시간을 보여주는 표를 원한다고 가정합니다. *EMPNUMBER*, *PROJNUMBER*, *STARTDATE* 및 *ENDDATE* 열이 있는 EMPTIME 표를 작성한 다음, 다음의 INSERT문을 사용하여 표를 채울 수 있습니다.

```
INSERT INTO CORPDATA.EMPTIME
   (EMPNUMBER, PROJNUMBER, STARTDATE, ENDDATE)
SELECT EMPNO, PROJNO, EMSTDATE, EMENDATE
         FROM CORPDATA.EMPPROJACT
```

INSERT문에 삽입된 SELECT문은 자료 검색에 사용된 SELECT문과 다르지 않습니다. FOR READ ONLY, FOR UPDATE 또는 OPTIMIZE절을 제외하고는 자료 검색에 사용된 모든 키워드, 열 함수 및 기법을 사용할 수 있습니다. SQL은 탐색 조건을 만족시키는 모든 행을 사용자가 지정하는 표에 삽입합니다. 하나의 표에서 다른 표로 행을 삽입하는 것은 소스 표나 목표 표 내의 기존 행에 영향을 미치지 않습니다.

복수 행 삽입에 관한 주의사항

표에 여러 행을 삽입할 경우에는 다음을 고려하십시오.

- INSERT문에서 내재적으로 또는 명시적으로 리스트된 열 수는 SELECT문에 열람된 열 수와 동일해야 합니다.
- 선택중인 열의 자료는 SELECT문에 INSERT를 사용할 때 삽입중인 열과 호환되는 것이어야 합니다.
- INSERT문에 삽입된 SELECT문이 행을 리턴시키지 않으면 행이 삽입되지 않았음을 경고하기 위해 SQLCODE에 100이 리턴됩니다. 정상적으로 행이 삽입되면 SQLCA의 SQLERRD(3) 필드에 실제로 SQL이 삽입한 행의 수를 나타내는 정수가 포함됩니다.
- INSERT문 수행시 SQL이 오류를 발견하면 SQL은 조작을 중단합니다. 만일 COMMIT (*CHG), COMMIT(*CS), COMMIT (*ALL) 또는 COMMIT(*RR)을 지정한다면 표에는 아무 것도 삽입되지 않고 음의 SQLCODE가 리턴됩니다. COMMIT(*NONE)를 지정하면 오류 이전에 삽입된 행은 표에 그대로 남아 있습니다.

- 두 개 이상의 표를 INSERT문의 SELECT문과 결합할 수 있습니다. 이와 같은 방법으로 로드되면 행들이 표에 실제로 저장된 행으로 존재하기 때문에 표가 UPDATE, DELETE, INSERT문으로 조작될 수 있습니다.

블록 INSERT 명령문으로 표에 복수 행 삽입

블록 INSERT문을 사용해 복수 행을 단일 명령문으로 된 표에 삽입할 수 있습니다. 블록 INSERT문은 REXX를 제외한 모든 언어에 지원됩니다. 표에 삽입된 자료는 호스트 구조 배열에 있어야 합니다. 인디케이터 변수에 블록 INSERT가 사용되면 변수는 또한 호스트 구조 배열에 있어야 합니다. 특정 언어에 대한 호스트 구조 배열 정보는 호스트 언어를 사용한 SQL 프로그래밍에서 해당 언어에 대한 장을 참조하십시오.

예를 들어 CORPDATA.EMPLOYEE 표에 10명의 사원을 추가하려면 다음과 같이 하십시오.

```
INSERT INTO CORPDATA.EMPLOYEE
                    (EMPNO,FIRSTNME,MIDINIT,LASTNAME,WORKDEPT)
             10 ROWS VALUES(:DSTRUCT:ISTRUCT)
```

DSTRUCT는 프로그램에서 선언된 다섯 개의 요소로 된 호스트 구조 배열입니다. 다섯 개의 요소는 EMPNO, FIRSTNME, MIDINIT, LASTNAME 및 WORKDEPT입니다. DSTRUCT에는 10개 행을 삽입할 수 있는 최소한 10차원이 있습니다. ISTRUCT는 프로그램에서 선언된 호스트 구조 배열입니다. ISTRUCT는 인디케이터에 대해 최소한 10차원의 작은 정수 필드를 가집니다.

블록화된 INSERT문은 어플리케이션 서버와 어플리케이션 리퀘스터가 모두 iSeries 시스템인 비분산 SQL 어플리케이션 및 분산 어플리케이션에 대해 지원됩니다.

식별 열에 삽입

식별 열에 값을 삽입하거나 시스템이 값을 삽입하도록 허용할 수 있습니다. 예를 들어, 표는 ORDERNO(식별 열), SHIPPED_TO (varchar(36)) 및 ORDER_DATE(날짜) 열이 있는 56 페이지의 『식별 열 작성 및 변경』에 작성됩니다. 다음 명령문을 발행하여 이 표로 행을 삽입할 수 있습니다.

```
INSERT INTO ORDERS (SHIPPED_TO, ORDER_DATE)
    VALUES ('BME TOOL', 2002-02-04)
```

이 경우, 값이 식별 열에 대한 시스템에 의해 자동으로 생성됩니다. 또한 DEFAULT 키워드를 사용하여 이 명령문 기록할 수도 있습니다.

```
INSERT INTO ORDERS (SHIPPED_TO, ORDER_DATE, ORDERNO)
    VALUES ('BME TOOL', 2002-02-04, DEFAULT)
```

삽입 후, IDENTITY_VAL_LOCAL 함수를 사용하여 시스템이 열에 지정한 값을 판별할 수 있습니다. 자세한 내용 및 예는 SQL 참조서의 IDENTITY_VAL_LOCAL 함수를 참조하십시오.

| 때로 식별 열의 값을 SELECT를 사용하는 다음과 같은 INSERT문에서와 같이 사용자가 지정하는 경우:

```
INSERT INTO ORDERS OVERRIDING USER VALUE
   (SELECT * FROM TODAYS_ORDER)
```

| 이 경우, OVERRIDING USER VALUE는 시스템에게 SELECT로부터의 식별 열에 제공되는 값을 무시하고 식별 열에 새로운 값을 생성하도록 지시합니다. GENERATED ALWAYS를 사용하여 식별 열을 작성한 경우 OVERRIDING USER VALUE를 사용해야 합니다. GENERATED BY DEFAULT에 대해서는 선택적입니다.
| OVERRIDING USER VALUE가 GENERATED BY DEFAULT 식별 열에 대해 저정되지 않은 경우, 값은 SELECT에 삽입된 열에 대해 제공됩니다.

| OVERRIDING SYSTEM VALUE를 지정하여 시스템이 GENERATED ALWAYS 식별 열에 select로부터의 값을 사용하도록 강제할 수 있습니다. 예를 들어, 다음 명령문을 발행합니다.

```
INSERT INTO ORDERS OVERRIDING SYSTEM VALUE
   (SELECT * FROM TODAYS_ORDER)
```

| 이 INSERT문은 SELECT로부터의 값을 사용합니다. 식별 열을 위해 새 값을 생성하지 않습니다. OVERRIDING SYSTEM VALUE절을 사용하지 않고 GENERATED ALWAYS를 사용하여 작성된 식별 열에 대해 값을 제공할 수 없습니다.

UPDATE 명령문을 사용하여 표의 자료 변경

표에서 자료를 변경하려면 UPDATE문을 사용하십시오. UPDATE문을 사용하면 WHERE절의 탐색 조건을 만족시키는 각 행에서 하나 이상의 열값을 변경할 수 있습니다. UPDATE문은(WHERE절에서 지정된 탐색 조건을 만족시키는 행 수에 따라) 표의 0개 이상의 행에서 하나 이상의 열값을 변경합니다. UPDATE문은 다음과 같습니다.

```
UPDATE table-name
  SET column-1 = value-1,
      column-2 = value-2, ...
  WHERE search-condition ...
```

예를 들어 한 사원의 자리가 이동된다고 가정합니다. 이동을 반영하기 위해 CORPDATA.EMPLOYEE 표의 여러 사원 자료 항목을 갱신하려는 경우 다음과 같이 지정할 수 있습니다.

```
UPDATE CORPDATA.EMPLOYEE
  SET JOB = :PGM-CODE,
      PHONENO = :PGM-PHONE
    WHERE EMPNO = :PGM-SERIAL
```

갱신하려는 각 열에 새로운 값을 지정하려면 SET절을 사용하십시오. SET절은 갱신하려는 열을 명명하고 변경하려는 값을 제공합니다. 사용자가 지정하는 값은 다음을 수행할 수 있습니다.

열 이름. 열의 현재 값을 같은 행의 다른 열에 있는 내용으로 대체합니다.

상수. 열의 현재 값을 SET절에 제공된 값으로 대체합니다.

널 값. 열의 현재 값을 키워드 NULL을 사용하여 널 값으로 대체합니다. 표 작성 시 열이 널 값을 포함할 수 있다고 정의해야 합니다. 그렇지 않으면 오류가 발생합니다.

호스트 변수. 열의 현재 값을 호스트 변수의 내용으로 대체합니다.

특수 레지스터. 열의 현재 값을 특수 레지스터 값(예: USER)으로 대체합니다.

표현식. 열의 현재 값을 표현식에서 나온 값으로 대체합니다.

A **스칼라 부속 선택.** 열의 현재 값을 부속 조회가 리턴한 값으로 대체합니다.

DEFAULT 키워드. 열의 현재 값을 열의 디폴트 값으로 대체합니다. 이 열은 그에 대해 정의된 디폴트 값을 가지거나 널 값을 허용해야 하며 그렇지 않으면 오류가 발생합니다.

다음은 서로 다른 값을 사용하는 명령문의 예입니다.

```
UPDATE WORKTABLE
   SET COL1 = 'ASC',
       COL2 = NULL,
       COL3 = :FIELD3,
       COL4 = CURRENT TIME,
       COL5 = AMT - 6.00,
       COL6 = COL7
    WHERE EMPNO = :PGM-SERIAL
```

행이 갱신되도록 지정하려면 WHERE절을 사용하십시오.

- 단일 행을 갱신하려면 단지 하나의 행만을 선택하는 WHERE절을 사용하십시오.
- 복수 행을 갱신하려면 갱신하려는 행만을 선택하는 WHERE절을 사용하십시오.

WHERE절은 생략할 수도 있습니다. 생략한 경우, SQL은 표나 보기에 있는 각 열을 사용자가 공급하는 값으로 갱신합니다.

UPDATE문 수행시 데이터베이스 관리자가 오류를 발견하면 갱신을 중단하고 음의 SQLCODE를 리턴시킵니다. COMMIT(*ALL), COMMIT(*CS), COMMIT(*CHG) 또는 COMMIT(*RR)를 지정하면 표 내의 행이 변경되지 않습니다(이 명령문에 의해 이미 변경된 행이 있을 경우, 이전 값으로 복원됩니다). COMMIT(*NONE)가 지정되는 경우, 이미 변경된 행은 이전 값으로 *복원되지 않습니다*.

데이터베이스 관리자가 탐색 조건을 만족시키는 행을 찾지 못하면 +100인 SQLCODE가 리턴됩니다.

주: UPDATE문에는 두 개 이상의 갱신된 행이 있을 수도 있습니다. 갱신된 행 수는 SQLCA의 SQLERRD(3)에 반영됩니다.

UPDATE문의 SET절은 갱신중인 각 행에 설정되는 실제 값을 결정하기 위해 다양한 방법으로 사용될 수 있습니다. 다음 예에는 각 열이 해당하는 값과 함께 나타납니다.

```
UPDATE EMPLOYEE
  SET WORKDEPT = 'D11',
      PHONENO = '7213',
      JOB = 'DESIGNER'
  WHERE EMPNO = '000270'
```

모든 열과 값들을 지정하면 이전의 갱신도 기록될 수 있습니다.

```
UPDATE EMPLOYEE
  SET (WORKDEPT, PHONENO, JOB)
      = ('D11', '7213', 'DESIGNER')
  WHERE EMPNO = '000270'
```

표의 데이터를 갱신하는 자세한 방법은 다음 절을 참조하십시오.

- 『스칼라 부속 선택을 사용하여 표 갱신』
- 『또다른 표에서 행을 사용하여 표 갱신』
- 113 페이지의『표 검색 시 자료 갱신』

UPDATE문에 대한 완벽한 설명은 SQL 참조서의 UPDATE를 참조하십시오.

스칼라 부속 선택을 사용하여 표 갱신

갱신 값을 선택하는 또다른 방법으로는 스칼라 부속 선택을 사용하는 것입니다. 스칼라 부속 선택을 사용하면 하나 이상의 열이 다른 표에서 선택한 하나 이상의 값에 설정되어 갱신될 수 있습니다. 다음 예에서 사원들은 다른 부서로 이동하지만, 계속 같은 프로젝트를 수행합니다. 새로운 부서 번호를 수용하기 위해 사원 표가 이미 갱신되었습니다. 이제 프로젝트 표가 이 사원의 새로운 부서 번호(부서 번호는 '000030'임)를 반영하기 위해 갱신되어야 합니다.

```
UPDATE PROJECT
  SET DEPTNO =
        (SELECT WORKDEPT FROM EMPLOYEE
           WHERE PROJECT.RESPEMP = EMPLOYEE.EMPNO)
  WHERE RESPEMP='000030'
```

이같은 기법은 단일 선택에서 리턴된 복수 값으로 열 리스트를 갱신하는데 사용할 수 있습니다.

또다른 표에서 행을 사용하여 표 갱신

또한 하나의 표에 있는 전체 행을 다른 표에 있는 행 값으로 갱신할 수도 있습니다. 표의 사본에 변경된 대로 갱신되어야 하는 마스터 클래스 스케줄 표가 있다고 가정합

니다. 작업 사본에서 변경되고 매일 밤 마스터표와 병합됩니다. 두 표에는 똑같은 열이 있으며 하나의 열, CLASS_CODE는 고유 키 열입니다.

```
UPDATE CL_SCHED
  SET ROW =
       (SELECT * FROM MYCOPY
          WHERE CL_SCHED.CLASS_CODE = MYCOPY.CLASS_CODE)
```

이것은 CL_SCHED에 있는 모든 행을 MYCOPY의 값으로 갱신할 것입니다.

식별 열 갱신

식별 열의 값을 지정된 값으로 갱신하거나 시스템이 새 값을 생성하도록 할 수 있습니다. 예를 들어, 표를 사용하면 ORDERNO(식별 열), SHIPPED_TO (varchar(36)) 및 ORDER_DATE(날짜) 열이 있는 56 페이지의 『식별 열 작성 및 변경』에 작성되고, 다음 명령문을 발행하여 식별 열의 값을 갱신할 수 있습니다.

```
UPDATE ORDERS
  SET (ORDERNO, ORDER_DATE)=
       (DEFAULT, 2002-02-05)
  WHERE SHIPPED_TO = 'BME TOOL'
```

값은 식별 열에 대한 시스템에 의해 자동으로 생성됩니다. OVERRIDING SYSTEM VALUE 절을 사용하여 시스템이 값을 생성하도록 하는 것을 대체할 수 있습니다.

```
UPDATE ORDERS OVERRIDING SYSTEM VALUE
  SET (ORDERNO, ORDER_DATE)=
       (553, '2002-02-05')
  WHERE SHIPPED_TO = 'BME TOOL'
```

표 검색 시 자료 갱신

커서를 사용하여 자료 행을 검색함과 동시에 갱신할 수 있습니다. 커서에 대한 자세한 정보는 135 페이지의 제 9 장 『커서 사용』을 참조하십시오. SELECT문에 FOR UPDATE OF와 그 뒤에 갱신될 열의 리스트를 사용하십시오. 그런 후 커서 제어 UPDATE문을 사용하십시오. WHERE CURRENT OF절이 갱신하려는 행을 가리키는 커서를 명명합니다. DYNAMIC절 없이 FOR UPDATE OF절, ORDER BY절, FOR READ ONLY절 또는 SCROLL절이 지정되지 않으면 모든 열이 갱신됩니다.

복수 행 FETCH문이 지정되고 수행되었다면, 커서는 블록의 마지막 행에 있게 됩니다. 그러므로 UPDATE문에 WHERE CURRENT OF절을 지정하면 블록의 마지막 행이 갱신됩니다. 블록 내의 행이 갱신되어야 하는 경우 프로그램이 그 행의 맨 앞으로 커서를 위치시켜야 합니다. 그런 후 UPDATE WHERE CURRENT OF를 지정할 수 있습니다. 다음 예를 고려합니다.

표 8. 표 갱신

화면이동 커서 SQL문	주석
EXEC SQL **DECLARE** THISEMP **DYNAMIC SCROLL CURSOR FOR** **SELECT** EMPNO, WORKDEPT, BONUS **FROM** CORPDATA.EMPLOYEE **WHERE** WORKDEPT = 'D11' **FOR UPDATE OF** BONUS END-EXEC.	
EXEC SQL **OPEN** THISEMP END-EXEC.	
EXEC SQL **WHENEVER NOT FOUND** GO TO CLOSE-THISEMP END-EXEC.	
EXEC SQL **FETCH NEXT FROM** THISEMP **FOR** 5 **ROWS** **INTO** :DEPTINFO :IND-ARRAY END-EXEC.	DEPTINFO와 IND-ARRAY가 프로그램 내에 호스트 구조 배열과 인디케이터 배열로 선언됩니다.
... 부서 D11의 사원 중 $500.00 미만의 보너스를 받은 사람이 있는지 조사합니다. 만약 있다면, 해당 레코드의 값을 $500.00으로 조정합니다.	
EXEC SQL **FETCH** RELATIVE :NUMBACK **FROM** THISEMP END-EXEC.	... 역순으로, 페치하여 갱신하려는 블록의 레코드에 위치합니다.
EXEC SQL **UPDATE** CORPDATA.EMPLOYEE **SET** BONUS = 500 **WHERE CURRENT OF** THISEMP END-EXEC.	... 부서 D11의 보너스가 $500.00 미만인 사원의 보너스를 갱신합니다.
EXEC SQL **FETCH** RELATIVE :NUMBACK **FROM** THISEMP **FOR** 5 **ROWS** **INTO** :DEPTINFO :IND-ARRAY END-EXEC.	... 이미 페치된 블록의 처음으로 옮겨 다시 그 블록을 페치합니다. (NUMBACK -(5 - NUMBACK - 1))
... 해당 블록에 $500.00 미만의 보너스를 받는 사원이 더 있는지 알아보기 위해 분기합니다. ... 다음 행의 블록을 페치하여 처리하기 위해 분기합니다.	
CLOSE-THISEMP. EXEC SQL **CLOSE** THISEMP END-EXEC.	

제한사항

FOR UPDATE OF는 다음 요소 중 하나를 포함하는 SELECT문과 함께 사용할 수 없습니다.

- 첫 번째 FROM절이 둘 이상의 표나 보기를 식별합니다.
- 첫 번째 FROM절이 읽기 전용 보기를 식별합니다.
- 첫 번째 SELECT절이 키워드 DISTINCT를 지정합니다.
- 첫 번째 FROM절이 사용자 정의 표 함수를 식별합니다.
- 외부 부속 선택에 GROUP BY절이 포함됩니다.
- 외부 부속 선택에 HAVING절이 포함됩니다.
- 첫 번째 SELECT절에 열 함수가 포함됩니다.
- SELECT문에 UNION 또는 UNION ALL 연산자가 포함됩니다.
- 선택문이 ORDER BY절을 포함하며 FOR UPDATE OF절과 DYNAMIC SCROLL이 지정되지 않습니다.
- SELECT문에 FOR FETCH ONLY절이 포함됩니다.
- SCROLL 키워드가 DYNAMIC 없이 지정됩니다.
- 선택 리스트에 DATALINK 열이 포함되고 FOR UPDATE OF절은 지정되지 않습니다.
- 첫 번째 부속 선택에는 임시 결과 표가 필요합니다.
- SELECT문에 FETCH FIRST *n* ROWS ONLY가 포함됩니다.

FOR UPDATE OF절이 지정되면 FOR UPDATE OF절에 지정되지 않은 열을 갱신할 수 없습니다. 그러나 다음 예에서와 같이 SELECT 리스트에는 없는 열을 FOR UPDATE OF절에 지정할 수 있습니다.

```
SELECT A, B, C FROM TABLE
FOR UPDATE OF A,E
```

FOR UPDATE OF절에는 필요 이상의 열 이름을 지정하지 마십시오. 이러한 행의 색인은 표를 액세스할 때 사용되지 않습니다.

DELETE 명령문을 사용하여 표에서 행 제거

표에서 행을 제거하려면 DELETE문을 사용하십시오. DELETE문으로 행을 삭제하면 전체 행이 제거됩니다. DELETE문은 행에서 특정 열을 제거하지 않습니다. DELETE문은(WHERE절에 지정된 탐색 조건을 만족시키는 행의 수에 따라) 0 이상의 표 행을 제거합니다. DELETE문에서 WHERE절을 생략하면 SQL은 표의 모든 행을 제거합니다. DELETE문은 다음과 같습니다.

```
DELETE FROM table-name
  WHERE search-condition ...
```

예를 들어 부서 D11이 다른 장소로 이동했다고 가정합니다. 다음과 같이 D11의 WORKDEPT 값으로 CORPDATA.EMPLOYEE 표 내의 각 행을 삭제할 수 있습니다.

```
DELETE FROM CORPDATA.EMPLOYEE
   WHERE WORKDEPT = 'D11'
```

WHERE절이 표에서 삭제하려는 행을 SQL에게 알려줍니다. SQL은 기본 표에서 탐색 조건을 만족시키는 모든 행을 삭제합니다. 보기에서 행을 삭제하면 기본 표의 행이 삭제됩니다. WHERE절은 생략할 수 있으나 WHERE이 없는 DELETE문은 표나 보기에서 모든 행을 삭제하기 때문에 하나의 WHERE절을 포함시키는 것이 좋습니다. 표 내용과 더불어 표 정의를 삭제하려면, DROP문을 발행하십시오. DROP문에 대한 자세한 정보는 *SQL 참조서*의 DROP문 주제를 참조하십시오.

DELETE문 수행시 SQL이 오류를 발견하면 자료 삭제를 중단하고 음의 SQLCODE를 리턴시킵니다. COMMIT(*ALL), COMMIT(*CS), COMMIT(*CHG) 또는 COMMIT(*RR)를 지정하면 표 내의 행이 삭제되지 않습니다(이 명령문에 의해 이미 삭제된 행이 있는 경우, 이전 값으로 복원됩니다). COMMIT(*NONE)이 지정되는 경우, 이미 삭제된 행은 이전 값으로 *복원되지 않습니다*.

SQL이 탐색 조건을 만족시키는 행을 찾지 못하면 +100인 SQLCODE가 리턴됩니다.

주: DELETE문에는 두 개 이상의 삭제된 행이 있을 수도 있습니다. 삭제된 행의 수는 SQLERRD(3)에 반영됩니다.

DELETE문에 대한 자세한 정보는 *SQL 참조서*의 DELETE문 주제를 참조하십시오.

제 7 장 부속 조회 사용

데이터를 선택하는 또 다른 방법으로 탐색 조건에서 부속 조회를 사용할 수 있습니다. 부속 조회는 표현식, 선택 목록, ORDER BY 및 GROUP BY 절로 사용될 수 있습니다. 자세한 내용은 다음의 절을 참조하십시오.

- 『SELECT 명령문의 부속 조회』
- 121 페이지의 『부속 조회 사용에 관한 주의사항』
- 122 페이지의 『상관 부속 조회』
- 126 페이지의 『UPDATE문의 상관 부속 조회 사용』
- 127 페이지의 『DELETE문에서 상관 부속 조회 사용』

SELECT 명령문의 부속 조회

단순 WHERE 및 HAVING절에서, 리터럴 값, 열 이름, 표현식 또는 레지스터를 사용하여 탐색 조건을 지정할 수 있습니다. 해당 탐색 조건에서, 특정 값에 대해 탐색 중이라는 것을 알 수 있습니다. 그러나, 때때로 표에서 다른 자료를 검색할 때까지 해당 값을 제공할 수 없을 때가 있습니다. 예를 들어 특정 프로젝트에서 작업중인 모든 사원의 사원 번호, 이름 및 작업 코드의 리스트를 원한다고 가정하고 프로젝트 번호를 MA2100이라고 지정하십시오. 명령문의 첫 번째 부분을 쉽게 작성할 수 있습니다.

```
SELECT EMPNO, LASTNAME, JOB
   FROM CORPDATA.EMPLOYEE
WHERE EMPNO ...
```

그러나 CORPDATA.EMPLOYEE 표에는 프로젝트 번호 자료가 들어 있지 않으므로 더 이상 계속 진행할 수 없습니다. CORPDATA.EMP_ACT 표에 대해 다른 SELECT 문을 발행하지 않고는 프로젝트 MA2100에 대해 작업하는 사원이 누구인지 알 수 없습니다.

SQL의 경우, 이러한 문제점 해결을 위해 하나의 SELECT문을 다른 SELECT문 내에 내포시킬 수 있습니다. 내부 SELECT문은 **부속 조회**라고 합니다. 부속 조회 주위의 SELECT문은 **외부 레벨 SELECT**라고 합니다. 부속 조회를 사용하면, 하나의 SQL문만을 발행하여 프로젝트 MA2100에 대해 작업하는 직원의 사원 번호, 이름 및 작업 코드를 검색할 수 있습니다.

```
SELECT EMPNO, LASTNAME, JOB
   FROM CORPDATA.EMPLOYEE
WHERE EMPNO IN
   (SELECT EMPNO
      FROM CORPDATA.EMPPROJACT
      WHERE PROJNO = 'MA2100')
```

이 SQL문의 결과를 보다 쉽게 이해하기 위해 SQL이 다음 프로세스로 진행된다고 가정합니다.

단계 1: SQL이 부속 조회를 평가하여 EMPNO 값 리스트를 얻습니다.

```
(SELECT EMPNO
          FROM CORPDATA.EMPPROJACT
     WHERE PROJNO= 'MA2100')
```

임시적인 결과 표를 작성합니다.

CORPDATA.EMPPROJACT로부터의 EMPNO
000010
000110

단계 2: 임시적인 결과 표는 외부 레벨 SELECT의 탐색 조건에서 리스트의 역할을 담당합니다. 본래, 이것은 실행하는 명령문입니다.

```
SELECT EMPNO, LASTNAME, JOB
        FROM CORPDATA.EMPLOYEE
     WHERE EMPNO IN
           ('000010', '000110')
```

최종 결과 표는 다음과 같습니다.

EMPNO	LASTNAME	JOB
000010	HAAS	PRES
000110	LUCCHESSI	SALESREP

자세한 내용은 다음의 절을 참조하십시오.

- 『상관(Correlation)』
- 119 페이지의 『부속 조회 및 탐색 조건』
- 119 페이지의 『부속 조회 사용 방법』
- 121 페이지의 『부속 조회 사용에 관한 주의사항』
- 122 페이지의 『상관 부속 조회』
- 126 페이지의 『UPDATE문의 상관 부속 조회 사용』
- 127 페이지의 『DELETE문에서 상관 부속 조회 사용』

상관(Correlation)

부속 조회의 목적은 행(WHERE절) 또는 행 그룹(HAVING절)에 대한 술부를 평가하는 데 필요한 정보를 제공하는 것입니다. 이는 부속 조회를 생성하는 결과 표를 통해 수행됩니다. 개념적으로, 새로운 행 또는 행 그룹이 처리될 때마다 부속 조회가 평가됩니다. 사실 부속 조회가 모든 행이나 그룹에 대해 동일하면 한번만 평가됩니다. 이와 같은 부속 조회를 **비상관**이라고 합니다.

일부 부속 조회는 행에서 행으로 또는 그룹에서 그룹으로 다른 값을 리턴합니다. 이를 허용하는 메카니즘을 **상관**이라고 하며 부속 조회가 **상관되었다고** 합니다. 상관 부속 조회에 대한 자세한 정보는 122 페이지의 『상관 부속 조회』에서 찾을 수 있습니다.

부속 조회 및 탐색 조건

부속 조회는 탐색 조건의 일부가 될 수 있습니다. 탐색 조건은 *피연산자 연산자 피연산자* 양식입니다. 피연산자 중 하나는 부속 조회가 될 수 있습니다. 다음 예에서, 첫 번째 **피연산자**는 EMPNO이고 **피연산자**는 IN입니다. 탐색 조건은 WHERE 또는 HAVING절의 일부가 될 수 있습니다. 절에는 부속 조회가 들어 있는 두 개 이상의 탐색 조건이 포함될 수 있습니다. 다른 탐색 조건과 같이 부속 조회가 들어 있는 탐색 조건은 괄호로 묶여지고 키워드 NOT이 앞에 올 수 있으며 키워드 AND와 OR을 통해 다른 탐색 조건에 연결될 수 있습니다. 예를 들어, 조회의 WHERE절은 다음과 같을 수 있습니다.

```
WHERE  (subquery1) = X AND (Y > SOME (subquery2) OR Z = 100)
```

부속 조회는 다른 부속 조회의 탐색 조건에도 표시될 수 있습니다. 그러한 부속 조회는 몇몇 내포 레벨에 **내**포됩니다. 예를 들어 외부 레벨 SELECT 내의 부속 조회 내의 부속 조회는 내포 레벨 2에 내포됩니다. SQL은 네스팅 레벨을 32로 낮추도록 합니다.

부속 조회 사용 방법

WHERE 또는 HAVING절 중 하나에 부속 조회를 포함시키는 여러 가지 방법이 있습니다.

- 『기본 비교』
- 120 페이지의 『수량 비교(ALL, ANY 및 SOME)』
- 120 페이지의 『IN 키워드』
- 121 페이지의 『EXISTS 키워드』

기본 비교

비교 연산자 중 한 연산자 앞뒤에 부속 조회를 사용할 수 있습니다. 부속 조회는 적어도 하나의 값을 리턴시킬 수 있습니다. 그 값은 열 함수 또는 연산식의 결과일 수 있습니다. 그런 다음, SQL이 부속 조회에서 발생되는 값을 비교 연산자의 외부의 값과 비교합니다. 예를 들어 회사 전체에서 교육 수준이 평균보다 높은 사원의 번호, 이름 및 급여를 찾으려 한다고 가정합니다.

```
    SELECT EMPNO, LASTNAME, SALARY
       FROM CORPDATA.EMPLOYEE
 WHERE EDLEVEL >
(SELECT AVG(EDLEVEL)
        FROM CORPDATA.EMPLOYEE)
```

SQL은 먼저 부속 조회를 평가한 후 SELECT문 WHERE절 내의 결과를 대체합니다. 이 예에서, 결과는 회사 전체의 평균 교육 수준입니다. 부속 조회는 단일 값을 리턴시킬 뿐만 아니라 값을 전혀 리턴시킬 수 없습니다. 이 경우 비교 결과를 알 수는 없습니다.

수량 비교(ALL, ANY 및 SOME)

비교 연산자 다음에는 키워드 ALL, ANY 또는 SOME이 오는 부속 조회를 사용할 수 있습니다. 이러한 방법으로 사용할 때 부속 조회는 널(null)을 포함해 0, 1 또는 여러 값을 리턴시킬 수 있습니다. 다음과 같은 방법에서는 ALL, ANY 및 SOME을 사용할 수 있습니다.

- 제공한 값이 부속 조회가 리턴하는 **모든** 값에 지정된 방법으로 비교되도록 하려면 ALL을 사용하십시오. 예를 들어 ALL과 함께 보다 큰(>) 비교 연산자를 사용한다고 가정합니다.

  ```
  ... WHERE expression > ALL (subquery)
  ```

  이 WHERE절을 만족시키려면 표현식의 값은 부속 조회에 의해 리턴된 모든 값(즉, 최대값보다 큰값)보다 커야 합니다. 부속 조회가 빈 세트를 리턴하면(즉, 값이 선택되지 않으면) 조건이 만족됩니다.
- 사용자가 제공하는 값이 지시에 따라 부속 조회가 리턴시키는 값 중 최소한 하나에 비교되어야 함을 표시하려면 ANY 또는 SOME을 사용하십시오. 예를 들어, **ANY**와 함께 보다 큰(>) 비교 연산자를 사용한다고 가정해 보십시오.

  ```
  ... WHERE expression > ANY (subquery)
  ```

  이 WHERE절을 만족시키려면 표현식의 값은 부속 조회에 의해 리턴된 값(즉, 최소값보다 큰값) 중 하나보다 최소한 커야 합니다. 부속 조회가 리턴시키는 값이 빈 세트이면 조건은 만족되지 않습니다.

주: 부속 조회가 하나 이상의 널을 리턴시킬 때 사용자가 공식 논리에 익숙하지 않는 한 예상하지 못한 결과를 나타낼 수 있습니다. 자세한 내용은 SQL 참조서의 한정술부에 대한 설명을 참조하십시오.

IN 키워드

IN을 사용하면 표현식의 값이 부속 조회에 의해 리턴된 값 사이에 있어야 함을 표시할 수 있습니다. IN 사용은 *=ANY* 또는 *=SOME*의 사용과 동등합니다. ANY와 SOME 사용에 대해서는 앞에 설명되어 있습니다. 값이 부속 조회에 의해 리턴된 값 사이에 없을 때 행을 선택하려면 NOT 키워드와 함께 IN 키워드도 사용할 수 있습니다. 예를 들어, 다음을 사용할 수 있습니다.

```
... WHERE WORKDEPT NOT IN (SELECT ...)
```

EXISTS 키워드

지금까지 표시된 부속 조회에서 SQL은 부속 조회를 평가하고 결과를 외부 레벨 SELECT의 WHERE절의 일부로 사용합니다. 반대로 키워드 EXISTS 사용시 SQL은 단순히 부속 조회가 하나 이상의 행을 리턴시키는지 여부를 검사합니다. 행을 리턴시키면 조건이 충족됩니다. 행을 리턴시키지 않은 경우, 조건이 만족되지 않습니다. 예를 들면 다음과 같습니다.

```
SELECT EMPNO,LASTNAME
   FROM CORPDATA.EMPLOYEE
WHERE EXISTS
  (SELECT *
     FROM CORPDATA.PROJECT
      WHERE PRSTDATE > '1982-01-01');
```

이 예에서, CORPDATA.PROJECT 표에 표시된 프로젝트에 1982년 1월 1일 이후로 추정된 시작일이 있으면 탐색 조건이 참입니다. 결과가 외부 레벨 SELECT에 대해 검사된 모든 행에 대해 항상 동일하기 때문에 이 예제는 EXISTS의 전체 성능을 표시하지 않습니다. 그 결과 모든 행이 결과에 표시되거나 아무것도 표시되지 않습니다. 보다 확장된 예제에서는 부속 조회 자체가 상관되어 있으며 행에서 행으로 변경됩니다. 상관 부속 조회에 대한 자세한 내용은 122 페이지의 『상관 부속 조회』를 참조하십시오.

예에 표시된 대로 EXISTS절의 부속 조회의 선택 리스트에 열 이름을 지정할 필요는 없습니다. 대신, SELECT *를 코딩해야 합니다.

또한 NOT 키워드와 함께 EXISTS 키워드를 사용하여 사용자가 지정한 자료나 조건이 존재하지 않는 경우 행을 선택할 수 있습니다. 다음을 사용할 수 있습니다.

```
... WHERE NOT EXISTS (SELECT ...)
```

부속 조회 사용에 관한 주의사항

1. SELECT문 내포시 각 추가 부속 조회에 대해 성능상의 문제점이 있다 할지라도 요구사항(1-31 부속 조회)을 만족시키기 위해 필요한 만큼 충분히 사용할 수 있습니다.
2. 외부 명령문이 SELECT문(모든 내포 레벨)인 경우:
   - 부속 조회는 외부 명령문과 동일한 표나 보기 또는 다른 표나 보기를 기초로 할 수 있습니다.
   - 외부 레벨 SELECT가 DECLARE CURSOR, CREATE TABLE, CREATE VIEW 또는 INSERT문의 일부일 경우에도 외부 레벨 SELECT문의 WHERE 절에서 부속 조회를 사용할 수 있습니다.
   - SELECT문의 HAVING절에서 부속 조회를 사용할 수 있습니다. 이 경우 SQL은 부속 조회를 평가하여 각 그룹을 규정하는데 사용합니다.

3. 명령문이 UPDATE나 DELETE문인 경우, UPDATE 또는 DELETE문의 WHERE 절에서 부속 조회를 사용할 수 있습니다. 또한 UPDATE문의 SET절에서 부속 조회를 사용할 수도 있습니다.
4. 부속 조회가 UPDATE문의 SET절에 사용되는 경우, 부속 선택의 결과 표에는 갱신되는 열의 해당 리스트와 같은 값들이 있어야 합니다. 이와 다른 경우, 반드시 부속 조회의 결과 표는 부속 조회가 EXISTS 키워드와 함께 사용중인 경우를 제외하고는 단일 열로 구성되어야 합니다. 키워드 ALL을 사용하는 술부의 경우, ANY, SOME 또는 EXISTS, 행 수는 0에서 많은 수로 변경할 수 있는 부속 조회로부터 리턴됩니다. 기타 모든 부속 조회의 경우, 행 수는 0 또는 1이어야 합니다.
5. 부속 조회에는 ORDER BY, UNION, UNION ALL, FOR READ ONLY, FETCH FIRST *n* ROWS, UPDATE 또는 OPTIMIZE절이 포함됩니다.

상관 부속 조회

이전에 설명한 부속 조회에서 SQL은 부속 조회를 한 번 평가하고 탐색 조건의 부속 조회 결과를 대체하고 탐색 조건의 값에 따라 외부 레벨 SELECT를 평가합니다. 또한 외부 레벨 SELECT 내의 새로운 각 행(WHERE절) 또는 그룹 행(HAVING절) 검사 시 SQL이 재평가해야 할 부속 조회를 작성할 수도 있습니다. 이것을 **상관 부속 조회**라고 합니다.

상관명과 상관 참조

상관 참조는 부속 조회 내의 탐색 조건에 표시됩니다. 참조의 형식은 항상 X.C입니다. 여기서 X는 상관명이고 C는 X가 표시되는 표 내의 열 이름입니다.

FROM절에 표시되는 표에 대해 상관명을 정의할 수 있습니다. 상관명은 조회의 표에 대한 고유한 이름을 제공합니다. 같은 표 이름이 조회 및 중첩 부속 선택 내에서 여러 번 사용될 수도 있습니다. 각 표 참조에 대하여 서로 다른 상관명을 지정하면 컬럼이 참조하는 표를 고유하게 지정할 수 있습니다.

상관명은 조회의 FROM절에 정의됩니다. 이 조회는 외부 레벨 SELECT나 참조를 갖는 조회를 포함하는 부속 조회 중 하나가 될 수 있습니다. 예를 들어 조회에 부속 조회 A, B 및 C가 포함되고 A는 B를 B는 C를 포함한다고 가정해 보십시오. 그러면 C에 사용되는 상관명은 B, A 또는 외부 레벨 SELECT에 정의될 수 있습니다. 상관명을 정의하려면 표 이름 다음에 상관명을 포함시키면 됩니다. 표 이름과 상관명 사이에 한 개 이상의 공백을 두고 그 뒤에 다른 표 이름이 오면 상관명 다음에 쉼표를 사용하십시오. 다음 FROM절은 표 TABLEA 및 TABLEB에 대해 상관명 TA 및 TB를 정의하고 표 TABLEC에 대해서는 상관명을 정의하지 않습니다.

```
FROM TABLEA TA, TABLEC, TABLEB TB
```

모든 수의 상관 참조가 부속 조회에 표시될 수 있습니다. 예를 들어, 탐색 조건에 하나의 상관명이 외부 레벨 SELECT에 정의될 수 있는 반면 다른 상관명은 포함된 부속 조회에 정의될 수 있습니다.

부속 조회를 실행하기 전에 참조된 열의 값은 항상 상관 참조로 대체됩니다.

예: WHERE절의 상관 부속 조회

교육 수준이 각 부서 내의 평균보다 높은 모든 사원의 리스트를 원한다고 가정합니다. 이 정보를 얻으려면 SQL이 CORPDATA.EMPLOYEE 표를 탐색해야 합니다. 표에 있는 각 사원의 경우, SQL은 사원 부서의 평균 교육 등급과 그 사원의 교육 등급을 비교해야 합니다. 부속 조회에서 SQL은 현재 행의 부서 번호에 대해 평균 교육 등급을 계산하도록 지정합니다. 예를 들면 다음과 같습니다.

```
SELECT EMPNO, LASTNAME, WORKDEPT, EDLEVEL
    FROM CORPDATA.EMPLOYEE X
 WHERE EDLEVEL >
(SELECT AVG(EDLEVEL)
      FROM CORPDATA.EMPLOYEE
       WHERE WORKDEPT = X.WORKDEPT)
```

상관 부속 조회는 하나 이상의 상관 참조가 표시되는 경우를 제외하고는 무상관 부속 조회처럼 표시됩니다. 예에서, 단일 상관 참조는 부속 선택의 FROM절 가운데 X.WORKDEPT의 어커런스입니다. 여기서 규정자 *X*는 외부 SELECT문의 FROM절에 정의됩니다. 이 절에서 *X*는 표 CORPDATA.EMPLOYEE의 상관명으로 소개됩니다.

이제 CORPDATA.EMPLOYEE의 주어진 행에 대해 부속 조회를 실행할 때 발생되는 상황에 대해 생각해 봅시다. 부속 조회가 실행되기 전에 X.WORKDEPT 어커런스는 그 행에 대한 WORKDEPT의 값으로 대체됩니다. 예를 들어 그 행이 CHRISTINE I HAAS에 대한 것이라고 가정합니다. 그녀의 작업 부서는 A00이며 이는 이 행에 대한 WORKDEPT의 값입니다. 이 행에 대해 실행되는 부속 조회는 다음과 같습니다.

```
(SELECT AVG(EDLEVEL)
      FROM CORPDATA.EMPLOYEE
   WHERE WORKDEPT = 'A00')
```

이렇게 고려된 행에 대해 부속 조회는 Christine이 속한 부서의 평균 교육 등급을 생성합니다. 그런 다음, 이것이 외부 명령문에서 Christine 자신의 교육 수준과 비교됩니다. WORKDEPT가 다른 값을 갖는 일부 행의 경우, 그 값은 A00 위치 내의 부속 조회에 표시됩니다. 예를 들어 MICHAEL L THOMPSON에 대한 행의 경우, 이 값은 B01이 되고 그의 행에 대한 부속 조회가 부서 B01에 대해 평균 교육 수준을 나타냅니다.

조회에 의해 생성되는 결과 표의 값은 다음과 같습니다.

표 9. 이전 조회에 대한 결과 세트

EMPNO	LASTNAME	WORKDEPT	EDLEVEL
000010	HAAS	A00	18
000030	KWAN	C01	20
000070	PULASKI	D21	16
000090	HENDERSON	E11	16
000110	LUCCHESSI	A00	19
000160	PIANKA	D11	17
000180	SCOUTTEN	D11	17
000210	JONES	D11	17
000220	LUTZ	D11	18
000240	MARINO	D21	17
000260	JOHNSON	D21	16
000280	SCHNEIDER	E11	17
000320	MEHTA	E21	16
000340	GOUNOT	E21	16
200010	HEMMINGER	A00	18
200220	JOHN	D11	18
200240	MONTEVERDE	D21	17
200280	SCHWARTZ	E11	17
200340	ALONZO	E21	16

예: HAVING절의 상관 부속 조회

평균 급여가 그들 영역(WORKDEPT가 같은 영역에 속하는 동일한 문자로 시작되는 모든 부서)의 평균 급여보다 높은 모든 부서의 리스트를 원한다고 가정합니다. 이 정보를 얻으려면 SQL이 CORPDATA.EMPLOYEE 표를 탐색해야 합니다. 표에 있는 각 부서의 경우, SQL이 부서의 평균 급여를 그 분야의 평균 급여와 비교합니다. 부속 조회에서 SQL은 현재 그룹에 있어서 부서에 대한 평균 급여를 계산합니다. 예를 들면 다음과 같습니다.

```
SELECT WORKDEPT, DECIMAL(AVG(SALARY),8,2)
 FROM CORPDATA.EMPLOYEE X
  GROUP BY WORKDEPT      HAVING AVG(SALARY) >
    (SELECT AVG(SALARY)
   FROM CORPDATA.EMPLOYEE
       WHERE SUBSTR(X.WORKDEPT,1,1) = SUBSTR(WORKDEPT,1,1))
```

CORPDATA.EMPLOYEE의 주어진 부서에 대해 부속 조회 실행시 발생하는 일을 생각해 봅시다. 부속 조회가 실행되기 전에 X.WORKDEPT의 어커런스는 그 그룹에 대한 WORKDEPT 열의 값으로 대체됩니다. 예를 들어 선택된 첫 번째 그룹의 WORKDEPT 값이 A00이라고 가정합니다. 이 그룹에 대해 실행되는 부속 조회는 다음과 같습니다.

```
  (SELECT AVG(SALARY)
        FROM CORPDATA.EMPLOYEE
     WHERE SUBSTR('A00',1,1) = SUBSTR(WORKDEPT,1,1))
```

이렇게 고려된 그룹에 대해 부속 조회가 그 영역의 평균 급여를 생성합니다. 그런 다음, 이 값은 외부 명령문에서 부서 'A00'에 대한 평균 급여와 비교됩니다. WORKDEPT가 'B01'인 기타 다른 그룹의 경우, 부속 조회는 부서 B01이 속하는 영역에 대한 평균 급여를 표시합니다.

조회에 의해 생성되는 결과 표의 값은 다음과 같습니다.

WORKDEPT	AVG SALARY
D21	25668.57
E01	40175.00
E21	24086.66

예: 선택 리스트의 상관 부속 조회

모든 부서(부서명, 번호 및 관리자의 이름 포함)의 리스트를 원한다고 가정합니다. 부서명 및 번호는 CORPDATA.DEPARTMENT 표에 있습니다. 그러나 DEPARTMENT에는 관리자 번호만있고 관리자 이름은 없습니다. 각 부서별 관리자 이름을 찾으려면 DEPARTMENT 표의 관리자 번호와 일치하는 EMPLOYEE 표의 직원 번호를 찾아 일치하는 행의 이름을 리턴해야 합니다. 현재 관리자가 지정된 부서만 리턴되어야 합니다.다음을 실행하십시오.

```
SELECT DEPTNO, DEPTNAME,
    (SELECT FIRSTNME CONCAT ' ' CONCAT
         MIDINIT CONCAT ' ' CONCAT LASTNAME
       FROM EMPLOYEE X
       WHERE X.EMPNO = Y.MGRNO) AS MANAGER_NAME
   FROM DEPARTMENT Y
   WHERE MGRNO IS NOT NULL
```

DEPTNO 및 DEPTNAME에 대하여 리턴된 각 행의 경우 시스템은 EMPNO = MGRNO인 곳을 찾아 관리자 이름을 리턴합니다.조회에 의해 생성되는 결과 표의 값은 다음과 같습니다.

표 10.

DEPTNO	DEPTNAME	MANAGER_NAME
A00	SPIFFY COMPUTER SERVICE DIV.	CHRISTINE I HAAS
B01	PLANNING	MICHAEL L THOMPSON
C01	INFORMATION CENTER	SALLY A KWAN
D11	MANUFACTURING SYSTEMS	IRVING F STERN
D21	ADMINISTRATION SYSTEMS	EVA D PULASKI
E01	SUPPORT SERVICES	JOHN B GEYER
E11	OPERATIONS	EILEEN W HENDERSON

표 10. (계속)

DEPTNO	DEPTNAME	MANAGER_NAME
E21	SOFTWARE SUPPORT	THEODORE Q SPENSER

UPDATE문의 상관 부속 조회 사용

UPDATE문에서 상관 부속 조회 사용시 상관명은 갱신하려는 행을 참조합니다. 예를 들어 프로젝트의 모든 활동이 1983년 9월 이전에 완료되어야 하는 경우, 부서는 그 프로젝트가 우선순위 프로젝트가 되도록 고려합니다. CORPDATA.PROJECT 표 내의 프로젝트를 평가하려면 아래의 SQL문을 사용할 수 있고 각 우선 순위 프로젝트에 대해 PRIORITY열(이 목적을 위해 CORPDATA.PROJECT에 추가한 열)에 1(우선순위를 표시하기 위한 플래그)을 작성할 수 있습니다.

```
UPDATE CORPDATA.PROJECT X
  SET PRIORITY = 1
  WHERE '1983-09-01' >
     (SELECT MAX(EMENDATE)
         FROM CORPDATA.EMPPROJACT
         WHERE PROJNO = X.PROJNO)
```

SQL이 CORPDATA.EMPPROJACT 표에서 각 행을 검사할 때, 프로젝트의 모든 활동에 대한 최대 활동 종료 날짜(EMENDATE)를 결정합니다(CORPDATA.PROJECT 표에서). 프로젝트에 연관된 각 활동의 종료일이 1983년 9월 이전일 경우, 표 CORPDATA.PROJECT에 있는 현재 행이 규정되고 갱신됩니다.

주문 수량에서 변경된 내용을 가지고 마스터 주문 표를 갱신하십시오. 주문 표의 수량이 설정되어 있지 않으면(널 값) 마스터 주문 표의 값을 그대로 보존하십시오.

```
UPDATE MASTER_ORDERS X
   SET QTY=(SELECT COALESCE (Y.QTY, X.QTY)
               FROM ORDERS Y
               WHERE X.ORDER_NUM = Y.ORDER_NUM)
   WHERE X.ORDER_NUM IN (SELECT ORDER_NUM
               FROM ORDERS)
```

이 예에서 ORDERS 표에 대응하는 행이 있는지 MASTER_ORDERS 표의 각 행을 점검합니다. ORDERS 표에 일치하는 행이 있으면 COALESCE 기능을 사용하여 QTY 열에 대한 값을 리턴합니다. ORDERS 표의 QTY에 널이 아닌 값이 들어 있는 경우 이 값은 MASTER_ORDERS 표의 QTY 열을 갱신하는 데 사용됩니다. ORDERS 표의 QTY 값이 NULL인 경우 MASTER_ORDERS QTY 열은 자신의 값을 사용하여 갱신됩니다.

DELETE문에서 상관 부속 조회 사용

DELETE문에서 상관 부속 조회 사용시 상관명은 삭제한 행을 표시합니다. SQL은 행 삭제 여부를 결정하기 위해 DELETE문에서 명명된 표 내의 각 행에 대해 상관 부속 조회를 한번 평가합니다.

CORPDATA.PROJECT 표 내의 행이 삭제되었다고 가정합니다. CORPDATA.EMPPROJACT 표에서 삭제된 프로젝트에 관련된 행도 삭제해야 합니다. 이를 수행하기 위해 다음을 사용할 수 있습니다.

```
DELETE FROM CORPDATA.EMPPROJACT X
  WHERE NOT EXISTS
     (SELECT *
        FROM CORPDATA.PROJECT
         WHERE PROJNO = X.PROJNO)
```

SQL은 CORPDATA.EMP_ACT 표 내의 각 행에 대해 동일한 프로젝트 번호로 된 행이 CORPDATA.PROJECT 표에 존재하는지의 여부를 결정합니다. 그렇지 않으면, CORPDATA.EMP_ACT 행은 삭제됩니다.

제 8 장 SQL 정렬 순서

정렬 순서는 문자 세트 내 문자가 비교 또는 순서화될 때 서로 관련되는 방법을 정의합니다. 정렬 순서에 대한 완벽한 설명은 SQL 참조서의 정렬 순서 절을 참조하십시오.

정렬 순서는 SQL문에서 수행되는 모든 문자와 UCS-2 그래픽 비교에 사용됩니다. 1바이트 및 2바이트 문자 자료에는 정렬 순서표가 있습니다. 각각의 1바이트 정렬 순서표는 연관된 2바이트 정렬 순서표를 가지며 그 역 또한 같습니다. 두 표간의 비교는 조회 수행이 필요할 경우 이루어집니다. 추가적으로 CREATE INDEX 명령문에는 색인에서 참조된 문자열을 적용하는 정렬 순서(명령문 수행시 효력이 발생함)가 있습니다.

자세한 내용은 다음의 주제를 참조하십시오.

- 『ORDER BY 및 행 선택을 사용하는 정렬 순서』
- 130 페이지의 『정렬 순서 및 ORDER BY』
- 131 페이지의 『행 선택』
- 132 페이지의 『정렬 순서 및 보기』
- 133 페이지의 『정렬 순서와 CREATE INDEX문』
- 133 페이지의 『정렬 순서 및 제한조건』

ORDER BY 및 행 선택을 사용하는 정렬 순서

정렬 순서 사용법을 참조하려면, 다음 표의 STAFF 표에 대한 이 절에서 예를 실행하십시오. JOB 열의 값이 대소문자로 이루어져 있음에 주의하십시오. 한 예로 값 'Mgr', 'MGR' 및 'mgr'를 볼 수 있습니다.

표 11. STAFF 표

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
10	Sanders	20	Mgr	7	18357.50	0
20	Pernal	20	Sales	8	18171.25	612.45
30	Merenghi	38	MGR	5	17506.75	0
40	OBrien	38	Sales	6	18006.00	846.55
50	Hanes	15	Mgr	10	20659.80	0
60	Quigley	38	SALES	0	16808.30	650.25
70	Rothman	15	Sales	7	16502.83	1152.00
80	James	20	Clerk	0	13504.60	128.20
90	Koonitz	42	sales	6	18001.75	1386.70
100	Plotz	42	mgr	6	18352.80	0

예에서는 다음을 사용한 각 명령문의 결과가 표시됩니다.

- *HEX 정렬 순서
- 언어 식별자 ENU를 사용하는 공유 가중치 정렬 순서
- 언어 식별자 ENU를 사용하는 고유 가중치 정렬 순서

주: ENU는 SRTSEQ(*LANGIDUNQ) 또는 SRTSEQ(*LANGIDSHR)와 LANGID(ENU)를 CRTSQLxxx, STRSQL 또는 RUNSQLSTM 명령에 지정하거나 SET OPTION문을 사용함으로써 언어 ID로서 선택됩니다.

정렬 순서 및 ORDER BY

다음 SQL문은 JOB 열의 값을 사용하여 결과 표가 정렬되도록 합니다.

```
SELECT * FROM STAFF ORDER BY JOB
```

표 12에서는 *HEX 정렬 순서를 사용한 결과 표를 보여줍니다. 정렬된 행은 JOB 열에 있는 EBCDIC 값을 근거로 합니다. 이 경우에 모든 영문 소문자는 대문자 앞에 정렬됩니다.

*표 12. *HEX 정렬 순서를 사용한 "SELECT * FROM STAFF ORDER BY JOB"*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
100	Plotz	42	mgr	6	18352.80	0
90	Koonitz	42	sales	6	18001.75	1386.70
80	James	20	Clerk	0	13504.60	128.20
10	Sanders	20	Mgr	7	18357.50	0
50	Hanes	15	Mgr	10	20659.80	0
30	Merenghi	38	MGR	5	17506.75	0
20	Pernal	20	Sales	8	18171.25	612.45
40	OBrien	38	Sales	6	18006.00	846.55
70	Rothman	15	Sales	7	16502.83	1152.00
60	Quigley	38	SALES	0	16808.30	650.25

표 13에서는 고유 가중치 정렬 순서로 정렬되는 방법을 보여줍니다. 정렬 순서가 JOB 열에 있는 값에 적용된 후 행이 정렬됩니다. 정렬 후 영문 소문자가 동일한 대문자 앞에 있는지, 값 'mgr', 'Mgr' 및 'MGR'이 서로 인접하고 있는지에 유의하십시오.

*표 13. ENU 언어 식별자에 대한 고유 가중치 정렬 순서를 사용한 "SELECT * FROM STAFF ORDER BY JOB"*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
80	James	20	Clerk	0	13504.60	128.20
100	Plotz	42	mgr	6	18352.80	0
10	Sanders	20	Mgr	7	18357.50	0
50	Hanes	15	Mgr	10	20659.80	0
30	Merenghi	38	MGR	5	17506.75	0
90	Koonitz	42	sales	6	18001.75	1386.70

*표 13. ENU 언어 식별자에 대한 고유 가중치 정렬 순서를 사용한 "SELECT * FROM STAFF ORDER BY JOB" (계속)*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
20	Pernal	20	Sales	8	18171.25	612.45
40	OBrien	38	Sales	6	18006.00	846.55
70	Rothman	15	Sales	7	16502.83	1152.00
60	Quigley	38	SALES	0	16808.30	650.25

표 14에서는 공유 가중치 정렬 순서로 정렬되는 방법을 보여줍니다. 정렬 순서가 JOB 열에 있는 값에 적용된 후 행이 정렬됩니다. 정렬 비교의 경우, 각 영문 소문자는 해당 대문자와 같이 처리됩니다. 표 14에서 모든 값 'MGR', 'mgr' 및 'Mgr'이 함께 섞여 있음에 주의하십시오.

*표 14. ENU 언어 ID에 대한 공유 가중치 정렬 순서를 사용한 "SELECT * FROM STAFF ORDER BY JOB"*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
80	James	20	Clerk	0	13504.60	128.20
10	Sanders	20	Mgr	7	18357.50	0
30	Merenghi	38	MGR	5	17506.75	0
50	Hanes	15	Mgr	10	20659.80	0
100	Plotz	42	mgr	6	18352.80	0
20	Pernal	20	Sales	8	18171.25	612.45
40	OBrien	38	Sales	6	18006.00	846.55
60	Quigley	38	SALES	0	16808.30	650.25
70	Rothman	15	Sales	7	16502.83	1152.00
90	Koonitz	42	sales	6	18001.75	1386.70

행 선택

다음 SQL문은 JOB 열에서 값 'MGR'이 있는 행을 선택합니다.

```
SELECT * FROM STAFF WHERE JOB='MGR'
```

표 15에서는 *HEX 정렬 순서로 행 선택이 수행되는 방법을 보여줍니다. 표 15에서 'JOB' 열에 대한 행 선택 기준에 일치하는 행이 SELECT문에서 지정된 것처럼 선택됩니다. 대문자 'MGR'만 선택됩니다.

*표 15. *HEX 정렬 순서를 사용한 "SELECT * FROM STAFF/WHERE JOB='MGR'"*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
30	Merenghi	38	MGR	5	17506.75	0

132 페이지의 표 16에서는 고유 가중치 정렬 순서로 행 선택이 수행되는 방법을 보여줍니다. 132 페이지의 표 16에서 소문자 및 대문자 영문자는 하나로 취급됩니다. 소문자 'mgf'는 대문자 'MGR'와 동일하게 취급되지 않습니다. 그러므로 소문자 'mgr'은 선

택되지 않습니다.

*표 16. ENU 언어 ID에 대한 고유 가중치 정렬 순서를 사용한 "SELECT * FROM STAFF WHERE JOB = 'MGR'"*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
30	Merenghi	38	MGR	5	17506.75	0

표 17에서는 공유 가중치 정렬 순서로 행 선택이 수행되는 방법을 보여줍니다. 표 17에서 열 'JOB'에 대한 행 선택 기준에 일치되는 행은 대문자 영문자가 소문자 영문자와 같이 취급되어 선택됩니다. 표 17에서는 모든 값 'mgr', 'Mgr' 및 'MGR'이 선택됨에 유의하십시오.

*표 17. ENU 언어 ID에 대한 공유 가중치 정렬 순서를 사용한 "SELECT * FROM STAFF WHERE JOB = 'MGR'"*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
10	Sanders	20	Mgr	7	18357.50	0
30	Merenghi	38	MGR	5	17506.75	0
50	Hanes	15	Mgr	10	20659.80	0
100	Plotz	42	mgr	6	18352.80	0

정렬 순서 및 보기

보기는 CREATE VIEW문 수행시 효율적인 정렬 순서로 작성됩니다. FROM절에서 보기를 참조할 때 정렬 순서는 CREATE VIEW의 부속 선택에서 문자 비교를 위해 사용됩니다. 그 때 중간 결과 표는 보기 부속 선택으로부터 작성됩니다. 조회 수행시 유효한 정렬 순서는 조회에 지정된 모든 문자와 UCS-2 그래픽 비교(문자 또는 UCS-2 그래픽으로의 내재적 비교 포함)에 적용됩니다.

다음 SQL문 및 표에서는 보기 및 정렬 순서 방법을 보여줍니다. 다음 예제에 사용된 보기 V1은 SRTSEQ(*LANGIDSHR) 및 LANGID(ENU)의 공유 가중치 정렬 순서로 작성되었습니다. CREATE VIEW문은 다음과 같습니다.

```
CREATE VIEW V1 AS SELECT *  FROM STAFF
  WHERE JOB = 'MGR' AND ID < 100
```

표 18에서는 보기에서 나온 결과 표를 보여줍니다.

*표 18. "SELECT * FROM V1"*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
10	Sanders	20	Mgr	7	18357.50	0
30	Merenghi	38	MGR	5	17506.75	0
50	Hanes	15	Mgr	10	20659.80	0

보기 V1에 있는 조회가 132 페이지의 표 18의 결과 표에 나타납니다. 아래에 보이는 조회는 SRTSEQ(*LANGIDUNQ) 및 LANGID(ENU)의 정렬 순서로 수행됩니다.

*표 19. 언어 ID ENU에 대한 고유 가중치 정렬 순서를 사용한 "SELECT * FROM V1 WHERE JOB = 'MGR'"*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
30	Merenghi	38	MGR	5	17506.75	0

정렬 순서와 CREATE INDEX문

색인은 정렬 순서를 사용한 CREATE INDEX문 수행시 효율적으로 작성됩니다. 입력 항목이 색인에 추가될 때마다 정의된 표에 색인이 삽입됩니다. 색인 항목에는 문자 키와 UCS-2 그래픽 키 열에 대해 가중치가 포함됩니다. 시스템은 색인의 정렬 순서에 있는 키 값을 변환하여 가중치를 구합니다.

정렬 순서와 해당 색인을 사용하여 선택이 이루어지는 경우에는 문자 또는 UCS-2 그래픽 키를 비교에 앞서 변환시킬 필요가 없습니다. 따라서 조회의 성능이 향상됩니다. 효과적인 색인 및 정렬 순서 작성에 대한 자세한 정보는 *데이터베이스 성능 및 조회 최적화*의 큰 표에 빠르게 액세스하기 위한 색인 사용을 참조하십시오.

정렬 순서 및 제한조건

고유 제한조건은 색인과 함께 수행됩니다. 고유 제한조건이 추가되는 표가 정렬 순서로 정의될 경우, 색인은 이와 동일한 정렬 순서로 작성됩니다.

참조 제한조건을 정의할 경우, 상위 표와 종속 표 사이에 정렬 순서가 일치해야 합니다. 정렬 순서와 제한사항에 대한 자세한 내용은 iSeries Information Center에서 데이터베이스 프로그래밍 책의 참조 제한사항과의 자료 통합성 확인 주제를 참조하십시오.

체크 제한조건이 정의될 때 사용되는 정렬 순서는 시스템이 INSERT 또는 UPDATE 시 제한조건의 유효성 검사에 사용하는 것과 같은 정렬 순서입니다.

제 9 장 커서 사용

SQL이 SELECT문을 수행하면 결과 행은 결과 표를 구성합니다. 커서가 결과 표의 액세스 방식을 제공합니다. 이는 SQL 프로그램 내에서 결과 표의 위치를 유지보수하는 데 사용됩니다. SQL은 커서를 사용하여 결과 표 내의 행을 식별하고 사용자 프로그램에서 사용할 수 있도록 합니다. 사용자 프로그램은 각각 고유명을 가져야 하지만 몇 개의 커서를 가질 수 있습니다.

커서를 사용하는 것과 관련된 명령문에는 다음이 포함됩니다.

- 커서를 정의하고 명명하며 행이 삽입된 선택문과 함께 검색되도록 지정하는 DECLARE CURSOR문
- 프로그램 내에서 사용할 커서를 열고 닫기 위한 OPEN 및 CLOSE문. 모든 행을 검색하기 전에 커서를 열어야 합니다.
- 커서의 결과 표로부터 행을 검색하거나 다른 행에 커서를 위치시키기 위한 FETCH문
- 커서의 현재 행을 갱신하기 위한 UPDATE ... WHERE CURRENT OF문
- 커서의 현재 행을 삭제하기 위한 DELETE ... WHERE CURRENT OF문

이러한 명령문에 대한 완벽한 설명은 SQL 참조서를 참조하십시오.

커서에 대한 자세한 내용은 다음의 주제를 참조하십시오.

- 『커서의 유형』
- 137 페이지의 『커서 사용 예』
- 143 페이지의 『복수 행 FETCH문 사용』
- 149 페이지의 『작업 단위 및 열린 커서』

주: 코드 예에 관한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

커서의 유형

SQL은 연속 및 화면이동 커서를 제공합니다. 커서의 유형은 커서에 사용될 수 있는 위치지정 방법을 결정합니다. 자세한 내용은 다음을 참조하십시오.

- 『연속 커서』
- 136 페이지의 『화면이동 커서』

연속 커서

연속 커서는 SCROLL 키워드 없이 정의되는 커서입니다.

연속 커서의 경우, 결과 표의 각 행은 커서의 OPEN당 한 번만 페치될 수 있습니다. 커서가 열릴 때 커서는 결과 표의 첫 번째 행 앞에 위치됩니다. FETCH가 발행되면 커서는 결과 표 내의 다음 행으로 이동됩니다. 그러면 그 행이 현재 행이 됩니다. 호스트 변수가 지정되면(FETCH문에 INTO절로) SQL이 현재 행의 내용을 사용자 프로그램의 호스트 변수로 이동시킵니다.

이 순서는 자료의 끝(SQLCODE = 100)에 도달할 때까지 FETCH문이 발행될 때마다 반복됩니다. 자료의 끝에 도달하면 커서를 닫으십시오. 자료의 끝에 도달하면 사용자는 결과 표의 어떤 행에도 액세스할 수 없습니다. 직렬 커서를 다시 사용하려면 커서를 반드시 먼저 닫고 나서 OPEN문을 다시 발행해야 합니다. 직렬 커서를 사용하여 백업할 수 없습니다.

화면이동 커서

화면이동 커서의 경우, 결과 표의 행은 여러 번 페치될 수 있습니다. 커서는 FETCH문에서 지정된 위치 옵션에 따라 결과 표를 통해 이동됩니다. 커서가 열릴 때 커서는 결과 표의 첫 번째 행 앞에 위치지정됩니다. FETCH가 발행될 때 커서는 결과 표에서 위치 옵션에 의해 지정된 행으로 위치지정됩니다. 그러면 그 행이 현재 행이 됩니다. 호스트 변수가 지정되면(FETCH문에 INTO절로) SQL이 현재 행의 내용을 사용자 프로그램의 호스트 변수로 이동시킵니다. 호스트 변수는 BEFORE 및 AFTER 위치 옵션에 대해서는 지정할 수 없습니다.

이 순서는 FETCH문이 발행될 때마다 반복됩니다. 자료 끝이나 자료 시작 조건이 일어날 때에는 커서를 닫을 필요가 없습니다. 위치 옵션에 의해 프로그램은 표로부터 행 페칭을 계속할 수 있습니다.

다음의 화면이동 옵션이 FETCH문 발행시에 커서를 위치지정하는데 사용됩니다. 이러한 위치는 결과 표 내의 현재 커서 위치에 상대적입니다.

NEXT	커서를 다음 행에 위치시킴. 이는 위치가 지정되지 않을 경우의 디폴트입니다.
PRIOR	커서를 이전 행에 위치시킴
FIRST	커서를 첫 번째 행에 위치시킴
LAST	커서를 최종 행에 위치시킴
BEFORE	커서를 첫 번째 행 앞에 위치시킴
AFTER	커서를 최종 행 뒤에 위치시킴
CURRENT	커서 위치를 변경시키지 않음
RELATIVE n	커서의 현재 위치와의 관계에서 호스트 변수 또는 정수 *n*을 평가합니다. 예를 들어 *n*이 -1인 경우, 커서는 결과 표의 이전 행에 위치합니다. *n*이 +3인 경우, 커서는 현재 행의 3행 다음에 위치합니다.

화면이동이 가능한 커서 경우, 다음에 의해 표의 끝이 판별될 수 있습니다.

```
FETCH AFTER FROM C1
```

커서가 표의 끝에 오면 프로그램이 PRIOR이나 RELATIVE 화면이동 옵션을 사용하여 표의 끝에서 시작하는 자료의 위치를 지정하여 페치할 수 있습니다.

커서 사용 예

프로그램에서 부서 D11 사원들에 관한 자료를 체크한다고 가정합니다. 다음 예제에서는 연속 커서와 화면이동 커서의 정의 및 사용을 위해 프로그램에 포함시키려는 SQL 문을 보여줍니다. 커서를 사용하면 CORPDATA.EMPLOYEE 표에서 부서에 대한 정보를 확보할 수 있습니다.

연속 커서 예제에서 프로그램은 부서 D11에 속한 모든 구성원의 업무를 갱신하고 다른 부서에서 가져온 사원의 기록을 삭제하면서 표의 모든 행을 처리합니다.

표 20. 연속 커서의 예

연속 커서 **SQL**문	찾아볼 곳
EXEC SQL **DECLARE** THISEMP **CURSOR FOR** **SELECT** EMPNO, LASTNAME, WORKDEPT, JOB **FROM** CORPDATA.EMPLOYEE **FOR UPDATE OF** JOB END-EXEC.	139 페이지의『단계 1: 커서 정의』.
EXEC SQL **OPEN** THISEMP END-EXEC.	140 페이지의『단계 2: 커서 열기』.
EXEC SQL **WHENEVER NOT FOUND** **GO TO** CLOSE-THISEMP END-EXEC.	141 페이지의『단계 3: 자료의 끝에 도달했을 때 수행할 작업 지정』.
EXEC SQL **FETCH** THISEMP **INTO** :EMP-NUM, :NAME2, :DEPT, :JOB-CODE END-EXEC.	141 페이지의『단계 4: 커서를 사용하는 행 검색』.
... for all employees in department D11, update the JOB value: EXEC SQL **UPDATE** CORPDATA.EMPLOYEE **SET** JOB = :NEW-CODE **WHERE CURRENT OF** THISEMP END-EXEC. ... then print the row.	142 페이지의『단계 5a: 현재 행 갱신』.

표 20. 연속 커서의 예 (계속)

연속 커서 SQL문	찾아볼 곳
... for other employees, delete the row: EXEC SQL **DELETE FROM** CORPDATA.EMPLOYEE **WHERE CURRENT OF** THISEMP END-EXEC.	142 페이지의『단계 5b: 현재 행 삭제』.
다음 행을 페치하고 처리하기 위해 뒤로 분기	
CLOSE-THISEMP. EXEC SQL **CLOSE** THISEMP END-EXEC.	143 페이지의『단계 6: 커서 닫기』.

화면이동 커서 예제에서 프로그램은 부서 D11의 대표 샘플을 확보하기 위해 RELATIVE 위치 옵션을 사용할 수 있습니다.

표 21. 화면이동 커서 예제

화면이동 커서 SQL문	찾아볼 곳
EXEC SQL **DECLARE** THISEMP **DYNAMIC SCROLL CURSOR FOR** **SELECT** EMPNO, LASTNAME, SALARY **FROM** CORPDATA.EMPLOYEE **WHERE** WORKDEPT = 'D11' END-EXEC.	139 페이지의『단계 1: 커서 정의』.
EXEC SQL **OPEN** THISEMP END-EXEC.	140 페이지의『단계 2: 커서 열기』.
EXEC SQL **WHENEVER NOT FOUND** **GO TO** CLOSE-THISEMP END-EXEC.	141 페이지의『단계 3: 자료의 끝에 도달했을 때 수행할 작업 지정』.
...initialize program summation salary variable EXEC SQL **FETCH RELATIVE** 3 **FROM** THISEMP **INTO** :EMP-NUM, :NAME2, :JOB-CODE END-EXEC. ...add the current salary to program summation salary ...branch back to fetch and process the next row.	141 페이지의『단계 4: 커서를 사용하는 행 검색』.
...calculate the average salary	

표 21. 화면이동 커서 예제 (계속)

화면이동 커서 SQL문	찾아볼 곳
CLOSE-THISEMP. EXEC SQL **CLOSE** THISEMP END-EXEC.	143 페이지의『단계 6: 커서 닫기』.

단계 1: 커서 정의

결과 표가 커서를 이용해 액세스되도록 정의하려면 DECLARE CURSOR문을 사용하십시오.

DECLARE CURSOR문은 커서를 명명하고 SELECT문을 지정합니다. SELECT문은 결과 표를 개념적으로 구성하는 일련의 행을 정의합니다. 연속 커서의 경우, 명령문은 다음과 같습니다(FOR UPDATE OF절은 선택가능함).

```
EXEC SQL
 DECLARE cursor-name CURSOR FOR
 SELECT column-1, column-2 ,...
   FROM table-name , ...
   FOR UPDATE OF column-2 ,...
END-EXEC.
```

이동할 수 있는 커서의 경우, 명령문은 다음과 같습니다(WHERE절은 생략가능함).

```
EXEC SQL
 DECLARE cursor-name DYNAMIC SCROLL CURSOR FOR
 SELECT column-1, column-2 ,...
   FROM table-name ,...
   WHERE column-1 = expression ...
END-EXEC.
```

여기서 표시된 SELECT문은 비교적 간단합니다. 그러나 연속 커서와 화면이동 커서에 대한 DECALRE CURSOR문 내의 SELECT문에 몇 가지 다른 유형의 절을 작성할 수 있습니다.

식별된 표(FROM절에서 명명된 표)의 일부 행 또는 모든 행에서 열을 갱신하려면 FOR UPDATE OF절을 포함시키십시오. 이는 갱신하려는 각 열을 명명합니다. 나중에 갱신할 열 이름을 지정하지 않고 ORDER BY절 또는 FOR READ ONLY절을 지정하면 갱신하려 할 때 음의 SQLCODE가 리턴됩니다. 사용자가 FOR UPDATE OF절, FOR READ ONLY절 또는 ORDER BY절을 지정하고 않고 결과 표가 읽기 전용이 아닌 경우, 지정된 표의 모든 열을 갱신할 수 있습니다.

결과 표의 일부가 아니더라도 식별된 표의 열을 갱신할 수 있습니다. 이 경우 SELECT 문에서 열을 명명할 필요는 없습니다. 사용자가 갱신하려는 열값을 포함하고 있는 행을 커서가 검색(FETCH를 사용하여)할 때 사용자는 UPDATE ... WHERE CURRENT OF를 사용하여 그 행을 갱신할 수 있습니다.

예를 들어 결과 표의 각 행에 *EMPNO*, *LASTNAME* 및 *WORKDEPT* 열 (CORPDATA.EMPLOYEE 표)이 포함된다고 가정합니다. *JOB* 열 (CORPDATA.EMPLOYEE 표의 각 행에 있는 열들 중 하나)을 갱신하려는 경우, JOB이 SELECT문에서 생략되었으면 DECLARE CURSOR문이 FOR UPDATE OF JOB ...을 포함해야 합니다.

결과 표와 커서는 다음 중 하나가 만족되는 경우 *읽기 전용*입니다.

- 첫 번째 FROM절이 둘 이상의 표나 보기를 식별합니다.
- 첫 번째 FROM절이 읽기 전용 보기를 식별합니다.
- 첫 번째 FROM절이 사용자 정의 표 함수를 식별합니다.
- 첫 번째 SELECT절이 키워드 DISTINCT를 지정합니다.
- 외부 부속 선택에 GROUP BY절이 포함됩니다.
- 외부 부속 선택에 HAVING절이 포함됩니다.
- 첫 번째 SELECT절에 열 함수가 포함됩니다.
- SELECT문에 부속 조회가 포함되므로 외부 부속 선택과 부속 조회의 기본 오브젝트가 동일한 표가 됩니다.
- SELECT문에 UNION 또는 UNION ALL 연산자가 포함됩니다.
- 선택문이 ORDER BY절을 포함하며 FOR UPDATE OF절과 DYNAMIC SCROLL이 지정되지 않습니다.
- SELECT문에 FOR READ ONLY절이 포함됩니다.
- SCROLL 키워드가 DYNAMIC 없이 지정됩니다.
- 선택 리스트에는 DataLink 열이 포함되고 FOR UPDATE OF절은 지정되지 않습니다.
- 첫 번째 부속 선택에는 임시 결과 표가 필요합니다.
- SELECT문에 FETCH FIRST *n* ROWS ONLY가 포함됩니다.

단계 2: 커서 열기

결과 표의 행 처리를 시작하려면 OPEN문을 발행하십시오. 프로그램이 OPEN문을 발행하면 SQL은 SELECT문에 지정된 호스트 변수의 현재 값을 사용하여 결과 표라고 하는 일련의 행을 식별하기 위해 DECLARE CURSOR문 내의 SELECT문을 처리합니다. 결과 표에는 탐색 조건을 만족시키는 정도에 따라 0, 1 또는 여러 개의 행이 포함될 수 있습니다. OPEN문은 다음과 같습니다.

```
EXEC SQL
 OPEN cursor-name
END-EXEC.
```

단계 3: 자료의 끝에 도달했을 때 수행할 작업 지정

결과 표에 도달하는 시점을 알려면 100의 값에 대해 SQLCODE 필드를 테스트하거나 '02000'의 값(즉, 자료의 끝)에 대해 SQLSTATE 필드를 테스트하십시오. FETCH문이 결과 표 내의 최종 행을 검색할 때 이 조건이 발생되면 프로그램은 후속 FETCH를 발행합니다. 예를 들면 다음과 같습니다.

```
...
IF SQLCODE =100 GO TO DATA-NOT-FOUND.
```

또는

```
IF SQLSTATE ='02000' GO TO DATA-NOT-FOUND.
```

이러한 기법 대신 사용할 수 있는 방법으로는 WHENEVER문을 코딩하는 것입니다. WHENEVER NOT FOUND를 사용하면 CLOSE문이 발행되는 프로그램의 다른 부분으로 분기됩니다. WHENEVER문은 다음과 같습니다.

```
EXEC SQL
 WHENEVER NOT FOUND GO TO symbolic-address
END-EXEC.
```

사용자 프로그램은 커서가 행을 페치하는데 사용될 때마다 자료의 끝 조건을 예측해야 하며 그러한 상황을 처리할 준비가 되어 있어야 합니다.

사용자가 연속 커서를 사용중이고 자료의 끝에 도달하면 뒤따르는 모든 FETCH문이 자료의 끝 조건을 리턴시킵니다. 커서를 이미 처리된 행에 위치시킬 수는 없습니다. CLOSE문이 커서에 수행될 수 있는 유일한 조작입니다.

화면이동 커서를 사용하여 자료의 끝에 도달했을 때 결과 표는 계속 자료를 처리할 수 있습니다. 위치 옵션의 조합을 사용하면 커서를 결과 표의 모든 위치에 위치시킬 수 있습니다. 자료의 끝에 도달하면 커서의 CLOSE 조작은 수행시킬 필요가 없습니다.

단계 4: 커서를 사용하는 행 검색

선택된 행의 내용을 프로그램의 호스트 변수로 이동하려면 FETCH문을 사용하십시오. DECLARE CURSOR문에 있는 SELECT문은 프로그램이 원하는 열값이 포함된 행을 식별합니다. 그러나 SQL은 FETCH문이 제공된 후 어플리케이션 프로그램에 대한 자료를 검색합니다.

프로그램이 FETCH문을 발행할 때 SQL은 현재 커서 위치를 시작 지점으로 사용하여 결과 표에 요구된 행을 위치시킵니다. 이는 해당 행을 **현재 행**으로 변경합니다. INTO 절이 지정되면 SQL은 현재 행의 내용을 프로그램의 호스트 변수로 이동합니다. 이 순서는 FETCH문이 제공될 때마다 반복됩니다.

SQL이 커서에 대한 그 다음 FETCH문이 발행되기까지 현재 행의 위치(즉, 현재 행에 대한 커서 포인트)를 유지보수합니다. DELETE문이 수행되어도 UPDATE문은 결과 표 내의 현재 행 위치를 변경하지 않습니다.

연속 커서 FETCH문은 다음과 같습니다.

```
EXEC SQL
 FETCH cursor-name
   INTO :host variable-1[, :host variable-2] ...
END-EXEC.
```

화면이동 커서 FETCH문은 다음과 같습니다.

```
EXEC SQL
 FETCH RELATIVE integer
   FROM cursor-name
   INTO :host variable-1[, :host variable-2] ...
END-EXEC.
```

단계 5a: 현재 행 갱신

프로그램이 커서를 행에 위치시키고 나면 사용자는 WHERE CURRENT OF절을 가진 UPDATE문을 사용하여 그 행의 자료를 갱신할 수 있습니다. WHERE CURRENT OF절은 갱신하려는 행을 지시하는 커서를 지정합니다. UPDATE ... WHERE CURRENT OF문은 다음과 같습니다.

```
EXEC SQL
 UPDATE table-name
   SET column-1 = value [, column-2 = value] ...
   WHERE CURRENT OF cursor-name
END-EXEC.
```

커서와 함께 사용될 때 UPDATE문은 다음과 같습니다.

- 오직 한 행, 현재의 행만을 갱신합니다.
- 갱신될 행을 가리키는 커서를 식별합니다.
- ORDER BY절, 또한 지정되었을 경우, 갱신되는 열이 이전에 DECLARE CURSOR 문의 FOR UPDATE OF절에서 명명될 것을 요구합니다.

행을 갱신한 후 그 다음 행을 위한 FETCH문을 발행하기까지 커서의 위치가 해당 행(즉, 커서의 현재 행이 변하지 않음)에 남습니다.

단계 5b: 현재 행 삭제

프로그램이 현재 행을 검색했으면 DELETE문을 사용하여 그 행을 삭제할 수 있습니다. 그렇게 하려면 커서와 함께 사용되도록 작성된 DELETE문을 발행하십시오. WHERE CURRENT OF절이 삭제하려는 행을 가리키는 커서를 지정합니다. DELETE ... WHERE CURRENT OF문은 다음과 같습니다.

```
EXEC SQL
 DELETE FROM table-name
   WHERE CURRENT OF cursor-name
END-EXEC.
```

커서와 함께 사용될 때, DELETE문은 다음과 같습니다.

- 오직 한 행, 현재의 행만을 삭제합니다.

- WHERE CURRENT OF절을 사용하면 삭제될 행을 가리키는 커서를 식별할 수 있습니다.

행을 삭제하면 다음 행에 대해 FETCH문을 발행할 때까지 그 커서를 사용하여 다른 행을 갱신 또는 삭제할 수 없습니다.

115 페이지의 『DELETE 명령문을 사용하여 표에서 행 제거』에서는 특정 탐색 조건을 만족시키는 모든 행을 삭제하기 위한 DELETE문의 사용 방법을 보여줍니다. 또한 행의 사본을 취해 그것을 관찰한 후 삭제하고자 할 때에도 FETCH 및 DELETE ... WHERE CURRENT OF문을 사용할 수 있습니다.

단계 6: 커서 닫기

연속 커서에 대한 결과 표의 행을 처리한 후 커서를 다시 사용하려면, 커서를 다시 열기 전에 CLOSE문을 발행하여 커서를 닫으십시오.

```
EXEC SQL
 CLOSE cursor-name
END-EXEC.
```

결과 표의 행을 처리한 후 커서를 다시 사용하지 않으려면 시스템이 커서를 닫도록 할 수 있습니다. 다음의 경우 시스템이 자동으로 커서를 닫습니다.

- HOLD문 없이 COMMIT가 발행되고 커서가 WITH HOLD절을 사용하여 선언하지 않은 경우
- HOLD문 없이 ROLLBACK문이 발행되는 경우
- 작업이 종료될 경우
- 활성 그룹이 종료되고 사전컴파일에 CLOSQLCSR(*ENDACTGRP)이 지정된 경우
- 호출 스택의 첫 번째 SQL 프로그램이 종료되고 프로그램 사전컴파일시 CLOSQLCSR(*ENDJOB)이나 CLOSQLCSR(*ENDACTGRP) 중 어느 것도 지정되지 않은 경우
- DISCONNECT문을 사용하여 어플리케이션 서버로의 연결이 종료된 경우
- 어플리케이션 서버로의 연결이 해제되었고 정상적인 COMMIT가 발생한 경우
- *RUW CONNECT가 발생한 경우

  열린 커서는 참조된 표나 보기에 대해 계속 잠금을 보유하고 있으므로 열린 커서가 더 이상 필요하지 않게 되면 즉시 열린 모든 커서를 반드시 닫아야 합니다.

복수 행 FETCH문 사용

복수 행 FETCH문은 단일 FETCH가 있는 표 또는 보기에서 복수 행을 검색하는데 사용될 수 있습니다. 프로그램은 FETCH문에서 요구된 행 수별로 행 블록화를 제어합니다. 단일 페치 호출에서 요구될 수 있는 최대 행 수는 32767입니다. 자료가 일단 검색되면 커서는 검색되는 마지막 행에 위치지정됩니다.

페치된 행이 놓일 기억영역을 정의하는 방법에는 호스트 구조 배열이나 연관된 설명자를 사용하는 행 기억영역의 두 가지 방법이 있습니다. REXX의 호스트 구조 배열을 제외하면 두 가지 방법 모두 SQL 사전컴파일러에 의해 지원되는 모든 언어로 코드화될 수 있습니다. 프로그래밍 언어에 대한 자세한 내용은 호스트 언어를 사용한 SQL 프로그래밍 정보를 참조하십시오. 복수 행 FETCH문의 두 형식에 의해 어플리케이션은 분리 인디케이터 배열을 코딩할 수 있습니다. 인디케이터 배열에는 널(null) 허용의 각 호스트 변수에 대해 하나의 인디케이터가 포함되어야 합니다.

복수 행 FETCH문은 연속 커서와 화면이동 가능 커서로 사용될 수 있습니다. 복수 행 FETCH에 대한 커서 정의, 열기 및 닫기에 사용된 작업은 동일합니다. FETCH문만을 변경하면 검색될 행 수와 행이 위치할 기억영역을 지정할 수 있습니다.

각 복수 행 FETCH 후 정보는 SQLCA를 통해 프로그램에 리턴됩니다. SQLCODE 및 SQLSTATE 필드 이외에도 SQLERRD에서는 다음 정보를 제공합니다.

- SQLERRD3에는 복수 행 FETCH문에서 검색된 행의 번호가 포함됩니다. SQLERRD3이 요구된 행의 번호보다 적다면 오류 또는 자료의 끝 조건이 발생합니다.
- SQLERRD4에는 검색되는 각 행의 길이가 포함됩니다.
- SQLERRD5에는 표의 마지막 행이 페치되었음을 나타내는 표시가 포함됩니다. 이는 갱신에 필요한 즉시 감지성이 커서에 없는 경우, 표 내의 최종 행이 페치되는 것을 감지하는데 사용할 수 있습니다. 갱신에 대한 즉시 감지성을 갖는 커서는 자료의 끝 조건을 검출하기 위해 SQLCODE +100이 수신될 때까지 페칭을 계속해야 합니다.

호스트 구조 배열을 사용하는 복수 행 FETCH

호스트 구조 배열이 있는 복수 행 FETCH를 사용하려면 SQL로 사용될 수 있는 호스트 구조 배열이 정의되어야 합니다. 각 언어에는 호스트 구조 배열을 정의하는 각각의 규칙과 프로토콜이 있습니다. 호스트 구조 배열은 변수 선언을 사용하거나 외부 파일 설명 검색을 위한 컴파일러 지시문(COBOL COPY 지시문과 같은)을 사용하여 정의될 수 있습니다.

호스트 구조 배열은 구조들의 배열로 구성됩니다. 각 구조는 결과 표의 한 행에 해당됩니다. 배열의 첫 구조는 첫 번째 행에 해당하고 배열의 두 번째 구조는 두 번째 행에 해당합니다. SQL은 호스트 구조 배열의 선언에 기초한 호스트 구조 배열 내 기본 항목의 속성을 판별합니다. 성능을 최대화하려면 호스트 구조 배열을 구성하는 항목의 속성은 검색되는 열의 속성과 일치해야 합니다.

다음의 COBOL 예를 고려해 보십시오.

```
    EXEC SQL INCLUDE SQLCA
END-EXEC.
```

```
...

      01 TABLE-1.
          02 DEPT  OCCURS 10 TIMES.
             05  EMPNO   PIC   X(6).
             05 LASTNAME.
                49 LASTNAME-LEN  PIC S9(4) BINARY.
                49 LASTNAME-TEXT PIC X(15).
             05  WORKDEPT PIC  X(3).
             05  JOB      PIC  X(8).
      01 TABLE-2.
         02 IND-ARRAY   OCCURS 10 TIMES.
            05 INDS PIC  S9(4) BINARY OCCURS 4 TIMES.

...
     EXEC SQL
     DECLARE D11 CURSOR FOR
     SELECT EMPNO, LASTNAME, WORKDEPT, JOB
        FROM CORPDATA.EMPLOYEE
        WHERE WORKDEPT = "D11"
  END-EXEC.

...

     EXEC SQL
       OPEN D11
  END-EXEC.
      PERFORM FETCH-PARA UNTIL SQLCODE NOT EQUAL TO ZERO.
  ALL-DONE.
        EXEC SQL CLOSE D11 END-EXEC.

...

      FETCH-PARA.
         EXEC SQL WHENEVER NOT FOUND GO TO ALL-DONE END-EXEC.
      EXEC SQL FETCH D11 FOR 10 ROWS INTO :DEPT :IND-ARRAY
  END-EXEC.

...
```

이 예에서 커서는 CORPDATA.EMPLOYEE 표에 대해 WORKDEPT열이 'D11'에 일치하는 모든 행을 선택하도록 정의되었습니다. 결과 표는 8행을 포함하고 있습니다. DECLARE CURSOR 및 OPEN문이 복수 행 FETCH문과 함께 사용될 때 이들 문에는 특별한 구문이 없습니다. 동일한 커서에 대해 단일 행을 리턴시킨 다른 FETCH문은 프로그램의 다른 곳에서 작성될 수 있습니다. 복수 행 FETCH문은 결과 표에 있는 모든 행을 검색하기 위해 사용됩니다. FETCH를 따르는 동안 커서 위치는 검색되는 마지막 행 위에 그대로 유지됩니다.

호스트 구조 배열 DEPT와 연관된 인디케이터 배열 IND-ARRAY는 어플리케이션에 정의되어 있습니다. 두 배열은 모두 10차원입니다. 인디케이터 배열에는 결과 표 내의 각 열에 대한 항목이 있습니다.

DEPT 호스트 구조 배열 기본 항목의 유형과 길이 속성은 검색될 열과 일치합니다.

복수 행 FETCH문이 제대로 완료되었을 때 호스트 구조에는 총 8행에 대한 자료가 포함됩니다. 인디케이터 배열 IND_ARRAY는 NULL 값이 리턴되지 않았기 때문에 모든 행의 모든 열에 0을 포함합니다.

어플리케이션으로 리턴되는 SQLCA에는 다음 정보가 포함되어 있습니다.

- SQLCODE는 0을 포함합니다.
- SQLSTATE는 '00000'을 포함합니다.
- SQLERRD3은 8(페치된 행의 수)을 포함합니다.
- SQLERRD4는 34(행의 길이)를 포함합니다.
- SQLERRD5는 결과 표의 최종 행이 블록에 있음을 나타내는 +100을 포함합니다.

SQLCA의 설명은 SQL 참조서의 부록 B를 참조하십시오.

행 기억장치 영역을 사용하는 복수 행 FETCH

어플리케이션이 행 기억영역이 있는 복수 행 FETCH를 사용할 수 있게 되기 전에 어플리케이션은 행 기억영역 및 연관된 설명 영역을 정의해야 합니다. 행 기억영역은 어플리케이션 프로그램에서 정의된 호스트 변수입니다. 행 기억영역에는 복수 행 FETCH의 결과가 포함됩니다. 행 기억영역은 복수 행 FETCH에서 요구된 모든 행을 보유하도록 충분한 바이트를 가진 문자 변수가 될 수 있습니다.

연관된 설명자에 의해 정의된 각 리턴 열에 대한 SQLTYPE 및 SQLLEN을 포함한 SQLDA는 복수 행 FETCH의 행 기억영역 형식에서 사용됩니다. 설명자가 제공한 정보는 데이터베이스에서 행 기억영역까지 자료 맵핑을 판별합니다. 성능을 최대화하려면 설명자 내의 정보 속성은 리턴된 열의 속성과 일치해야 합니다.

SQLDA의 설명은 SQL 참조서의 부록 C를 참조하십시오.

다음의 PL/I 예를 고려해 보십시오.

```
*....+....1....+....2....+....3....+....4....+....5....+....6....+....7...*
     EXEC SQL INCLUDE SQLCA;
     EXEC SQL INCLUDE SQLDA;

...

    DCL DEPTPTR PTR;
    DCL 1 DEPT(20) BASED(DEPTPTR),
        3 EMPNO CHAR(6),
        3 LASTNAME CHAR(15) VARYING,
        3 WORKDEPT CHAR(3),
        3 JOB CHAR(8);
    DCL I BIN(31) FIXED;
    DEC J BIN(31) FIXED;
    DCL ROWAREA CHAR(2000);

...

    ALLOCATE SQLDA SET(SQLDAPTR);
     EXEC SQL
     DECLARE D11 CURSOR FOR
     SELECT EMPNO, LASTNAME, WORKDEPT, JOB
        FROM CORPDATA.EMPLOYEE
     WHERE WORKDEPT = 'D11';
```

그림 1. 행 기억영역을 사용한 복수 행 FETCH의 예 (1/2)

```
...

     EXEC SQL
      OPEN D11;
    /* SET UP THE DESCRIPTOR FOR THE MULTIPLE-ROW FETCH */
    /* 4 COLUMNS ARE BEING FETCHED                 */
    SQLD = 4;
    SQLN = 4;
    SQLDABC = 366;
    SQLTYPE(1) = 452; /* FIXED LENGTH CHARACTER -  */
                      /* NOT NULLABLE              */
    SQLLEN(1) = 6;
    SQLTYPE(2) = 456; /*VARYING LENGTH CHARACTER   */
                      /* NOT NULLABLE              */
    SQLLEN(2) = 15;
    SQLTYPE(3) = 452; /* FIXED LENGTH CHARACTER -  */
    SQLLEN(3) = 3;
    SQLTYPE(4) = 452; /* FIXED LENGTH CHARACTER -  */
                      /* NOT NULLABLE              */
    SQLLEN(4) = 8;
    /*ISSUE THE MULTIPLE-ROW FETCH STATEMENT TO RETRIEVE*/
    /*THE DATA INTO THE DEPT ROW STORAGE AREA      */
    /*USE A HOST VARIABLE TO CONTAIN THE COUNT OF  */
    /*ROWS TO BE RETURNED ON THE MULTIPLE-ROW FETCH     */

    J = 20;        /*REQUESTS 20 ROWS ON THE FETCH */
   ...
     EXEC SQL
  WHENEVER NOT FOUND
  GOTO FINISHED;
     EXEC SQL
  WHENEVER SQLERROR
  GOTO FINISHED;
     EXEC SQL
  FETCH D11 FOR :J ROWS
      USING DESCRIPTOR :SQLDA INTO :ROWAREA;
/* ADDRESS THE ROWS RETURNED                   */
DEPTPTR = ADDR(ROWAREA);
/*PROCESS EACH ROW RETURNED IN THE ROW STORAGE */
/*AREA BASED ON THE COUNT OF RECORDS RETURNED  */
/*IN SQLERRD3.                                 */
DO I = 1 TO SQLERRD(3);
 IF EMPNO(I) = '000170'  THEN
  DO;
      :
  END;
  END;
IF SQLERRD(5) = 100 THEN
  DO;
    /* PROCESS END OF FILE  */
  END;
FINISHED:
```

그림 1. 행 기억영역을 사용한 복수 행 FETCH의 예 (2/2)

이 예에서 커서는 CORPDATA.EMPLOYEE 표에 대해 WORKDEPT열이 'D11'에 일치하는 모든 행을 선택하도록 정의되었습니다. 부록 A『iSeries용 DB2 UDB 샘플 표』

의 샘플 EMPLOYEE 표는 결과 표에 복수 행이 포함됨을 나타냅니다. DECLARE CURSOR 및 OPEN문은 복수 행 FETCH문과 함께 사용될 경우 특수 구문을 갖지 않습니다. 동일한 커서에 대해 단일 행을 리턴시킨 다른 FETCH문은 프로그램의 다른 곳에서 작성될 수 있습니다. 복수 행 FETCH문은 결과 표에 있는 모든 행을 검색하기 위해 사용됩니다. FETCH에 뒤이어 커서 위치는 블록 내의 최종 행에 남아 있게 됩니다.

행 영역인 ROWAREA는 문자 배열로 정의됩니다. 결과 표에 대한 자료는 호스트 변수에 놓여집니다. 다음 예제에서 포인터 변수는 ROWAREA의 주소에 할당됩니다. 리턴된 행의 각 항목은 기본 구조 DEPT와 함께 체크되고 사용됩니다.

설명자 내의 항목 속성(유형과 길이)은 검색된 열과 일치합니다. 이 경우, 인디케이터 영역은 제공되지 않습니다.

FETCH문이 완료된후 ROWAREA에는 'D11'(이 경우 11행)과 동일한 모든 행이 포합됩니다. 어플리케이션으로 리턴되는 SQLCA에는 다음 정보가 들어 있습니다.

- SQLCODE는 0을 포함합니다.
- SQLSTATE는 ‘00000’을 포함합니다.
- SQLERRD3은 11(리턴된 행의 수)을 포함합니다.
- SQLERRD4는 34(페치된 행의 길이)를 포함합니다.
- SQLERRD5는 결과 표의 최종 행이 페치되었음을 나타내는 +100을 포함합니다.

이 예에서 어플리케이션은 파일의 끝에 도달했음을 나타내는 표시가 SQLERRD5에 들어 있는 사실을 이용합니다. 그 결과 어플리케이션은 행을 더 검색하기 위해 SQL을 다시 호출할 필요가 없습니다. 삽입에 대한 즉시 감지성이 커서에 있다면 레코드가 추가된 경우 SQL을 호출해야 합니다. 커서는 확약 제어 레벨이 *RR이 아닌 경우 즉시 감지성을 가집니다.

작업 단위 및 열린 커서

사용자 프로그램이 작업 단위를 완료하면 사용자가 수행한 변경사항이 확약 또는 롤백됩니다. COMMIT 또는 ROLLBACK문에서 HOLD를 지정하지 않는 한 열린 모든 커서는 SQL에 의해 자동으로 닫힙니다. WITH HOLD절에 의해 선언된 커서는 COMMIT에서 자동으로 닫히지 않습니다. 그런 커서는 ROLLBACK에서 자동으로 닫힙니다(DECLARE CURSOR문에서 지정된 WITH HOLD절이 무시됩니다).

COMMIT 또는 ROLLBAK문 실행 다음에 현재 커서 위치에서 처리를 계속하려면 COMMIT HOLD 또는 ROLLBAK HOLD를 지정해야 합니다. HOLD를 지정하면 열린 모든 커서가 열려진 채 커서 위치에 있으므로 처리를 재개할 수 있습니다. 커서 위치는 COMMIT문에서 관리됩니다. ROLLBACK문에서 커서 위치는 이전 작업 단위에서 검색된 마지막 행의 바로 뒤로 복원됩니다. 모든 레코드 잠금은 계속 해제됩니다.

COMMIT 또는 ROLLBACK문을 HOLD 없이 발행하고 나면 모든 잠금은 해제되고 모든 커서는 닫혀집니다. 커서를 다시 열 수는 있으나 결과 표의 첫 번째 행에서 처리가 시작됩니다.

주: CRTSQLxxx 명령의 ALWBLK(*ALLREAD) 매개변수를 지정하여 읽기 전용 커서에 대해 커서 위치 복원을 변경할 수 있습니다. ALWBLK 매개변수의 사용과 CRTSQLxxx 명령의 옵션에 관련된 다른 성능에 대해 자세히 알려면 제 14 장『동적 SQL 어플리케이션』을 참조하십시오.

확약 제어 및 작업 단위에 대한 자세한 정보는 확약 제어 주제를 참조하십시오.

제 10 장 자료 무결성

자료 무결성은 스키마의 표 사이에 자료 값을 업무에 맞는 상태로 보유되도록 보장하는 원리입니다. 예를 들어 은행에서 표 A에 고객 리스트를 가지고 있고 표 B에 고객 계좌 리스트를 가지고 있다면 연관된 고객이 표 A에 있지 않는 한 새로운 계좌를 표 B에 추가한다는 것은 무리입니다.

이 장에서는 시스템이 자동으로 이런 종류의 관계를 강화하는 여러 가지 방법에 대해 설명합니다. 참조 무결성, 체크 제한사항 및 트리거는 모두 자료 무결성을 달성하는 방법입니다. 또한 CREATE VIEW의 WITH CHECK OPTION절에서는 보기를 통해 자료를 삽입하거나 갱신하는 것이 제한됩니다. 자세한 내용은 다음의 주제를 참조하십시오.

- 『체크 제한조건 추가 및 사용』
- 152 페이지의 『참조 무결성』
- 162 페이지의 『보기에서 WITH CHECK OPTION』
- 165 페이지의 『iSeries용 DB2 UDB 트리거 지원』

자료 무결성에 대한 포괄적인 정보는 데이터베이스 프로그래밍을 참조하십시오.

체크 제한조건 추가 및 사용

*제한조건 체크*는 열이나 열 그룹에 허용되는 값을 제한하여 삽입과 갱신시에 자료의 유효성을 보장합니다. SQL CREATE TABLE 및 ALTER TABLE문을 사용하여 점검 제한조건을 추가하십시오.

아래 예에서, 다음 명령문은 3열과, 해당 열에서 허용되는 값을 양수로 제한하는 COL2에 대한 점검 제한조건이 있는 표를 작성합니다.

```
CREATE TABLE T1 (COL1 INT, COL2 INT CHECK (COL2>0), COL3 INT)
```

이 표에 대해 다음 명령문을 사용하는 경우:

```
INSERT INTO T1 VALUES (-1, -1, -1)
```

COL2에 삽입될 값이 체크 제한조건을 충족시키지 않기 때문에 위 명령문은 실패합니다. 즉, -1은 0보다 크지 않습니다.

다음 명령문은 성공적으로 수행될 것입니다.

```
INSERT INTO T1 VALUES (1, 1, 1)
```

일단 해당 행이 삽입되면 다음 명령문은 실패합니다.

```
ALTER TABLE T1 ADD CONSTRAINT C1 CHECK (COL1=1 AND COL1<COL2)
```

이 ALTER TABLE문은 COL1에 허용되는 값을 제한하는 두 번째 체크 제한조건과 1보다 큰 COL2의 값인 규칙도 효율적으로 추가합니다. 이 제한조건은 허용되지 않는데 이는 이 제한조건의 두 번째 부분이 기존 자료와 맞지 않기 때문입니다(COL2의 값 '1'은 COL1의 값 '1'보다 작지 않음).

참조 무결성

참조 무결성은 한 표에서 다른 표로의 모든 참조가 유효한 데이터베이스 표 집합의 조건입니다.

다음과 같은 예를 고려합니다(이 샘플 표는 부록 A『iSeries용 DB2 UDB 샘플 표』에 나타나 있습니다).

- CORPDATA.EMPLOYEE는 사원의 마스터 리스트로 사용됩니다.
- CORPDATA.DEPARTMENT는 유효한 모든 부서 번호의 마스터 리스트로 사용됩니다.
- CORPDATA.EMP_ACT는 프로젝트에 대해 수행된 활동의 마스터 리스트를 제공합니다.

다른 표는 이 표에서 설명된 것과 같은 엔티티를 참조합니다. 표가 마스터 리스트가 있는 자료를 포함할 때 이 자료는 실제로 마스터 리스트에 나타나야 하며 그렇지 않을 경우에는 그 자료에 대한 참조가 유효하지 않습니다. 마스터 리스트를 포함하는 표는 *상위 표*이고 이를 참조하는 표는 *종속 표*입니다. 종속 표에서 상위 표로의 참조가 유효한 경우, 표 집합의 조건을 *참조 무결성*이라고 합니다.

다시 말하면 참조 무결성이란 모든 외부 키의 모든 값이 유효한 데이터베이스의 상태를 말합니다. 외부 키의 각각의 값은 상위 키에도 존재하거나 널(null)이어야 합니다. 이러한 참조 무결성 정의를 위해서는 다음 용어를 이해하고 있어야 합니다.

- *고유 키*는 고유하게 한 행을 식별하는 한 표 내의 열 또는 열 세트입니다. 하나의 표에 여러 개의 고유 키가 있을 수 있지만 한 표에 있는 두 개의 행이 같은 고유 키 값을 가질 수는 없습니다.
- *1차 키*는 널(null)을 허용하지 않는 고유 키입니다. 하나의 표가 둘 이상의 1차 키를 가질 수는 없습니다.
- *상위 키*는 참조 제한조건에서 참조되는 고유 키 또는 1차 키입니다.
- *외부 키*는 그 값이 상위 키의 값과 일치하는 열 또는 열 세트입니다. 외부 키를 빌드하기 위해 사용된 열값이 널인 경우, 이 규칙은 적용되지 않습니다.
- *상위 표*는 상위 키를 포함하는 표입니다.
- *종속 표*는 외부 키를 포함하는 표입니다.
- *하부 표*는 종속 표이거나 종속 표의 하부인 표입니다.

참조 무결성을 강제하면 모든 널이 아닌 외부 키가 맵핑 상위 키를 가져야 한다고 명시된 규칙이 위반되지 않도록 할 수 있습니다.

참조 무결성에 대한 자세한 내용은 다음의 주제를 참조하십시오.

- 『참조 제한조건 추가 또는 제거』
- 155 페이지의 『참조 제한조건 제거』
- 155 페이지의 『참조 제한조건이 있는 표에 삽입』
- 156 페이지의 『참조 제한조건이 있는 표 갱신』
- 158 페이지의 『참조 제한조건이 있는 표에서 삭제』
- 161 페이지의 『체크 지연』

SQL은 CREATE TABLE 및 ALTER TABLE문의 참조 무결성 개념을 지원합니다. 이러한 명령에 대한 자세한 설명은 SQL 참조서를 참조하십시오. 또한 iSeries Navigator를 사용하여 기존의 표를 추가하거나 표를 작성할 때 사용하는 제한조건을 추가할 수도 있습니다.

참조 제한조건 추가 또는 제거

제한사항은 하나의 표, 종속 표에서 다른 표, 상위 표의 자료로의 참조가 유효한지를 확인하는 규칙입니다. 참조 제한조건을 사용하여 참조 무결성을 확인할 수 있습니다.

SQL CREATE TABLE 및 ALTER TABLE문을 사용하여 참조 제한조건을 추가하거나 변경하십시오.

참조 제한조건을 사용하여, 외부 키 값이 상위 키 값에도 나타나는 경우에만 널(null)이 아닌 외부 키 값이 유효합니다. 참조 제한조건을 정의할 때에는 다음을 지정하십시오.

- 1차 또는 고유 키
- 외부 키
- 상위 행이 삭제되거나 갱신될 때 종속 행에 대해 취해지는 조치를 지정하는 삭제 및 갱신 규칙

제한조건에 대해 선택적으로 이름을 지정할 수 있습니다. 이름을 지정하지 않으면 이름이 자동으로 생성됩니다.

일단 참조 제한사항이 정의되면, 시스템은 iSeries Navigator, CL 명령, 유틸리티 또는 고급 언어 명령문을 비롯한 다른 인터페이스나 SQL을 통해 수행된 모든 INSERT, DELETE 및 UPDATE 조작에 제한사항을 실시합니다.

예: 참조 제한조건 추가

샘플 사원 표에 나타난 모든 부서 번호가 부서 표에도 나타나야 한다는 규칙이 참조 제한조건입니다. 이 제한조건은 모든 사원이 기존의 부서에 소속되도록 보장합니다. 다

음 SQL문은 제한조건 관계를 정의하여 CORPDATA.DEPARTMENT 및 CORPDATA.EMPLOYEE 표를 작성합니다.

```
CREATE TABLE CORPDATA.DEPARTMENT
      (DEPTNO    CHAR(3)     NOT NULL PRIMARY KEY,
       DEPTNAME  VARCHAR(29) NOT NULL,
       MGRNO     CHAR(6),
       ADMRDEPT  CHAR(3)     NOT NULL
                             CONSTRAINT REPORTS_TO_EXISTS
                               REFERENCES CORPDATA.DEPARTMENT (DEPTNO)
                               ON DELETE CASCADE)

CREATE TABLE CORPDATA.EMPLOYEE
      (EMPNO     CHAR(6)     NOT NULL PRIMARY KEY,
       FIRSTNAME VARCHAR(12) NOT NULL,
       MIDINIT   CHAR(1)     NOT NULL,
       LASTNAME  VARCHAR(15) NOT NULL,
       WORKDEPT  CHAR(3)     CONSTRAINT WORKDEPT_EXISTS
                               REFERENCES CORPDATA.DEPARTMENT (DEPTNO)
                               ON DELETE SET NULL ON UPDATE RESTRICT,
       PHONENO   CHAR(4),
       HIREDATE  DATE,
       JOB       CHAR(8),
       EDLEVEL   SMALLINT    NOT NULL,
       SEX       CHAR(1),
       BIRTHDATE DATE,
       SALARY    DECIMAL(9,2),
       BONUS     DECIMAL(9,2),
       COMM      DECIMAL(9,2),
       CONSTRAINT  UNIQUE_LNAME_IN_DEPT UNIQUE (WORKDEPT, LASTNAME))
```

이 경우 DEPARTMENT 표에는 1차 키로 기능하는 고유 부서 번호(DEPTNO) 열이 있으며, 이 표는 2개의 제한조건 관계 상위 표입니다.

REPORTS_TO_EXISTS

DEPARTMENT 표가 같은 관계의 상위 표 및 종속 표인 자체 참조 제한조건입니다. ADMRDEPT의 모든 널이 아닌 값은 DEPTNO의 값과 일치해야 합니다. 부서는 데이터베이스 내의 기존 부서에 보고해야 합니다. DELETE CASCADE 규칙은 DEPTNO 값이 *n*인 행이 삭제되는 경우 ADMRDEPT가 *n*인 표의 모든 행도 삭제됩니다.

WORKDEPT_EXISTS

EMPLOYEE 표를 종속 표로 설정하며 사원 부서 할당(WORKDEPT) 열을 외부 키로 설정합니다. 따라서 WORKDEPT의 모든 값은 DEPTNO 값과 일치해야 합니다. DELETE SET NULL 규칙은 행이 DEPTNO 값이 *n*인 DEPARTMENT에서 삭제되는 경우, EMPLOYEE의 WORKDEPT 값은 값이 *n*인 모든 행에서 널(null)로 설정됨을 나타냅니다. UPDATE RESTRICT 규칙은 DEPARTMENT의 DEPTNO 값이 EMPLOYEE의 WORKDEPT 값이 현재 DEPTNO 값과 일치하는 경우 갱신될 수 없음을 나타냅니다.

EMPLOYEE 표에서 제한조건 UNIQUE_LNAME_IN_DEPT는 성(last name)이 부서 내에서 고유하도록 합니다. 이 제한조건이 불확실하게 보이지만 여러 개의 열로 구성된 제한조건이 표 레벨로 정의될 수 있는 방법을 보여줍니다.

참조 제한조건 제거

ALTER TABLE문은 한 표에 대해 하나의 제한조건을 동시에 추가하거나 제거하는데 사용될 수 있습니다. 제거되는 제한조건이 참조 제한조건 관계에 있는 상위 키일 경우, 이 상위 파일과 종속 파일간의 제한조건도 역시 제거됩니다.

DROP TABLE문과 DROP SCHEMA문도 드롭되는 표나 스키마에 대한 제한사항을 제거합니다.

예: 제한조건 제거

다음 예는 표 DEPARTMENT의 열 DEPTNO에 대한 1차 키를 제거합니다. 제거되는 1차 키가 제한조건 관계에 있는 상위 키이므로 표 DEPARTMENT와 EMPLOYEE에서 정의된 제한조건 REPORTS_TO_EXISTS 및 WORKDEPT_EXISTS 또한 각각 제거될 것입니다.

```
ALTER TABLE CORPDATA.EMPLOYEE DROP PRIMARY KEY
```

또한 다음 예에서와 같이 이름별로 제한조건을 제거할 수도 있습니다.

```
ALTER TABLE CORPDATA.DEPARTMENT
            DROP CONSTRAINT UNIQUE_LNAME_IN_DEPT
```

참조 제한조건이 있는 표에 삽입

참조 제한조건이 있는 표에 자료를 삽입할 때 명심해야 할 몇 가지 중요한 사항이 있습니다. 상위 키를 가진 상위 표에 자료를 삽입할 경우 SQL에서는 다음을 허용하지 않습니다.

- 상위 키에 대한 복제값
- 상위 키가 1차 키일 경우, 1차 키의 모든 열에 대한 널 값

외부 키를 가진 종속 표에 자료를 삽입할 경우

- 외부 키 열에 삽입하는 각각의 널이 아닌 값은 상위 표의 해당 상위 키에 있는 일부 값과 같아야 합니다.
- 외부 키에 있는 열의 일부가 널이면 전체 외부 키가 널로 간주됩니다. 열을 포함한 모든 외부 키가 널이면 INSERT가 계속됩니다(고유 색인 위반이 없을 때까지).

예: 제한조건이 있는 자료 삽입

두 외부 키를 정의하기 위해 샘플 어플리케이션 프로젝트 표(PROJECT)를 변경하십시오.

- 부서 표를 참조하는 부서 번호(DEPTNO)에서 외부 키
- 사원 표를 참조하는 사원 번호(RESPEMP)에서 외부 키

```
ALTER TABLE CORPDATA.PROJECT ADD CONSTRAINT RESP_DEPT_EXISTS
                    FOREIGN KEY (DEPTNO)
                    REFERENCES CORPDATA.DEPARTMENT
                ON DELETE RESTRICT

ALTER TABLE CORPDATA.PROJECT ADD CONSTRAINT RESP_EMP_EXISTS
                FOREIGN KEY (RESPEMP)
                REFERENCES CORPDATA.EMPLOYEE
                ON DELETE RESTRICT
```

REFERENCES절에서는 상위 표 열이 지정되지 않음을 유의하십시오. 참조된 표에 1차 키가 있거나 상위 키로 사용될 수 있는 자격이 있는 고유 키가 있는 한 열 지정은 필요없습니다.

PROJECT 표에 삽입된 모든 행은 부서 표에 있는 DEPTNO의 일부 값과 같은 DEPTNO의 값을 가져야 합니다(프로젝트 표의 DEPTNO가 NOT NULL로 정의되었으므로 널 값이 허용되지 않습니다). 행은 사원 표의 EMPNO의 일부 값과 같거나 널인 RESPEMP의 값과 같아야 합니다.

부록 A『iSeries용 DB2 UDB 샘플 표』에 표시된 대로 샘플 자료를 가진 표는 이 제한조건을 따릅니다. 다음 INSERT문은 DEPARTMENT 표 내에 맵핑 DEPTNO값('A01')이 없으므로 실패합니다.

```
INSERT INTO CORPDATA.PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
       VALUES  ('AD3120', 'BENEFITS ADMIN', 'A01', '000010')
```

마찬가지로 다음 INSERT문은 EMPLOYEE 표에 EMPNO값 '000011'이 없으므로 실패합니다.

```
INSERT INTO CORPDATA.PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
       VALUES ('AD3130', 'BILLING', 'D21', '000011')
```

다음 INSERT문은 DEPARTMENT 표에 맵핑 DEPTNO 값 'E01'이 있으며 EMPLOYEE 표에 맵핑 EMPNO 값 '000010'이 있으므로 성공적으로 완료됩니다.

```
INSERT INTO CORPDATA.PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
       VALUES ('AD3120', 'BENEFITS ADMIN', 'E01', '000010')
```

참조 제한조건이 있는 표 갱신

모표를 갱신중이라면 종속 행이 존재하는 1차 키를 수정할 수 없습니다. 키의 변경은 종속 표에 대한 참조 제한조건을 위반하게 되며 일부 행에 상위(parent)가 없게 합니다. 게다가 1차 키의 어느 부분에도 널 값을 부여할 수 없습니다.

갱신 규칙

상위 표에 UPDATE가 수행될 때 종속 표에 취해지는 조치는 참조 제한조건에 대해 지정된 갱신 규칙에 따라 다릅니다. 참조 제한조건에 대한 갱신 규칙이 정의되어 있지 않을 경우, UPDATE NO ACTION 규칙이 사용됩니다.

UPDATE NO ACTION

다른 행이 상위 표 행에 종속되어 있지 않을 경우, 상위 표 행이 갱신될 수 있도록 지정합니다. 하나의 종속 행이 관계에 존재할 경우, UPDATE는 실패합니다. 종속 행에 대한 체크는 명령문 종료시에 수행됩니다.

UPDATE RESTRICT

다른 행이 상위 표 행에 종속되어 있지 않을 경우, 상위 표 행이 갱신될 수 있도록 지정합니다. 하나의 종속 행이 관계에 존재할 경우, UPDATE는 실패합니다. 종속 행에 대한 체크가 즉시 수행됩니다.

RESTRICT 규칙과 NO ACTION 규칙 간의 미세한 차이점은 트리거 및 참조 제한조건의 상호 작용을 검토함으로써 가장 쉽게 알 수 있습니다. 트리거는 조작 전후에 작동하도록 정의될 수 있습니다(이 경우에는 UPDATE문). *이전 트리거*는 UPDATE 수행 전과 제한조건 체크 전에 시작됩니다. *이후 트리거*는 DELETE가 수행된 후, 즉 RESTRICT의 제한조건 규칙 이후(체크가 즉시 수행될 때) 제한조건 규칙 NO ACTION 이전에(체크가 명령문 종료 시에 수행될 때) 작동됩니다. 트리거와 규칙은 다음 순서로 발생합니다.

1. *이전 트리거*는 UPDATE 전과 RESTRICT 또는 NO ACTION의 제한조건 규칙 전에 시작됩니다.
2. *이후 트리거*는 RESTRICT 규칙 후와 NO ACTION 규칙 전에 시작됩니다.

종속표를 갱신중이면 변경하는 모든 널이 아닌 외부 키값은 표가 종속 관계에 있는 각각의 관계에 대한 1차 키와 일치해야 합니다. 예를 들어 사원 표의 부서 번호는 부서 표의 부서 번호에 따라 다릅니다. 사원을 어떤 부서에도 할당할 수 있으나 (널 값) 존재하지 않는 부서는 예외입니다.

참조 제한조건이 있는 표에 대한 UPDATE가 실패할 경우 갱신 조작 동안 수행된 모든 변경사항이 취소됩니다. 제한사항에 대해 작업할 때 확약 제어와 저널링의 포함에 대한 자세한 내용은 349 페이지의 『저널링』 및 350 페이지의 『확약 제어』를 참조하십시오.

예: UPDATE 규칙

예를 들어 어떤 부서가 여전히 일부 프로젝트에 대해 책임이 있으며 그 프로젝트가 프로젝트 표 내의 종속 행에 의해 설명되고 있다면 부서 표로부터 부서 번호를 갱신할 수 없습니다.

PROJECT 표에 ‘D01’의 값(WHERE문에 의해 목표된 행)이 있는 DEPARTMENT.DEPTNO에 종속되는 행이 있으므로 다음 UPDATE는 실패합니다. 이 UPDATE가 허용될 경우, PROJECT 및 DEPARTMENT 표간의 참조 제한조건은 파기될 것입니다.

```
UPDATE CORPDATA.DEPARTMENT
        SET DEPTNO = 'D99'
        WHERE DEPTNAME = 'DEVELOPMENT CENTER'
```

다음 명령문은 DEPARTMENT에 있는 1차 키 DEPTNO와 PROJECT에 있는 외부 키 DEPTNO 사이에 존재하는 참조 제한조건을 위반하기 때문에 실패합니다.

```
UPDATE CORPDATA.PROJECT
  SET DEPTNO = 'D00'
  WHERE DEPTNO = 'D01';
```

명령문이 D01의 모든 부서 번호를 부서 번호 D00으로 변경하려고 시도합니다. D00이 DEPARTMENT의 1차 키 DEPTNO의 값이 아니므로 명령문은 실패합니다.

참조 제한조건이 있는 표에서 삭제

표에 1차 키는 있으나 종속 키가 없을 경우, 참조 제한조건이 없을 때와 같이 DELETE가 수행됩니다. 표에 외부 키는 있으나 1차 키가 없을 때에도 이와 같습니다. 표에 1차 키와 종속 표가 있을 경우, DELETE는 지정된 삭제 규칙에 따라 행을 삭제하거나 갱신합니다. 삭제 조작을 성공시키려면 영향받은 모든 관계의 전체 삭제 규칙이 충족되어야 합니다. 참조 제한조건이 위반될 경우 DELETE는 실패합니다.

상위 표에서 DELETE가 수행될 때 종속 표에 대해 취해질 조치는 참조 제한조건에 대해 지정된 삭제 규칙에 따라 다릅니다. 삭제 규칙이 정의되어 있지 않으면 DELETE NO ACTION 규칙이 사용됩니다.

DELETE NO ACTION

다른 행이 상위 표 행에 종속되어 있지 않을 경우 상위 표 행이 삭제될 수 있도록 지정합니다. 하나의 종속 행이 관계에 존재할 경우 DELETE는 실패합니다. 종속 행에 대한 체크는 명령문 종료시에 수행됩니다.

DELETE RESTRICT

다른 행이 상위 표 행에 종속되어 있지 않을 경우 상위 표 행이 삭제될 수 있도록 지정합니다. 하나의 종속 행이 관계에 존재할 경우 DELETE는 실패합니다. 종속 행에 대한 체크가 즉시 수행됩니다.

예를 들어 어떤 부서가 여전히 일부 프로젝트에 대해 책임이 있으며 프로젝트 표 내의 종속 행에 의해 서술되고 있다면 부서 표로부터 그 부서를 삭제할 수 없습니다.

DELETE CASCADE

상위 표에서 지정된 행이 먼저 삭제되도록 지정합니다. 그런 후 종속 행이 삭제됩니다.

예를 들어 부서 표 내의 행을 삭제함으로써 부서를 삭제할 수 있습니다. 부서 표에서 행을 삭제하면 다음 항목도 삭제됩니다.

- 그 부서에 보고하는 모든 부서에 대한 행
- 그러한 부서에 보고하는 모든 부서

DELETE SET NULL

각 종속 행에서 외부 키의 널 가능 열이 디폴트 값으로 설정되도록 지정합니

다. 이는 이것이 삭제될 행을 참조하는 외부 키의 멤버일 경우 그 열이 디폴트 값으로만 설정됨을 의미합니다. 인접 하부인 종속 행에만 영향을 받습니다.

DELETE SET DEFAULT

각 종속 행 내 외부 키의 각 열이 디폴트 값으로 설정되도록 지정합니다. 이는 이것이 삭제될 행을 참조하는 외부 키의 멤버일 경우 그 열이 디폴트 값으로만 설정됨을 의미합니다. 인접 하부인 종속 행만 영향을 받습니다.

예를 들어, 사원이 어떤 부서를 관리하는 경우 사원 표에서 사원을 삭제할 수 있습니다. 이 경우, 관리자에게 보고하는 각 사원에 대한 값 MGRNO는 부서 표(DEPARTMENT) 내에서 공백으로 설정됩니다. 다른 몇 가지 디폴트 값이 표 작성시 지정될 경우 이 값이 사용됩니다.

이는 부서 표에 대해 정의된 REPORTS_TO_EXISTS 제한조건에 기인합니다.

하부 표에 삭제 규칙 RESTRICT 또는 NO ACTION이 있고 하부 행이 삭제될 수 없는 행이 있으면 전체 DELETE는 실패합니다.

하나의 프로그램으로 이 명령문을 수행할 때 삭제된 행의 수는 SQLCA의 SQLERRD(3)로 리턴됩니다. 이 수에는 DELETE문에 지정된 표에서 삭제된 행의 수만 포함됩니다. 여기에는 CASCADE 규칙에 따라 삭제된 행은 포함되지 않습니다. SQLCA의 SQLERRD(5)에는 모든 표 내의 참조 제한조건에 의해 영향받은 행의 수가 들어 있습니다.

RESTRICT 규칙과 NO ACTION 규칙간의 미세한 차이점은 트리거 및 참조 제한조건의 상호 작용을 검토함으로써 가장 쉽게 알 수 있습니다. 트리거는 조작 전후에 작동하도록 정의될 수 있습니다(이 경우에는 DELETE문). *이전 트리거*는 DELETE 수행 전과 제한조건 체크 전에 시작됩니다. *이후 트리거*는 DELETE가 수행된 후, 즉 RESTRICT의 제한조건 규칙 이후(체크가 즉시 수행될 때) 제한조건 규칙 NO ACTION 이전에(체크가 명령문 종료시에 수행될 때) 작동됩니다. 트리거와 규칙은 다음 순서로 발생합니다.

1. *이전 트리거*는 DELETE 전과 RESTRICT 또는 NO ACTION의 제한조건 규칙 전에 시작됩니다.
2. *이후 트리거*는 RESTRICT 규칙 후와 NO ACTION 규칙 전에 시작됩니다.

예: DELETE 케스케이드 규칙

DEPARTMENT 표로부터 부서를 삭제하여 이 부서에 할당된 모든 사원에 대한 WORKDEPT(EMPLOYEE 표에서)를 널(null)로 설정합니다. 아래 DELETE문을 고려합니다.

```
DELETE FROM CORPDATA.DEPARTMENT
       WHERE DEPTNO = 'E11'
```

부록 A『iSeries용 DB2 UDB 샘플 표』에 표시된 것과 같은 표 및 자료가 주어질 경우, 하나의 행이 표 DEPARTMENT로부터 삭제되고 값이 'E11'일 때는 언제나

WORKDEPT값을 디폴트 값으로 설정하기 위해 표 EMPLOYEE가 갱신됩니다. 위의 샘플 자료에서 물음표('?')는 널 값을 나타냅니다. 결과는 다음과 같이 나타납니다.

표 22. DEPARTMENT 표. DELETE문이 완료된 후의 표 내용

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00
B01	PLANNING	000020	A00
C01	INFORMATION CENTER	000030	A00
D01	DEVELOPMENT CENTER	?	A00
D11	MANUFACTURING SYSTEMS	000060	D01
D21	ADMINISTRATION SYSTEMS	000070	D01
E01	SUPPORT SERVICES	000050	A00
E21	SOFTWARE SUPPORT	000100	E01
F22	BRANCH OFFICE F2	?	E01
G22	BRANCH OFFICE G2	?	E01
H22	BRANCH OFFICE H2	?	E01
I22	BRANCH OFFICE I2	?	E01
J22	BRANCH OFFICE J2	?	E01

부서 'E11'에 보고하는 부서가 없으므로 DEPARTMENT 표에 연쇄된 삭제가 없음을 유의하십시오.

다음은 DELETE문이 완료되기 전과 후에 EMPLOYEE 표에서 영향을 받은 어떤 부분에 대한 스냅샷입니다.

표 23. EMPLOYEE 부분 표. DELETE문 전의 부분 내용.

EMPNO	FIRSTNME	MI	LASTNAME	WORKDEPT	PHONENO	HIREDATE
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05
000250	DANIEL	S	SMITH	D21	0961	1960-10-30
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11
000270	MARIA	L	PEREZ	D21	9001	1980-09-30
000280	ETHEL	R	SCHNEIDER	E11	0997	1967-03-24
000290	JOHN	R	PARKER	E11	4502	1980-05-30
000300	PHILIP	X	SMITH	E11	2095	1972-06-19
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07
000330	WING		LEE	E21	2103	1976-02-23
000340	JASON	R	GOUNOT	E21	5696	1947-05-05

표 24. EMPLOYEE 부분 표. DELETE문 후의 부분 내용.

EMPNO	FIRSTNME	MI	LASTNAME	WORKDEPT	PHONENO	HIREDATE
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21

표 24. EMPLOYEE 부분 표 (계속). DELETE문 후의 부분 내용.

EMPNO	FIRSTNME	MI	LASTNAME	WORKDEPT	PHONENO	HIREDATE
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05
000250	DANIEL	S	SMITH	D21	0961	1960-10-30
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11
000270	MARIA	L	PEREZ	D21	9001	1980-09-30
000280	ETHEL	R	SCHNEIDER	?	0997	1967-03-24
000290	JOHN	R	PARKER	?	4502	1980-05-30
000300	PHILIP	X	SMITH	?	2095	1972-06-19
000310	MAUDE	F	SETRIGHT	?	3332	1964-09-12
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07
000330	WING		LEE	E21	2103	1976-02-23
000340	JASON	R	GOUNOT	E21	5696	1947-05-05

체크 지연

참조 제한조건과 체크 제한조건은 제한조건의 잠재적인 위반 존재시 체크 지연중으로 알려진 상태일 수 있습니다. 참조 제한조건의 경우, 잠재적인 불일치가 상위 키와 외부 키 간에 존재할 때 발생합니다. 체크 제한조건의 경우, 잠재적인 값이 체크 제한조건으로 제한되는 열에 존재할 때 발생합니다. 시스템이 이 제한조건의 위반을 판별하면(복원 조작 후와 같은 경우) 제한조건은 체크 지연으로 표시됩니다. 이 경우 제한조건에 속하는 표 사용에 제한이 일어납니다. 참조 제한조건의 경우, 다음의 제한이 적용됩니다.

- 종속 파일에서 입/출력 조작은 허용되지 않습니다.
- 상위 파일에서는 읽기 및 삽입 조작만 허용됩니다.

체크 제한조건이 체크 지연 상태인 경우, 다음의 제한이 적용됩니다.

- 읽기 조작이 이 파일에 대해 허용되지 않습니다.
- 삽입과 갱신은 허용되고 제한조건은 강제됩니다.

체크 지연으로부터 제한조건을 구하려면 반드시 다음과 같이 해야 합니다.

1. CHGPFCST(실제 파일 제한조건 변경) CL 명령을 사용하여 관계를 사용할 수 없게 하십시오.
2. 참조 제한조건에 대한 키(외부 키, 상위 키 또는 둘다)나 체크 제한사항에 대한 열 자료를 정정하십시오.
3. CHGPFCST CL 명령으로 다시 제한조건을 사용할 수 있게 하십시오.

DSPCPCST(체크 지연 제한조건 표시) CL 명령을 사용하여 제한조건을 위반하는 행을 식별할 수 있습니다.

검사 지연 상태에 있는 표에 대한 작업의 자세한 내용은 데이터베이스 프로그래밍 책을 참조하십시오.

보기에서 WITH CHECK OPTION

WITH CHECK OPTION은 보기를 통해 자료를 삽입하거나 갱신할 때 수행될 체크 레벨을 지정하는 CREATE VIEW문에서 선택할 수 있는 절입니다. 옵션이 지정될 경우, 보기를 통해 삽입되거나 갱신되는 모든 행은 이 보기의 정의를 따라야 합니다.

보기가 읽기 전용일 경우 WITH CHECK OPTION은 지정될 수 없습니다. 보기의 정의에는 부속 조회가 포함되지 않습니다.

보기가 WITH CHECK OPTION절 없이 작성될 경우, 보기에 대해 수행되는 삽입 조작과 갱신 조작이 보기 정의를 따르는지에 대해서는 체크되지 않습니다. 보기가 WITH CHECK OPTION을 포함한 다른 보기에 직접 또는 간접적으로 종속될 경우에는 몇 가지 체크가 계속해서 이루어집니다. 보기의 정의가 사용되지 않으므로 행이 보기의 정의에 맞지 않는 보기를 통해 삽입되거나 갱신될 수도 있습니다. 이는 행이 보기를 사용하여 다시 선택될 수 없음을 의미합니다.

검사는 『WITH CASCADED CHECK OPTION』 또는 163 페이지의 『WITH LOCAL CHECK OPTION』일 수 있습니다. WITH CHECK OPTION에 관한 추가 논의 내용은 SQL 참조서의 CREATE VIEW 주제를 참조하십시오.

WITH CASCADED CHECK OPTION

WITH CASCADED CHECK OPTION은 보기를 통해 삽입되거나 갱신되는 모든 행이 보기의 정의를 따라야 함을 지정합니다. 또한 행이 삽입되거나 갱신될 때의 모든 종속 보기의 탐색 조건이 체크됩니다. 보기의 정의를 따르지 않을 경우, 행은 보기를 사용하여 검색될 수 없습니다.

예를 들면, 다음과 같은 갱신할 수 있는 보기를 고려합니다.

```
CREATE VIEW  V1 AS SELECT COL1
     FROM T1 WHERE COL1 > 10
```

WITH CHECK OPTION이 지정되어 있지 않으므로 삽입될 값이 보기의 탐색 조건에 맞지 않아도 다음의 INSERT문은 성공합니다.

```
INSERT INTO V1 VALUES (5)
```

WITH CASCADED CHECK OPTION을 지정하여 V1에 다른 보기를 작성합니다.

```
CREATE VIEW V2 AS SELECT COL1
      FROM V1 WITH CASCADED CHECK OPTION
```

아래 INSERT문은 V2 정의에 따르지 않는 행을 생성하기 때문에 실패합니다.

```
INSERT INTO V2 VALUES (5)
```

V2에 작성된 보기를 하나 더 고려합니다.

```
CREATE VIEW V3 AS SELECT COL1
     FROM V2 WHERE COL1 < 100
```

V3이 V2에 종속되고 V2에 WITH CASCADED CHECK OPTION이 있는 것만으로도 다음 INSERT문은 실패합니다.

```
INSERT INTO V3 VALUES (5)
```

그러나 V2 정의를 따르므로 다음 INSERT문은 성공합니다. V3에 WITH CASCADED CHECK OPTION이 없으므로 명령문이 V3 정의를 따르지 않는 것은 문제가 되지 않습니다.

```
INSERT INTO V3 VALUES (200)
```

WITH LOCAL CHECK OPTION

행을 갱신할 수 있으므로 보기를 통해 더 이상 행을 검색할 수 없다는 점을 제외하면 WITH LOCAL CHECK OPTION은 WITH CASCADED CHECK OPTION과 동일합니다. 이는 보기가 WITH CHECK OPTION절이 없는 것으로 정의된 보기에 직접 또는 간접적으로 종속될 때 발생할 수 있습니다.

예를 들어, 앞의 예에서 사용된 것과 같으며 갱신할 수 있는 보기를 고려합니다.

```
CREATE VIEW  V1 AS SELECT COL1
     FROM T1 WHERE COL1 > 10
```

이번에는 WITH LOCAL CHECK OPTION을 지정하여 V1에 두 번째 보기를 작성합니다.

```
CREATE VIEW V2 AS SELECT COL1
     FROM V1 WITH LOCAL CHECK OPTION
```

V2에 탐색 조건이 없으므로 이전의 CASCADED CHECK OPTION 예에서는 실패했던 동일한 INSERT가 성공하게 되며 V1이 체크 옵션을 지정하지 않으므로 V1의 탐색 조건은 체크하지 않아도 됩니다.

```
INSERT INTO V2 VALUES (5)
```

V2에 작성된 보기를 하나 더 고려합니다.

```
CREATE VIEW V3 AS SELECT COL1
FROM V2 WHERE COL1 < 100
```

V1의 탐색 조건이 V2에서 WITH LOCAL CHECK OPTION으로 인해 체크되지 않으므로 다음 INSERT는 역시 성공하게 되며 이전 예에서 WITH CASCADED CHECK OPTION도 이와 같습니다.

```
INSERT INTO V3 VALUES (5)
```

LOCAL 및 CASCADED CHECK OPTION간의 차이는 행이 갱신될 때 얼마나 많은 종속 보기의 탐색 조건이 체크되는지에 있습니다.

- WITH LOCAL CHECK OPTION은 행이 삽입되거나 갱신될 때 WITH LOCAL CHECK OPTION 또는 WITH CASCADED CHECK OPTION이 있는 종속 보기의 탐색 조건이 체크되도록 지정합니다.
- WITH CASCADED CHECK OPTION은 행이 삽입되거나 갱신될 때 모든 종속 보기의 탐색 조건이 체크되도록 지정합니다.

예: 케스케이드 체크 옵션

아래 표와 보기를 사용합니다.

```
CREATE TABLE T1 (COL1 CHAR(10))

CREATE VIEW  V1 AS SELECT COL1
      FROM T1 WHERE COL1 LIKE 'A%'

CREATE VIEW V2 AS SELECT COL1
      FROM V1 WHERE COL1 LIKE '%Z'
                           WITH LOCAL CHECK OPTION

CREATE VIEW V3 AS SELECT COL1
      FROM V2 WHERE COL1 LIKE 'AB%'

CREATE VIEW V4 AS SELECT COL1
      FROM V3 WHERE COL1 LIKE '%YZ'
                            WITH CASCADED CHECK OPTION

CREATE VIEW V5 AS SELECT COL1
      FROM V4 WHERE COL1 LIKE 'ABC%'
```

보기가 INSERT 또는 UPDATE를 사용하여 조작되는 다른 탐색 조건이 체크 지연될 것입니다.

- V1이 조작될 경우, V1에 지정된 WITH CHECK OPTION이 없으므로 어떤 조건도 체크되지 않습니다.
- V2가 조작될 경우,
  - COL1은 문자 Z로 끝나야 하며 문자 A로 시작할 필요가 없습니다. 이는 체크 옵션이 LOCAL이고 보기 V1에 지정된 체크 옵션이 없기 때문입니다.
- V3이 조작될 경우,
  - COL1은 문자 Z로 끝나야 하며 문자 A로 시작할 필요가 없습니다. V3에는 지정된 체크 옵션이 없으므로 소유하고 있는 탐색 조건이 일치되지 않습니다. 그러나 V3이 V2에서 정의되고 V2에는 체크 옵션이 있으므로 V2에 대한 탐색 조건이 체크되어야 합니다.
- V4가 조작될 경우,
  - COL1은 'AB'로 시작되고 'YZ'로 끝나야 합니다. V4에는 지정된 WITH CASCADED CHECK OPTION이 있으므로 V4가 종속되는 모든 보기에 대해 모든 탐색 조건이 체크되어야 합니다.
- V5가 조작될 경우,

– COL1은 ‘AB’로 시작되어야 하지만 반드시 ‘ABC’일 필요는 없습니다. 이는 V5가 체크 옵션을 지정하지 않기 때문이며 소유하고 있는 탐색 조건을 체크할 필요는 없습니다. 그러나 V5가 V4에서 정의되고 V4에는 중첩된 체크 옵션이 있으므로 V4, V3, V2 및 V1에 대한 모든 탐색 조건이 체크되어야 합니다. 즉, COL1은 ‘AB’로 시작하여 ‘YZ’로 끝나야 합니다.

V5가 WITH LOCAL CHECK OPTION을 작성할 경우, V5에서 조작은 COL1이 ‘ABC’로 시작되어 ‘YZ’로 끝나야 함을 의미합니다. LOCAL CHECK OPTION은 세 번째 문자가 ‘C’여야 한다는 추가 요건을 부가합니다.

iSeries용 DB2 UDB 트리거 지원

*트리거*는 지정된 변경 조작이 지정된 표에 대해 수행될 때 자동으로 실행되는 조치 세트입니다. 변경 조작은 SQL INSERT, UPDATE 또는 DELETE문이거나 어플리케이션 프로그램의 insert, update 또는 delete 고급 언어 명령문일 수 있습니다. 트리거는 업무 규칙 강제, 입력 자료 확인 및 감사 추적 보유 등의 업무에 유용합니다.

트리거는 두 개의 다른 방법으로 정의할 수 있습니다.

- 166 페이지의『SQL 트리거』
- 171 페이지의『외부 트리거』

외부 트리거의 경우, CRTPFTRG CL 명령이 사용됩니다. 트리거 조치 세트가 포함된 프로그램을 지원되는 고급 언어로 정의할 수 있습니다. 외부 트리거는 삽입, 갱신, 삭제 또는 읽기 트리거가 될 수 있습니다.

SQL 트리거의 경우, CREATE TRIGGER문이 사용됩니다. 트리거 프로그램은 SQL을 사용하여 전체적으로 정의됩니다. SQL 트리거는 삽입, 갱신 또는 삭제 트리거가 될 수 있습니다.

일단 트리거가 표와 연관되기만 하면 트리거 지원은 변경 조작이 표에 대해 개시되거나 논리 파일이나 보기가 표에 대해 작성될 때마다 트리거 프로그램을 호출합니다. SQL 트리거 및 외부 트리거를 동일한 표에 대해 정의할 수 있습니다. 최대 200개의 트리거를 단일 표에 대해 정의할 수 있습니다.

각각의 변경 조작은 변경 조작 발생 전후에 트리거를 호출할 수 있습니다. 또한 액세스되는 표를 매번 호출하는 *읽기* 트리거를 추가할 수 있습니다. 따라서, 하나의 표를 많은 트리거 유형과 연관시킬 수 있습니다.

- 이전 삭제 트리거
- 이전 삽입 트리거
- 이전 갱신 트리거
- 이후 삭제 트리거

- 이후 삽입 트리거
- 이후 갱신 트리거
- 읽기 전용 트리거(외부 트리거 전용)

SQL 표에 대해 정의할 수 있는 트리거 수와 최대 트리거 네스팅 레벨을 비롯한 트리거 한계에 대한 정보와 트리거 코드화 시의 권장사항 및 예방책은 데이터베이스 프로그래밍 책에서 ″데이터베이스에서 자동 이벤트 트리거″ 장을 참조하십시오.

SQL 트리거

SQL CREATE TRIGGER문은 삽입, 갱신 또는 삭제 조작이 수행될 때마다 데이터베이스 관리 시스템이 표 그룹을 활동적으로 제어, 모니터 및 관리할 수 있는 방법을 제공합니다. SQL 트리거에 지정된 명령문은 SQL 삽입, 갱신 또는 삭제 조작이 수행될 때마다 실행됩니다. SQL 트리거는 저장 프로시듀어나 사용자 정의 기능을 호출하여 트리거 실행 시에 추가 처리를 수행합니다.

저장 프로시듀어와 달리 SQL 트리거는 어플리케이션에서 직접 호출할 수 있습니다. 그 대신, SQL 트리거는 삽입, 갱신 또는 삭제 조작의 트리거 실행에 따라 데이터베이스 관리 시스템에서 호출합니다. SQL 트리거의 정의는 데이터베이스 관리 시스템에 저장되며 트리거가 정의된 SQL 표가 수정될 때 데이터베이스 관리 시스템에 의해 호출됩니다.

SQL 트리거 작성에 대한 자세한 내용은 『SQL 트리거 작성』을 참조하십시오.

CREATE TRIGGER문 사용에 대한 완벽한 내용은 SQL 참조서의 CREATE TRIGGER문 주제를 참조하십시오.

SQL 트리거 작성

SQL 트리거는 CREATE TRIGGER SQL문을 지정하여 작성할 수 있습니다. SQL 트리거의 일상 업무에서 명령문은 SQL에서 프로그램(*PGM) 오브젝트로 전송됩니다. 프로그램은 트리거 이름 규정자에 의해 지정된 스키마에 작성됩니다. 지정된 트리거는 SYSTRIGGERS, SYSTRIGDEP, SYSTRIGCOL 및 SYSTRIGUPD SQL 카탈로그에 등록됩니다. SQL 트리거의 변수 제어문 사용 방법에 대한 추가 정보와 SQL문 레벨에서 SQL 트리거 디버그 방법에 대한 정보는 SQL 참조서의 ″SQL 프로시듀어, 함수 및 트리거″ 장을 참조하십시오.

SQL 트리거 작성의 예와 고려사항은 다음을 참조하십시오.

- 167 페이지의 『BEFORE SQL 트리거』
- 168 페이지의 『AFTER SQL 트리거』
- 169 페이지의 『SQL 트리거의 핸들러』
- 170 페이지의 『SQL 트리거 변환 표』

BEFORE SQL 트리거

BEFORE 트리거는 표를 수정할 수 없지만 입력 열 값을 확인하고 표에서 삽입되었거나 갱신된 열 값을 수정하는 데에도 사용할 수 있습니다. 다음의 예에서 트리거는 목표 표에 행을 삽입하기 전에 회사의 회계 사분기를 설정하는 데 사용됩니다.

```
CREATE TABLE TransactionTable (DateOfTransaction DATE, FiscalQuarter SMALLINT)

CREATE TRIGGER TransactionBeforeTrigger BEFORE INSERT ON TransactionTable
REFERENCING NEW AS new_row
FOR EACH ROW MODE DB2ROW
BEGIN
  DECLARE newmonth SMALLINT;
SET newmonth = MONTH(new_row.DateOfTransaction);
  IF newmonth < 4 THEN
    SET new_row.FiscalQuarter=3;
  ELSEIF newmonth < 7 THEN
    SET new_row.FiscalQuarter=4;
  ELSEIF newmonth < 10 THEN
    SET new_row.FiscalQuarter=1;
  ELSE
    SET new_row.FiscalQuarter=2;
  END IF;
END
```

아래의 SQL 삽입 명령문의 경우, 현재 날짜가 2000년 11월 14일이면 ″FiscalQuarter″ 열이 2로 설정됩니다.

```
INSERT INTO TransactionTable(DateOfTransaction)
           VALUES(CURRENT DATE)
```

SQL 트리거는 사용자 정의 고유 유형(UDT)와 저장 프로시듀어에 대한 액세스 권한을 가지고 있으며 이를 사용할 수 있습니다. 다음의 예에서, SQL 트리거는 저장 프로시듀어를 호출하여 일부 사전정의 업무 논리를 실행하며 이 경우, 업무에 대한 사전정의 값으로 열을 설정합니다.

```
CREATE DISTINCT TYPE enginesize AS DECIMAL(5,2) WITH COMPARISONS

CREATE DISTINCT TYPE engineclass AS VARCHAR(25) WITH COMPARISONS

CREATE PROCEDURE SetEngineClass(IN SizeInLiters enginesize,
                                OUT CLASS engineclass)
LANGUAGE SQL CONTAINS SQL
BEGIN
  IF SizeInLiters<2.0 THEN
    SET CLASS = 'Mouse';
  ELSEIF SizeInLiters<3.1 THEN
    SET CLASS ='Economy Class';
  ELSEIF SizeInLiters<4.0 THEN
    SET CLASS ='Most Common Class';
  ELSEIF SizeInLiters<4.6 THEN
    SET CLASS = 'Getting Expensive';
  ELSE
    SET CLASS ='Stop Often for Fillups';
  END IF;
END
```

```
CREATE TABLE EngineRatings (VariousSizes enginesize, ClassRating engineclass)

CREATE TRIGGER SetEngineClassTrigger BEFORE INSERT ON EngineRatings
REFERENCING NEW AS new_row
FOR EACH ROW MODE DB2ROW
  CALL SetEngineClass(new_row.VariousSizes, new_row.ClassRating)
```

아래의 SQL 삽입 명령문의 경우, ″VariousSizes″ 열의 값이 3.0이면 ″ClassRating″ 열은 ″Economy Class″로 설정됩니다.

```
INSERT INTO EngineRatings(VariousSizes) VALUES(3.0)
```

SQL에서는 SQL 트리거를 작성하기 전에 모든 표, 사용자 정의 기능, 프로시듀어 및 사용자 정의 유형이 존재해야 합니다. 위의 예에서 모든 표, 저장 프로시듀어 및 사용자 정의 유형은 트리거가 작성되기 전에 정의됩니다.

AFTER SQL 트리거

SQL 트리거에서 WHEN 조건을 사용하여 조건을 지정할 수 있습니다. 조건이 참으로 평가하면, SQL 트리거 루틴 본문의 SQL문이 실행됩니다. 조건이 거짓으로 평가하면, SQL 트리거 루틴 본문의 SQL문이 실행되지 않으며 제어가 데이터베이스 시스템으로 반환됩니다. 다음의 예에서, 트리거가 활성화되면 트리거 루틴 본문의 명령문을 실행해야 하는지의 여부를 판별하기 위해 조회가 평가됩니다.

```
CREATE TABLE TodaysRecords(TodaysMaxBarometricPressure FLOAT,
  TodaysMinBarometricPressure FLOAT)

CREATE TABLE OurCitysRecords(RecordMaxBarometricPressure FLOAT,
  RecordMinBarometricPressure FLOAT)

CREATE TRIGGER UpdateMaxPressureTrigger
AFTER UPDATE OF TodaysMaxBarometricPressure ON TodaysRecords
REFERENCING NEW AS new_row
FOR EACH ROW MODE DB2ROW
WHEN (new_row.TodaysMaxBarometricPressure>
     (SELECT MAX(RecordMaxBarometricPressure) FROM
     OurCitysRecords))
  UPDATE OurCitysRecords
         SET RecordMaxBarometricPressure =
             new_row.TodaysMaxBarometricPressure

CREATE TRIGGER UpdateMinPressureTrigger
AFTER UPDATE OF TodaysMinBarometricPressure
ON TodaysRecords
REFERENCING NEW AS new_row
FOR EACH ROW MODE DB2ROW
WHEN(new_row.TodaysMinBarometricPressure<
     (SELECT MIN(RecordMinBarometricPressure) FROM
     OurCitysRecords))
  UPDATE OurCitysRecords
         SET RecordMinBarometricPressure =
             new_row.TodaysMinBarometricPressure
```

먼저 현재 값이 표에 대해 초기화됩니다.

```
INSERT INTO TodaysRecords VALUES(0.0,0.0)
INSERT INTO OurCitysRecords VALUES(0.0,0.0)
```

아래의 SQL 갱신 명령문의 경우, OurCitysRecords의 RecordMaxBarometricPressure는 UpdateMaxPressureTrigger에 의해 갱신됩니다.

```
UPDATE TodaysRecords SET TodaysMaxBarometricPressure = 29.95
```

그러나 추후에는 TodaysMaxBarometricPressure가 단지 29.91이면 RecordMaxBarometricPressure가 갱신되지 않습니다.

```
UPDATE TodaysRecords SET TodaysMaxBarometricPressure = 29.91
```

SQL은 단일 트리거 조치에 대해 여러 트리거를 정의할 수 있도록 합니다. 이전 예에서, 두 개의 AFTER UPDATE 트리거인 UpdateMaxPressureTrigger와 UpdateMinPressureTrigger가 있습니다. 이러한 트리거들은 표 TodaysRecords의 특정 열이 갱신될 때에만 활성화됩니다.

AFTER 트리거는 표를 수정할 수 있습니다. 위의 예에서, UPDATE 조작은 두 번째 표에 적용됩니다. 순환 삽입 및 갱신 조작은 피해야 한다는 점을 유의하십시오. 데이터베이스 관리 시스템은 최대 트리거 네스팅 레벨에 도달하면 조작을 종료합니다. 삽입 또는 갱신 조작이 최대 네스팅 레벨에 도달하기 전에 종료되도록 조건부 논리를 추가하여 순환을 방지할 수 있습니다. 트리거의 네트워크를 통해 반복적으로 연속되는 트리거의 네트워크에서 동일한 상황을 방지해야 합니다.

SQL 트리거의 핸들러

SQL 트리거의 핸들러는 SQL 트리거에 오류로부터 회복하거나 트리거 루틴 본문의 SQL 문을 실행하는 동안에 발생한 오류에 대한 정보를 기록하는 능력을 부여합니다.

다음의 예에서는 두 개의 핸들러가 정의되어 있는데, 하나는 넘침 상태를 처리하기 위한 것이고 두 번째 핸들러는 SQL 예외를 처리하기 위한 것입니다.

```
CREATE TABLE ExcessInventory(Description VARCHAR(50), ItemWeight SMALLINT)
CREATE TABLE YearToDateTotals(TotalWeight SMALLINT)


CREATE TABLE FailureLog(Item VARCHAR(50), ErrorMessage VARCHAR(50), ErrorCode INT)

CREATE TRIGGER InventoryDeleteTrigger
AFTER DELETE ON ExcessInventory
REFERENCING OLD AS old_row
FOR EACH ROW MODE DB2ROW
BEGIN
  DECLARE sqlcode INT;
  DECLARE invalid_number condition FOR '22003';
  DECLARE exit handler FOR invalid_number
  INSERT INTO FailureLog VALUES(old_row.Description,
        'Overflow occurred in YearToDateTotals', sqlcode);
  DECLARE exit handler FOR sqlexception
  INSERT INTO FailureLog VALUES(old_row.Description,
```

```
        'SQL Error occurred in InventoryDeleteTrigger', sqlcode);
  UPDATE YearToDateTotals SET TotalWeight=TotalWeight +
        old_row.itemWeight;
END
```

먼저, 표의 현재 값이 초기화됩니다.

```
INSERT INTO ExcessInventory VALUES('Desks',32500)
INSERT INTO ExcessInventory VALUES('Chairs',500)
INSERT INTO YearToDateTotals VALUES(0)
```

아래의 첫 번째 SQL 삭제 명령문이 실행되면, 품목 ″Desks″에 대한 ItemWeight가 표 YearToDateTotals의 TotalWeight에 대한 열 합계에 더해집니다. 두 번째 SQL 삭제 명령문이 실행되면, 열이 최대 32767까지의 값만을 처리하므로 품목 ″Chairs″에 대한 ItemWeight가 TotalWeight의 열 합계에 더해져서 넘침이 발생합니다. 넘침이 발생하면, invalid_number 나감 핸들러가 실행되며 FailureLog 표에 행이 기록됩니다. 예를 들어, YearToDateTotals 표가 실수로 삭제된 경우, sqlexception 나감 핸들러가 실행됩니다. 이 예에서는 나중에 문제점을 진단할 수 있도록 핸들러를 사용하여 기록부를 작성합니다.

```
DELETE FROM ExcessInventory WHERE Description='Desks'
DELETE FROM ExcessInventory WHERE Description='Chairs'
```

SQL 트리거 변환 표

SQL 트리거는 SQL 삽입, 갱신 또는 삭제 조작에 대해 영향을 받은 모든 행을 참조해야 할 수 있습니다. 이 점은 예를 들어, 트리거가 영향을 받은 행의 특정 열에 MIN 또는 MAX와 같은 총계 함수를 적용해야 하는 경우에 필수적입니다. OLD_TABLE 및 NEW_TABLE 변환 표를 이와 같은 용도로 사용할 수 있습니다. 다음의 예에서, 트리거는 총계 함수 MAX를 표 StudentProfiles의 영향을 받은 모든 행에 적용합니다.

```
CREATE TABLE StudentProfiles(StudentsName VARCHAR(125),
      StudentsYearInSchool SMALLINT, StudentsGPA DECIMAL(5,2))CREATE TABLE CollegeBoundStudentsProfile
      (YearInSchoolMin SMALLINT, YearInSchoolMax SMALLINT, StudentGPAMin
      DECIMAL(5,2), StudentGPAMax DECIMAL(5,2))

CREATE TRIGGER UpdateCollegeBoundStudentsProfileTrigger
AFTER UPDATE ON StudentProfiles
REFERENCING NEW_TABLE AS ntable
FOR EACH STATEMENT MODE DB2SQL
BEGIN
  DECLARE maxStudentYearInSchool SMALLINT;
  SET maxStudentYearInSchool =
       (SELECT MAX(StudentsYearInSchool) FROM ntable);
  IF maxStudentYearInSchool >
       (SELECT MAX (YearInSchoolMax) FROM
          CollegeBoundStudentsProfile) THEN
    UPDATE CollegeBoundStudentsProfile SET YearInSchoolMax =
         maxStudentYearInSchool;
  END IF;
END
```

앞의 예에서, 트리거는 트리거 갱신 명령문의 실행 이후에 한 번 실행되는데, FOR EACH STATEMENT 트리거로 정의되어 있기 때문입니다. 변환 표를 참조하는 트리거를 정의할 때 변환 표 수록을 위해 데이터베이스 관리 시스템에 필요한 오버헤드의 처리를 고려해야 합니다.

외부 트리거

외부 트리거의 경우, 트리거 조치 세트가 들어 있는 프로그램을 *PGM 오브젝트를 작성하는 지원되는 고급 언어로 정의할 수 있습니다. 트리거 프로그램은 그 속에 삽입된 SQL을 가질 수 있습니다. 시스템 트리거를 정의하려면, 트리거 프로그램을 작성하고 ADDPFTRG CL 명령을 사용하여 이를 표에 추가해야 하거나 iSeries Navigator를 사용하여 이를 추가할 수 있습니다. 표에 트리거를 추가하려면, 다음과 같이 수행해야 합니다.

- 표 식별
- 조작의 종류 식별
- 필요한 조치를 수행하는 프로그램 식별

외부 트리거의 예는 『외부 트리거 예 프로그램』을 참조하십시오.

외부 트리거 예 프로그램

샘플 외부 트리거 프로그램은 다음과 같습니다. 이 프로그램들은 삽입된 SQL이 있는 ILE C로 작성된 것입니다.

iSeries용 DB2 UDB에서의 외부 트리거 사용에 대한 완전한 논의 및 추가 예는 데이터베이스 프로그래밍의 ″데이터베이스에서 자동 이벤트 트리거″ 장을 참조하십시오.

주: 코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

```
#include "string.h"
#include "stdlib.h"
#include "stdio.h"
#include <recio.h>
#include <xxcvt.h>
#include "qsysinc/h/trgbuf"       /* Trigger input parameter         */
#include "lib1/csrc/msghand1"     /* User defined message handler    */
/*****************************************************************/
/* This is a trigger program which is called whenever there is an    */
/* update to the EMPLOYEE table. If the employee's commission is     */
/* greater than the maximum commission, this trigger program will    */
/* increase the employee's salary by 1.04 percent and insert into    */
/* the RAISE table.                                                  */
/*                                                                   */
/* The EMPLOYEE record information is passed from the input parameter*/
/* to this trigger program.                                          */
/*****************************************************************/

Qdb_Trigger_Buffer_t *hstruct;
char *datapt;

   /*******************************************************/
   /* Structure of the EMPLOYEE record which is used to   */
   /* store the old or the new record that is passed to   */
   /* this trigger program.                               */
   /*                                                     */
   /* Note : You must ensure that all the numeric fields  */
   /*        are aligned at 4 byte boundary in C.         */
   /*        Used either Packed struct or filler to reach */
   /*        the byte boundary alignment.                 */
   /*******************************************************/

_Packed struct rec{
         char  empn[6];
  _Packed struct { short fstlen ;
                   char  fstnam[12];
                 } fstname;
         char  minit[1];
_Packed struct { short lstlen;
                 char lstnam[15];
               } lstname;
       char  dept[3];
       char  phone[4];
       char  hdate[10];
       char  jobn[8];
       short edclvl;
       char  sex1[1];
       char  bdate[10];
       decimal(9,2) salary1;
       decimal(9,2) bonus1;
       decimal(9,2) comm1;
       } oldbuf, newbuf;
 EXEC SQL INCLUDE SQLCA;
```

그림 2. 샘플 트리거 프로그램 (1/5)

```
main(int argc, char **argv)
{
int i;
int obufoff;                        /* old buffer offset            */
int nuloff;                         /* old null byte map offset     */
int nbufoff;                        /* new buffer offset            */
int nul2off;                        /* new null byte map offset     */
short work_days  = 253;             /* work days during in one year */
decimal(9,2) commission = 2000.00; /* cutoff to qualify for        */
decimal(9,2) percentage = 1.04;    /* raised salary as percentage  */
char raise_date[12] = "1982-06-01";/* effective raise date         */


struct {
       char  empno[6];
       char  name[30];
       decimal(9,2) salary;
       decimal(9,2) new_salary;
       } rpt1;

   /*******************************************************/
   /* Start to monitor any exception.                     */
   /*******************************************************/

  _FEEDBACK fc;
  _HDLR_ENTRY hdlr = main_handler;
                             /***************************************/
                             /* Make the exception handler active.  */
                             /***************************************/
  CEEHDLR(&hdlr, NULL, &fc);
                             /***************************************/
                             /* Ensure exception handler OK         */
                             /***************************************/
  if (fc.MsgNo != CEE0000)
    {
      printf("Failed to register exception handler.\n");
      exit(99);
    };

   /*******************************************************/
   /* Move the data from the trigger buffer to the local  */
   /* structure for reference.                            */
   /*******************************************************/

hstruct = (Qdb_Trigger_Buffer_t *)argv[1];
datapt  = (char *) hstruct;

obufoff = hstruct ->Old_Record_Offset;       /* old buffer   */
memcpy(&oldbuf,datapt+obufoff,; hstruct->Old_Record_Len);

nbufoff = hstruct ->New_Record_Offset;       /* new buffer   */
memcpy(&newbuf,datapt+nbufoff,; hstruct->New_Record_Len);
```

그림 2. 샘플 트리거 프로그램 (2/5)

```
EXEC SQL WHENEVER SQLERROR  GO TO ERR_EXIT;

   /*****************************************************/
   /* Set the transaction isolation level to the same as  */
   /* the application based on the input parameter in the */
   /* trigger buffer.                                     */
   /*****************************************************/

if(strcmp(hstruct->Commit_Lock_Level,"0") == 0)
     EXEC SQL SET TRANSACTION ISOLATION LEVEL NONE;
else{
  if(strcmp(hstruct->Commit_Lock_Level,"1") == 0)
       EXEC SQL SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED, READ
                 WRITE;
  else {
     if(strcmp(hstruct->Commit_Lock_Level,"2") == 0)
       EXEC SQL SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
  else
     if(strcmp(hstruct->Commit_Lock_Level,"3") == 0)
       EXEC SQL SET TRANSACTION ISOLATION LEVEL ALL;
  }
}

 /*****************************************************/
 /* If the employee's commission is greater than maximum */
 /* commission, then increase the employee's salary      */
 /* by 1.04 percent and insert into the RAISE table.     */
 /*****************************************************/

if (newbuf.comm1 >= commission)
{
  EXEC SQL  SELECT EMPNO, EMPNAME, SALARY
          INTO :rpt1.empno, :rpt1.name, :rpt1.salary
          FROM TRGPERF/EMP_ACT
          WHERE EMP_ACT.EMPNO=:newbuf.empn ;

  if (sqlca.sqlcode == 0) then
  {
     rpt1.new_salary = salary * percentage;
     EXEC SQL INSERT INTO TRGPERF/RAISE VALUES(:rpt1);
  }
  goto finished;
 }
 err_exit:
   exit(1);

  /* All done  */
 finished:
return;
}  /* end of main line  */
```

그림 2. 샘플 트리거 프로그램 (3/5)

```
/*****************************************************************/
/*   INCLUDE NAME : MSGHAND1                                     */
/*                                                               */
/*   DESCRIPTION  : Message handler to signal an exception to    */
/*                  the application to inform that an            */
/*                  error occured in the trigger program.        */
/*                                                               */
/*   NOTE : This message handler is a user defined routine.      */
/*                                                               */
/*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
#include <leawi.h>

#pragma linkage (QMHSNDPM, OS)
void QMHSNDPM(char *,             /* Message identifier             */
              void *,             /* Qualified message file name    */
              void *,             /* Message data or text           */
              int,                /* Length of message data or text */
              char *,             /* Message type                   */
              char *,             /* Call message queue             */
              int,                /* Call stack counter             */
              void *,             /* Message key                    */
              void *,             /* Error code                     */
              ...);               /* Optionals:
                                       length of call message queue
                                        name
                                       Call stack entry qualification
                                       display external messages
                                        screen wait time           */
/*****************************************************************/
/********  This is the start of the exception handler function.     */
/*****************************************************************/
  void main_handler(_FEEDBACK *cond, _POINTER *token, _INT4 *rc,
                    _FEEDBACK *new)
   {
                           /*************************************/
                           /* Initialize variables for call to  */
                           /* QMHSNDPM.                         */
                           /* User must create a message file and */
                           /* define a message ID to match the  */
                           /* following data.                   */
                           /*************************************/
  char      message_id[7] = "TRG9999";
  char      message_file[20] = "MSGF      LIB1      ";
  char      message_data[50] = "Trigger error                " ;
  int       message_len = 30;
  char      message_type[10] = "*ESCAPE   ";
  char      message_q[10] = "_C_pep    ";
  int       pgm_stack_cnt = 1;
  char      message_key[4];
```

그림 2. 샘플 트리거 프로그램 (4/5)

```
                          /****************************************/
                          /* Declare error code structure for     */
                          /* QMHSNDPM.                            */
                          /****************************************/
struct error_code {
  int bytes_provided;
  int bytes_available;
  char message_id[7];
} error_code;

error_code.bytes_provided = 15;
                          /****************************************/
                          /* Set the error handler to resume and  */
                          /* mark the last escape message as      */
                          /* handled.                             */
                          /****************************************/
*rc = CEE_HDLR_RESUME;
                          /****************************************/
                          /* Send my own *ESCAPE message.         */
                          /****************************************/
QMHSNDPM(message_id,
        &message_file,
        &message_data,
         message_len,
         message_type,
         message_q,
         pgm_stack_cnt,
        &message_key,
        &error_code );
                          /****************************************/
                          /* Check that the call to QMHSNDPM      */
                          /* finished correctly.                  */
                          /****************************************/
if (error_code.bytes_available != 0)
    {
      printf("Error in QMHOVPM : %s\n", error_code.message_id);
    }
}
```

그림 2. 샘플 트리거 프로그램 (5/5)

제 11 장 저장 프로시듀어

프로시듀어(종종 저장된 프로시듀어라고 함)는 호스트 언어 명령문과 SQL문이 포함될 수 있는 조작을 수행하기 위해 호출될 수 있는 프로그램입니다. SQL의 프로시듀어는 호스트 언어의 프로시듀어와 비슷한 장점을 제공합니다.

iSeries용 DB2 SQL 저장 프로시듀어 지원은 SQL 어플리케이션이 SQL문을 통해 프로시듀어를 정의하고 호출할 수 있는 방법을 제공합니다. 저장 프로시듀어는 분산 및 비분산 iSeries용 DB2 SQL 어플리케이션에서 모두 사용할 수 있습니다. 저장 프로시듀어를 사용함으로써 얻는 큰 이점은 분산된 어플리케이션에 대한 것으로서, 어플리케이션 리퀘스터 또는 클라이언트상에서 하나의 CALL문 실행으로 어플리케이션 서버에서 어떤 *크기*의 작업이라도 수행할 수 있다는 것입니다.

프로시듀어는 SQL 프로시듀어 또는 외부 프로시듀어로 정의될 수 있습니다. 외부 프로시듀어는 지원되는 상위 단계 언어 프로그램(System/36* 프로그램과 프로시듀어 제외) 또는 REXX 프로시듀어일 수 있습니다. 이 프로시듀어가 SQL문을 포함해야 할 필요는 없으나 SQL문을 포함할 수도 있습니다. SQL 프로시듀어는 전적으로 SQL에 정의되며 SQL 제어문이 들어 있는 SQL문을 포함할 수 있습니다.

저장 프로시듀어를 코딩하기 위해서는 먼저 다음을 이해하고 있어야 합니다.

- CREATE PROCEDURE문을 통한 저장 프로시듀어 정의
- CALL문을 통한 저장 프로시듀어 호출
- 매개변수 전달 규칙
- 프로시듀어를 호출하는 프로그램에 완료 상태를 리턴시키는 방법

CREATE PROCEDURE문을 사용하면 저장 프로시듀어를 정의할 수 있습니다. CREATE PROCEDURE문은 프로시듀어와 매개변수 정의를 카탈로그 표 SYSROUTINES와 SYSPARMS에 추가합니다. 이러한 정의는 시스템에서 SQL CALL 문에 의해 액세스될 수 있습니다.

외부 프로시듀어 또는 SQL 프로시듀어를 작성하기 위해 SQL CREATE PROCEDURE 명령문을 사용할 수 있습니다. 또는, iSeries Navigator를 사용하여 프로시듀어를 정의할 수 있습니다.

다음 섹션에서는 저장 프로시듀어를 정의하고 호출하기 위해 사용된 SQL문, 저장 프로시듀어의 매개변수 전달에 관한 정보 및 저장 프로시듀어 사용의 예에 대해 설명합니다.

- 178 페이지의 『외부 프로시듀어 정의』
- 179 페이지의 『SQL 프로시듀어 정의』

- 185 페이지의『저장 프로시듀어 디버그』
- 186 페이지의『저장 프로시듀어 호출』
- 191 페이지의『저장 프로시듀어 및 UDF에 대한 매개변수 전달 규약』
- 196 페이지의『인디케이터 변수 및 저장 프로시듀어』
- 198 페이지의『완료 상태를 호출 프로그램에 리턴하는 방법』
- 199 페이지의『CALL문 예』

Java로 코딩된 저장된 프로시듀어에 대한 설명은 IBM Developer Kit for Java의 Java SQL 루틴을 참조하십시오.

DRDA와 함께 저장 프로시듀어를 사용하는 것에 대한 정보는 394 페이지의『DRDA 저장 프로시듀어 고려사항』을 참조하십시오.

주: 코드 예에 대한 정보는 x 페이지의『코드 면책사항 정보』를 참조하십시오.

외부 프로시듀어 정의

외부 프로시듀어에 대한 CREATE PROCEDURE문은 다음을 수행합니다.

- 프로시듀어를 명명합니다.
- 매개변수 및 속성을 정의합니다.
- 프로시듀어 호출시 시스템이 사용하는 프로시듀어에 관한 기타 정보를 제공합니다.

다음 예를 고려합니다.

```
CREATE PROCEDURE P1
         (INOUT PARM1 CHAR(10))
         EXTERNAL NAME MYLIB.PROC1
              LANGUAGE C
         GENERAL WITH NULLS;
```

이 CREATE PROCEDURE문은 다음을 수행합니다.

- 프로시듀어 P1을 명명합니다.
- 입력 매개변수 및 출력 매개변수로 모두 사용될 수 있는 하나의 매개변수를 정의합니다. 이 매개변수는 길이가 10인 문자 필드입니다. 매개변수는 유형 IN, OUT 또는 INOUT으로 정의될 수 있습니다. 매개변수 유형은 매개변수 값이 프로시듀어간에 언제 전달될 것인지를 결정합니다.
- MYLIB에서 PROC1인 프로시듀어에 해당되는 프로그램의 이름을 정의합니다. MYLIB.PROC1은 프로시듀어가 CALL문에서 호출될 때 호출되는 프로그램입니다.
- 프로시듀어 P1(프로그램 MYLIB.PROC1)이 C로 작성됨을 나타냅니다. 언어는 전달될 수 있는 매개변수의 유형에 영향을 미치므로 중요합니다. 이는 또한 매개변수가 프로시듀어로 전달되는 방법에도 영향을 미칩니다(예를 들어 ILE C 프로시듀어의 경우, NUL 종료자는 문자, 그래픽, 날짜, 시간 및 시간소인 매개변수에 전달됨).

- CALL 유형이 SIMPLE CALL WITH NULLS가 되도록 정의합니다. 이는 프로시듀어에 대한 매개변수가 NULL 값을 포함할 수 있으므로 CALL문에서 프로시듀어로 추가 인수가 전달되기 원함을 나타냅니다. 추가 인수는 N 단정수의 배열로서 여기서 N은 CREATE PROCEDURE문에 선언된 매개변수의 개수입니다. 이 예에는 매개변수만 있으므로 배열은 단지 하나의 요소만을 포함합니다.

호출하기 위해 프로시듀어를 정의할 필요가 없음을 주목해야 합니다. 그러나 프로시듀어 정의를 찾을 수 없는 경우, CALL문에서 프로시듀어가 호출될 때 이 프로그램의 CREATE PROCEDURE나 DECLARE PROCEDURE로부터 특정 제한사항이나 가정이 만들어집니다. 예를 들어 NULL 인디케이터는 전달될 수 없습니다. 187 페이지의 『프로시듀어 정의가 없는 삽입 CALL문 사용』에는 해당 프로시듀어 정의가 없는 CALL문의 예가 나옵니다.

SQL 프로시듀어 정의

SQL 프로시듀어에 대한 CREATE PROCEDURE문은 다음을 수행합니다.

- 프로시듀어를 명명합니다.
- 매개변수 및 속성을 정의합니다.
- 프로시듀어 호출시 사용될 프로시듀어에 관한 기타 정보를 제공합니다.
- 프로시듀어 본문을 정의합니다. 프로시듀어 본문은 프로시듀어의 실행 가능 부분으로 하나의 SQL문입니다.

사원 번호와 등급을 입력항목으로 처리하여 사원의 급여를 갱신하는 다음의 간단한 예를 보겠습니다.

```
CREATE PROCEDURE UPDATE_SALARY_1
  (IN EMPLOYEE_NUMBER CHAR(10),
   IN RATE DECIMAL(6,2))
   LANGUAGE SQL MODIFIES SQL DATA
      UPDATE CORPDATA.EMPLOYEE
     SET SALARY = SALARY * RATE
        WHERE EMPNO = EMPLOYEE_NUMBER;
```

이 CREATE PROCEDURE문은 다음을 수행합니다.

- 프로시듀어 UPDATE_SALARY_1을 명명합니다.
- 입력 매개변수인 길이가 6인 문자 자료 유형의 EMPLOYEE_NUMBER 매개변수와 입력 매개변수인 십진 자료 유형의 RATE 매개변수를 정의합니다.
- 프로시듀어가 SQL 자료를 수정하는 SQL 프로시듀어임을 나타냅니다.
- 프로시듀어 본문을 하나의 UPDATE문으로 정의합니다. 프로시듀어 호출시 UPDATE문은 EMPLOYEE_NUMBER 및 RATE에 대해 전달된 값을 사용하여 실행됩니다.

단일 UPDATE문 대신, 논리가 SQL 제어문을 사용하여 SQL 프로시듀어에 추가될 수 있습니다. SQL 제어문은 다음으로 구성됩니다.

- 지정문
- CALL문
- CASE문
- 복합 명령문
- FOR문
- GET DIAGNOSTICS문
- GOTO문
- IF문
- ITERATE문
- LEAVE문
- LOOP문
- REPEAT문
- RESIGNAL문
- RETURN문
- SIGNAL문
- WHILE문

다음은 사원 번호와 최종 평가로 얻어진 등급을 입력항목으로 처리하는 예입니다. 이 프로시듀어는 CASE문을 사용하여 갱신을 위한 적절한 증가분과 상여금을 결정합니다.

```
CREATE PROCEDURE UPDATE_SALARY_2
        (IN EMPLOYEE_NUMBER CHAR(6),
         IN RATING INT)
         LANGUAGE SQL MODIFIES SQL DATA
        CASE RATING
          WHEN 1 THEN
            UPDATE CORPDATA.EMPLOYEE
              SET SALARY = SALARY * 1.10,
              BONUS = 1000
              WHERE EMPNO = EMPLOYEE_NUMBER;
          WHEN 2 THEN
            UPDATE CORPDATA.EMPLOYEE
              SET SALARY = SALARY * 1.05,
              BONUS = 500
              WHERE EMPNO = EMPLOYEE_NUMBER;
          ELSE
            UPDATE CORPDATA.EMPLOYEE
              SET SALARY = SALARY * 1.03,
              BONUS = 0
              WHERE EMPNO = EMPLOYEE_NUMBER;
         END CASE;
```

이 CREATE PROCEDURE문은 다음을 수행합니다.

- 프로시듀어 UPDATE_SALARY_2를 명명합니다.

- 입력 매개변수인 길이가 6인 문자 자료 유형의 EMPLOYEE_NUMBER 매개변수와 입력 매개변수인 정수 자료 유형의 RATING 매개변수를 정의합니다.
- 프로시듀어가 SQL 자료를 수정하는 SQL 프로시듀어임을 나타냅니다.
- 프로시듀어 본문을 정의합니다. 프로시듀어 호출시 입력 매개변수 RATING이 체크되고 적절한 갱신 명령문이 실행됩니다.

복합 명령문을 추가함으로써 여러 명령문을 프로시듀어 본문에 추가할 수 있습니다. 하나의 복합 명령문 안에는 원하는 만큼의 SQL문을 지정할 수 있습니다. 또한 SQL 변수, 커서 및 처리자를 선언할 수 있습니다.

다음 예는 입력으로 DEPT_NUMBER를 선택합니다. 이 예는 해당 부서 내의 모든 직원의 총 급여와 해당 부서 내에서 상여금을 받은 직원 수를 리턴합니다.

```
CREATE PROCEDURE RETURN_DEPT_SALARY
         (IN DEPT_NUMBER CHAR(3),
          OUT DEPT_SALARY DECIMAL(15,2),
          OUT DEPT_BONUS_CNT INT)
         LANGUAGE SQL READS SQL DATA
         P1: BEGIN
          DECLARE EMPLOYEE_SALARY DECIMAL(9,2);
          DECLARE EMPLOYEE_BONUS DECIMAL(9,2);
          DECLARE TOTAL_SALARY DECIMAL(15,2)DEFAULT 0DECLARE BONUS_CNT INT DEFAULT 0;
          DECLARE END_TABLE INT DEFAULT 0;
          DECLARE C1 CURSOR FOR
           SELECT SALARY, BONUS FROM CORPDATA.EMPLOYEE
            WHERE WORKDEPT = DEPT_NUMBER;
          DECLARE CONTINUE HANDLER FOR NOT FOUND
            SET END_TABLE = 1;
          DECLARE EXIT HANDLER FOR SQLEXCEPTION
            SET DEPT_SALARY = NULL;
          OPEN C1;
             FETCH C1 INTO EMPLOYEE_SALARY, EMPLOYEE_BONUS;
          WHILE END_TABLE = 0 DO
             SET TOTAL_SALARY = TOTAL_SALARY + EMPLOYEE_SALARY + EMPLOYEE_BONUS;
             IF EMPLOYEE_BONUS > 0 THEN
               SET BONUS_CNT = BONUS_CNT + 1;
               END IF;
             FETCH C1 INTO EMPLOYEE_SALARY, EMPLOYEE_BONUS;
           END WHILE;
           CLOSE C1;
           SET DEPT_SALARY = TOTAL_SALARY;
           SET DEPT_BONUS_CNT = BONUS_CNT;
         END P1;
```

이 CREATE PROCEDURE문은 다음을 수행합니다.

- 프로시듀어 RETURN_DEPT_SALARY를 명명합니다.
- 입력 매개변수이고 길이가 3인 문자 자료 유형의 매개변수 DEPT_NUMBER, 입력 매개변수인 십진 자료 유형의 매개변수 DEPT_SALARY 그리고 출력 매개변수인 정수 자료 유형의 매개변수 DEPT_BONUS_CNT를 정의합니다.
- 프로시듀어가 SQL 자료를 읽는 SQL 프로시듀어임을 나타냅니다.
- 프로시듀어 본문을 정의합니다.

- SQL 변수 EMPLOYEE_SALARY와 TOTAL_SALARY를 십진 필드로 선언합니다.
- 정수이고 0으로 초기설정된 SQL 변수 BONUS_CNT와 END_TABLE을 선언합니다.
- EMPLOYEE 표에서 열을 선택하는 커서 C1을 선언합니다.
- 변수 END-TABLE을 1로 설정할 때 NOT FOUND에 대한 계속 처리자를 선언합니다. 이 처리자는 FETCH에 리턴된 행이 더 이상 없는 경우에 호출됩니다. 처리자 호출시 SQLCODE와 SQLSTATE는 1로 다시 초기설정됩니다.
- SQLEXCEPTION에 대한 나감 처리자를 선언합니다. 호출시 DEPT_SALARY는 널(null)로 설정되고 복합 명령문의 처리는 종료됩니다. 이 처리자는 오류 발생시 예를 들어 SQLSTATE 클래스가 '00', '01' 또는 '02'가 아닌 경우에 호출됩니다. 인디케이터는 항상 SQL 프로시듀어로 전달되므로 DEPT_SALARY에 대한 인디케이터 값은 프로시듀어 리턴시 -1입니다. 이 처리자가 호출되는 경우, SQLCODE와 SQLSTATE는 0으로 다시 초기설정됩니다.

  SQLEXCEPTION에 대한 처리자가 지정되지 않고 다른 처리자에서 처리되지 않는 오류가 발생하는 경우, 복합 명령문의 실행이 종료되고 오류가 SQLCA에 리턴됩니다. 인디케이터와 마찬가지로 SQLCA는 항상 SQL 프로시듀어로부터 리턴됩니다.
- 커서 C1의 OPEN, FETCH 및 CLOSE를 포함합니다. 커서의 CLOSE가 지정되지 않을 경우, 이 커서는 SET RESULT SETS가 CREATE PROCEDURE문에 지정되지 않았으므로 복합 명령문의 끝에서 닫힙니다.
- 최종 레코드까지 루프가 페치되는 WHILE문을 포함합니다. 검색되는 각각의 행에 대해 TOTAL_SALARY가 증가되고 직원의 상여금이 0보다 클 경우, BONUS_CNT가 증가됩니다.
- DEPT_SALARY와 DEPT_BONUS_CNT를 출력 매개변수로 리턴합니다.

복합 명령문은 아토믹이 될 수 있으므로 예상치 못한 오류가 발생하는 경우, 아토믹 명령문 안에서 명령문들이 롤백됩니다. 아토믹(atomic) 복합 명령문은 SAVEPOINTS를 사용하여 구현됩니다. 복합 명령문이 성공적인 경우, 트랜잭션은 확약됩니다. SAVEPOINTS 사용에 대한 자세한 정보는 354 페이지의 『저장점』을 참조하십시오.

다음 예는 입력으로 DEPT_NUMBER를 선택합니다. EMPLOYEE_BONUS 표가 존재하는지 확인한 후 해당 부서 내에서 상여금을 받은 모든 직원의 이름을 삽입합니다. 프로시듀어는 상여금을 받은 모든 직원의 총수를 리턴합니다.

```
CREATE PROCEDURE CREATE_BONUS_TABLE
        (IN DEPT_NUMBER CHAR(3),
         INOUT CNT INT)
         LANGUAGE SQL MODIFIES SQL DATA
          CS1: BEGIN ATOMIC
          DECLARE NAME VARCHAR(30) DEFAULT NULL;
          DECLARE CONTINUE HANDLER FOR SQLSTATE '42710'
             SELECT COUNT(*) INTO CNT
```

```
   FROM DATALIB.EMPLOYEE_BONUS;
DECLARE CONTINUE HANDLER FOR SQLSTATE '23505'
   SET  CNT = CNT - 1;
DECLARE UNDO HANDLER FOR SQLEXCEPTION
   SET CNT = NULL;
IF DEPT_NUMBER IS NOT NULL THEN
  CREATE TABLE DATALIB.EMPLOYEE_BONUS
       (FULLNAME VARCHAR(30),
        BONUS DECIMAL(10,2),
       PRIMARY KEY (FULLNAME));          FOR_1:FOR V1 AS C1 CURSOR FOR
    SELECT FIRSTNME, MIDINIT, LASTNAME, BONUS
       FROM CORPDATA.EMPLOYEE
       WHERE WORKDEPT = CREATE_BONUS_TABLE.DEPT_NUMBER
    DO
    IF BONUS > 0 THEN
      SET NAME = FIRSTNME CONCAT ' ' CONCAT
                 MIDINIT  CONCAT ' 'CONCAT LASTNAME;
      INSERT INTO DATALIB.EMPLOYEE_BONUS
         VALUES(CS1.NAME, FOR_1.BONUS);
      SET CNT = CNT + 1;
    END IF;
  END FOR FOR_1;
END IF;           END CS1
```

이 CREATE PROCEDURE문은 다음을 수행합니다.

- 프로시듀어 CREATE_BONUS_TABLE을 명명합니다.
- 입력 매개변수이고 길이가 3인 문자 자료 유형의 DEPT_NUMBER 매개변수와 입출력 매개변수인 정수 자료 유형의 CNT 매개변수를 정의합니다.
- 프로시듀어가 SQL 자료를 수정하는 SQL 프로시듀어임을 나타냅니다.
- 프로시듀어 본문을 정의합니다.
  - 가변 문자로서 SQL 변수 NAME을 선언합니다.
  - SQLSTATE 42710에 대한 계속 처리자를 선언합니다. EMPLOYEE_BONUS 표가 이미 존재하는 경우, 처리자가 호출되고 표 내의 레코드 수를 검색합니다. SQLCODE와 SQLSTATE는 0으로 재설정되고 FOR문으로 처리가 계속됩니다.
  - SQLSTATE 23505, 복제 키에 대한 계속 처리자를 선언합니다. 프로시듀어가 표에 이미 존재하는 이름을 삽입하려는 경우, 처리자가 호출되고 CNT가 감소됩니다. INSERT문 다음의 SET문에 대한 처리는 계속됩니다.
  - SQLEXCEPTION에 대한 UNDO 처리자를 선언합니다. 호출되는 경우, 이전의 명령문이 롤백되고 CNT는 0으로 설정되며 복합 명령문 이후의 처리는 계속됩니다. 이 경우 복합 명령문 다음에 명령문이 없으므로 프로시듀어는 리턴됩니다.
  - FOR문을 사용하여 커서 C1을 선언하고 EMPLOYEE 표에서 레코드를 읽습니다. FOR문 내에서 선택 리스트의 열 이름은 페치된 행의 자료를 포함하는 SQL 변수로 사용됩니다. 각 행에 대해 열 FIRSTNME, MIDINIT 및 LASTNAME의 자료는 사이에 공백을 포함시켜 결합되고 그 결과는 SQL 변수 NAME에 놓입니다. SQL 변수 NAME과 BONUS는 EMPLOYEE_BONUS 표에 삽입됩니다. 선택 리스트 항목의 자료 유형이 프로시듀어 작성시 알려져야 하므로 FOR문에 지정된 표는 프로시듀어 작성시 존재해야 합니다.

SQL 변수명은 그 변수명이 정의된 FOR문 또는 복합 명령문의 레이블 이름으로 규정될 수 있습니다. 이 예에서 FOR_1.BONUS는 선택된 각 행에 대한 열 BONUS의 값을 포함하는 SQL 변수를 참조합니다. CS1.NAME은 레이블 CS1로 시작되는 복합 명령문에 정의된 NAME 변수입니다. 매개변수명도 프로시듀어 명으로 규정될 수 있습니다. CREATE_BONUS_TABLE.DEPT_NUMBER는 프로시듀어 CREATE_BONUS_TABLE에 대한 DEPT_NUMBER 매개변수입니다. 규정되지 않은 SQL 변수명이 열 이름도 허용되는 SQL문에 사용되고 변수명이 열 이름과 같은 경우, 이 이름은 해당 열을 참조하는데 사용될 것입니다.

SQL 프로시듀어에 동적 SQL을 사용할 수도 있습니다. 다음의 예는 특정 부서의 사원 모두가 있는 표를 작성합니다. 부서 번호는 프로시듀어에 입력시 전달되고 표 이름에 연결됩니다.

```
CREATE PROCEDURE CREATE_DEPT_TABLE (IN P_DEPT CHAR(3))
        LANGUAGE SQL
   BEGIN
     DECLARE STMT CHAR(1000);
     DECLARE MESSAGE CHAR(20);
     DECLARE TABLE_NAME CHAR(30);
     DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
        SET MESSAGE = 'ok';
     SET TABLE_NAME = 'CORPDATA.DEPT_' CONCAT P_DEPT CONCAT '_T';
     SET STMT = 'DROP TABLE ' CONCAT TABLE_NAME;
     PREPARE S1 FROM STMT;
     EXECUTE S1;
        SET STMT = 'CREATE TABLE ' CONCAT TABLE_NAME CONCAT
       '( EMPNO CHAR(6) NOT NULL,
          FIRSTNME VARCHAR(12) NOT NULL,          MIDINIT CHAR(1) NOT NULL,
          LASTNAME CHAR(15) NOT NULL,
          SALARY DECIMAL(9,2))';
     PREPARE S2 FROM STMT;
     EXECUTE S2;
     SET STMT = 'INSERT INTO ' CONCAT TABLE_NAME CONCAT
       'SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, SALARY
         FROM CORPDATA.EMPLOYEE
         WHERE  WORKDEPT = ?';
     PREPARE S3 FROM STMT;
     EXECUTE S3 USING P_DEPT;
END;
```

이 CREATE PROCEDURE문은 다음을 수행합니다.

- 프로시듀어 CREATE_DEPT_TABLE을 명명합니다.
- 입력 매개변수이고 길이가 3인 문자 자료 유형인 P_DEPT 매개변수를 정의합니다.
- 프로시듀어가 SQL 프로시듀어임을 나타냅니다.
- 프로시듀어 본문을 정의합니다.
  - SQL 변수 STMT 및 SQL 변수 TABLE_NAME을 문자로 선언합니다.

- CONTINUE 처리자를 선언합니다. 프로시듀어는 표(이미 있는 경우)를 DROP하려 합니다. 표가 존재하지 않다면 처음 EXECUTE가 실패합니다. 처리자를 사용하여 처리 과정은 지속될 것입니다.
- 뒷부분이 '_T'인 매개변수 P_DEPT에 전달된 문자가 변수 TABLE_NAME 뒤에 오는 'DEPT_'에 설정합니다.
- 변수 STMT를 DROP문에 설정하고 이 명령문을 준비하여 실행합니다.
- 변수 STMT를 CREATE문에 설정하고 이 명령문을 준비하여 실행합니다.
- 변수 STMT를 INSERT문에 설정하고 이 명령문을 준비하여 실행합니다. 매개변수 마커는 where절에 지정됩니다. 이 명령문이 수행될 때 변수 P_DEPT는 USING 절에서 전달됩니다.

프로시듀어가 부서에 대한 전달 값 'D21'이라면 표 DEPT_D21_T가 작성되고 이 표는 부서 'D21'에 있는 모든 사원으로 초기설정됩니다.

저장 프로시듀어 디버그

Create SQL 프로시듀어, Create SQL 기능 또는 Create Trigger문에 SET OPTION DBGVIEW = *SOURCE를 지정하여 SQL문 레벨에서 생성된 프로그램이나 모듈을 디버그할 수 있습니다. 또한 RUNSQLSTM 명령에서 매개변수로 DBGVIEW (*SOURCE)를 지정하고 RUNSQLSTM 내의 모든 루틴에 적용할 수도 있습니다.

소스 보기는 시스템에 의해 원래 루틴 본체에서 QTEMP/QSQDSRC로 작성됩니다. 소스 보기는 프로그램이나 서비스 프로그램과 함께 저장되지 않습니다. 이는 디버그할 때 정지하는 위치에 대응하는 행 수대로 구분됩니다. 매개변수 및 변수 이름을 포함하여 텍스트는 대문자로 바뀝니다.

모든 변수 및 매개변수는 구조의 일부로 생성됩니다. 디버그 시 변수를 평가할 때 구조 이름이 사용되어야 합니다. 변수는 현재 레이블명에 의해 규정됩니다. 매개변수는 프로시듀어명 또는 기능명에 의해 규정됩니다. 트리거에서 전환 변수는 적절한 상관명에 의해 규정됩니다. 각 복합 명령문이나 FOR문에 대하여 레이블 이름을 지정하는 것이 강력하게 권장됩니다. 사용자가 지정하지 않으면 시스템에서 대신 생성됩니다. 그러면 변수를 평가하는 것이 거의 불가능해집니다. 모든 변수 및 매개변수는 대문자 이름으로 평가되어야 합니다. 또한 구조 이름도 평가할 수 있습니다. 그러면 구조 내의 모든 변수를 표시합니다. 변수 또는 매개변수에 널이 사용될 수 있는 경우 해당 변수 또는 매개변수에 대한 인디케이터는 구조에서 해당 변수나 매개변수 바로 다음에 나옵니다.

SQL 루틴이 C로 작성되었으므로 SQL 소스 디버그에 영향을 미치는 몇 가지 C에서의 제한사항이 있습니다. SQL 루틴 본체에 지정된 구분된 이름들은 C에서 지정할 수 없습니다. 이러한 이름을 위해 이름이 생성되는데, 이것이 다시 디버그 또는 평가를 어렵게 만듭니다. 문자 변수의 내용을 평가하려면 변수 이름 앞에 *를 지정하십시오.

시스템에서 대부분의 변수 및 매개변수 이름에 대한 인디케이터를 생성하므로 변수에 SQL 널 값이 들어 있는지 직접 확인할 방법이 없습니다. 변수 평가 시 인디케이터가 널 값을 표시하도록 설정되어 있어도 항상 값을 표시합니다.

핸들러가 호출되었는지 판별하기 위해서는 핸들러 내의 첫 번째 명령문에 중단점을 설정하십시오. 핸들러 내의 복합 명령문이나 FOR문에서 선언된 변수는 평가될 수 있습니다.

저장 프로시듀어 호출

SQL CALL문은 저장 프로시듀어를 호출합니다. CALL문에서 저장 프로시듀어명 및 모든 인수가 지정됩니다. 인수는 상수, 특수 레지스터 또는 호스트 변수일 수 있습니다. CALL문에 지정된 외부 저장 프로시듀어에는 해당되는 CREATE PROCEDURE문이 없어도 됩니다. SQL 프로시듀어로 작성된 프로그램은 CREATE PROCEDURE문에 지정된 프로시듀어명을 호출해서만 호출될 수 있습니다.

프로시듀어가 시스템 프로그램 오브젝트이더라도 CALL CL 명령 사용은 대개 프로시듀어 호출 작업을 수행하지 않습니다. CALL CL 명령은 입력 및 출력 매개변수를 맵핑시키기 위해 프로시듀어 정의를 사용하지 않을 뿐더러 프로시듀어의 매개변수 스타일을 사용하여 프로그램에 매개변수를 전달하지도 합니다.

iSeries용 DB2 SQL은 각 유형에 대해 다른 규칙을 가지고 있으므로 주소지정이 필요한 CALL문에는 3가지 유형이 있습니다. 다음과 같습니다.

- 프로시듀어 정의가 존재하는 삽입 또는 동적 CALL문
- 어떤 프로시듀어 정의도 존재하지 않는 삽입 CALL문
- 어떤 CREATE PROCEDURE도 존재하지 않는 동적 CALL문

주: 여기서 동적이라는 말은 다음을 나타냅니다.

- 동적으로 준비되고 실행되는 CALL문
- 대화식 환경에서 발행된 CALL문(예를 들면, STRSQL 또는 조회 관리자를 통해)
- EXECUTE IMMEDIATE문에서 실행된 CALL문

다음은 각 유형에 대한 설명입니다.

- 187 페이지의 『프로시듀어 정의가 있는 CALL문 사용』
- 187 페이지의 『프로시듀어 정의가 없는 삽입 CALL문 사용』
- 188 페이지의 『SQLDA가 있는 삽입 CALL문 사용』
- 189 페이지의 『CREATE PROCEDURE가 없는 동적 CALL문 사용』

프로시듀어 정의가 있는 CALL문 사용

이 유형의 CALL문은 DECLARE PROCEDURE 카탈로그 정의로부터 프로시듀어 및 인수 속성에 대한 모든 정보를 읽습니다. 다음의 PL/I 예에서는 표시된 CREATE PROCEDURE문에 맵핑하는 CALL문을 보여줍니다.

```
DCL HV1 CHAR(10);
DCL IND1 FIXED BIN(15);
   :
EXEC SQL CREATE P1 PROCEDURE
          (INOUT PARM1 CHAR(10))
          EXTERNAL NAME MYLIB.PROC1
          LANGUAGE C
          GENERAL WITH NULLS;
   :
EXEC SQL CALL P1 (:HV1 :IND1);
   :
```

이 CALL문이 호출될 때 프로그램 MYLIB/PROC1에 대한 호출이 이루어지고 두 개의 인수가 전달됩니다. 프로그램의 언어가 ILE C이므로 첫 번째 인수는 호스트 변수 HV1의 내용이 들어 있는 11개의 문자 길이를 갖는 C NUL 종료 스트링입니다. ILE C 프로시듀어를 호출할 때, iSeries용 DB2 SQL은 매개변수가 문자, 그래픽, 날짜, 시간 또는 시간소인 변수로 선언되는 경우 한 문자를 매개변수 선언에 추가한다는 점을 유의하십시오. 두 번째 인수는 인디케이터 배열입니다. 이 경우 CREATE PROCEDURE 문에는 하나의 매개변수만 있으므로 두 번째 인수가 하나의 짧은 정수입니다. 이 인수에는 프로시듀어에 대한 항목 가운데 인디케이터 변수 IND1의 내용이 들어 있습니다.

첫 번째 매개변수가 INOUT으로 선언되므로 SQL은 호스트 변수 HV1과 인디케이터 변수 IND1을 사용자 프로그램으로 리턴하기 전에 MYLIB.PROC1로부터 리턴되는 값으로 선언됩니다.

주:

1. CREATE PROCEDURE 및 CALL문에 지정된 프로시듀어명은 프로그램의 SQL 사전컴파일중에 둘간의 링크가 작성되도록 정확하게 일치해야 합니다.
2. CREATE PROCEDURE와 DECLARE PROCEDURE문이 존재하는 삽입 CALL 문의 경우에는 DECLARE PROCEDURE문이 사용됩니다.

프로시듀어 정의가 없는 삽입 CALL문 사용

해당되는 CREATE PROCEDURE문이 없는 정적 CALL문은 다음 규칙에 따라 처리됩니다.

- 모든 호스트 변수 인수는 INOUT 유형 매개변수로 취급됩니다.
- CALL 유형은 GENERAL입니다(인디케이터 인수가 전달되지 않습니다).
- 호출 프로그램은 CALL에 지정된 프로시듀어명 및 필요할 경우 명명 규칙에 기초하여 결정됩니다.

- 호출 프로그램의 언어는 프로그램에 관해 시스템으로부터 검색된 정보에 기초하여 결정됩니다.

예: 프로시듀어 정의가 없는 삽입 CALL문

주: 코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

다음은 프로시듀어 정의가 없는 삽입 CALL문의 PL/I 예입니다.

```
DCL HV2 CHAR(10);
   :
EXEC SQL CALL P2 (:HV2);
   :
```

CALL문이 호출되면, iSeries용 DB2 SQL은 표준 SQL 명명 규칙에 기초하여 프로그램을 찾으려고 시도합니다. 위의 예에서는 명명 옵션 *SYS(시스템 명명)가 사용되며 DFTRDBCOL 매개변수가 CRTSQLPLI 명령에 지정되지 않은 것으로 가정합니다. 이 경우 P2로 명명된 프로그램에 대한 라이브러리 리스트가 탐색됩니다. 호출 유형이 GENERAL이므로 인디케이터 변수에 대한 프로그램으로 추가 인수가 전달되지 않습니다.

주: CALL문이 실행될 때 인디케이터 변수가 CALL문에 지정되고 그 값이 0보다 적을 경우, 인디케이터를 프로시듀어로 전달할 방법이 없으므로 오류가 발생합니다.

프로그램 P2가 라이브러리 리스트에서 발견되었다고 가정할 때 호스트 변수 HV2의 내용이 CALL에서 프로그램으로 전달되며 P2로부터 리턴된 인수는 P2의 실행이 완료된 후 호스트 변수로 다시 맵핑됩니다.

SQLDA가 있는 삽입 CALL문 사용

어떤 유형의 삽입 CALL에서나 (프로시듀어 정의의 존재 여부와 관계없이) 다음의 예 C에서 설명하는 것처럼 매개변수 리스트가 아닌 SQLDA가 전달됩니다. 저장 프로시듀어가 2개의 매개변수 즉, 첫 번째로 SHORT INT 유형과 두 번째로 길이 4인 유형 CHAR를 기대하고 있다고 가정합니다.

```
#define SQLDA_HV_ENTRIES 2
#define SHORTINT 500
#define NUL_TERM_CHAR 460

        exec sql include sqlca;
        exec sql include sqlda;
...
typedef struct sqlda Sqlda;
typedef struct sqlda* Sqldap;
...
        main()
{
 Sqldap dap;
 short col1;
 char col2[4];
```

```
 int bc;
 dap = (Sqldap) malloc(bc=SQLDASIZE(SQLDA_HV_ENTRIES));
        /* SQLDASIZE is a macro defined in the sqlda include */
 col1 = 431;
 strcpy(col2,"abc");
 strncpy(dap->sqldaid,"SQLDA   ",8);
 dap->sqldabc = bc;         /* bc set in the malloc statement above */
 dap->sqln = SQLDA_HV_ENTRIES;
 dap->sqld = SQLDA_HV_ENTRIES;
 dap->sqlvar[0].sqltype = SHORTINT;
 dap->sqlvar[0].sqllen  = 2;
 dap->sqlvar[0].sqldata = (char*) &col1;
 dap->sqlvar[0].sqlname.length = 0;
 dap->sqlvar[1].sqltype = NUL_TERM_CHAR;
 dap->sqlvar[1].sqllen  = 4;
 dap->sqlvar[1].sqldata = col2;
 ...
 EXEC SQL CALL P1 USING DESCRIPTOR :*dap;
 ...
}
```

또한 호출 프로시듀어명이 호스트 변수에 저장되어 하드코드 프로시듀어명 대신 호스트 변수가 CALL문에 사용될 수도 있습니다. 예를 들면 다음과 같습니다.

```
...
        main()
{
 char proc_name[15];
 ...
 strcpy (proc_name, "MYLIB.P3");
 ...
 EXEC SQL CALL :proc_name ...;
 ...
}
```

위의 예에서 MYLIB.P3이 매개변수를 기대하고 있으면 USING DESCRIPTOR절과 함께 전달된 SQLDA나 매개변수 리스트가 이전 예에서와 같이 사용될 수도 있습니다.

프로시듀어명이 들어 있는 호스트 변수가 CALL문에 사용되고 CREATE PROCEDURE 카탈로그 정의가 존재하는 경우, 사용되지 않습니다. 프로시듀어명은 매개변수 마커로 지정될 수 없습니다.

저장 프로시듀어 호출 예는 이 장의 후반부에서 추가로 찾을 수 있습니다.

CREATE PROCEDURE가 없는 동적 CALL문 사용

다음 규칙은 CREATE PROCEDURE 정의가 없을 경우 동적 CALL문의 처리에 관한 것입니다.

- 모든 인수는 IN 유형 매개변수로 취급됩니다.
- CALL 유형은 GENERAL입니다(인디케이터 인수가 전달되지 않습니다).
- 호출 프로그램은 CALL 및 명명 규칙에서 지정된 프로시듀어명에 기초하여 결정됩니다.

- 호출 프로그램의 언어는 프로그램에 관해 시스템으로부터 검색된 정보에 기초하여 결정됩니다.

예: CREATE PROCEDURE가 없는 동적 CALL문

주: 코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

다음은 동적 CALL문의 C 예입니다.

```
char hv3[10],string[100];
    :
strcpy(string,"CALL MYLIB.P3 ('P3 TEST')");
EXEC SQL EXECUTE IMMEDIATE :string;
    :
```

이 예에서는 EXECUTE IMMEDIATE문을 통해 실행된 동적 CALL문을 보여줍니다. 'P3 TEST'가 들어 있는 문자 변수로서 전달된 하나의 매개변수로 프로그램 MYLIB.P3에 대한 호출이 이루어집니다.

이전의 예에서와 같이 CALL문을 실행하고 상수를 전달할 때 프로그램에서 예상되는 인수의 길이를 기억해야 합니다. 프로그램 MYLIB.P3이 5자로 이루어진 인수를 예상한 경우, 프로그램에 대해 예제에 지정된 상수의 마지막 2자가 없어집니다.

주: 이런 이유로 CALL문에서 호스트 변수를 사용하여 프로시듀어의 속성이 정확히 일치되고 문자가 유실되지 않도록 하는 것이 보다 안전합니다. 동적 SQL에 있어 PREPARE 및 EXECUTE문이 처리를 위해 사용될 경우, 호스트 변수가 CALL문 인수로 지정될 수 있습니다.

CALL문에 전달된 숫자 상수의 경우 다음 규칙이 적용됩니다.

- 모든 정수 상수는 단어 전체가 2진 정수로 전달됩니다.
- 모든 10진 상수는 팩 10진값으로 전달됩니다. 정밀도 및 소수 자릿수는 상수값에 기초하여 결정됩니다. 예를 들어 값 123.45는 팩 10진수(5,2)로 전달됩니다. 이와 같이 값 001.01 역시 각각 5와 2의 정밀도 및 소수 자릿수로 전달됩니다.
- 모든 부동 소수점 상수는 배정밀도 부동 소수점으로 전달됩니다.

동적 CALL문에 지정된 특수 레지스터는 다음과 같이 전달됩니다.

CURRENT DATE
ISO 형식의 10바이트 문자 스트링으로 전달됩니다.

CURRENT TIME
ISO 형식의 8바이트 문자 스트링으로 전달됩니다.

CURRENT TIMESTAMP
IBM SQL 형식의 26바이트 문자 스트링으로 전달됩니다.

CURRENT TIMEZONE
정밀도 6 및 소수 자릿수의 0의 팩 10진수로 전달됩니다.

CURRENT SCHEMA

128바이트의 가변 길이 문자 스트링으로 전달됩니다.

CURRENT SERVER

18바이트의 가변 길이 문자 스트링으로 전달됩니다.

USER

18바이트의 가변 길이 문자 스트링으로 전달됩니다.

CURRENT PATH

3483바이트의 가변 길이 문자 스트링으로 전달됩니다.

저장 프로시듀어 및 UDF에 대한 매개변수 전달 규약

CALL문은 모든 지원되는 호스트 언어 및 REXX 프로시듀어로 작성되어 있는 프로그램으로 인수를 전달할 수 있습니다. 각각의 언어는 각 언어에 조정되어 있는 다른 자료 유형을 지원합니다. SQL 자료 유형은 다음 표의 맨 왼쪽 열에 들어 있습니다. 이 행의 다른 열에는 이 자료 유형이 특정 언어에 대한 매개변수 유형으로 지원되는지의 여부를 나타내는 표시가 들어 있습니다. 이 열에 대시(-) 기호가 들어 있는 경우, 그 자료 유형은 이 언어에 대한 매개변수 유형으로 지원되지 않습니다. 호스트 변수 선언은 iSeries용 DB2 SQL이 이 자료 유형을 이 언어의 매개변수로 지원함을 나타냅니다. 이 선언은 호스트 변수가 프로시듀어에 의해 올바르게 수신되고 설정되도록 선언되어야 함을 나타냅니다. SQL 프로시듀어 호출시 모든 SQL 자료 유형이 지원되므로 표에는 열이 하나도 제공되지 않습니다.

자세한 내용은 호스트 언어용 SQL 프로그래밍 및 IBM Devekioer's Kit for Java의 Java SQL 루틴 절을 참조하십시오.

표 25. 매개변수의 자료 유형

SQL 자료 유형	**C 및 C++**	**CL**	**iSeries용 COBOL 및 iSeries용 ILE COBOL**
SMALLINT	짧은	-	PIC S9(4) BINARY
INTEGER	긴	-	PIC S9(9) BINARY
BIGINT	매우 긴	-	PIC S9(18) BINARY **주:** iSeries용 ILE COBOL에 대해서만 지원됩니다.
DECIMAL(p,s)	10진수(p,s)	TYPE(*DEC) LEN(p s)	PIC S9(p-s)V9(s) PACKED-DECIMAL **주:** 정밀도가 18보다 크지 않아야 합니다.
NUMERIC(p,s)	-	-	PIC S9(p-s)V9(s) DISPLAY SIGN LEADING SEPARATE **주:** 정밀도가 18보다 크지 않아야 합니다.

표 25. 매개변수의 자료 유형 (계속)

SQL 자료 유형	C 및 C++	CL	iSeries용 COBOL 및 iSeries용 ILE COBOL
REAL 또는 FLOAT(p)	부동	-	COMP-1 **주:** iSeries용 ILE COBOL에 대해서만 지원됩니다.
DOUBLE PRECISION, FLOAT 또는 FLOAT(p)	배정밀도	-	COMP-2 **주:** iSeries용 ILE COBOL에 대해서만 지원됩니다.
CHARACTER(n)	char ... [n+1]	TYPE(*CHAR) LEN(n)	PIC X(n)
VARCHAR(n)	char ... [n+1]	-	가변 길이 문자 스트링(호스트 언어에 의한 SQL 프로그래밍의 COBOL 장 참조)
VARCHAR(n) FOR BIT DATA	VARCHAR 구조화 양식(호스트 언어에 대한 SQL 프로그래밍의 C 참조)	-	가변 길이 문자 스트링(호스트 언어에 의한 SQL 프로그래밍의 COBOL 장 참조)
GRAPHIC(n)	wchar_t ... [n+1]	-	PIC G(n) DISPLAY-1 또는 PIC N(n) **주:** iSeries용 ILE COBOL에만 지원됩니다.
VARGRAPHIC(n)	VARGRAPHIC 구조화 형식(C 장 참조)	-	가변 길이 그래픽 스트링(호스트 언어에 의한 SQL 프로그래밍의 COBOL 장 참조). **주:** iSeries용 ILE COBOL에만 지원됩니다.
DATE	char ... [11]	TYPE(*CHAR) LEN(10)	PIC X(10) iSeries용 ILE COBOL의 경우만, FORMAT DATE.
TIME	char ... [9]	TYPE(*CHAR) LEN(8)	PIC X(8) iSeries용 ILE COBOL의 경우만, FORMAT TIME.
TIMESTAMP	char ... [27]	TYPE(*CHAR) LEN(26)	PIC X(26) iSeries용 ILE COBOL의 경우만, FORMAT TIMESTAMP.
CLOB	CLOB 구조화 양식(호스트 언어에 의한 SQL 프로그래밍에서 C 장 참조)	-	CLOB 구조화 양식(호스트 언어에 의한 SQL 프로그래밍에서 COBOL 장 참조) **주:** iSeries용 ILE COBOL에 대해서만 지원됩니다.
BLOB	BLOB 구조화 양식(호스트 언어에 의한 SQL 프로그래밍에서 C 장 참조)	-	BLOB 구조화 양식(호스트 언어에 의한 SQL 프로그래밍에서 COBOL 장 참조) **주:** iSeries용 ILE COBOL에 대해서만 지원됩니다.
DBCLOB	DBCLOB 구조화 양식(호스트 언어에 의한 SQL 프로그래밍에서 C 장 참조)	-	DBCLOB 구조화 양식(호스트 언어에 의한 SQL 프로그래밍에서 COBOL 장 참조) **주:** iSeries용 ILE COBOL에 대해서만 지원됩니다.

표 25. 매개변수의 자료 유형 (계속)

SQL 자료 유형	C 및 C++	CL	iSeries용 COBOL 및 iSeries용 ILE COBOL
ROWID	ROWID 구조화 양식(호스트 언어에 의한 SQL 프로그래밍에서 C 장 참조)	-	ROWID 구조화 양식(호스트 언어에 의한 SQL 프로그래밍의 COBOL 장 참조)
DataLink	-	-	-
인디케이터 변수	짧은	-	PIC S9(4) BINARY

표 26. 매개변수의 자료 유형

SQL 자료 유형	FORTRAN	Java 매개변수 양식 JAVA	Java 매개변수 양식 DB2GENERAL	PL/I
SMALLINT	INTEGER*2	짧은	짧은	FIXED BIN(15)
INTEGER	INTEGER*4	int	int	FIXED BIN(31)
BIGINT	-	긴	긴	-
DECIMAL(p,s)	-	BigDecimal	BigDecimal	FIXED DEC(p,s)
NUMERIC(p,s)	-	BigDecimal	BigDecimal	-
REAL 또는 FLOAT(p)	REAL*4	부동	부동	FLOAT BIN(p)
DOUBLE PRECISION, FLOAT 또는 FLOAT(p)	REAL*8	배정밀도	배정밀도	FLOAT BIN(p)
CHARACTER(n)	CHARACTER*n	스트링	스트링	CHAR(n)
VARCHAR(n)	-	스트링	스트링	CHAR(n) VAR
VARCHAR(n) FOR BIT DATA	-	-	com.ibm.db2.app.Blob	CHAR(n) VAR
GRAPHIC(n)	-	스트링	스트링	-
VARGRAPHIC(n)	-	스트링	스트링	-
DATE	CHARACTER*10	날짜	스트링	CHAR(10)
TIME	CHARACTER*8	시간	스트링	CHAR(8)
TIMESTAMP	CHARACTER*26	시간소인	스트링	CHAR(26)
CLOB	-	-	com.ibm.db2.app.Clob	CLOB 구조화 형식(호스트 언어에 의한 SQL 프로그래밍에서 PL/I 장 참조)
BLOB	-	-	com.ibm.db2.app.Blob	BLOB 구조화 형식(호스트 언어에 의한 SQL 프로그래밍에서 PL/I 장 참조)
DBCLOB	-	-	com.ibm.db2.app.Clob	DBCLOB 구조화 형식(호스트 언어에 의한 SQL 프로그래밍에서 PL/I 장 참조)

표 26. 매개변수의 자료 유형 (계속)

SQL 자료 유형	FORTRAN	Java 매개변수 양식 JAVA	Java 매개변수 양식 DB2GENERAL	PL/I
ROWID	-	-	-	ROWID 구조화 양식(호스트 언어에 의한 SQL 프로그래밍의 PL/I 장 참조)
DataLink	-	-	-	-
인디케이터 변수	INTEGER*2	-	-	FIXED BIN(15)

표 27. 매개변수의 자료 유형

SQL 자료 유형	REXX	RPG	ILE RPG
SMALLINT	-	단일 서브필드가 들어 있는 자료 구조. 서브필드 스펙의 43열은 ***B***, 52열은 그 길이가 각각 2와 0이어야 합니다.	자료 스펙. 서브필드 스펙에서 40열은 ***B***이고 길이는 <= 4여야 하고 41-42열은 00이어야 합니다. 또는 자료 스펙. 서브필드 스펙에서 위치 40은 ***I***이고 길이는 5이어야 하고, 위치 41-42는 00이어야 합니다.
INTEGER	소수가 없는 숫자 스트링 (및 선택적 선행 부호)	단일 서브필드가 들어 있는 자료 구조. 서브필드 스펙의 43열은 ***B***, 52열은 그 길이가 각각 4와 0이어야 합니다.	자료 스펙. 40열의 ***B***의 길이는 서브필드 스펙의 41-42열에서 <=09이고 >=05이며, 00이어야 합니다. 또는 자료 스펙. 서브필드 스펙에서 위치 40은 ***I***이고 길이는 10여야 하고, 위치 41-42는 00이어야 합니다.
BIGINT	-	-	자료 스펙. 서브필드 스펙에서 위치 40은 ***I***이고 길이는 20여야 하고, 위치 41-42는 00이어야 합니다.
DECIMAL(p,s)	소수가 있는 숫자 스트링 (및 선택적 선행 부호).	단일 서브필드가 들어 있는 자료 구조. 서브필드 스펙의 43열은 ***P***, 52열은 0-9여야 합니다. 또는 숫자 입력 필드나 연산 결과 필드.	자료 스펙. 서브필드 스펙에서 40열은 ***P***이고 41-42열은 00-31이어야 합니다.
NUMERIC(p,s)	-	단일 서브필드가 들어 있는 자료 구조. 서브필드 스펙의 43열은 ***공백***, 52열은 0-9여야 합니다.	자료 스펙. 서브필드 스펙에서 40열은 ***S***이거나 ***공백***이고 41-42열은 00-31이어야 합니다.
REAL 또는 FLOAT(p)	숫자 그리고 E(및 선택적 부호), 다음에 숫자	-	자료 스펙. 40열은 ***F***이고 길이는 4여야 합니다.
DOUBLE PRECISION, FLOAT 또는 FLOAT(p)	숫자 그리고 E(및 선택적 부호), 다음에 숫자	-	자료 스펙. 40열은 ***F***이고 길이는 8이어야 합니다.
CHARACTER(n)	두 개의 어포스트로피 사이에 n자가 있는 스트링	서브필드가 없는 자료 구조 또는 단일 서브필드가 들어 있는 자료 구조. 서브필드 스펙의 43열과 52열은 ***공백***이고 또는 문자 입력 필드나 연산 결과 필드여야 합니다.	자료 스펙. 서브필드 스펙에서 40열과 41-42열은 ***A***이거나 ***공백***이어야 합니다.

표 27. 매개변수의 자료 유형 (계속)

SQL 자료 유형	REXX	RPG	ILE RPG
VARCHAR(n)	두 개의 어포스트로피 사이에 n자가 있는 스트링	-	자료 스펙. 40열은 ***A***이거나 서브필드 스펙의 40과 41-42열은 ***공백***이고 44-80열은 키워드 VARYING입니다.
VARCHAR(n) FOR BIT DATA	두 개의 어포스트로피 사이에 n자가 있는 스트링	-	자료 스펙. 40열은 ***A***이거나 서브필드 스펙의 40과 41-42열은 ***공백***이고 44-80열은 키워드 VARYING입니다.
GRAPHIC(n)	G'로 시작되는 스트링 그리고 n자의 2바이트 문자, 다음에 '	-	자료 스펙. 서브필드 스펙의 40열은 ***G***입니다.
VARGRAPHIC(n)	G'로 시작되는 스트링 그리고 n자의 2바이트 문자, 다음에 '	-	자료 스펙. 서브필드 스펙의 40열은 ***G***이고 위치 44-80은 키워드 VARYING입니다.
DATE	두 개의 어포스트로피 사이에 10자가 있는 스트링	서브필드가 없는 자료 구조 또는 단일 서브필드가 들어 있는 자료 구조. 서브필드 스펙의 43열과 52열은 ***공백***이고 길이는 10 또는 문자 입력 필드나 연산 결과 필드여야 합니다.	자료 스펙. 서브필드 스펙의 40열은 ***D***입니다. 위치 44-80은 DATFMT(*ISO)입니다.
TIME	두 개의 어포스트로피 사이에 8자가 있는 스트링	서브필드가 없는 자료 구조 또는 단일 서브필드가 들어 있는 자료 구조. 서브필드 스펙의 43열과 52열은 ***공백***이고 길이는 8이어야 합니다. 또는 문자 입력 필드나 연산 결과 필드여야 합니다.	자료 스펙. 서브필드 스펙의 40열은 ***T***입니다. 위치 44-80은 TIMFMT(*ISO)입니다.
TIMESTAMP	두 개의 어포스트로피 사이에 26자가 있는 스트링	서브필드가 없는 자료 구조 또는 단일 서브필드가 들어 있는 자료 구조. 서브필드 스펙의 43열과 52열은 ***공백***이고 길이는 26이어야 합니다. 또는 문자 입력 필드나 연산 결과 필드여야 합니다.	자료 스펙. 서브필드 스펙의 40열은 ***Z***입니다.
CLOB	-	-	CLOB 구조화 형식(호스트 언어에 의한 SQL 프로그래밍에서 RPG 장 참조)
BLOB	-	-	BLOB 구조화 형식(호스트 언어에 의한 SQL 프로그래밍에서 RPG 장 참조)
DBCLOB	-	-	DBCLOB 구조화 형식(호스트 언어에 의한 SQL 프로그래밍에서 RPG 장 참조)
ROWID	-	-	ROWID 구조화 양식(호스트 언어에 의한 SQL 프로그래밍의 RPG 장 참조)
DataLink	-	-	-
인디케이터 변수	소수가 없는 숫자 스트링 (및 선택적 선행 부호)	단일 서브필드가 들어 있는 자료 구조. 서브필드 스펙의 43열은 ***B***, 52열은 그 길이가 각각 2와 0이어야 합니다.	자료 스펙. 서브필드 스펙에서 40열은 ***B***이고 길이는 <= 4여야 하고 41-42열은 00이어야 합니다.

인디케이터 변수 및 저장 프로시듀어

인디케이터 변수는 CALL문에 사용될 수 있으며 제공되는 호스트 변수는 프로시듀어 간에서 추가 정보를 주고 받기 위한 매개변수에 사용됩니다. 인디케이터 변수는 연관된 호스트 변수가 널 값을 포함하는 것으로 해석되도록 하는 SQL의 표준 설명 수단이며 이것이 1차적인 용도입니다.

호스트 변수에 널 값이 포함됨을 나타내기 위해서는 2바이트 정수인 인디케이터 변수를 음의 값으로 설정해야 합니다. 인디케이터 변수가 있는 CALL문은 다음과 같이 처리됩니다.

- 인디케이터 변수가 음수이면 이것은 널 값을 나타냅니다. 디폴트 값이 CALL문의 연관된 호스트 변수로 전달되며 인디케이터 변수는 변경되지 않은 채 전달됩니다.
- 인디케이터 변수가 음수이면 이것은 호스트 변수에 널이 아닌 값이 들어 있음을 나타냅니다. 이 경우 호스트 변수와 인디케이터 변수가 변경되지 않고 전달됩니다.

이러한 처리 규칙은 프로시듀어로부터 리턴되는 출력 매개변수뿐만 아니라 프로시듀어에 대한 입력 매개변수에서도 동일합니다. 인디케이터 변수가 저장 프로시듀어와 함께 사용될 때 처리를 위한 올바른 코딩 방식은 연관된 호스트 변수를 사용하기에 앞서 인디케이터 변수의 값을 우선 체크하는 것입니다.

다음 예는 CALL문에서 인디케이터 변수 처리를 보여줍니다. 연관된 호스트 변수를 사용하기에 앞서 인디케이터 변수의 값을 체크하는 논리에 유의하십시오. 또한 인디케이터 변수가 프로시듀어 PROC(2바이트 값의 배열로 이루어진 세 번째 값으로서)로 전달되는 방식에도 유의하십시오.

프로시듀어가 다음과 같이 정의되었다고 가정합니다.

```
CREATE PROCEDURE PROC1
   (INOUT DECIMALOUT DECIMAL(7,2), INOUT DECOUT2 DECIMAL(7,2))
   EXTERNAL NAME LIB1.PROC1 LANGUAGE RPGLE
   GENERAL WITH NULLS)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Program CRPG
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    D INOUT1          S              7P 2
    D INOUT1IND       S              4B 0
    D INOUT2          S              7P 2
    D INOUT2IND       S              4B 0
    C                   EVAL      INOUT1 = 1
    C                   EVAL      INOUT1IND  = 0
    C                   EVAL      INOUT2 = 1
    C                   EVAL      INOUT2IND = -2
    C/EXEC SQL CALL PROC1 (:INOUT1 :INOUT1IND , :INOUT2
    C+                         :INOUT2IND)
    C/END-EXEC
    C                   EVAL      INOUT1 = 1
    C                   EVAL      INOUT1IND  = 0
    C                   EVAL      INOUT2 = 1
    C                   EVAL      INOUT2IND = -2
    C/EXEC SQL CALL PROC1 (:INOUT1 :INOUT1IND , :INOUT2
    C+                         :INOUT2IND)
    C/END-EXEC
    C     INOUT1IND     IFLT      0
    C*                   :
    C*                  HANDLE NULL INDICATOR
    C*                   :
    C                   ELSE
    C*                   :
    C*                  INOUT1 CONTAINS VALID DATA
    C*                   :
    C                   ENDIF
    C*                   :
    C*                  HANDLE ALL OTHER PARAMETERS
    C*                  IN A SIMILAR FASHION
    C*                   :
    C                   RETURN
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
End of PROGRAM CRPG
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

그림 3. CALL문에서 인디케이터 변수 처리 (1/2)

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Program PROC1
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    D INOUTP          S              7P 2
    D INOUTP2         S              7P 2
    D NULLARRAY       S              4B 0 DIM(2)
    C     *ENTRY        PLIST
    C                   PARM                    INOUTP
    C                   PARM                    INOUTP2
    C                   PARM                    NULLARRAY
    C     NULLARRAY(1)  IFLT      0
    C*                   :
    C*                  INOUTP DOES NOT CONTAIN MEANINGFUL DATA
    C*
    C                   ELSE
    C*                   :
    C*                  INOUTP CONTAINS MEANINGFUL DATA
    C*                   :
    C                   ENDIF
    C*                  PROCESS ALL REMAINING VARIABLES
    C*
    C*                  BEFORE RETURNING, SET OUTPUT VALUE FOR FIRST
    C*                  PARAMETER AND SET THE INDICATOR TO A NON-NEGATIV
    C*                  VALUE SO THAT THE DATA IS RETURNED TO THE CALLING
    C*                  PROGRAM
    C*
    C                   EVAL      INOUTP2 = 20.5
    C                   EVAL      NULLARRAY(2) = 0
    C*
    C*                  INDICATE THAT THE SECOND PARAMETER IS TO CONTAIN
    C*                  THE NULL VALUE UPON RETURN. THERE IS NO POINT
    C*                  IN SETTING THE VALUE IN INOUTP SINCE IT WON'T BE
    C*                  PASSED BACK TO THE CALLER.
    C                   EVAL      NULLARRAY(1) = -5
    C                   RETURN
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
End of PROGRAM PROC1
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

그림 3. CALL문에서 인디케이터 변수 처리 (2/2)

완료 상태를 호출 프로그램에 리턴하는 방법

SQL 프로시듀어의 경우, 프로시듀어에서 처리되지 않은 오류는 SQLCA의 호출자에게 리턴됩니다. SIGNAL 및 RESIGNAL 제어문을 오류 정보를 송신하는 데에 사용할 수도 있습니다. 자세한 정보는 SQL 참조서의 SQL 프로시듀어, 함수 및 트리거 주제를 참조하십시오.

외부 프로시듀어의 경우, 상태 정보를 리턴하는 두 가지 방법이 있습니다. CALL문을 발행하는 SQL 프로그램으로 상태를 리턴시키는 한 가지 방법은 임시로 INOUT 유형

매개변수를 코딩하고 프로시듀어로부터 리턴되기 전에 설정하는 것입니다. 호출되는 프로시듀어가 기존 프로그램인 경우 이것이 가능하지 않을 수도 있습니다.

CALL문을 발행하는 SQL 프로그램으로 상태를 리턴하는 또 다른 방법은 이탈 메세지를 프로시듀어를 호출하는 호출 프로그램(오퍼레이팅 시스템 프로그램 QSQCALL)으로 송신하는 것입니다. 프로시듀어를 호출하는 호출 프로그램은 QSQCALL입니다. 각 언어에는 조건을 신호하고 메세지를 송신하는 방법이 있습니다. 메세지를 신호하기 위한 적절한 방법을 결정하려면 각각의 언어 참조서를 보십시오. 메세지가 신호될 때 QSQCALL은 오류를 SQLCODE/SQLSTATE-443/38501로 만듭니다.

CALL문 예

이 예에서는 CALL문의 인수가 여러 언어에 대한 프로시듀어로 전달되는 방법을 보여줍니다. 이 예에서는 또한 프로시듀어에서 인수들을 로컬 변수로 수신하는 방법을 보여줍니다.

첫 번째 예제는 CREATE PROCEDURE 정의를 사용하여 P1과 P2 프로시듀어를 호출하는 ILE C 프로그램을 보여줍니다. 프로시듀어 P1은 C로 작성되며 10개의 매개변수가 있습니다. 프로시듀어 P2는 PL/I로 작성되며 역시 10개의 매개변수가 있습니다.

두 개의 프로시듀어가 다음과 같이 정의되었다고 가정합니다.

```
EXEC SQL CREATE PROCEDURE P1 (INOUT PARM1 CHAR(10),
                              INOUT PARM2 INTEGER,
                              INOUT PARM3 SMALLINT,
                              INOUT PARM4 FLOAT(22),
                              INOUT PARM5 FLOAT(53),
                              INOUT PARM6 DECIMAL(10,5),
                              INOUT PARM7 VARCHAR(10),
                              INOUT PARM8 DATE,
                              INOUT PARM9 TIME,
                              INOUT PARM10 TIMESTAMP)
          EXTERNAL NAME  TEST12.CALLPROC2
          LANGUAGE C GENERAL WITH NULLS

EXEC SQL CREATE PROCEDURE P2 (INOUT PARM1 CHAR(10),
                              INOUT PARM2 INTEGER,
                              INOUT PARM3 SMALLINT,
                              INOUT PARM4 FLOAT(22),
                              INOUT PARM5 FLOAT(53),
                              INOUT PARM6 DECIMAL(10,5),
                              INOUT PARM7 VARCHAR(10),
                              INOUT PARM8 DATE,
                              INOUT PARM9 TIME,
                              INOUT PARM10 TIMESTAMP)
          EXTERNAL NAME  TEST12.CALLPROC
          LANGUAGE PLI GENERAL WITH NULLS
```

예 1: ILE C 어플리케이션에서 호출된 ILE C 및 PL/I 프로시듀어

주: 코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

```
/*************************************************************/
/*********** START OF SQL C Application **********************/

#include <stdio.h>
#include <string.h>
        #include <decimal.h>
        main()
{
 EXEC SQL INCLUDE SQLCA;
 char PARM1[10];
 signed long int PARM2;
 signed short int PARM3;
 float PARM4;
 double PARM5;
 decimal(10,5) PARM6;
 struct { signed short int parm7l;
          char parm7c[10];
        } PARM7;
char PARM8[10];        /* FOR DATE */
char PARM9[8];         /* FOR TIME */
char PARM10[26];       /* FOR TIMESTAMP */
```

그림 4. CREATE PROCEDURE 및 CALL의 샘플 (1/2)

```
/*****************************************************/
/* Initialize variables for the call to the procedures */
/*****************************************************/
strcpy(PARM1,"PARM1");
PARM2 = 7000;
PARM3 = -1;
PARM4 = 1.2;
PARM5 = 1.0;
PARM6 = 10.555;
PARM7.parm7l = 5;
strcpy(PARM7.parm7c,"PARM7");
strncpy(PARM8,"1994-12-31",10);         /* FOR DATE      */
strncpy(PARM9,"12.00.00",8);            /* FOR TIME      */
strncpy(PARM10,"1994-12-31-12.00.00.000000",26);
                                        /* FOR TIMESTAMP */
/*********************************************/
/* Call the C procedure                      */
/*                                           */
/*                                           */
/*********************************************/
EXEC SQL CALL P1 (:PARM1, :PARM2, :PARM3,
                  :PARM4, :PARM5, :PARM6,
                  :PARM7, :PARM8, :PARM9,
                  :PARM10 );
if (strncmp(SQLSTATE,"00000",5))
 {
  /* Handle error or warning returned on CALL statement */
 }

/* Process return values from the CALL.                */
:


/*********************************************/
/* Call the PLI procedure                    */
/*                                           */
/*                                           */
/*********************************************/
/* Reset the host variables prior to making the CALL    */
/*                                                      */
:
EXEC SQL CALL P2 (:PARM1, :PARM2, :PARM3,
                  :PARM4, :PARM5, :PARM6,
                  :PARM7, :PARM8, :PARM9,
                  :PARM10 );
if (strncmp(SQLSTATE,"00000",5))
 {
  /* Handle error or warning returned on CALL statement */
}
/* Process return values from the CALL.                */
:
}

/******** END OF C APPLICATION *********************************/
/***************************************************************/
```

그림 4. CREATE PROCEDURE 및 CALL의 샘플 (2/2)

```
/******** START OF C PROCEDURE P1 ******************************/
/*        PROGRAM TEST12/CALLPROC2                              */
/*************************************************************/

#include <stdio.h>
#include <string.h>
        #include <decimal.h>
main(argc,argv)
  int argc;
  char *argv[];
  {
    char parm1[11];
    long int parm2;
    short int parm3,i,j,*ind,ind1,ind2,ind3,ind4,ind5,ind6,ind7,
              ind8,ind9,ind10;
    float parm4;
    double parm5;
    decimal(10,5) parm6;
    char parm7[11];
        char parm8[10];
        char parm9[8];
        char parm10[26];
    /* ********************************************************/
    /* Receive the parameters into the local variables -       */
    /* Character, date, time, and timestamp are passed as      */
    /* NUL terminated strings - cast the argument vector to    */
    /* the proper data type for each variable. Note that       */
    /* the argument vector could be used directly instead of   */
    /* copying the parameters into local variables - the copy  */
    /* is done here just to illustrate the method.             */
    /* ********************************************************/

    /* Copy 10 byte character string into local variable      */
    strcpy(parm1,argv[1]);

    /* Copy 4 byte integer into local variable                */
    parm2 = *(int *) argv[2];

    /* Copy 2 byte integer into local variable                */
    parm3 = *(short int *) argv[3];

    /* Copy floating point number into local variable         */
    parm4 = *(float *) argv[4];

    /* Copy double precision number into local variable       */
    parm5 = *(double *) argv[5];

    /* Copy decimal number into local variable                */
    parm6 = *(decimal(10,5) *) argv[6];
```

그림 5. 프로시듀어 P1의 예 (1/2)

```
/*********************************************************/
/* Copy NUL terminated string into local variable.         */
/* Note that the parameter in the CREATE PROCEDURE was   */
/* declared as varying length character. For C, varying   */
/* length are passed as NUL terminated strings unless     */
/* FOR BIT DATA is specified in the CREATE PROCEDURE     */
/*********************************************************/
strcpy(parm7,argv[7]);

/*********************************************************/
/* Copy date into local variable.                         */
/* Note that date and time variables are always passed in */
/* ISO format so that the lengths of the strings are      */
/* known. strcpy would work here just as well.            */
/*********************************************************/
strncpy(parm8,argv[8],10);

/* Copy time into local variable                          */
strncpy(parm9,argv[9],8);

/*********************************************************/
/* Copy timestamp into local variable.                    */
/* IBM SQL timestamp format is always passed so the length*/
/* of the string is known.                                */
/*********************************************************/
strncpy(parm10,argv[10],26);

/*********************************************************/
/* The indicator array is passed as an array of short     */
/* integers. There is one entry for each parameter passed */
/* on the CREATE PROCEDURE (10 for this example).        */
/* Below is one way to set each indicator into separate   */
/* variables.                                             */
/*********************************************************/
   ind = (short int *) argv[11];
   ind1 = *(ind++);
   ind2 = *(ind++);
   ind3 = *(ind++);
   ind4 = *(ind++);
   ind5 = *(ind++);
   ind6 = *(ind++);
   ind7 = *(ind++);
   ind8 = *(ind++);
   ind9 = *(ind++);
   ind10 = *(ind++);
  :
/* Perform any additional processing here                 */
  :
return;
}
/******** END OF C PROCEDURE P1 *****************************/
```

그림 5. 프로시듀어 P1의 예 (2/2)

```
/******** START OF PL/I PROCEDURE P2 *************************/
/******** PROGRAM TEST12/CALLPROC ****************************/
/************************************************************/


CALLPROC :PROC( PARM1,PARM2,PARM3,PARM4,PARM5,PARM6,PARM7,
                      PARM8,PARM9,PARM10,PARM11);
 DCL  SYSPRINT FILE STREAM OUTPUT EXTERNAL;
 OPEN FILE(SYSPRINT);
 DCL PARM1 CHAR(10);
 DCL PARM2 FIXED BIN(31);
 DCL PARM3 FIXED BIN(15);
 DCL PARM4 BIN FLOAT(22);
 DCL PARM5 BIN FLOAT(53);
 DCL PARM6 FIXED DEC(10,5);
 DCL PARM7 CHARACTER(10) VARYING;
 DCL PARM8 CHAR(10);      /* FOR DATE */
 DCL PARM9 CHAR(8);       /* FOR TIME */
 DCL PARM10 CHAR(26);     /* FOR TIMESTAMP */
 DCL PARM11(10) FIXED BIN(15);   /* Indicators */

 /* PERFORM LOGIC - Variables can be set to other values for  */
 /* return to the calling program.                            */

 :

 END CALLPROC;
```

그림 6. 프로시듀어 P2의 예

다음 예는 ILE C 프로그램에서 호출된 REXX 프로시듀어를 보여줍니다.

프로시듀어가 다음과 같이 정의되었다고 가정합니다.

```
EXEC SQL CREATE PROCEDURE REXXPROC
              (IN PARM1 CHARACTER(20),
               IN PARM2 INTEGER,
               IN PARM3 DECIMAL(10,5),
               IN PARM4 DOUBLE PRECISION,
               IN PARM5 VARCHAR(10),
               IN PARM6 GRAPHIC(4),
               IN PARM7 VARGRAPHIC(10),
               IN PARM8 DATE,
               IN PARM9 TIME,
               IN PARM10 TIMESTAMP)
         EXTERNAL NAME 'TEST.CALLSRC(CALLREXX)'
         LANGUAGE REXX GENERAL WITH NULLS
```

예 2. C 어플리케이션에서 호출된 REXX 프로시듀어 예

주: 코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

```
/************************************************************/
/*********** START OF SQL C Application **********************/

        #include <decimal.h>
        #include <stdio.h>
        #include <string.h>
        #include <wcstr.h>
        /*-----------------------------------------------------------*/
        exec sql include sqlca;
        exec sql include sqlda;
        /* ************************************************************/
        /* Declare host variable for the CALL statement              */
        /* ************************************************************/
        char parm1[20];
        signed long int parm2;
        decimal(10,5) parm3;
        double parm4;
        struct {  short  dlen;
                  char dat[10];
                } parm5;
        wchar_t parm6[4] = { 0xC1C1, 0xC2C2, 0xC3C3, 0x0000 };
        struct {  short  dlen;
                  wchar_t dat[10];
                } parm7 = {0x0009, 0xE2E2,0xE3E3,0xE4E4, 0xE5E5, 0xE6E6,
                           0xE7E7, 0xE8E8, 0xE9E9, 0xC1C1, 0x0000 };

        char parm8[10];
        char parm9[8];
        char parm10[26];
        main()
        {
```

그림 7. C 어플리케이션으로부터 호출된 REXX 프로시듀어의 예 (1/4)

```
        /* ************************************************************/
        /* Call the procedure - on return from the CALL statement the  */
        /* SQLCODE should be 0. If the SQLCODE is non-zero,            */
        /* the procedure detected an error.                            */
        /* ************************************************************/
        strcpy(parm1,"TestingREXX");
        parm2 = 12345;
        parm3 = 5.5;
        parm4 = 3e3;
        parm5.dlen = 5;
        strcpy(parm5.dat,"parm6");
        strcpy(parm8,"1994-01-01");
        strcpy(parm9,"13.01.00");
        strcpy(parm10,"1994-01-01-13.01.00.000000");

         EXEC SQL CALL REXXPROC (:parm1, :parm2,
                                 :parm3,:parm4,
                                 :parm5, :parm6,
                                 :parm7,
                                 :parm8, :parm9,
                                 :parm10);

         if (strncpy(SQLSTATE,"00000",5))
           {
           /* handle error or warning returned on CALL   */
           :
           }
                :
       }

/****** END OF SQL C APPLICATION   ***********************************/
/*********************************************************************/
```

그림 7. C 어플리케이션으로부터 호출된 REXX 프로시듀어의 예 (2/4)

```
/*********************************************************************/
/****** START OF REXX MEMBER TEST/CALLSRC CALLREXX *******************/
/*********************************************************************/
       /* REXX source member TEST/CALLSRC CALLREXX              */
       /* Note the extra parameter being passed for the indicator*/
       /* array.                                                */
       /*                                                       */
       /* ACCEPT THE FOLLOWING INPUT VARIABLES SET TO THE       */
       /* SPECIFIED VALUES :                                    */
       /* AR1        CHAR(20)          =  'TestingREXX'         */
       /* AR2        INTEGER           = 12345                  */
       /* AR3        DECIMAL(10,5)     = 5.5                    */
       /* AR4        DOUBLE PRECISION  = 3e3                    */
       /* AR5        VARCHAR(10)       = 'parm6'                */
       /* AR6        GRAPHIC           = G'C1C1C2C2C3C3'        */
       /* AR7        VARGRAPHIC        =                        */
       /*             G'E2E2E3E3E4E4E5E5E6E6E7E7E8E8E9E9EAEA'   */
       /* AR8        DATE              = '1994-01-01'           */
       /* AR9        TIME              = '13.01.00'             */
       /* AR10       TIMESTAMP         =                        */
       /*                       '1994-01-01-13.01.00.000000'     */
       /* AR11       INDICATOR ARRAY   = +0+0+0+0+0+0+0+0+0+0    */

       /*********************************************************/
       /* Parse the arguments into individual parameters        */
       /*********************************************************/
       parse arg ar1 ar2 ar3 ar4 ar5 ar6 ar7 ar8 ar9 ar10 ar11

       /*********************************************************/
       /* Verify that the values are as expected                */
       /*********************************************************/
       if ar1<>"'TestingREXX'" then signal ar1tag
       if ar2<>12345 then signal ar2tag
       if ar3<>5.5 then signal ar3tag
       if ar4<>3e3 then signal ar4tag
       if ar5<>"'parm6'" then signal ar5tag
       if ar6 <>"G'AABBCC'" then signal ar6tag
       if ar7 <>"G'SSTTUUVVWWXXYYZZAA'" then ,
        signal ar7tag
       if ar8 <> "'1994-01-01'" then signal ar8tag
       if ar9 <> "'13.01.00'" then signal ar9tag
       if ar10 <> "'1994-01-01-13.01.00.000000'" then signal ar10tag
       if ar11 <> "+0+0+0+0+0+0+0+0+0+0" then signal ar11tag
```

그림 7. C 어플리케이션으로부터 호출된 REXX 프로시듀어의 예 (3/4)

```
      /*************************************************************/
      /* Perform other processing as necessary ..                  */
      /*************************************************************/
         :
      /*************************************************************/
      /* Indicate the call was successful by exiting with a        */
      /* return code of 0                                          */
      /*************************************************************/
      exit(0)

      ar1tag:
      say "ar1 did not match" ar1
      exit(1)
      ar2tag:
      say "ar2 did not match" ar2
      exit(1)
      :
      :

/************ END OF REXX MEMBER   *********************************/
```

그림 7. C 어플리케이션으로부터 호출된 REXX 프로시듀어의 예 (4/4)

제 12 장 오브젝트 관계 기능 사용

이 장은 DB2의 오브젝트 지향 기능에 대해 설명합니다.

- DB2 오브젝트 확장을 사용하는 이유
- 오브젝트 지원에 대한 DB2 접근
- 대형 오브젝트(LOB) 사용
- 사용자 정의 기능(UDF)
- 사용자 정의된 고유한 유형(UDT)
- UDT, UDF 및 LOB 사이의 공동 작용

현대 프로그래밍 언어 기술에서 최근에 개발된 가장 중요한 개념 중 하나는 *오브젝트 지향*입니다. 오브젝트 지향은 어플리케이션 정의역의 엔티티를 클래스화하여 서로 연관된 독립적 오브젝트로 모델링할 수 있다는 개념입니다. 오브젝트 지향을 통해 어플리케이션 정의역의 오브젝트의 유사점과 차이점을 파악할 수 있고 이 오브젝트를 관련된 유형으로 그룹화할 수 있습니다. 같은 유형의 오브젝트는 같은 집합의 특정 유형 기능을 공유하므로 같은 방법으로 작동합니다. 이 기능은 어플리케이션 정의역의 오브젝트 작동의 영향을 받습니다.

DB2 오브젝트 확장을 사용하는 이유

DB2의 오브젝트 확장을 사용하여, 오브젝트 지향(OO) 개념과 방법론을 관계형 데이터베이스(RDB)에 결합시킬 수 있습니다. 더 풍부한 유형과 기능 집합으로 확장하면 됩니다. 이러한 확장으로 표 열에 오브젝트 지향 자료 유형의 인스턴스를 저장할 수 있으며 SQL문의 기능을 사용하여 조작할 수 있습니다. 뿐만 아니라, 저장된 오브젝트의 의미 작동을, 데이터베이스를 사용하여 모든 사용자 어플리케이션이 공유할 수 있는 중요한 자원으로 만들 수 있습니다.

오브젝트 지향을 관계형 데이터베이스(RDB) 시스템에 결합하여 사용자의 새로운 유형과 기능을 정의할 수 있습니다. 이 새로운 유형과 기능은 어플리케이션 정의역의 오브젝트의 의미를 반영해야 합니다. 모델링하려는 일부 자료 오브젝트는 크고 복잡한 것일 수도 있습니다(예를 들면, 텍스트, 음성, 이미지, 재무 자료). 따라서 대형 오브젝트의 조작과 기억장치에 대한 메카니즘도 필요합니다. 사용자 정의 고유한 유형(UDT), 사용자 정의 기능(UDF), 그리고 대형 오브젝트(LOB)는 DB2에 의해 제공되는 메카니즘입니다. DB2를 사용하여, 이제 데이터베이스 내의 어플리케이션 오브젝트를 조작하고 저장하여 사용자의 새로운 유형과 기능을 정의할 수 있습니다.

다음 섹션에서 설명되겠지만 이 오브젝트 지향 피처는 중요한 시너지 효과를 냅니다. 사용자는 어플리케이션 정의역의 복잡한 오브젝트를 UDT로 모델링할 수 있습니다. UDT는 내부적으로 LOB로 표시될 수 있습니다. UDT의 작동은 UDF의 표현으로 구현될

수 있습니다. 이 섹션은 UDT와 UDF를 정의하기 위해 필요한 단계에 따라 LOB를 사용하는 방법을 보여줍니다. 또한 UDT, UDF, LOB가 사용자 어플리케이션의 오브젝트를 어떻게 더 잘 표현하여 함께 작업할 수 있는지 알려줍니다.

주: DB2 오브젝트 지향 메카니즘(UDT, UDF, LOB)은 오브젝트 지향 어플리케이션을 지원하기 위해서만 사용하는 것은 아닙니다. C++ 프로그래밍 언어가 모든 종류의 비 오브젝트 지향 어플리케이션을 구현하듯이 DB2에 의해 제공되는 오브젝트 지향 메카니즘은 모든 종류의 비 오브젝트 지향 어플리케이션도 지원합니다. UDT, UDF, LOB는 모든 데이터베이스 어플리케이션을 모델링하기 위해 사용할 수 있는 범용 메카니즘입니다. 따라서, 이 DB2 오브젝트 확장은 이전의 어플리케이션 뿐 아니라 오브젝트 지향 어플리케이션을 확장해서 지원합니다.

오브젝트 지원에 대한 DB2 접근

DB2의 오브젝트 확장으로 관계형 기술의 강점을 가지면서 오브젝트 기술의 장점을 알 수 있습니다. 관계형 시스템에서 자료 유형은 이 자료 유형의 인스턴스(또는 오브젝트)가 저장된 표의 열에 저장된 자료를 서술합니다. 이 인스턴스에 대한 조작은 표현식이 허용하는 곳에서 호출될 수 있는 기능 또는 오퍼레이터에 의해 지원됩니다.

오브젝트 확장 지원에 대한 DB2 접근은 관계형 개념과 정확하게 들어 맞습니다. UDT는 사용자가 정의하는 자료 유형입니다. 내장 유형과 마찬가지로 UDT는 표의 열에 저장된 자료를 서술하기 위해 사용될 수 있습니다. UDF는 사용자가 정의하는 기능입니다. 내장 기능 또는 조작과 마찬가지로 UDF는 UDT 인스턴스의 조작을 지원합니다. 따라서 UDT 인스턴스는 표의 열에 저장되며 SQL 조회문의 UDF에 의해 조작됩니다. UDT는 다른 방법으로 내부적으로 표현될 수 있습니다. LOB는 이 중의 한 예에 불과합니다.

대형 오브젝트(LOB) 사용

VARCHAR와 VARGRAPHIC 자료 유형은 32K바이트의 기억장치로 제한됩니다. 이 *크기*는 작거나 중간 *크기*의 텍스트 자료에 대해서는 충분하지만 어플리케이션은 종종 큰 텍스트 문서를 저장해야 합니다. 오디오, 비디오, 그림, 텍스트와 그래픽, 이미지와 같은 다양한 추가 자료 유형도 저장해야 합니다. DB2는 *크기*가 최대 2기가바이트인 스트링으로 이러한 자료 오브젝트를 저장하기 위해 3가지 자료 유형을 제공합니다. 3 가지 자료 유형은 2진 대형 오브젝트(BLOB), 1바이트 문자 대형 오브젝트(CLOB), 2바이트 문자 대형 오브젝트(DBCLOB)입니다.

대형 오브젝트(LOB)를 저장하는 것 외에도 데이터베이스의 각 LOB를 수정, 사용, 참조하는 방법도 필요합니다. 각 DB2 표에는 많은 양의 연관된 LOB 자료가 있을 수 있습니다. 하나 이상의 LOB 값이 들어 있는 단일 행이 3.5기가바이트를 초과할 수 없지만 표에는 거의 256기가바이트의 LOB 자료가 포함될 수도 있습니다. 임의의 시점의 특정 행의 LOB 열의 내용에는 *대형 오브젝트 값*이 있습니다.

호스트 변수를 기타 자료 유형과 같이 사용하여 LOB를 참조하고 조작할 수 있습니다. 그러나, 호스트 변수는 LOB 값을 보유할 수 있을 만큼 충분히 크지 않은 클라이언트 메모리 버퍼를 사용합니다. 이 큰 값을 조작하려면 다른 방법이 필요합니다. *위치 지정자*는 데이터베이스 서버에서 대형 오브젝트 값을 식별하고 조작하고, LOB 값들을 추출하는데 유용합니다. *파일 참조 변수*는 대형 오브젝트 값을 실제로 클라이언트와 서로 주고 받는데 유용합니다.

다음의 하위 절에서는 위에 소개된 주제들을 보다 자세하게 설명합니다.

대형 오브젝트 자료 유형(BLOB, CLOB, DBCLOB) 이해

대형 오브젝트 자료 유형은 *크기*가 0바이트인 자료부터 2기가바이트인 자료까지 저장합니다.

3 가지의 대형 오브젝트 자료 유형은 다음과 같은 정의를 가집니다.

- 큰 문자 오브젝트(CLOB) -- 연관된 코드 페이지와 함께 1바이트 문자로 구성된 문자 스트링. 이 자료 유형은 정보의 양이 정규 VARCHAR 자료 유형의 한계(상한은 32K바이트) 이상으로 커질 수 있는 텍스트 기반 정보를 보유하기에 아주 좋습니다. 다른 문자 유형과의 호환 뿐만 아니라 정보의 코드 페이지 변환도 지원됩니다.
- 2바이트 큰 문자 오브젝트(DBCLOB) -- 연관된 코드 페이지와 함께 2바이트 문자로 구성된 문자 스트링. 이 자료 유형은 2바이트 문자 집합이 사용되는 텍스트 기반 정보를 보유하기에 아주 좋습니다. 또한 다른 2바이트 문자 유형과의 호환 뿐만 아니라 정보의 코드 페이지 변환도 지원됩니다.
- 큰 2진 오브젝트(BLOB) -- 연관된 코드 페이지가 없이 바이트로 구성된 2진 스트링. 이 자료 유형은 2진 자료를 저장할 수 있으므로 가장 유용합니다. 따라서, 사용자 정의 고유한 유형(UDT)에 의해 사용되는 완벽한 소스 유형입니다. BLOB를 소스 유형으로 사용하는 UDT는 이미지, 음성, 그래픽, 기타 업무 또는 특정 어플리케이션 자료를 저장하기 위해 만들어집니다. UDT에 대한 자세한 내용은 240 페이지의 『사용자 정의된 고유한 유형(UDT)』을 참조하십시오.

대형 오브젝트 위치 지정자 이해

개념적으로, LOB 위치 지정자는 작고, 쉽게 관리할 수 있는 값을 사용하여 훨씬 더 큰 값을 참조한다는, 현재 통용되고 있는 간단한 생각을 표현합니다. 특별히, LOB 위치 지정자는 데이터베이스 시스템에 보유된 LOB 값(또는 LOB 표현식)을 참조하기 위해 프로그램이 사용하는 호스트 변수에 저장된 4바이트 값입니다. LOB 위치 지정자를 사용하여, 프로그램은 LOB 값이 정규 호스트 변수에 저장된 것처럼 LOB 값을 조작할 수 있습니다. LOB 위치 지정자를 사용할 때, LOB 값을 서버에서 어플리케이션으로 가져올 (또한 다시 뒤로 가져갈) 필요는 없습니다.

LOB 위치 지정자는 데이터베이스의 행 또는 실제 기억장치 위치가 아니라, LOB 값 또는 LOB 표현식과 연관됩니다. 따라서 LOB 값을 위치 지정자로 선택한 후, 위치 지정자에 의해 참조되는 값에 영향을 줄 원래 행 또는 표에 대해 조작을 수행할 수 없습니다. 위치 지정자와 연관된 값은 작업 단위가 끝나거나, 위치 지정자가 명시적으로 해제될 때(둘 중 먼저 일어나는 작업)까지 유용합니다. FREE LOCATOR문은 위치 지정자를 연관된 값에서 해제합니다. 마찬가지로 확약 또는 롤백 조작은 트랜잭션과 연관된 모든 LOB 위치 지정자를 해제합니다.

LOB 위치 지정자는 DB2와 UDF 사이에서도 전달될 수 있습니다. UDF 내에서, LOB 위치 지정자를 사용하여 LOB 값을 조작하기 위해 LOB 자료에 대한 작업을 수행하는 해당 기능이 제공됩니다.

LOB 값을 선택할 때 3 가지 옵션이 있습니다.

- 전체 LOB 값을 호스트 변수로 선택합니다. 전체 LOB 값은 호스트 변수로 복사됩니다.
- LOB 값을 LOB 위치 지정자로 선택합니다. LOB 값은 서버에 남아 있으며, 호스트 변수로 복사되지 않습니다.
- 전체 LOB 값을 파일 참조 변수로 선택합니다. LOB 값은 통합 파일 시스템(IFS) 파일로 이동됩니다.

프로그램 내의 LOB 값 사용으로 프로그래머는 어떤 방법이 최선인지를 판별할 수 있습니다. LOB 값이 매우 크고 하나 이상의 연속되는 SQL문에 대한 입력 값으로만 필요한 경우, 값을 위치 지정자에 보관합니다.

프로그램에서 크기에 상관없이 전체 LOB 값을 필요로 하는 경우, LOB를 전송할 수 밖에 없습니다. 이 경우에도 선택할 수 있는 옵션이 있습니다. 전체 값을 정규 또는 파일 참조 호스트 변수로 선택할 수 있습니다. LOB 값을 위치 지정자로 선택하여 다음 예인 213 페이지의 『예: CLOB 값으로 작업하기 위해 위치 지정자 사용』에서 제안하는 것처럼 위치 지정자에서 정규 호스트 변수로 조금씩 읽어들일 수 있습니다.

예: CLOB 값으로 작업하기 위해 위치 지정자 사용

이 예에서, 어플리케이션 프로그램은 LOB 값에 대한 위치 지정자를 검색한 후 LOB 값에서 자료를 추출하기 위해 위치 지정자를 사용합니다. 이 방법을 사용하여 프로그램은 한 조각의 LOB 자료(크기는 프로그램에 의해 결정됨)에 대해 충분한 기억장치만을 할당합니다. 뿐만 아니라, 커서를 사용하여 프로그램은 하나의 페치 호출만을 발행해야 합니다.

주: 코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

샘플 LOBLOC 프로그램 작업 방법

1. **호스트 변수를 선언합니다.** BEGIN DECLARE SECTION과 END DECLARE SECTION문은 호스트 변수 선언을 구분합니다. SQL문에서 참조될 때 호스트 변수에는 콜론(:)의 접두부가 붙습니다. CLOB LOCATOR 호스트 변수를 선언합니다.
2. **LOB 값을 위치 지정자 호스트 변수로 페치합니다.** 데이터베이스의 LOB 필드의 위치를 위치 지정자 호스트 변수로 가져오기 위해 CURSOR와 FETCH 루틴이 사용됩니다.
3. **LOB LOCATORS를 해제합니다.** 이 예에서 사용된 LOB LOCATORS가 해제되며 이전에 연관된 값에서 위치 지정자를 해제합니다.

CHECKERR 매크로/기능은 프로그램 외부의 오류 검사 유틸리티입니다. 이 오류 검사 유틸리티의 위치는 사용된 프로그래밍 언어에 따라 다릅니다.

C check_error는 CHECKERR로 다시 정의되며 util.c 파일에 위치합니다.

C 샘플: LOBLOC.SQC

```
        #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "util.h"

 EXEC SQL INCLUDE SQLCA;

#define  CHECKERR(CE_STR)   if (check_error (CE_STR, &sqlca) != 0) return 1;

int main(int argc, char *argv[]) {

#ifdef DB2MAC
   char * bufptr;
  #endif

   EXEC SQL BEGIN DECLARE SECTION; 1
      char number[7];
      long deptInfoBeginLoc;
      long deptInfoEndLoc;
      SQL TYPE IS CLOB_LOCATOR resume;
      SQL TYPE IS CLOB_LOCATOR deptBuffer;
      short lobind;
      char buffer[1000]="";
      char userid[9];
      char passwd[19];
     EXEC SQL END DECLARE SECTION;

   printf( "Sample C program: LOBLOC\n" );

   if (argc == 1) {
```

```
      EXEC SQL CONNECT TO sample;
  CHECKERR ("CONNECT TO SAMPLE");
   }
   else if (argc == 3) {
      strcpy (userid, argv[1]);
      strcpy (passwd, argv[2]);
      EXEC SQL CONNECT TO sample USER :userid USING :passwd;
      CHECKERR ("CONNECT TO SAMPLE");
   }
   else {
      printf ("\nUSAGE: lobloc [userid passwd]\n\n");
      return 1;
      } /* endif */

   /* Employee A10030 is not included in the following select, because
      the lobeval program manipulates the record for A10030 so that it is
      not compatible with lobloc */

     EXEC SQL DECLARE c1 CURSOR FOR
            SELECT empno, resume FROM emp_resume WHERE resume_format='ascii'
            AND empno <> 'A00130';

   EXEC SQL OPEN c1;
   CHECKERR ("OPEN CURSOR");

   do {
      EXEC SQL FETCH c1 INTO :number, :resume :lobind;  2
      if (SQLCODE != 0) break;
      if (lobind < 0) {
         printf ("NULL LOB indicated\n");
   } else {
         /* EVALUATE the LOB LOCATOR */
         /* Locate the beginning of "Department Information" section */
     EXEC SQL VALUES (POSSTR(:resume, 'Department Information'))
            INTO :deptInfoBeginLoc;
         CHECKERR ("VALUES1");

         /* Locate the beginning of "Education" section (end of "Dept.Info" */
     EXEC SQL VALUES (POSSTR(:resume, 'Education'))
            INTO :deptInfoEndLoc;
         CHECKERR ("VALUES2");

         /* Obtain ONLY the "Department Information" section by using SUBSTR */
         EXEC SQL VALUES(SUBSTR(:resume, :deptInfoBeginLoc,
            :deptInfoEndLoc - :deptInfoBeginLoc)) INTO :deptBuffer;
         CHECKERR ("VALUES3");

         /* Append the "Department Information" section to the :buffer var. */
         EXEC SQL VALUES(:buffer || :deptBuffer) INTO :buffer;
         CHECKERR ("VALUES4");
      } /* endif */
   } while ( 1 );

#ifdef DB2MAC
   /* Need to convert the newline character for the Mac */
   bufptr = &(buffer[0]);
   while ( *bufptr != '\0' ) {
      if ( *bufptr == 0x0A ) *bufptr = 0x0D;
  bufptr++;
   }
  #endif

   printf ("%s\n",buffer);

   EXEC SQL FREE LOCATOR :resume, :deptBuffer;  3
   CHECKERR ("FREE LOCATOR");

   EXEC SQL CLOSE c1;
   CHECKERR ("CLOSE CURSOR");

   EXEC SQL CONNECT RESET;
   CHECKERR ("CONNECT RESET");
   return 0;
}
/* end of program : LOBLOC.SQC */
```

COBOL 샘플: LOBLOC.SQB

```
 Identification Division.
 Program-ID. "lobloc".

 Data Division.
```

```
 Working-Storage Section.
     copy "sqlenv.cbl".
     copy "sql.cbl".
     copy "sqlca.cbl".

     EXEC SQL BEGIN DECLARE SECTION END-EXEC. 1
 01 userid            pic x(8).
 01 passwd.
   49 passwd-length   pic s9(4) comp-5 value 0.
   49 passwd-name     pic x(18).
 01 empnum            pic x(6).
 01 di-begin-loc      pic s9(9) comp-5.
 01 di-end-loc        pic s9(9) comp-5.
 01 resume            USAGE IS SQL TYPE IS CLOB-LOCATOR.
 01 di-buffer         USAGE IS SQL TYPE IS CLOB-LOCATOR.
 01 lobind            pic s9(4) comp-5.
 01 buffer            USAGE IS SQL TYPE IS CLOB(1K).
     EXEC SQL END DECLARE SECTION END-EXEC.

 77 errloc          pic x(80).

 Procedure Division.
 Main Section.
     display "Sample COBOL program: LOBLOC".

* Get database connection information.
     display "Enter your user id (default none): "
          with no advancing.
     accept userid.

     if userid = spaces
       EXEC SQL CONNECT TO sample END-EXEC
  else
       display "Enter your password : " with no advancing
       accept passwd-name.

* Passwords in a CONNECT statement must be entered in a VARCHAR
* format with the length of the input string.
     inspect passwd-name tallying passwd-length for characters
        before initial " ".

     EXEC SQL CONNECT TO sample USER :userid USING :passwd
  END-EXEC.
     move "CONNECT TO" to errloc.
     call "checkerr" using SQLCA errloc.

* Employee A10030 is not included in the following select, because
* the lobeval program manipulates the record for A10030 so that it is
* not compatible with lobloc

     EXEC SQL DECLARE c1 CURSOR FOR
              SELECT empno, resume FROM emp_resume
              WHERE resume_format = 'ascii'
              AND empno <> 'A00130' END-EXEC.

     EXEC SQL OPEN c1 END-EXEC.
     move "OPEN CURSOR" to errloc.
     call "checkerr" using SQLCA errloc.

     Move 0 to buffer-length.

     perform Fetch-Loop thru End-Fetch-Loop
        until SQLCODE not equal 0.

* display contents of the buffer.
     display buffer-data(1:buffer-length).

     EXEC SQL FREE LOCATOR :resume, :di-buffer END-EXEC. 3
     move "FREE LOCATOR" to errloc.
     call "checkerr" using SQLCA errloc.

     EXEC SQL CLOSE c1 END-EXEC.
     move "CLOSE CURSOR" to errloc.
     call "checkerr" using SQLCA errloc.

     EXEC SQL CONNECT RESET END-EXEC.
     move "CONNECT RESET" to errloc.
     call "checkerr" using SQLCA errloc.
 End-Main.
        go to End-Prog.

 Fetch-Loop Section.
     EXEC SQL FETCH c1 INTO :empnum, :resume :lobind 2
  END-EXEC.
```

```
     if SQLCODE not equal 0
     go to End-Fetch-Loop.

* check to see if the host variable indicator returns NULL.
     if lobind less than 0 go to NULL-lob-indicated.

* Value exists.  Evaluate the LOB locator.
* Locate the beginning of "Department Information" section.
     EXEC SQL VALUES (POSSTR(:resume, 'Department Information'))
              INTO :di-begin-loc END-EXEC.
     move "VALUES1" to errloc.
     call "checkerr" using SQLCA errloc.

* Locate the beginning of "Education" section (end of Dept.Info)
     EXEC SQL VALUES (POSSTR(:resume, 'Education'))
               INTO :di-end-loc END-EXEC.
     move "VALUES2" to errloc.
     call "checkerr" using SQLCA errloc.

     subtract di-begin-loc from di-end-loc.

* Obtain ONLY the "Department Information" section by using SUBSTR
     EXEC SQL VALUES (SUBSTR(:resume, :di-begin-loc,
              :di-end-loc))
              INTO :di-buffer END-EXEC.
     move "VALUES3" to errloc.
     call "checkerr" using SQLCA errloc.

* Append the "Department Information" section to the :buffer var
     EXEC SQL VALUES (:buffer || :di-buffer) INTO :buffer
  END-EXEC.
     move "VALUES4" to errloc.
     call "checkerr" using SQLCA errloc.

     go to End-Fetch-Loop.

 NULL-lob-indicated.
     display "NULL LOB indicated".

 End-Fetch-Loop. exit.

 End-Prog.
            stop run.
```

인디케이터 변수 및 LOB 위치 지정자

어플리케이션 프로그램의 일반 호스트 변수에 대해 널 값을 호스트 값으로 선택할 때, 값이 널(null)임을 나타내는 음수 값이 인디케이터 변수에 할당됩니다. 그러나, LOB 위치 지정자의 경우 인디케이터 변수의 의미는 약간 다릅니다. 위치 지정자 호스트 변수 자체는 널일 수 없으므로, 음수 인디케이터 변수는 LOB 위치 지정자에 의해 표시된 LOB 값이 널임을 나타냅니다. 인디케이터 변수 값을 사용하여 널 정보는 클라이언트에 대해 로컬로 유지됩니다. 즉, 서버는 유효한 위치 지정자를 사용하여 널 값을 추적하지 않습니다.

LOB 파일 참조 변수

파일 참조 변수는 호스트 변수와 비슷하지만, (메모리 버퍼가 아니라) IFS 파일과 자료를 주고 받는 데 사용된다는 점은 다릅니다. LOB 위치 지정자가 LOB 값을 (포함하는 것이 아니라) 나타내는 것처럼 파일 참조 변수는 파일을 나타냅니다. 데이터베이스 조회, 갱신, 삽입은 파일 참조 변수를 사용하여 단일 LOB 값을 저장(또는 검색)할 수 있습니다.

매우 큰 오브젝트의 경우 파일이 자연적인 컨테이너입니다. 대부분의 LOB는 서버의 데이터베이스로 이동되기 전에 클라이언트의 파일에 저장된 자료로 시작합니다. 파일 참조 변수 사용은 LOB 자료 이동을 돕습니다. 프로그램은 파일 참조 변수를 사용하여

LOB 자료를 IFS 파일에서 직접 데이터베이스 엔진으로 전송합니다. LOB 자료를 이동하기 위해, 어플리케이션은 호스트 변수를 사용하여 파일을 읽고 쓰는데 유틸리티 루틴을 작성할 필요가 없습니다.

주: 파일 참조 변수에 의해 참조된 파일은 프로그램이 실행되는 시스템에서(꼭 시스템에 상주할 필요는 없음) 액세스할 수 있어야 합니다. 저장된 프로시듀어의 경우 이 시스템은 서버일 것입니다.

파일 참조 변수는 BLOB, CLOB, 또는 DBCLOB의 자료 유형을 가집니다. 자료 소스(입력) 또는 자료 목표(출력)으로 사용됩니다. 파일 참조 변수는 상대 파일명 또는 파일의 완전한 경로명(후자가 권장됨)을 가질 수 있습니다. 파일명 길이는 어플리케이션 프로그램 내에서 지정됩니다. 파일 참조 변수의 자료 길이 부분은 입력되는 동안 사용되지 않습니다. 출력되는 동안 자료 길이는 어플리케이션 요청자 코드에 의해 파일로 기록되는 새로운 자료 길이로 설정됩니다.

파일 참조 변수를 사용할 때 입력과 출력에 대해 서로 다른 옵션이 있습니다. 파일 참조 변수 구조의 `file_options` 필드를 설정하여 파일에 대한 조치를 선택해야 합니다. 입력과 출력 값에 대한 필드에 할당할 수 있는 선택 값이 아래 보여집니다.

입력 파일 참조 변수를 사용할 때의 값(C에 대해 표시됨)과 옵션은 다음과 같습니다.

- **SQL_FILE_READ**(정규 파일) -- 이 옵션의 값은 2입니다. 이 옵션은 열고, 읽고, 닫을 수 있는 파일입니다. DB2는 파일을 열 때의 파일의 자료 길이(바이트)를 결정합니다. 그런 다음 DB2는 파일 참조 변수 구조의 `data_length` 필드를 통해 길이를 리턴합니다(COBOL에 대한 값은 SQL-FILE-READ입니다).

출력 파일 참조 변수를 사용할 때의 값과 옵션은 다음과 같습니다.

- **SQL_FILE_CREATE**(새로운 파일) -- 이 옵션의 값은 8입니다. 이 옵션은 새로운 파일을 작성합니다. 파일이 이미 존재하면 오류 메세지가 리턴됩니다(COBOL에 대한 값은 SQL-FILE-CREATE입니다).
- **SQL_FILE_OVERWRITE**(파일 겹쳐쓰기) -- 이 옵션의 값은 16입니다. 파일이 아직 존재하지 않으면 이 옵션은 새로운 파일을 작성합니다. 파일이 이미 존재하면, 새로운 자료는 파일의 자료를 겹쳐씁니다(COBOL에 대한 값은 SQL-FILE-OVERWRITE입니다).
- **SQL_FILE_APPEND**(파일 덧붙이기) -- 이 옵션의 값은 32입니다. 이 옵션은 파일이 존재하는 경우 파일에 자료를 덧붙입니다. 그렇지 않은 경우에는 새로운 파일을 작성합니다(COBOL에 대한 값은 SQL-FILE-APPEND입니다).

주: LOB 파일 참조 변수가 OPEN문에서 사용되면 커서가 닫힐 때까지 LOB 파일 참조 변수를 삭제하지 마십시오.

통합 파일 시스템에 대한 자세한 내용은 통합 파일 시스템을 참조하십시오.

예: 문서를 파일로 추출

이 프로그램 예에서는 CLOB 요소가 표에서 외부 파일로 검색되는 방법을 보여줍니다.

샘플 LOBFILE 프로그램 작업 방법

1. **호스트 변수를 선언합니다.** BEGIN DECLARE SECTION과 END DECLARE SECTION문은 호스트 변수 선언을 구분합니다. SQL문에서 참조될 때 호스트 변수에는 콜론(:)의 접두부가 붙습니다. CLOB FILE REFERENCE 호스트 변수를 선언합니다.
2. **CLOB FILE REFERENCE 호스트 변수가 설정됩니다.** FILE REFERENCE의 속성이 설정됩니다. 완전히 선언된 경로가 없는 파일명은 디폴트로 사용자의 현재 디렉토리에 위치합니다. 경로명이 선행 슬래시(/) 문자로 시작되지 않는 경우 규정화되지 않습니다.
3. **CLOB FILE REFERENCE 호스트 변수로 선택합니다.** resume 필드의 자료는 호스트 변수에 의해 참조된 파일명으로 선택됩니다.

CHECKERR 매크로/기능은 프로그램 외부의 오류 체크 유틸리티입니다. 이 오류 체크 유틸리티의 위치는 사용된 프로그래밍 언어에 따라 다릅니다.

C check_error는 CHECKERR로 다시 정의되며 util.c 파일에 위치합니다.

COBOL CHECKERR는 이름이 checkerr.cbl인 외부 프로그램입니다.

주: 코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

C 샘플: LOBFILE.SQC

```
       #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sql.h>
#include "util.h"

 EXEC SQL INCLUDE SQLCA;

#define  CHECKERR(CE_STR)   if (check_error (CE_STR, &sqlca) != 0) return 1;

int main(int argc, char *argv[]) {

   EXEC SQL BEGIN DECLARE SECTION; 1
      SQL TYPE IS CLOB_FILE resume;
      short lobind;
      char userid[9];
      char passwd[19];
     EXEC SQL END DECLARE SECTION;

   printf( "Sample C program: LOBFILE\n" );

   if (argc == 1) {
      EXEC SQL CONNECT TO sample;
  CHECKERR ("CONNECT TO SAMPLE");
   }
   else if (argc == 3) {
      strcpy (userid, argv[1]);
      strcpy (passwd, argv[2]);
      EXEC SQL CONNECT TO sample USER :userid USING :passwd;
      CHECKERR ("CONNECT TO SAMPLE");
   }
   else {
      printf ("\nUSAGE: lobfile [userid passwd]\n\n");
```

```
      return 1;
      } /* endif */

   strcpy (resume.name, "RESUME.TXT");   2
   resume.name_length = strlen("RESUME.TXT");
   resume.file_options = SQL_FILE_OVERWRITE;

   EXEC SQL SELECT resume INTO :resume :lobind FROM emp_resume   3
      WHERE resume_format='ascii' AND empno='000130';

      if (lobind < 0) {
      printf ("NULL LOB indicated \n");
   } else {
      printf ("Resume for EMPNO 000130 is in file : RESUME.TXT\n");
      } /* endif */

   EXEC SQL CONNECT RESET;
   CHECKERR ("CONNECT RESET");
   return 0;
}
/* end of program : LOBFILE.SQC */
```

COBOL 샘플: LOBFILE.SQB

```
 Identification Division.
 Program-ID. "lobfile".

 Data Division.
 Working-Storage Section.
     copy "sqlenv.cbl".
     copy "sql.cbl".
     copy "sqlca.cbl".

     EXEC SQL BEGIN DECLARE SECTION END-EXEC.  1
 01 userid            pic x(8).
 01 passwd.
   49 passwd-length   pic s9(4) comp-5 value 0.
   49 passwd-name     pic x(18).
 01 resume            USAGE IS SQL TYPE IS CLOB-FILE.
 01 lobind            pic s9(4) comp-5.
     EXEC SQL END DECLARE SECTION END-EXEC.

 77 errloc          pic x(80).

 Procedure Division.
 Main Section.
     display "Sample COBOL program: LOBFILE".

* Get database connection information.
     display "Enter your user id (default none): "
          with no advancing.
     accept userid.

     if userid = spaces
       EXEC SQL CONNECT TO sample END-EXEC
  else
       display "Enter your password : " with no advancing
       accept passwd-name.

* Passwords in a CONNECT statement must be entered in a VARCHAR
* format with the length of the input string.
     inspect passwd-name tallying passwd-length for characters
        before initial " ".

     EXEC SQL CONNECT TO sample USER :userid USING :passwd
  END-EXEC.
     move "CONNECT TO" to errloc.
     call "checkerr" using SQLCA errloc.

     move "RESUME.TXT" to resume-NAME.                              2
     move 10 to resume-NAME-LENGTH.
     move SQL-FILE-OVERWRITE to resume-FILE-OPTIONS.

     EXEC SQL SELECT resume INTO :resume :lobind  3
              FROM emp_resume
              WHERE resume_format = 'ascii'
              AND empno = '000130' END-EXEC.
     if lobind less than 0 go to NULL-LOB-indicated.

     display "Resume for EMPNO 000130 is in file : RESUME.TXT".
     go to End-Main.
```

```
NULL-LOB-indicated.
    display "NULL LOB indicated".

End-Main.
    EXEC SQL CONNECT RESET END-EXEC.
    move "CONNECT RESET" to errloc.
    call "checkerr" using SQLCA errloc.
End-Prog.
          stop run.
```

예: CLOB 열에 자료 삽입

다음 C 프로그램 세그먼트의 경로 설명에서

- userid는 사용자 중 한 명에 대한 디렉토리를 나타냅니다.
- dirname은 『userid』의 서브디렉토리를 나타냅니다.
- filnam.1은 사용자가 표에 삽입하려는 문서 중 하나의 이름일 수 있습니다.
- clobtab는 CLOB 자료 유형의 표 이름입니다.

다음 예는 :hv_text_file에 의해 참조되는 정규 파일의 자료를 CLOB 열에 삽입하는 방법을 보여줍니다.

```
strcpy(hv_text_file.name, "/home/userid/dirname/filnam.1");
     hv_text_file.name_length = strlen("/home/userid/dirname/filnam.1");
     hv_text_file.file_options = SQL_FILE_READ; /* this is a 'regular' file */

     EXEC SQL INSERT INTO CLOBTAB
       VALUES(:hv_text_file);
```

LOB 열의 배치 표시 화면

실제 파일(PF) 멤버 표시(DSPPFM)와 같은 CL 명령어를 사용하여 LOB 열을 가진 표의 자료 행이 표시되는 경우, 해당 행에 저장된 LOB 자료는 표시되지 않습니다. 그 대신 데이터베이스는 LOB 열에 대해 특별한 값을 표시합니다. 이 특별한 값의 배치는 다음과 같습니다.

- 13에서 28바이트의 16진수 0
- *POINTER로 시작하여 공백이 뒤따르는 16바이트

값의 첫 번째 부분의 바이트 수는 값의 두 번째 부분을 16바이트 경계에 맞추기 위한 필요한 숫자로 설정됩니다.

예를 들어, ColumnOne Char(10), ColumnTwo CLOB(40K), ColumnThree BLOB(10M)의 세 열을 갖는 표가 있다고 가정합니다. 이 표에 DSPPFM을 발행하려면, 자료의 각 행은 다음과 같이 표시되어야 합니다.

- ColumnOne의 경우: 10바이트가 문자 자료로 채워집니다.
- ColumnTwo의 경우: 22바이트는 16진수 0으로 채워지고 16바이트는 '*POINTER'로 채워집니다.
- ColumnThree의 경우: 16바이트는 16진수 0으로 채워지고 16바이트는 '*POINTER'로 채워집니다.

이런 식으로 LOB 열을 표시하는 전체 명령 집합은 다음과 같습니다.

- 실제 파일(PF) 멤버 표시(DSPPFM)
- TOFILE 키워드에 대해 *PRINT 값이 지정되었을 경우 파일 복사(CPYF)
- 저널 표시(DSPJRN)
- 저널 항목 검색(RTVJRNE)
- ENTFMT 키워드에 대해 *TYPE1, *TYPE2, *TYPE3, *TYPE4 값이 지정된 경우 저널 항목 수신(RCVJRNE)

LOB 열의 저널 항목 배치

두 명령은 저널된 LOB 자료를 사용자가 주소 지정할 수 있게 하는 버퍼를 리턴합니다.

- ENTFMT 키워드에 대해 *TYPEPTR 값이 지정된 경우, 저널 항목 수신(RCVJRNE) CL 명령
- 저널 항목 검색(QjoRetrieveJournalEntries) API

이 항목의 LOB 열 배치는 다음과 같습니다.

- 0 내지 15바이트의 16진수 0
- '00'x로 설정된 1바이트의 시스템 정보
- 아래의 포인터에 의해 주소가 지정된 LOB 자료의 길이가 있는 4바이트
- 16진수 0의 8바이트
- 저널 항목에 저장된 LOB 자료를 가리키는 포인터가 있는 16바이트

이 배치의 첫 부분은 LOB 자료를 가리키는 포인터를 16바이트 경계에 맞추기 위한 것입니다. 이 영역의 바이트 수는 LOB 열에 선행하는 열의 길이에 따라 달라집니다. 이 첫 부분의 길이를 계산하는 방법의 예에 대해서는 LOB 열의 표시 배치에 대한 위의 섹션을 참조하십시오.

LOB 열의 저널 처리에 대한 자세한 정보는 저널링 주제를 참조하십시오.

사용자 정의 기능(UDF)

사용자 정의 기능은 사용자가 자신의 SQL 확장을 작성할 수 있는 메카니즘입니다. DB2와 함께 제공되는 내장 기능은 유용한 기능 집합이지만 사용자의 모든 요구사항을 만족시키지는 않을 것입니다. 따라서 다음과 같은 이유로 SQL을 확장할 필요가 있습니다.

- **사용자 정의**.

  사용자 어플리케이션에 특정한 기능은 DB2에는 존재하지 않습니다. 기능이 단순한 변형, 간단한 계산이든, 또는 복잡한 분석이든, 작업을 수행하기 위해 UDF를 사용할 수 있습니다.

- **유연성**.

DB2 내장 기능은 사용자가 어플리케이션에 포함시키려는 다양성을 허용하지 않을 수 있습니다.

- **표준화**.

  사용자 사이트의 많은 프로그램은 같은 기본 기능 집합을 구현하지만 모든 구현 프로그램은 조금씩 다릅니다. 따라서 사용자가 수신하는 결과의 일치성을 확신할 수 없습니다. 이 기능을 UDF에서 일단 정확하게 구현해 놓으면, 모든 이들 프로그램은 SQL에서 직접 같은 구현 프로그램을 사용할 수 있으며 일치된 결과를 제공합니다.

- **오브젝트 관계형 지원**.

  240 페이지의 『사용자 정의된 고유한 유형(UDT)』에서 설명했듯이, UDT는 DB2의 안전성을 증가시키고 기능을 확장하는데 매우 유용할 수 있습니다. 작동을 제공하고 유형을 캡슐화하여 UDF는 UDT에 대한 *메*소드로 작용합니다.

뿐만 아니라, SQL UDF는 대형 오브젝트와 DataLink 유형을 지원합니다. 데이터베이스는 이 자료 유형으로 작업하는 데 유용한 여러 가지 내장 기능을 제공하지만, SQL UDF는 사용자가 이 영역에서 데이터베이스의 기능을 향상시키고 더 잘 조작할 수 있는 방법을 제공합니다(요구되는 특별한 기능 포함).

자세한 내용은 다음의 절을 참조하십시오.

- 『UDF를 사용하는 이유』
- 225 페이지의 『UDF 개념』
- 227 페이지의 『UDF 구현』
- 228 페이지의 『UDF 등록』
- 229 페이지의 『저장 및 복원 고려사항』
- 229 페이지의 『예: UDF 등록』
- 233 페이지의 『UDF 사용』

UDF를 사용하는 이유

DB2 어플리케이션을 작성하는 경우, 원하는 조치나 작업을 구현할 때 다음 선택을 할 수 있습니다.

- UDF로
- 사용자 어플리케이션의 서브루틴이나 기능으로

새로운 조작을 사용자 어플리케이션의 서브루틴이나 기능으로 구현하는 것이 쉽게 보이지만, 그래도 몇 가지를 고려해야 합니다.

- **재사용**.

  새로운 조작이 사이트의 다른 사용자 또는 프로그램의 기능을 향상시킬 수 있는 것이라면 UDF는 이를 재사용하도록 도울 수 있습니다. 뿐만 아니라, 데이터베이스의 모든 사용자가 사용할 수 있는 표현식이 있는 어느 SQL에서나 직접 기능을 호출할 수 있습니다. 데이터베이스는 기능 인수의 많은 자료 유형 승격을 자동으로 허용합

니다. 예를 들면, DECIMAL에서 DOUBLE로의 승격을 사용하여, 데이터베이스는 사용자 기능이 다른, 그러나 호환되는 자료 유형에 적용되도록 허용합니다.

새로운 기능을 정상 기능으로 구현하는 것이 쉬워 보일 것입니다(사용자는 기능을 DB2에 정의할 필요가 없습니다). 이렇게 하면, 다른 모든 관심있는 어플리케이션 개발자에게 알려주어야 하며, 사용할 수 있도록 기능을 효율적으로 패키지화해야 합니다. 그러나, 이러한 과정은 데이터베이스에 액세스하기 위해 정상적으로 명령 행 프로세서(CLP)를 사용하는 것과 같은 대화형 사용자를 무시하게 됩니다. 그러나, 프로그램 내에서만 사용되도록 작성된 기능은 연관된 프로그램을 갖고 있지 않은 (대화형) 사용자를 무시합니다. 여기에는 ODBC, JDBC 등과 같은 많은 클라이언트 뿐만 아니라 STRSQL, STRQM, RUNSQLSTM과 같은 명령이 포함됩니다. CLP 사용자는 기능이 데이터베이스 내의 UDF가 아니면 이 기능을 사용할 수 없습니다. 이는 또한 다시 컴파일되지 않는 SQL을 사용하는 다른 툴(Visualizer와 같은)에도 적용됩니다.

- 성능.

  특별한 경우, 어플리케이션이 아니라 데이터베이스 엔진에서 직접 UDF를 호출하면 상당한 성능 향상을 가져 옵니다. 이후의 처리를 위해 기능이 자료의 규정에서 사용되는 경우에 이를 알 수 있습니다. 기능이 행 선택 처리에서 사용될 때 이러한 일이 발생합니다.

  일부 자료를 처리하려는 간단한 시나리오를 고려합니다. SELECTION_CRITERIA() 기능으로 표현될 수 있는 선택 기준을 충족시킬 수 있습니다. 사용자 어플리케이션은 다음과 같은 SELECT문을 발행할 수 있습니다.

  ```
  SELECT A, B, C FROM T
  ```

  각 행을 수신할 때 자료에 대해 SELECTION_CRITERIA를 실행하여 자료를 더 처리하는데 관심이 있는지 결정합니다. 여기서, 표 T의 모든 행은 다시 어플리케이션으로 전달되어야 합니다. 그러나, SELECTION_CRITERIA()가 UDF로 구현되면, 사용자 어플리케이션은 다음 명령문을 발행할 수 있습니다.

  ```
  SELECT C FROM T WHERE SELECTION_CRITERIA(A,B)=1
  ```

  이 경우, 관심있는 행과 열이 어플리케이션과 데이터베이스의 인터페이스를 통해 전달됩니다.

UDF가 기능을 향상시킬 수 있는 다른 경우는 대형 오브젝트(LOB)를 처리할 때입니다. LOB 유형 중 하나의 값에서 어떤 정보를 추출하는 기능을 가정합니다. 이 추출을 데이터베이스에서 바로 수행하고 추출된 값만 다시 어플리케이션으로 전달할 수 있습니다. 이는 전체 LOB 값을 다시 어플리케이션에 전달하여 추출을 수행하는 것보다 더 효율적입니다. 이 기능을 UDF로 패키지화하여 얻을 수 있는 성능 향상은 특정한 상황에 따라 아주 클 수 있습니다(LOB 위치 지정자를 사용하여 LOB의 일부를 추출할 수도 있습니다. 비슷한 시나리오에 대한 예는 216 페이지의 『인디케이터 변수 및 LOB 위치 지정자』를 참조하십시오).

- **오브젝트 지향.**

  UDF를 사용하여 *고유한 유형*이라고도 불리는 사용자 정의 고유한 유형(UDT)의 작동을 구현할 수 있습니다. UDT에 대한 자세한 내용은 240 페이지의 『사용자 정의된 고유한 유형(UDT)』을 참조하십시오. UDT에 대한 추가 세부사항 및 여기에 설명되어 있는 *캐스트 능력*의 주요 개념은 SQL 참조서의 CREATE DISTINCT TYPE 문을 참조하십시오. 고유한 유형을 만들 때, 고유한 유형과 그 소스 유형 사이의 캐스트 기능이 자동으로 제공됩니다. 또한 =, >, < 등과 같은 비교 연산자도 소스 유형에 따라 제공됩니다. 추가적인 작동도 스스로 제공해야 합니다. 고유한 유형의 모든 사용자가 쉽게 액세스할 수 있는 데이터베이스의 고유한 유형의 작동을 유지하는 것이 제일 좋습니다. 그러므로 UDF를 구현 메카니즘으로 사용할 수 있습니다.

  예를 들면, 1MB BLOB 이상으로 정의된 BOAT 고유한 유형이 있다고 가정합니다. 유형 CREATE문은 다음과 같습니다.

  ```
  CREATE DISTINCT TYPE BOAT AS BLOB(1M)
  ```

  BLOB은 다양한 항해 스펙과 몇 개의 그림을 포함합니다. 보트의 *크기*를 비교할 수 있습니다. 그러나, BLOB 소스 유형 이상으로 정의된 고유한 유형으로 사용자를 위해 자동으로 생성된 비교 연산자를 갖지는 못합니다. 선택한 측정 단위를 기준으로 한 보트가 다른 보트보다 큰지를 결정하는 BOAT_COMPARE 기능을 구현할 수 있습니다. 이는 배수량, 전체 길이, 미터법의 톤수, 또는 BOAT 오브젝트를 바탕으로 한 다른 연산일 수 있습니다. 다음과 같이 BOAT_COMPARE 기능을 작성할 수 있습니다.

  ```
  CREATE SQL FUNCTION BOAT_COMPARE (BOAT, BOAT) RETURNS INTEGER ...
  ```

  사용자 기능이 리턴 값이

  – 1이면 첫 번째 BOAT가 더 큽니다.

  – 2이면 두 번째가 더 큽니다.

  – 0이면 같습니다.

  보트를 비교하기 위해 이 기능을 SQL 코드에서 사용할 수 있습니다. 다음 표를 작성한다고 가정합니다.

  ```
  CREATE TABLE BOATS_INVENTORY (
    BOAT_ID        CHAR(5),
    BOAT_TYPE      VARCHAR(25),
    DESIGNER       VARCHAR(40),
    OWNER          VARCHAR(40),
    DESIGN_DATE    DATE,
    SPEC           BOAT,
    ...                  )

  CREATE TABLE MY_BOATS (
    BOAT_ID        CHAR(5),
    BOAT_TYPE      VARCHAR(25),
    DESIGNER       VARCHAR(40),
    DESIGN_DATE    DATE,
  ```

```
ACQUIRE_DATE   DATE,
ACQUIRE_PRICE  CANADIAN_DOLLAR,
CURR_APPRAISL  CANADIAN_DOLLAR,
SPEC           BOAT,
...                  )
```

다음 SQL SELECT문을 실행할 수 있습니다.

```
SELECT INV.BOAT_ID, INV.BOAT_TYPE, INV.DESIGNER,
  INV.OWNER, INV.DESIGN_DATE
  FROM BOATS_INVENTORY INV, MY_BOATS MY
  WHERE MY.BOAT_ID = '19GCC'
  AND BOAT_COMPARE(INV.SPEC, MY.SPEC) = 1
  AND INV.DESIGNER = MY.DESIGNER
```

이 간단한 예는 MY_BOATS의 특정 보트보다 큰, 같은 디자이너의 BOATS_INVENTORY에서 모든 보트를 리턴합니다. 데이터베이스 서버에서 비교가 되므로 예에서는 관심있는 행만 어플리케이션에 다시 전달합니다. 사실, 여기서는 자료 유형 BOAT의 어떤 값도 전달하는 것을 피합니다. BOAT가 1MB의 BLOB 자료 유형을 바탕으로 하므로 이는 기억장치와 성능이 상당히 향상된 것입니다.

UDF 개념

다음은 UDF를 코딩하기 전에 알아야 할 중요한 개념을 설명합니다.

기능명

- 기능의 전체 이름

  *SQL 명명을 사용하는 기능의 전체 이름은 <schema-name>.<function-name>입니다.

  *SYS 명명에서 기능의 전체 이름은 <schema-name>/<function-name>입니다. 기능명은 DML문의 *SYS 명명을 사용하여 규정될 수 없습니다.

  기능을 참조하는 곳에서는 어디서나 이 전체 이름을 사용할 수 있습니다. 예를 들면 다음과 같습니다.

  ```
  QGPL.SNOWBLOWER_SIZE    SMITH.FOO    QSYS2.SUBSTR    QSYS2.FLOOR
  ```

  그러나 참조하려는 기능을 DB2가 판별해야하는 경우에는 <schema-name>.을 생략할 수도 있습니다. 예를 들면 다음과 같습니다.

  ```
  SNOWBLOWER_SIZE    FOO    SUBSTR    FLOOR
  ```

- 경로

  경로 개념은 schema-name이 지정되지 않을 때 발생하는 *규정되지* 않은 참조의 DB2 분석의 중심 개념입니다. 기능을 참조하는 DDL문의 경로 사용은 SQL 참조서에서 해당 CREATE FUNCTION문의 설명을 참조하십시오. 경로는 스키마명의 순서가 있는 리스트입니다. UDT 뿐만 아니라 UDF에 대한 규정되지 않은 참조를 해석하기 위한 스키마 집합을 제공합니다. 기능 참조가 경로의 하나 이상의 스키마의 기능과 일치하는 경우, 경로의 스키마 순서가 이 일치를 해석하기 위해 사용됩니다. 경로는

정적 SQL에 대한 사전컴파일과 바인드 명령의 SQLPATH 옵션에 의해 설정됩니다. 경로는 동적 SQL에 대한 SET PATH 명령문에 의해 설정됩니다. 활성 그룹에서 실행되는 첫 번째 SQL문이 SQL 명명으로 실행되면 경로는 다음의 디폴트 값을 가집니다.

```
"QSYS","QSYS2","<ID>"
```

이는 정적 및 동적 SQL에 모두 적용되는데, 여기서 <ID>는 현재 명령문의 권한부여 ID를 나타냅니다.

활성 그룹에서 실행되는 첫 번째 SQL문이 시스템 명명으로 실행되면 디폴트 경로는 *LIBL입니다.

- 과부하 기능명.

  기능명은 *과부하*될 수 있습니다. 과부하는 같은 스키마에서도 복수 기능이 같은 이름을 가질 수 있습니다. 그러나 두 기능이 동일한 *서명*을 가질 수는 없습니다. 기능 서명은 모든 기능 매개변수의 정의된 자료 유형이 정의된 순서로 이어진 규정된 기능명입니다. 과부하 기능의 예에 대해서는 230 페이지의 『예: BLOB 스트링 탐색』을 참조하십시오. 서명 및 기능 분석에 대한 자세한 정보는 SQL 참조서의 기능 주제를 참조하십시오.

- 기능 해석

  참조가 규정된 참조이건 규정되지 않은 참조이건, 모든 기능 참조에 대해 *가장 잘 맞는* 것을 선택하기 위해 과부하 사실과 기능 경로를 고려하는 것이 *기능 해석 알고리즘*입니다. 내장 기능을 포함한 모든 기능이 기능 선택 알고리즘에 의해 처리됩니다. 기능 분석 알고리즘은 기능 유형은 고려하지 않습니다.그러므로 참조 사용 시 스칼라 함수를 필요로 하거나 또는 그 반대의 경우라 하더라도 표 기능은 *최적 맞춤* 기능으로 분석될 수 있습니다.

  경로, SET PATH문 및 기능 해상도 알고리즘의 개념은 SQL 참조서에서 자세히 설명합니다. SQLPATH 사전컴파일 옵션은 명령 부록에서 설명됩니다.

- 기능 유형

  여러 유형의 기능이 있습니다.

  – *내장*. 데이터베이스에서 제공하는 기능입니다. SUBSTR()은 한 예입니다.
  – *시스템 생성*. DISTINCT TYPE이 작성될 때 데이터베이스 엔진에 의해 내포적으로 생성된 기능입니다. 이 기능은 DISTINCT TYPE과 기본 유형 사이의 캐스트 조작을 제공합니다.
  – *사용자 정의*. 사용자에 의해 작성되어 데이터베이스에 등록된 기능입니다.

  뿐만 아니라, 각 기능은 *스칼라* 또는 *열* 또는 *표* 함수로 더 세밀하게 분류할 수 있습니다.

  *스칼라 함수*는 호출될 때마다 단일 값 응답을 리턴합니다. 예를 들면, 내장 기능 SUBSTR()은 스칼라 함수이며 다른 많은 내장 기능도 스칼라 함수입니다. 시스템 생

성 기능은 항상 스칼라 함수입니다. 스칼라 UDF는 외부(C와 같은 프로그래밍 언어, 또는 SQL--SQL 기능에서 코딩됨), 또는 소스(기존 기능의 구현 프로그램 사용) UDF 입니다.

*열 함수*는 유사 값 집합(자료의 한 열)을 수신하여 이 값 집합으로부터 단일 값 응답을 리턴합니다. 이 함수는 DB2에서 *총계 함수*라고도 불립니다. 일부 내장 기능은 열 함수입니다. 열 함수의 예는 내장 기능 AVG()입니다. 외부 UDF는 열 함수로 정의될 수 없습니다. 그러나, 소스 UDF가 내장 열 함수 중 하나를 소스로 한 경우 열 함수로 정의됩니다. 후자가 고유한 유형에 대해 유용합니다. 예를 들면, 고유한 유형 SHOESIZE가 기본 유형 INTEGER와 함께 정의되어 존재한다면 UDF, AVG(SHOESIZE)를 기존 내장 열 함수, AVG(INTEGER)을 소스로 하는 열 함수로 정의할 수 있습니다.

*표 기능*은 표를 참조하는 SQL문에 표를 리턴합니다.이것은 SELECT의 FROM절에서 참조되어야 합니다. 표 기능은 SQL 언어 처리 기능을 DB2 데이터가 아닌 데이터에 적용하거나 이러한 데이터를 DB2 표로 변환할 때 사용할 수 있습니다.예를 들어 파일을 받아 표로 변환하고 WWW에서 데이터를 샘플링한 후 표로 만들어지거나 Lotus Notes 데이터베이스에 액세스하여 메시지의 날짜, 송신자 및 텍스트와 같은 메일 메시지 정보를 리턴할 수 있습니다. 이 정보를 데이터베이스의 다른 표와 결합할 수 있습니다. 표 기능은 외부 기능 또는 SQL 기능으로 정의될 수 있습니다. 그러나 피소스(sourced) 기능으로 정의될 수는 없습니다.

UDF 구현

소스, 외부, SQL의 세 가지 유형의 UDF가 있습니다. 각 유형의 구현은 상당히 다릅니다.

- 소스 UDF. 이 UDF는 단순히 다른 기능을 참조하는 데이터베이스에 등록된 기능입니다. 실제로 이 UDF는 소스 기능을 맵핑합니다. 그러므로, 이 기능을 구현하기 위해서는 CREATE FUNCTION문을 사용하여 데이터베이스에 등록하기만 하면 됩니다.
- 외부 기능. 이 기능은 C, COBOL, 또는 RPG와 같은 고급 언어로 작성된 서비스 프로그램과 프로그램에 대한 참조입니다. 일단 기능이 데이터베이스에 등록되면, 데이터베이스는 기능이 DML문에서 참조될 때마다 프로그램 또는 서비스 프로그램을 호출합니다. 따라서, UDF 작성자가 고급 언어를 알고 이 고급 언어에서 코드를 개발할 줄 알 뿐만 아니라 프로그램과 데이터베이스 사이의 인터페이스를 이해할 것을 외부 UDF는 요구합니다. 외부 기능 작성에 대한 자세한 내용은 259 페이지의 제 13 장 『사용자 정의 기능(UDF) 작성』을 참조하십시오.
- SQL UDF. SQL UDF는 전적으로 SQL 언어로 작성된 기능입니다. 이 '코드'는 실제로는 CREATE FUNCTION문에 삽입된 SQL문입니다. SQL UDF는 여러 가지 장점을 가집니다.
  - SQL로 작성되므로 전달하기 쉽습니다.

– 데이터베이스와 기능 사이의 인터페이스를 정의할 때는 실제 매개변수 전달의 상세한 내용에 대해 신경쓸 필요없이 SQL 선언문을 사용하면 됩니다.
– 대형 오브젝트, 자료 링크, UDT를 매개변수로 전달하여 그 다음에는 기능에서 이들을 조작하도록 허용합니다. SQL 기능에 대한 자세한 내용은 259 페이지의 제 13 장 『사용자 정의 기능(UDF) 작성』에서 설명됩니다.

UDF를 사용하려면 다음 단계를 수행하십시오.

1. DB2로 UDF 등록. 작성되는 UDF 유형에 관계없이 모두 CREATE FUNCTION 문을 사용하여 데이터베이스에 등록해야 합니다. 소스 기능의 경우, 이 등록 단계만으로 기능을 데이터베이스에 정의할 수 있습니다. SQL UDF의 경우 CREATE FUNCTION문만으로 기능을 정의할 수 있지만, CREATE문의 구문이 더 복잡합니다(실제 SQL 실행 코드가 포함됨). 외부 UDF의 경우 CREATE FUNCTION 문은 기능을 데이터베이스에 등록하기만 합니다. 기능을 실제로 구현하는 지원 코드는 별도로 작성해야 합니다. 자세한 내용은 『UDF 등록』을 참조하십시오.
2. UDF 디버깅. 259 페이지의 제 13 장 『사용자 정의 기능(UDF) 작성』을 참조하십시오.

이 단계가 성공적으로 끝나면 사용자 UDF는 CREATE VIEW와 같은 DDL(data definition language) 또는 DML(data manipulation language) 명령문에서 사용되도록 준비가 완료됩니다.

UDF 등록

데이터베이스가 기능을 인식하고 사용하기 전에 UDF는 데이터베이스에 등록되어야 합니다. CREATE FUNCTION 명령문을 사용하여 UDF를 등록할 수 있습니다.

명령문을 사용하여 DETERMINISTIC, ALLOW PARALLEL 및 RETURNS NULL ON NULL INPUT과 같은 옵션과 함께 프로그램의 언어와 이름을 지정할 수 있습니다. 이 옵션으로 기능의 목적과 데이터베이스 호출을 최적화할 수 있는 방법을 데이터베이스에 더 자세히 알려줄 수 있습니다.

실제 코드를 작성하고 완전히 테스트한 후 DB2에 외부 UDF를 등록해야 합니다. 실제로 작성하기 전에 UDF를 정의할 수 있습니다. 그러나, UDF를 실행하는 과정에서 문제를 피하기 위해 등록하기 전에 다양하게 작성하여 테스트하도록 권장됩니다. UDF 테스트에 대한 내용은 259 페이지의 제 13 장 『사용자 정의 기능(UDF) 작성』을 참조하십시오.

UDF 등록의 예는 229 페이지의 『예: UDF 등록』을 참조하십시오.

저장 및 복원 고려사항

ILE 외부 프로그램이나 서비스 프로그램과 연관된 외부 기능이 작성될 때기능 속성을 연관된 프로그램이나 서비스 프로그램 오브젝트에 저장하려 합니다.*PGM 또는 *SRVPGM 오브젝트가 저장되었다가 이 시스템 또는 다른 시스템으로 복원되는 경우 카탈로그는 자동으로 이 속성을 사용하여 갱신됩니다.함수의 속성을 저장할 수 없는 경우 카탈로그를 자동으로 갱신할 수 없으며 사용자는 새 시스템에 외부 기능을 작성해야 합니다. 속성은 다음과 같은 제한사항에 맞추어 외부 기능 주제에 저장될 수 있습니다.

- 외부 프로그램 라이브러리는 QSYS 또는 QSYS2가 아니어야 합니다.
- 외부 프로그램은 CREATE FUNCTION문이 발행될 때 존재해야 합니다.
- 외부 프로그램은 ILE *PGM 또는 *SRVPGM 오브젝트여야 합니다.
- 외부 프로그램 또는 서비스 프로그램에는 최소한 하나의 SQL문이 들어 있어야 합니다.

오브젝트를 갱신하지 못해도 기능은 작성할 수 있습니다.

예: UDF 등록

다음 예는 UDF가 등록될 수 있는 다양한 전형적인 상황을 보여줍니다. 예에는 다음이 포함됩니다.

- 예: 지수화
- 예: 스트링 탐색
- 예: UDT로 스트링 탐색
- 예: UDT 매개변수에 의한 외부 기능
- 예: UDT에서 AVG
- 예: 계산
- 예: 문서 ID를 리턴하는 표 기능

주: 코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

예: 지수화

외부 UDF를 작성하여 부동 소수점 값을 지수화하고, MATH 스키마에 등록하려고 한다고 가정합니다.

```
CREATE FUNCTION MATH.EXPON (DOUBLE, DOUBLE)
  RETURNS DOUBLE
EXTERNAL NAME 'MYLIB/MYPGM(MYENTRY)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
  RETURNS NULL ON NULL INPUT
  ALLOW PARALLEL
```

이 예에서 시스템은 RETURNS NULL ON NULL INPUT 디폴트 값을 사용합니다. 인수 중 하나가 널(null)인 경우 결과가 널이기를 원하므로 이는 바람직합니다. scratchpad를 요구하지 않고 마지막 호출이 필요하지 않으므로, NO SCRATCHPAD와 NO FINAL CALL 디폴트 값이 사용됩니다. EXPON이 병렬이 아닐 이유가 없으므로, ALLOW PARALLEL 값이 지정됩니다.

예: 스트링 탐색

동료 Willie가 인수로 전달된 주어진 CLOB 값 내에서 역시 전달된 인수인 주어진 짧은 문자열의 존재를 찾는 UDF를 작성했습니다. UDF는 스트링을 찾은 경우 CLOB 내의 스트링 위치를 리턴하고, 찾지 못했으면 0을 리턴합니다.

또한 Willie는 FLOAT 결과를 리턴하는 기능을 작성했습니다. 사용자가 SQL에서 언제 이 기능이 사용되는지 안다면, 이 기능은 항상 INTEGER를 리턴해야 합니다. 다음 기능을 작성할 수 있습니다.

```
CREATE FUNCTION FINDSTRING (CLOB(500K), VARCHAR(200))
  RETURNS INTEGER
  CAST FROM FLOAT
  SPECIFIC "willie_find_feb95"
  EXTERNAL NAME 'MYLIB/MYPGM(FINDSTR)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
  RETURNS NULL ON NULL INPUT
```

UDF에서 실제로는 FLOAT 값을 리턴하지만 사용자는 UDF를 사용했던 명령문에 값을 리턴하기 전에 이 값을 INTEGER로 캐스트하기를 원하다는 것을 지정하기 위해 CAST FROM절이 사용됩니다. SQL 참조서에서 설명한 바와 같이, INTEGER 내장 기능은 이러한 캐스트를 수행할 수 있습니다. 또한 사용자는 이 기능에 대한 자신만의 특정한 이름을 주고 나중에 이를 DDL에서 참조할 수도 있습니다(231 페이지의 『예: UDT로 스트링 탐색』 참조). UDF가 널 값을 처리하기 위해 작성된 것이 아니므로, RETURNS NULL ON NULL INPUT을 사용합니다. scratchpad가 없으므로, NO SCRATCHPAD와 NO FINAL CALL 디폴트 값이 사용됩니다. FINDSTRING이 병렬이 아닐 이유가 없으므로, ALLOW PARALLEL 디폴트 값이 사용됩니다.

예: BLOB 스트링 탐색

사용자는 이 기능이 CLOB에서 뿐만 아니라 BLOB에서도 작동되기를 원하므로 BLOB을 첫 번째 매개변수로 갖는 다른 FINDSTRING을 정의합니다.

```
CREATE FUNCTION FINDSTRING (BLOB(500K), VARCHAR(200))
  RETURNS INTEGER
  CAST FROM FLOAT
  SPECIFIC "willie_fblob_feb95"
  EXTERNAL NAME 'MYLIB/MYPGM(FINDSTR)'
  LANGUAGE C
```

```
PARAMETER STYLE DB2SQL
NO SQL
DETERMINISTIC
NO EXTERNAL ACTION
```

이 예에서는 UDF 이름의 과부하를 보여주며, 복수 UDF를 같은 본문에서 공유할 수 있음을 보여줍니다. BLOB은 CLOB에 할당될 수 없지만 같은 소스 코드를 사용할 수는 있습니다. DB2와 UDF 사이의 BLOB과 CLOB에 대한 프로그래밍 인터페이스는 같으므로 위의 예에서 프로그래밍 문제는 없습니다. 길이는 자료 뒤에 따라옵니다. DB2는 특정한 기능 본문을 사용하는 UDF가 같은 본문을 사용하는 다른 UDF와 일치하는지 체크하지 않습니다.

예: UDT로 스트링 탐색

이 예는 이전 예의 연속입니다. 230 페이지의 『예: BLOB 스트링 탐색』의 FINDSTRING 기능에 만족하지만, 소스 유형 BLOB으로 고유한 유형 BOAT를 정의했다고 가정합니다. FINDSTRING이 BOAT 자료 유형을 가진 값에 대해서도 조작하기를 원하여 다른 FINDSTRING 기능을 작성합니다. 이 기능은 230 페이지의 『예: BLOB 스트링 탐색』의 BLOB 값에 대해 조작을 하는 FINDSTRING을 소스로 사용합니다. 이 예에서 FINDSTRING은 더욱 과부하됩니다.

```
CREATE FUNCTION FINDSTRING (BOAT, VARCHAR(200))
  RETURNS INT
  SPECIFIC "slick_fboat_mar95"
  SOURCE SPECIFIC "willie_fblob_feb95"
```

이 FINDSTRING 기능은 230 페이지의 『예: BLOB 스트링 탐색』의 FINDSTRING과는 다른 서명을 가지므로 이름을 과부하하는데 문제가 없습니다. 사용자는 나중에 DDL에서 참조할 수도 있으므로 자신만의 특정한 이름을 주기를 원합니다. SOURCE 절을 사용하고 있으므로, EXTERNAL NAME절 또는 기능 속성을 지정하는 관련된 키워드 중 하나를 사용할 수 없습니다. 이 속성을 소스 기능에서 가져옵니다. 마지막으로, 사용하는 소스 기능을 식별하기 위해, 특정 기능명이 명시적으로 230 페이지의 『예: BLOB 스트링 탐색』에 제공됩니다. 이는 규정되지 않은 참조이므로 이 소스 기능이 상주하는 스키마는 기능 경로에 있어야 하며, 그렇지 않으면 참조가 해석되지 않습니다.

예: UDT 매개변수에 의한 외부 기능

사용자는 BOAT를 가져와서 디자인 속성을 조사하여 캐나다 달러로 보트의 비용을 생성하는 다른 UDF를 작성했습니다. 내부적으로는 노동 비용을 독일 마르크, 일본 엔, 미국 달러로 산정할 수 있지만, 이 기능은 보트 제작비용을 요구되는 화폐인 캐나다 달러로 빌드해야 합니다. 이는 현재의 환율 정보를 DB2 외부에서 관리되는 exchange_rate 파일에서 가져와야 하며, 응답은 이 파일의 기능에 따라 달라짐을 의미합니다. 이는 기능을 NOT DETERMINISTIC으로 만듭니다.

```
CREATE FUNCTION BOAT_COST (BOAT)
  RETURNS INTEGER
  EXTERNAL NAME 'MYLIB/COSTS(BOATCOST)'
  LANGUAGE C
```

```
PARAMETER STYLE DB2SQL
NO SQL
NOT DETERMINISTIC
NO EXTERNAL ACTION
```

CAST FROM과 SPECIFIC은 지정되지 않았지만 NOT DETERMINISTIC은 지정되었습니다.

예: UDT에서 AVG

이 예는 CANADIAN_DOLLAR 고유한 유형에 대해 AVG 열 기능을 구현합니다. CANADIAN_DOLLAR의 정의에 대해서는 242 페이지의 『예: 화폐』를 참조하십시오. 강력한 유형 때문에 내장 AVG 기능을 고유한 유형에서 사용할 수 없습니다. CANADIAN_DOLLAR에 대한 소스 유형은 DECIMAL이었으므로, AVG(DECIMAL) 내장 기능에서 이를 소스로 하여 AVG를 구현합니다. 이렇게 하기 위해서는 DECIMAL을 CANADIAN_DOLLAR로 캐스트할 수 있어야 하고 반대로도 캐스트할 수 있어야 합니다. 그러나 DECIMAL이 CANADIAN_DOLLAR에 대한 소스 유형이므로 이 캐스트가 작동됩니다.

```
CREATE FUNCTION AVG (CANADIAN_DOLLAR)
  RETURNS CANADIAN_DOLLAR
  SOURCE "QSYS2".AVG(DECIMAL(9,2))
```

사용자의 SQL 경로에 다른 AVG 기능이 숨어있을 수 있으므로, SOURCE절에서 사용자가 기능 이름을 규정하였습니다.

예: 계산

간단한 계산 기능은 처음에는 1을 리턴하고 매번 호출될 때마다 결과를 1씩 증가시킵니다. 이 기능은 SQL 인수를 갖지 않으며, 응답이 호출할 때마다 달라지므로 정의에 의해 NOT DETERMINISTIC 기능입니다. SCRATCHPAD를 사용하여 리턴된 최종 값을 저장하십시오. 매번 호출될 때마다 기능은 이 값을 증가시키고 리턴합니다.

```
CREATE FUNCTION COUNTER ()
  RETURNS INT
  EXTERNAL NAME 'MYLIB/MYFUNCS(CTR)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  NOT DETERMINISTIC
  NOT FENCED
  SCRATCHPAD 4
  DISALLOW PARALLEL
```

제공된 매개변수 정의가 없으며 빈 괄호만이 나타납니다. 위의 기능은 SCRATCHPAD를 지정하며 NO FINAL CALL의 디폴트 스펙을 사용합니다. 이 경우 scratchpad의 크기는 단지 4바이트로만 설정되는데, 이는 카운터로 충분합니다. COUNTER 기능에서 하나의 scratchpad만이 적절히 조작되어 사용되면 되므로, DB2가 이 기능을 병렬로 조작하지 못하도록 DISALLOW PARALLEL이 추가됩니다.

예: 문서 ID를 리턴하는 표 기능

주어진 주제 영역(첫 번째 매개변수)과 일치하고 주어진 스트링(두 번째 매개변수)을 포함하는 텍스트 관리 시스템의 각 알려진 문서에 대하여 단일 문서 식별자 열로 구성되는 행을 리턴하는 표 기능을 작성했습니다. 이 UDF는 텍스트 관리 시스템 기능을 사용하여 신속하게 문서를 식별합니다.

```
CREATE FUNCTION DOCMATCH (VARCHAR(30), VARCHAR(255))
  RETURNS TABLE (DOC_ID CHAR(16))
  EXTERNAL NAME 'DOCFUNCS/UDFMATCH(udfmatch)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
  NOT FENCED
  SCRATCHPAD
  NO FINAL CALL
  DISALLOW PARALLEL       CARDINALITY 20
```

단일 세션의 문맥 내에서 항상 동일한 표를 리턴하므로 DETERMINISTIC으로 정의됩니다. DOCMATCH로부터의 출력을 정의하는 RETURNS절에 컬럼 이름 DOC_ID가 포함되므로 주의하십시오. FINAL CALL은 각 표 기능에 지정되지 않아야 합니다. 또한 DISALLOW PARALLEL 키워드는 표 기능이 병렬로 작동할 수 없으므로 추가됩니다. DOCMATCH로부터의 출력 *크기*가 매우 가변적이지만 CARDINALITY 20은 대표 값으로 지정되어 DB2 옵티마이저가 최선의 결정을 내리는 데 도움이 됩니다.

일반적으로, 이 표 기능은 다음과 같이 문서 텍스트가 들어 있는 표로 결합에 사용됩니다.

```
SELECT T.AUTHOR, T.DOCTEXT
  FROM DOCS AS T, TABLE(DOCMATCH('MATHEMATICS', 'ZORN''S LEMMA')) AS F
  WHERE T.DOCID = F.DOC_ID
```

FROM절의 표 기능을 지정하기 위한 특수 구문(TABLE 키워드)에 주의하십시오. 이 호출에서 DOCMATCH() 표 기능은 ZORM'S LEMMA를 참조하는 각 MATHEMATICS 문서에 대한 단일 열 DOC_ID가 포함된 행을 리턴합니다. 이들 DOC_ID 변수는 작성자의 이름 및 문서 텍스트를 검색하여 마스터 문서 표에 결합됩니다.

UDF 사용

스칼라와 열 UDF는 표현식이 유효한 SQL문 내 어디에서나 대부분 호출될 수 있습니다. 표 UDF는 SELECT의 FROM절에 호출될 수 있습니다. 그러나 UDF 사용에는 몇 가지 제한사항이 있습니다.

- UDF와 시스템 생성 기능은 체크 제한사항에서 지정될 수 없습니다. 또한 점검 제한 조건은 내장 기능인 DLVALUE, DLURLPATH, DLURLPATHONLY, DLURLSCHEME, DLURLCOMPLETE, DLURLSERVER, ATAN2,

DIFFERENCE, RADIANS, RAND, SOUNDEX, NOW, CURDATE 및 CURTIME에 대한 참조를 포함할 수도 없습니다.

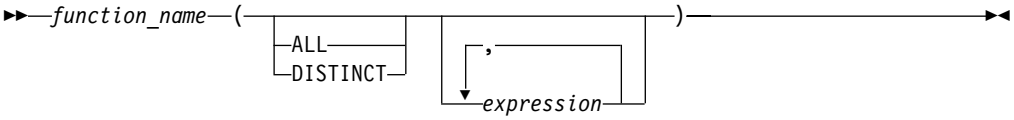
- SQL문이 읽기 전용이 아니고 임시 처리(ALWCPYDTA(*YES) 또는 (*OPTIMIZE))를 허용하지 않으면, 외부 UDF, SQL UDFS, 그리고 내장 기능 DLVALUE, DLURLPATH, DLURLPATHONLY, DLURLSCHEME, DLURLCOMPLETE, DLURLSERVER는 ORDER BY 또는 GROUP BY절에서 참조될 수 없습니다.

SQL 참조서는 이들 모든 문맥을 자세히 설명합니다. 그러나 이 절의 설명과 예는 상대적으로 간단한 SELECT문 문맥을 중점적으로 보여주지만, 이 문맥에만 사용이 제한되는 것은 아닙니다.

*경로*와 *기능 해석* 알고리즘의 중요성과 사용 요약에 대해서는 225 페이지의 『UDF 개념』을 참조하십시오. SQL 참조서에는 이 두 개념에 대한 자세한 내용이 들어 있습니다. 기능을 참조하는 DML(Data Manipulation Language) 해석시 기능 해석 알고리즘을 사용하므로, 이 알고리즘이 작동되는 방법을 이해하는 것이 중요합니다.

기능 참조

기능이 UDF이건 또는 내장 기능이건 기능에 대한 참조에는 다음 구문이 포함됩니다.



위에서 function_name은 규정되었거나 규정되지 않은 기능명일 수 있습니다. *SYS 명명 규칙을 사용할 경우 기능은 규정될 수 없습니다. 인수는 0에서 90까지의 숫자이며, 다음을 포함한 표현식일 수 있습니다.

- 규정되었거나 규정되지 않은 열 이름
- 상수
- 표현식
- 기능
- 호스트 변수
- CAST 기능과 함께 매개변수 마커
- 스칼라 부속 선택
- 특수 레지스터

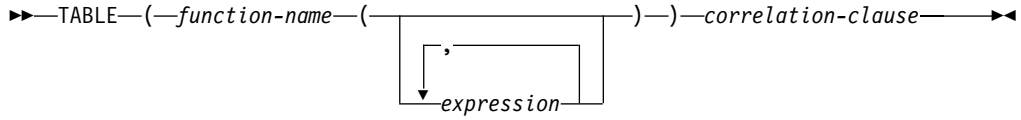
인수의 위치는 중요하므로 의미가 정확하도록 기능 정의에 순응해야 합니다. 인수의 위치와 기능 정의는 기능 본문에 순응해야 합니다. DB2는 기능 정의에 더 잘 일치시키기 위해 인수를 뒤섞으려고 시도하지 않으며, 개별 기능 매개변수의 의미를 결정하려고 시도하지도 않습니다.

기능 호출 예

기능 호출의 몇 가지 유효한 예는 다음과 같습니다.

```
AVG(FLOAT_COLUMN)
BLOOP(COLUMN1)
BLOOP(FLOAT_COLUMN + CAST(? AS INTEGER))
BLOOP(:hostvar :indicvar)
BRIAN.PARSE(CONCAT(CHAR_COLUMN,USER), 1, 0, 0, 1)
CTR()
FLOOR(FLOAT_COLUMN)
PABLO.BLOOP(A+B)
PABLO.BLOOP(:hostvar)
"search_schema"(CURRENT PATH, 'GENE')
SUBSTR(COLUMN2,8,3)
QSYS2.FLOOR(AVG(EMP.SALARY))
QSYS2.AVG(QSYS2.FLOOR(EMP.SALARY))
QSYS2.SUBSTR(COLUMN2,11,LENGTH(COLUMN3))
```

표 기능 참조



위 다이어그램에서, function_name은 규정되거나 규정되지 않은 기능명일 수 있습니다. *SYS 또는 *SQL 명명 규칙을 사용할 때 사용자 정의 표 기능이 규정될 수 있으므로 주의하십시오.

표 기능 호출의 몇 가지 유효한 예는 다음과 같습니다.

```
TABLE(TABFUNC()) X
TABLE(BLOOP_TAB(:hostvar, COL1+COL2))
NEW_TABLE TABLE(SCHEMA1.BLOOP_TAB2()) X
```

기능에 매개변수 마커 또는 널 사용

매개변수 마커와 널 값에는 중요한 제한사항이 관련되어 있습니다. 다음을 간단하게 코드화할 수 없습니다.

```
BLOOP(?)
```

또는

```
BLOOP(NULL)
```

기능 분석에서는 인수가 어떤 자료 유형이 될지 모르므로 참조를 분석할 수 없습니다. CAST 스펙을 사용하여 기능 분석이 사용할 수 있는 INTEGER와 같은 매개변수 마커나 널 값에 대한 유형을 제공할 수 있습니다.

```
BLOOP(CAST(? AS INTEGER))
```

또는

```
BLOOP(CAST(NULL AS INTEGER))
```

규정된 기능 참조 사용

규정된 기능 참조를 사용하는 경우, DB2의 일치되는 기능에 대한 탐색을 해당 스키마로 제한합니다. 예를 들어, 다음 명령문이 있습니다.

```
SELECT PABLO.BLOOP(COLUMN1) FROM T
```

스키마 PABLO의 BLOOP 기능만이 고려됩니다. 사용자 SERGE가 BLOOP 기능을 정의했거나, 내장 BLOOP 기능이 있는지 여부는 문제가 되지 않습니다. 사용자 PABLO가 해당 스키마에서 두 개의 BLOOP 기능을 정의했습니다.

```
CREATE FUNCTION BLOOP (INTEGER) RETURNS ...
CREATE FUNCTION BLOOP (DOUBLE) RETURNS ...
```

따라서 BLOOP이 PABLO 스키마 내에서 과부하되고 기능 선택 알고리즘은 인수, column1의 자료 유형에 따라 최선의 BLOOP을 선택합니다. 이 경우, PABLO.BLOOP은 둘 다 숫자 인수를 취하며 column1이 숫자 유형 중 하나가 아니면 명령문이 실패합니다. 다른 한편으로는 column1이 SMALLINT 또는 INTEGER이면, 기능 선택이 첫 번째 BLOOP으로 해석되는 반면, column1이 DECIMAL 또는 DOUBLE이면, 두 번째 BLOOP이 선택됩니다.

이 예에 대해 몇 가지 지적할 것이 있습니다.

1. 이 예는 인수 승격을 보여줍니다. 첫 번째 BLOOP이 INTEGER 매개변수로 정의되었지만, 사용자는 SMALLINT 인수로 전달할 수 있습니다. 기능 선택 알고리즘은 내장 자료 유형 사이의 승격을 지원하고(자세한 내용은 SQL 참조서를 참조하십시오) DB2는 적절한 자료 값 변환을 수행합니다.
2. 어떤 이유로 사용자가 SMALLINT 또는 INTEGER 인수와 함께 두 번째 BLOOP을 호출하기를 원하는 경우, 다음과 같이 명령문에서 명시적인 조치를 취해야 합니다.

   ```
   SELECT PABLO.BLOOP(DOUBLE(COLUMN1)) FROM T
   ```

3. 또는, DECIMAL 또는 DOUBLE 인수와 함께 첫 번째 BLOOP을 호출하기를 원하는 경우, 정확한 목적이 무엇인지에 따라 명시적 조치를 선택할 수 있습니다.

   ```
   SELECT PABLO.BLOOP(INTEGER(COLUMN1)) FROM T
   SELECT PABLO.BLOOP(FLOOR(COLUMN1)) FROM T
   ```

   SQL 참조서에서 기타 이러한 기능에 대해 조사해야 합니다. INTEGER 기능은 QSYS2 스키마에서 내장 기능입니다.

규정되지 않은 기능 참조 사용

규정된 기능 참조를 사용하는 대신 규정되지 않은 기능 참조를 사용하면, DB2의 일치되는 기능에 대한 탐색에서는 참조를 규정하기 위해 일반적으로 기능 경로를 사용합니다. DROP FUNCTION 또는 COMMENT ON FUNCTION 기능의 경우, 기능이 *SQL 명명, 또는 *SYS 명명에 대한 *LIBL에 대해 규정되지 않았다면 현재 권한부여 ID를 사용하여 참조가 규정됩니다. 따라서, *사용자가 사용자 기능의 경로, 현재 기*

능 경로의 스키마에 있는 충돌하는 기능(있는 경우)을 아는 것이 중요합니다. 예를 들어, 사용자가 PABLO이고 정적 SQL문이 다음과 같다고 가정합니다. 여기서 COLUMN1은 INTEGER 자료 유형입니다.

```
SELECT BLOOP(COLUMN1) FROM T
```

236 페이지의 『규정된 기능 참조 사용』에 인용된 두 개의 BLOOP 기능을 작성하였고, 그 중 하나를 선택합니다. 다음 디폴트 기능 경로가 사용되고 QSYS 또는 QSYS2 내에 충돌하는 BLOOP이 없으면, column1이 INTEGER이므로 첫 번째 BLOOP이 선택됩니다.

```
"QSYS","QSYS2","PABLO"
```

그러나, 사용자가 다른 목적으로 이전에 작성한 사전컴파일과 바인딩을 위한 스크립트를 사용하고 있다는 것을 잊어버렸다고 가정합니다. 이 스크립트에서, 사용자는 명시적으로 SQLPATH 매개변수를 코딩하여 현재 작업에는 적용되지 않는 다른 이유로 다음의 기능 경로를 지정했습니다.

```
"KATHY","QSYS","QSYS2","PABLO"
```

Kathy가 작업하기 위해 BLOOP 기능을 작성한 경우, 기능 선택은 Kathy의 기능을 아주 잘 해석할 수 있으며 사용자의 명령문도 오류 없이 실행됩니다. DB2는 사용자가 스스로의 작업을 잘 알고 수행하고 있다고 가정하므로 통지하지 않습니다. 정확하지 않은 출력과 명령문을 식별하여 필요한 정정을 하는 것은 사용자의 책임입니다.

기능 참조의 요약

규정된 기능 참조나 규정되지 않은 기능 참조 모두를 위해, 기능 선택 알고리즘은 모든 적용가능한 기능, 즉 다음의 항목을 가진 사용자 정의 기능과 내장 기능을 모두 조사합니다.

- 주어진 이름
- 기능 참조의 인수와 같은 수의 정의된 매개변수
- 대응하는 인수 유형에서 승격가능하거나 동일한 각 매개변수

(*적용가능한 기능*이란 규정된 참조의 경우 *명명된 스키마의 기능*을 의미하며, 규정되지 않은 참조의 경우 *기능 경로의 스키마의 기능*을 의미합니다.) 알고리즘은 정확한 일치를 찾거나, 찾지 못하는 경우에는 이 기능 중에서 가장 근사하는 일치를 찾습니다. 규정되지 않은 참조의 경우에만, 두 개의 동일하게 근사한 일치가 다른 스키마에서 발견되면 현재 기능 경로가 결정적인 요소로 사용됩니다. 알고리즘에 대한 세부사항은 SQL 참조서에 나타나 있습니다.

236 페이지의 『규정된 기능 참조 사용』의 끝 부분의 예에서 나타난 흥미있는 특징은 같은 피처에 대한 참조라도 *기능 참조가 내포될 수 있다는 사실*입니다. 이는 UDF 뿐만 아니라 내장 기능에 대해서도 일반적으로 사실입니다. 그러나, 열 함수가 관련되면 몇 가지 제한이 있습니다.

이전 예를 다시 나타내면 다음과 같습니다.

```
CREATE FUNCTION BLOOP (INTEGER) RETURNS INTEGER ...
CREATE FUNCTION BLOOP (DOUBLE) RETURNS INTEGER ...
```

이제 다음 DML문을 고려합니다.

```
SELECT BLOOP( BLOOP(COLUMN1)) FROM T
```

column1이 DECIMAL 또는 DOUBLE 열이면, 내부 BLOOP 참조는 위에서 정의된 두 번째 BLOOP로 해석됩니다. 이 BLOOP이 INTEGER를 리턴하므로 외부 BLOOP은 첫 번째 BLOOP으로 해석됩니다.

또는, column1이 SMALLINT 또는 INTEGER 열이면, 내부 bloop 참조는 위에서 정의된 첫 번째 BLOOP로 해석됩니다. 이 BLOOP이 INTEGER를 리턴하므로 외부 BLOOP도 첫 번째 BLOOP으로 해석됩니다. 이 경우, 같은 기능에 대한 내포된 참조를 보는 것입니다.

기능 참조에 대해 중요한 몇 가지 추가 사항은 다음과 같습니다.

- SQL 오퍼레이터 중 하나의 이름으로 기능을 정의할 수 있습니다. 예를 들면, 고유한 유형 BOAT를 갖는 값에 대한 "+" 연산자에 어떤 의미를 추가할 수 있습니다. 다음과 같은 UDF를 정의할 수 있습니다.

  ```
  CREATE FUNCTION "+" (BOAT, BOAT) RETURNS ...
  ```

  그런 다음 다음과 같은 유효한 SQL문을 작성할 수 있습니다.

  ```
  SELECT "+"(BOAT_COL1, BOAT_COL2)
  FROM BIG_BOATS
  WHERE BOAT_OWNER = 'Nelson Mattos'
  ```

  이런 방법으로 >, =, LIKE, IN 등과 같은 내장 조건 연산자를 과부하하도록 허용되지 않음을 주의하십시오.

- 기능 선택 알고리즘은 특정 기능으로 해석할 때 참조 문맥을 고려하지 않습니다. 이전에서 약간 수정된 다음 BLOOP 기능을 보십시오.

  ```
  CREATE FUNCTION BLOOP (INTEGER) RETURNS INTEGER ...
  CREATE FUNCTION BLOOP (DOUBLE) RETURNS CHAR(10)...
  ```

  이제 사용자는 다음과 같은 SELECT문을 작성합니다.

  ```
  SELECT 'ABCDEFG' CONCAT BLOOP(SMALLINT_COL) FROM T
  ```

  SMALLINT 인수를 사용하여 해석된 가장 근사한 일치가 위에서 정의된 첫 번째 BLOOP이므로, CONCAT의 두 번째 연산자는 자료 유형 INTEGER로 해석됩니다. CONCAT가 스트링 인수를 요구하므로 명령문은 실패합니다. 첫 번째 BLOOP이 존재하지 않았으므로, 다른 BLOOP이 선택되고 명령문 실행은 성공적입니다.

- UDF는 LOB 유형(BLOB, CLOB, 또는 DBCLOB)을 갖는 결과나 매개변수와 함께 정의될 수 있습니다. DB2는 이러한 기능을 호출하기 전에 기억장치 내의 전체

LOB 값을, 값의 소스가 *LOB 위치 지정자* 호스트 값일지라도, 구체적으로 나타냅니다. 예를 들어, 다음 C 언어 어플리케이션의 한 부분을 살펴봅니다.

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS CLOB(150K) clob150K ;        /* LOB host var */
  SQL TYPE IS CLOB_LOCATOR clob_locator1;  /* LOB locator host var */
  char                  string[40];       /* string host var */
EXEC SQL END DECLARE SECTION;
```

호스트 변수 :clob150K 또는 :clob_locator1은 해당 매개변수가 CLOB(500K)로 정의된 기능에 대한 인수로 유효합니다. 따라서 230 페이지의 『예: 스트링 탐색』에 정의된 FINDSTRING을 참조하는 경우, 다음 두 명령문 모두 프로그램에서 유효합니다.

```
... SELECT FINDSTRING (:clob150K, :string) FROM ...
... SELECT FINDSTRING (:clob_locator1, :string) FROM ...
```

- LOB 유형 중 하나인 비 SQL UDF 매개변수 또는 결과는 AS LOCATOR 수정자로 작성될 수 있습니다. 이 경우, 전체 LOB 값은 호출하기 전에 구체적으로 결정되지 않습니다. 대신 LOB LOCATOR가 UDF에 전달됩니다.

  사용자는 이러한 기능을 LOB를 바탕으로 한 고유한 유형의 결과나 UDF 매개변수에서 사용할 수도 있습니다. 이 기능은 비SQL UDF에 제한됩니다. 이러한 기능의 인수는 정의된 유형의 LOB 값이 될 수 있으며, LOCATOR 유형 중 하나로 정의된 호스트 변수일 필요는 없습니다. 호스트 변수 위치 지정자를 인수로 사용하는 것은 UDF 매개변수와 결과 정의에서 AS LOCATOR를 사용하는 것과는 전혀 무관합니다.

- UDF는 고유한 유형을 사용하여 매개변수 또는 결과로 정의될 수 있습니다(이전 예에 나타나 있습니다). DB2는 값을 고유한 유형의 소스 자료 유형 형식으로 UDF에 전달합니다.

  호스트 변수이며, 해당 매개변수가 고유한 유형으로 정의된 UDF에 전달되는 인수로 사용되는 고유한 유형은 **사용자에 의해 고유한 유형으로 명시적으로 캐스트되어야 합니다.** 고유한 유형에 대한 호스트 언어 유형은 없습니다. DB2의 강력한 유형에서 이를 필요로 합니다. 그렇지 않으면 결과가 확실하지 않을 수 있습니다. 따라서 BLOB에 대해 정의된 BOAT 고유한 유형을 고려하고, BOAT 유형의 오브젝트를 인수로 취하는 231 페이지의 『예: UDT 매개변수에 의한 외부 기능』의 BOAT_COST UDF를 고려합니다. 다음 C 언어 어플리케이션의 한 부분에서, 호스트 변수 :ship은 BOAT_COST 기능으로 전달된 BLOB 값을 가집니다.

  ```
  EXEC SQL BEGIN DECLARE SECTION;
    SQL TYPE IS BLOB(150K) ship;
  EXEC SQL END DECLARE SECTION;
  ```

  다음 명령문은 둘다 :ship 호스트 변수를 BOAT 유형으로 캐스트하므로 BOAT_COST 기능으로 정확하게 해석됩니다.

  ```
  ... SELECT BOAT_COST (BOAT(:ship)) FROM ...
  ... SELECT BOAT_COST (CAST(:ship AS BOAT)) FROM ...
  ```

여러 BOAT 고유한 유형이 데이터베이스에 있거나 다른 스키마에 여러 BOAT UDF가 있으면, 기능 경로에 세심한 주의를 기울여야 합니다. 그렇지 않으면 결과가 확실하지 않을 수 있습니다.

사용자 정의된 고유한 유형(UDT)

사용자 정의된 고유한 유형은 내장 자료 유형의 사용 가능한 범위를 넘어 DB2 기능을 확장할 수 있게 하는 메카니즘입니다. 사용자 정의된 고유한 유형을 사용하면 업무를 모델링하고 자료의 의미 규칙을 파악하기 위해 더 이상 시스템이 제공하는 내장 자료 유형만 사용하도록 제한하지 않기 때문에 사용자에게 상당한 권한을 주는 DB2에 대한 새로운 자료 유형을 정의할 수 있습니다. 고유한 자료 유형을 사용하면 현재의 데이터베이스 유형에 대해 1대 1의 기초로 맵할 수 있습니다.

다음 주제는 UDT에 대해 더 자세히 설명합니다.

- 『UDT를 사용하는 이유』
- 241 페이지의 『UDT 정의』
- 242 페이지의 『UDT로 표 정의』
- 243 페이지의 『UDT 조작』
- 249 페이지의 『UDT, UDF 및 LOB 사이의 공동 작용』

UDT를 사용하는 이유

UDT와 연관된 여러 가지 장점이 있습니다.

1. **확장성**.

   새로운 유형을 정의하여 DB2에 의해 제공된 유형 집합을 무한히 확장하여 어플리케이션을 지원할 수 있습니다.

2. **유연성**.

   사용자 정의 기능(UDF)을 사용하여 새로운 유형에 대한 동작과 의미를 지정하여 시스템에서 사용할 수 있는 유형의 다양성을 늘일 수 있습니다.

3. **일치된 작동**.

   강력한 유형으로 UDT가 적절히 작동되도록 할 수 있습니다. 이는 UDT에서 정의된 기능만이 UDT의 인스턴스에 적용되도록 보장합니다.

4. **캡슐화**.

   UDT의 작동은 이에 적용되는 조작과 기능에 의해 제한됩니다. 어플리케이션 실행은 유형에 대해 선택한 내부 표시에 종속되지 않으므로 이는 구현 프로그램에 유연성을 제공합니다.

5. **확장가능한 작동**.

   유형에 대한 사용자 정의 기능의 정의는 사용자 UDT를 조작하기 위해 제공된 기능을 언제나 확장할 수 있습니다(221 페이지의 『사용자 정의 기능(UDF)』 참조).

6. **오브젝트 지향 확장의 기초.**

   UDT는 대부분의 오브젝트 지향 피처의 기초입니다. 이는 오브젝트 지향 확장의 가장 중요한 단계를 표현합니다.

UDT 정의

UDT는 표, 색인, UDF 같은 다른 오브젝트처럼 CREATE문으로 정의해야 합니다.

CREATE DISTINCT TYPE문을 사용하여 새로운 UDT를 정의합니다. 명령문 구문 및 모든 옵션의 자세한 설명은 SQL 참조서의 CREATE DISTINCT TYPE을 참조하십시오.

CREATE DISTINCT TYPE문에 대해서 다음을 알 수 있습니다.

1. 새로운 UDT 이름은 규정되었거나 규정되지 않은 이름일 수 있습니다.
2. UDT의 소스 유형은 UDT를 내부적으로 표시하기 위해 DB2에 의해 사용된 유형입니다. 이런 이유 때문에 내장 자료 유형이어야 합니다. 이전에 정의된 UDT는 다른 UDT의 소스 유형으로 사용될 수 없습니다.

UDT 정의의 일부로 DB2는 다음을 위해 항상 캐스트 기능을 생성합니다.

- 소스 유형의 표준 이름을 사용하여 UDT를 소스 유형으로 캐스트하기 위해. 예를 들어, FLOAT를 바탕으로 고유 유형을 만든 경우, DOUBLE이라고 불리는 캐스트 기능이 만들어 집니다.
- 소스 유형을 UDT로 캐스트하기 위해. UDT에 대해 추가 캐스트를 생성하는 시기에 대한 설명은 SQL 참조서를 참조하십시오.

이 기능은 조회에서 UDT 조작시 중요합니다.

규정되지 않은 UDT 해제

유형 이름 또는 기능을 제외하고, 규정되지 않은 유형 이름 또는 기능에 대한 참조를 해석하기 위해 기능 경로가 사용됩니다.

- 작성됨
- 제거됨
- 주석이 붙음

규정되지 않은 기능 참조가 해석되는 방법에 대한 정보는 236 페이지의 『규정된 기능 참조 사용』을 참조하십시오.

예: CREATE DISTINCT TYPE 사용

다음은 CREATE DISTINCT TYPE을 사용하는 예입니다.

- 예: 화폐
- 예: 재개

주: 코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

예: 화폐

다른 통화를 처리할 필요가 있는 어플리케이션을 작성하면서 이 통화가 조회에서 직접 서로 비교되거나 조작되는 것을 DB2가 허용하지 않도록 하는 경우를 가정합니다. 다른 통화의 값을 비교하려는 경우 변환이 필요합니다. 따라서 표시할 필요가 있는 각 통화에 대해 하나씩 필요한 수만큼 UDT를 정의해야 합니다.

```
CREATE DISTINCT TYPE US_DOLLAR AS DECIMAL (9,2)
CREATE DISTINCT TYPE CANADIAN_DOLLAR AS DECIMAL (9,2)
CREATE DISTINCT TYPE GERMAN_MARK AS DECIMAL (9,2)
```

예: 재개

회사에 지원한 지원자가 작성한 어플리케이션 양식을 DB2 표로 저장하면서 이 양식에서 정보를 추출하는 기능을 사용한다고 가정합니다. 이 기능은 정규 문자 스트링에 적용될 수 없으므로(왜냐하면, 기능은 리턴해야 할 정보를 찾을 수 없으므로), 사용자는 작성된 양식을 나타내기 위해 UDT를 정의합니다.

```
CREATE DISTINCT TYPE PERSONAL.APPLICATION_FORM AS CLOB(32K)
```

UDT로 표 정의

여러 가지 UDT를 정의한 후 유형이 UDT인 열이 있는 표를 정의할 수 있습니다. 다음은 CREATE TABLE을 사용하는 예입니다.

- 예: 판매
- 예: 어플리케이션 양식

주: 코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

예: 판매

여러 국가에서 회사의 판매량을 다음과 같이 저장하는 표를 정의하는 경우를 가정합니다.

```
CREATE TABLE US_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1985),
   TOTAL         US_DOLLAR)

CREATE TABLE CANADIAN_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1985),
   TOTAL         CANADIAN_DOLLAR)

CREATE TABLE GERMAN_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1985),
   TOTAL         GERMAN_MARK)
```

위의 예의 UDT는 242 페이지의 『예: 화폐』의 명령문과 같은 CREATE DISTINCT TYPE을 사용하여 작성됩니다. 위의 예에서는 체크 제한사항을 사용합니다. 점검 제한조건에 대한 정보는 151 페이지의 『체크 제한조건 추가 및 사용』을 참조하십시오.

예: 어플리케이션 양식

다음과 같이 지원자가 작성한 양식을 보존하는 순서로 표를 정의할 경우를 가정합니다.

```
CREATE TABLE APPLICATIONS
  (ID                INTEGER,
   NAME              VARCHAR (30),
   APPLICATION_DATE  DATE,
   FORM              PERSONAL.APPLICATION_FORM)
```

UDT 이름의 규정자가 권한부여 ID와 같지 않으며 디폴트 기능 경로를 변경하지 않았으므로 UDT 이름이 완전히 규정되었습니다. 유형과 기능 이름이 완전히 규정되지 않을 때마다 DB2는 현재 기능 경로에 나열된 스키마를 탐색하여 주어진 규정되지 않은 이름과 일치하는 유형 또는 기능 이름을 찾습니다.

UDT 조작

UDT와 연관된 가장 중요한 개념 중 하나는 *강력한 유형*입니다. 강력한 유형은 UDT에서 정의된 오퍼레이터와 기능만이 UDT의 인스턴스에 적용될 수 있도록 보장합니다.

사용자 UDT의 인스턴스가 정확하다는 것을 확인하기 위해 강력한 유형이 중요합니다. 예를 들어, 현재 환율에 따라 미국 달러를 캐나다 달러로 변환하는 기능을 정의한 경우, 이 기능이 잘못된 값을 리턴할 것이므로 독일 마르크를 캐나다 달러로 변환하기 위해 이와 같은 기능을 사용하지 않을 것입니다.

강력한 유형으로 인해, DB2는 예를 들면, UDT 소스 유형의 인스턴스를 가진 UDT 인스턴스를 비교하는 조회를 작성하도록 허용하지 않습니다. 같은 이유로, DB2는 다른 유형에 정의된 기능을 UDT에 적용하도록 허용하지 않을 것입니다. UDT 인스턴스를 다른 유형의 인스턴스와 비교하려면 하나 또는 다른 유형의 인스턴스를 캐스트해야 합니다. 같은 의미로, 이 기능을 UDT 인스턴스에 적용하려면 UDT 인스턴스를 UDT에 정의되지 않은 기능의 매개변수 유형으로 캐스트해야 합니다.

UDT 조작의 예는 『UDT 조작의 예』를 참조하십시오.

UDT 조작의 예

다음은 UDT를 조작하는 예입니다.

- 예: UDT와 상수 사이의 비교
- 예: UDT 사이의 캐스트
- 예: UDT와 관련된 비교
- 예: UDT와 관련된 소스 UDF
- 예: UDT와 관련된 할당
- 예: 동적 SQL의 할당

- 예: 다른 UDT와 관련된 할당
- 예: UNION에서 UDT 사용

주: 코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

예: UDT와 상수 사이의 비교

1992년 7월(7/92)에 미국에서 $100 000.00 이상 판매된 제품을 알고 싶은 경우를 가정합니다.

```
SELECT PRODUCT_ITEM
  FROM   US_SALES
  WHERE  TOTAL > US_DOLLAR (100000)
  AND    month = 7
  AND    year  = 1992
```

사용자는 미국 달러를 미국 달러의 소스 유형의 인스턴스(즉, DECIMAL)와 직접 비교할 수 없으므로 DECIMAL을 미국 달러로 캐스트하는 DB2에서 제공하는 캐스트 기능을 사용합니다. DB2에서 제공하는 다른 캐스트 기능(즉, 미국 달러를 DECIMAL로 캐스트하는 기능)을 사용할 수도 있으며 열의 합계를 DECIMAL로 캐스트할 수도 있습니다. UDT로 캐스트하건 UDT를 캐스트하건, 사용자는 캐스트를 수행하기 위해 캐스트 스펙 표기법 또는 기능 표기법을 사용할 수 있습니다. 즉, 위의 조회문을 다음과 같이 작성해야 합니다.

```
SELECT PRODUCT_ITEM
  FROM   US_SALES
  WHERE  TOTAL > CAST (100000 AS us_dollar)
  AND    MONTH = 7
  AND    YEAR  = 1992
```

예: UDT 사이의 캐스트

캐나다 달러를 미국 달러로 변환하는 UDF를 정의하려는 경우를 가정합니다. 현재 환율을 DB2 외부에서 관리되는 파일에서 얻을 수 있습니다. 캐나다 달러로 값을 가져와서 환율 파일에 액세스하여 미국 달러에 해당하는 값을 리턴하는 UDF를 정의합니다.

처음 보기에 이런 UDF는 작성하기 쉬워 보입니다. 그러나 모든 C 컴파일러가 DECIMAL 값을 지원하는 것은 아닙니다. 서로 다른 환율을 표시하는 UDT가 DECIMAL로 정의되었습니다. 사용자의 UDF는 DOUBLE 값을 수신하여 리턴해야 합니다. 왜냐하면, DOUBLE이 십진 정밀도를 손상시키지 않고 DECIMAL 값을 표시할 수 있는 C에 의해 제공되는 유일한 자료 유형이기 때문입니다. 따라서, 사용자 UDF는 다음과 같이 정의되어야 합니다.

```
CREATE FUNCTION CDN_TO_US_DOUBLE(DOUBLE) RETURNS DOUBLE
  EXTERNAL NAME 'MYLIB/CURRENCIES(C_CDN_US)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  NOT DETERMINISTIC
```

캐나다 달러와 미국 달러 사이의 환율은 두 번의 UDF 호출에서 달라지므로, NOT DETERMINISTIC으로 선언해야 합니다.

이제 질문은 캐나다 달러를 이 UDF로 어떻게 전달하여 미국 달러 값을 얻느냐 입니다. 캐나다 달러는 DECIMAL 값으로 캐스트되어야 합니다. DECIMAL 값은 DOUBLE로 캐스트되어야 합니다. 또한 리턴된 DOUBLE 값을 DECIMAL로 캐스트하고 DECIMAL 값을 미국 달러로 캐스트해야 합니다.

이러한 캐스트는 사용자가 매개변수와 리턴 유형이 소스 기능의 리턴 유형 및 매개변수와 정확하게 일치하지 않는 소스 UDF를 정의할 때 자동으로 DB2에 의해 수행됩니다. 따라서, 두 개의 소스 UDF를 정의해야 합니다. 첫 번째 UDF는 DOUBLE 값을 DECIMAL 표시로 가져옵니다. 두 번째 UDF는 DECIMAL 값을 UDT로 가져옵니다. 다음을 정의하십시오.

```
CREATE FUNCTION CDN_TO_US_DEC (DECIMAL(9,2)) RETURNS DECIMAL(9,2)
  SOURCE CDN_TO_US_DOUBLE (DOUBLE)

CREATE FUNCTION US_DOLLAR (CANADIAN_DOLLAR) RETURNS US_DOLLAR
  SOURCE CDN_TO_US_DEC (DECIMAL())
```

US_DOLLAR 기능을 US_DOLLAR(C1)(C1은 유형이 캐나다 달러인 열)에서 처럼 호출하면 다음을 호출하는 것과 같은 효과가 있습니다.

```
US_DOLLAR (DECIMAL(CDN_TO_US_DOUBLE (DOUBLE (DECIMAL (C1)))))
```

즉, C1(캐나다 달러로 표시됨)은 decimal로 캐스트되고, CDN_TO_US_DOUBLE 기능으로 전달되는 double 값으로 다시 캐스트됩니다. 이 기능은 환율 파일에 액세스하여 decimal로 캐스트되고 다시 미국 달러로 캐스트되는 double 값(미국 달러 값을 표시)을 리턴합니다.

독일 마르크를 미국 달러로 변환하는 기능도 위의 예와 비슷합니다.

```
CREATE FUNCTION GERMAN_TO_US_DOUBLE(DOUBLE)
  RETURNS DOUBLE
  EXTERNAL NAME 'MYLIB/CURRENCIES(C_GER_US)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  NOT DETERMINISTIC

CREATE FUNCTION GERMAN_TO_US_DEC (DECIMAL(9,2))
  RETURNS DECIMAL(9,2)
  SOURCE GERMAN_TO_US_DOUBLE(DOUBLE)

CREATE FUNCTION US_DOLLAR(GERMAN_MARK) RETURNS US_DOLLAR
  SOURCE GERMAN_TO_US_DEC (DECIMAL())
```

예: UDT와 관련된 비교

1989년 3월(3/89)에 캐나다와 독일보다 미국에서 더 많이 판매된 제품을 알고 싶은 경우를 가정합니다.

```
SELECT US.PRODUCT_ITEM, US.TOTAL
  FROM US_SALES AS US, CANADIAN_SALES AS CDN, GERMAN_SALES AS GERMAN
  WHERE US.PRODUCT_ITEM = CDN.PRODUCT_ITEM
  AND US.PRODUCT_ITEM = GERMAN.PRODUCT_ITEM
  AND US.TOTAL > US_DOLLAR (CDN.TOTAL)
  AND US.TOTAL > US_DOLLAR (GERMAN.TOTAL)
  AND US.MONTH = 3
  AND US.YEAR  = 1989
  AND CDN.MONTH = 3
  AND CDN.YEAR  = 1989
  AND GERMAN.MONTH = 3
  AND GERMAN.YEAR  = 1989
```

미국 달러를 캐나다 달러나 독일 마르크와 직접 비교할 수 없으므로, 캐나다 달러로 된 값을 미국 달러로 캐스트하는 UDF와 독일 마르크로 된 값을 미국 달러로 캐스트하는 UDF를 사용합니다. 같은 통화가 아니기 때문에 금액을 비교할 수 없으므로 모두 DECIMAL로 캐스트하여 변환된 DECIMAL 값을 비교할 수는 없습니다.

예: UDT와 관련된 소스 UDF

독일 마르크의 SUM을 지원하기 위해 내장 SUM 기능에 대해 소스 UDF를 정의했다고 가정합니다.

```
CREATE FUNCTION SUM (GERMAN_MARKS)
  RETURNS GERMAN_MARKS
  SOURCE SYSIBM.SUM (DECIMAL())
```

사용자는 1994년의 각 제품에 대한 독일에서 판매 합계를 알려고 합니다. 판매 합계를 미국 달러로 구하려고 합니다.

```
SELECT PRODUCT_ITEM, US_DOLLAR (SUM (TOTAL))
  FROM GERMAN_SALES
  WHERE YEAR = 1994
  GROUP BY PRODUCT_ITEM
```

위와 같은 방식으로 미국 달러에서 SUM 기능을 정의하지 않았으면 **SUM**(us_dollar (total))을 작성할 수 없습니다.

예: UDT와 관련된 할당

새로운 지원자가 작성한 양식을 데이터베이스에 저장하려는 경우를 가정합니다. 사용자는 작성된 양식을 표시하기 위해 사용된 문자 스트링 값이 들어 있는 호스트 값을 정의했습니다.

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS CLOB(32K) hv_form;
EXEC SQL END DECLARE SECTION;

/* Code to fill hv_form */

INSERT INTO APPLICATIONS
  VALUES (134523, 'Peter Holland', CURRENT DATE, :hv_form)
```

사용자는 문자 스트링을 UDT personal.application_form으로 변환하는 캐스트 기능을 명시적으로 호출하지 않았습니다. 이는 DB2가 UDT의 소스 유형의 인스턴스를 해당 UDT를 가진 목표에 할당하도록 허용하기 때문입니다.

예: 동적 SQL의 할당

246 페이지의 『예: UDT와 관련된 할당』에 주어진 명령문과 같은 명령문을 동적 SQL에서 사용하기 위해, 다음과 같은 매개변수 마커를 사용할 수 있습니다.

```
EXEC SQL BEGIN DECLARE SECTION;
  long id;
  char  name[30];
  SQL TYPE IS CLOB(32K) form;
  char command[80];
EXEC SQL END DECLARE SECTION;

/* Code to fill host variables */

strcpy(command,"INSERT INTO APPLICATIONS VALUES");
strcat(command,"(?, ?, CURRENT DATE, ?)");

EXEC SQL PREPARE APP_INSERT FROM :command;
EXEC SQL EXECUTE APP_INSERT USING :id, :name, :form;
```

DB2의 캐스트 스펙을 사용하여 매개변수 마커의 유형이 UDT 열에 할당할 수 있는 유형인 CLOB(32K)임을 DB2에 알려줍니다. 호스트 언어가 UDT를 지원하지 않으므로 UDT 유형의 호스트 변수를 선언할 수 없습니다. 따라서, 매개변수 마커의 유형이 UDT임을 지정할 수 없습니다.

예: 다른 UDT와 관련된 할당

246 페이지의 『예: UDT와 관련된 소스 UDF』의 독일 마르크를 소스로 하는 UDF와 유사한, 미국과 캐나다 달러의 SUM을 지원하기 위해 내장 SUM 기능에 대해 두 개의 피소스(sourced) UDF를 정의했다고 가정합니다.

```
CREATE FUNCTION SUM (CANADIAN_DOLLAR)
       RETURNS CANADIAN_DOLLAR
       SOURCE SYSIBM.SUM (DECIMAL())

     CREATE FUNCTION SUM (US_DOLLAR)
       RETURNS US_DOLLAR
       SOURCE SYSIBM.SUM (DECIMAL())
```

이제 수퍼바이저가 별도의 표로 각 국가와 각 제품의 연간 판매 합계(미국 달러)를 관리하도록 요청한다고 가정합니다.

```
CREATE TABLE US_SALES_94
       (PRODUCT_ITEM  INTEGER,
        TOTAL         US_DOLLAR)

     CREATE TABLE GERMAN_SALES_94
       (PRODUCT_ITEM  INTEGER,
```

```
   TOTAL         US_DOLLAR)

CREATE TABLE CANADIAN_SALES_94
  (PRODUCT_ITEM  INTEGER,
   TOTAL         US_DOLLAR)

INSERT INTO US_SALES_94
  SELECT PRODUCT_ITEM, SUM (TOTAL)
  FROM US_SALES
  WHERE YEAR = 1994
  GROUP BY PRODUCT_ITEM

INSERT INTO GERMAN_SALES_94
  SELECT PRODUCT_ITEM, US_DOLLAR (SUM (TOTAL))
  FROM GERMAN_SALES
  WHERE YEAR = 1994
  GROUP BY PRODUCT_ITEM

INSERT INTO CANADIAN_SALES_94
  SELECT PRODUCT_ITEM, US_DOLLAR (SUM (TOTAL))
  FROM CANADIAN_SALES
  WHERE YEAR = 1994
  GROUP BY PRODUCT_ITEM
```

다른 UDT들을 서로 직접 할당할 수 없으므로, UDT 사용자는 캐나다 달러와 독일 마르크로 된 값을 미국 달러로 명시적으로 캐스트합니다. UDT는 소스 유형으로만 캐스트될 수 있으므로 캐스트 스펙 구문을 사용할 수 없습니다.

예: UNION에서 UDT 사용

미국 사용자들에게 회사의 모든 제품의 모든 판매량을 보여주기 위한 조회를 제공하는 경우를 가정합니다.

```
      SELECT PRODUCT_ITEM, MONTH, YEAR, TOTAL
      FROM US_SALES
UNION
      SELECT PRODUCT_ITEM, MONTH, YEAR, US_DOLLAR (TOTAL)
      FROM CANADIAN_SALES
UNION
      SELECT PRODUCT_ITEM, MONTH, YEAR, US_DOLLAR (TOTAL)
      FROM GERMAN_SALES
```

UDT는 같은 UDT끼리만 union에서 호환될 수 있으므로 캐나다 달러를 미국 달러로, 독일 마르크를 미국 달러로 캐스트합니다. 캐스트 스펙은 UDT와 그 소스 유형 사이에서만 캐스트되도록 허용하므로 UDT 사이에서 캐스트하려면 기능 표기법을 사용해야 합니다.

UDT, UDF 및 LOB 사이의 공동 작용

이전 섹션에서 개별 DB2 오브젝트 확장(UDT, UDF, LOB)을 사용하고 정의하는 방법에 대해 설명하였습니다. 그러나, 이 섹션에서 설명되겠지만 이 세 가지 오브젝트 확장 사이에는 많은 시너지가 발생합니다.

자세한 내용은 다음 절을 참조하십시오.

- 『UDT, UDF, LOB 결합』
- 『복잡한 어플리케이션의 예』

UDT, UDF, LOB 결합

오브젝트 지향의 개념에 따라 어플리케이션 정의역의 비슷한 오브젝트는 관련된 유형으로 그룹화됩니다. 이들 유형 각각은 이름, 내부 표시, 작동을 가집니다. UDT를 사용하여, 사용자의 새로운 유형의 이름과 내부적 표시 방법을 DB2에 알려줄 수 있습니다. LOB는 새로운 유형에 대한 가능한 내부 표시 중 하나이며 복잡한 대형 구조의 경우 가장 적합한 표시입니다. UDF를 사용하여 새로운 유형의 작동을 정의할 수 있습니다. 결과적으로 UDT, UDF, LOB 사이에는 중요한 시너지가 존재합니다. 복잡한 자료 구조와 작동을 가진 어플리케이션 유형은 내부적으로 LOB로 표시되고, 작동이 UDF에 의해 구현된 UDT로 모델링됩니다. 어플리케이션 유형의 의미 무결성을 제어하는 규칙은 제한사항과 트리거로 표시됩니다. 관련된 UDT와 UDF를 더 잘 구성하고 제어하려면, 같은 스키마에 두어야 합니다.

복잡한 어플리케이션의 예

다음 예는 복잡한 어플리케이션에서 UDT, UDF, LOB를 함께 사용하는 방법을 보여줍니다.

예: UDT와 UDF 정의
예: 데이터베이스를 채우는 데 LOB 기능 사용
예: UDF를 UDT의 조회 인스턴스로 사용
예: LOB 위치 지정자를 UDT 인스턴스 조작에 사용

주: 코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

예: UDT와 UDF 정의

회사에 송신된 전자 우편을 DB2 표로 저장하려는 경우를 가정합니다. 사생활에 관련된 문제는 무시하고, 주제, 고객 주문을 받기 위해 사용된 전자 우편 서비스 횟수 등을 알기 위해 이러한 전자 우편에 대해 조회문을 작성하려고 합니다. 전자 우편은 꽤 클 수도 있고, 복잡한 내부 구조(송신자, 리사버, 주제, 날짜, 전자 우편 내용)를 가집니다. 따라서, 사용자는 소스 유형이 대형 오브젝트인 UDT를 사용하여 전자 우편을 표시하기로 결정합니다. 전자 우편의 주제, 송신자, 날짜 등을 추출하는 함수와 같은 UDF 집합을 전자 우편 유형에 대해 정의합니다. 또한 전자 우편의 내용을 탐색할 수 있는 기능도 정의합니다. 다음 CREATE문을 사용하여 위의 작업을 수행합니다.

```
CREATE DISTINCT TYPE E_MAIL AS BLOB (1M)

CREATE FUNCTION SUBJECT (E_MAIL)
  RETURNS VARCHAR (200)
  EXTERNAL NAME 'LIB/PGM(SUBJECT)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION

CREATE FUNCTION SENDER (E_MAIL)
  RETURNS VARCHAR (200)
  EXTERNAL NAME 'LIB/PGM(SENDER)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION

CREATE FUNCTION RECEIVER (E_MAIL)
  RETURNS VARCHAR (200)
  EXTERNAL NAME 'LIB/PGM(RECEIVER)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION

CREATE FUNCTION SENDING_DATE (E_MAIL)
  RETURNS DATE CAST FROM VARCHAR(10)
  EXTERNAL NAME 'LIB/PGM(SENDING_DATE)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION

CREATE FUNCTION CONTENTS (E_MAIL)
  RETURNS BLOB (1M)
  EXTERNAL NAME 'LIB/PGM(CONTENTS)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION

CREATE FUNCTION CONTAINS (E_MAIL, VARCHAR (200))
  RETURNS INTEGER
  EXTERNAL NAME 'LIB/PGM(CONTAINS)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
```

```
CREATE TABLE ELECTRONIC_MAIL
  (ARRIVAL_TIMESTAMP TIMESTAMP,
   MESSAGE E_MAIL)
```

예: 데이터베이스를 채우는 데 LOB 기능 사용

파일에서 유지보수되는 전자 우편을 DB2로 전송하여 사용자의 표를 채우는 경우를 가정합니다. 모든 전자 우편을 DB2에 저장할 때까지 다음 INSERT문을 다른 HV_EMAIL_FILE 값으로 여러 번 실행합니다.

```
EXEC SQL BEGIN DECLARE SECTION
  SQL TYPE IS BLOB_FILE HV_EMAIL_FILE;

EXEC SQL END DECLARE SECTION
  strcpy (HV_EMAIL_FILE.NAME, "/u/mail/email/mbox");
  HV_EMAIL_FILE.NAME_LENGTH = strlen(HV_EMAIL_FILE.NAME);
  HV_EMAIL_FILE.FILE_OPTIONS = 2;

EXEC SQL INSERT INTO ELECTRONIC_MAIL
  VALUES (CURRENT TIMESTAMP, :hv_email_file);
```

DB2 LOB 지원에 의해 제공된 모든 기능은 소스 유형이 LOB인 모든 UDT에 적용할 수 있습니다. 따라서, 파일 내용을 UDT 열로 할당하기 위해 LOB 파일 참조 변수를 사용했습니다. BLOB 유형의 값을 전자 우편 유형으로 변환하기 위해 캐스트 기능을 사용하지는 않았습니다. 이는 DB2가 고유 유형의 소스 유형 값을 고유 유형의 목표에 할당하도록하기 때문입니다.

예: UDF를 UDT의 조회 인스턴스로 사용

고객 주문에 상관없이 특정 고객에 의해 송신된 전자 우편 수를 알려고 하며 고객의 전자 우편 주소는 고객 표에 있다고 가정합니다.

```
SELECT COUNT (*)
  FROM ELECTRONIC_MAIL AS EMAIL, CUSTOMERS
  WHERE SUBJECT (EMAIL.MESSAGE) = 'customer order'
  AND CUSTOMERS.EMAIL_ADDRESS = SENDER (EMAIL.MESSAGE)
  AND CUSTOMERS.NAME = 'Customer X'
```

이 SQL 조회의 UDT에서 정의된 UDF를 사용합니다. 왜냐하면 이 UDF가 UDT를 조작하는 유일한 방법이기 때문입니다. 이런 의미에서 사용자 UDT 전자 우편은 완전히 캡슐화됩니다. 이는 내부 표시 및 구조가 숨겨지고 정의된 UDF에 의해서만 조작됩니다. 이 UDF는 자료의 표시를 드러낼 필요 없이 자료를 해석하는 방법을 알고 있습니다.

시장에서 제품 성능과 관련된 1994년에 수신된 회사의 모든 전자 우편의 세부사항을 알려는 경우를 가정합니다.

```
SELECT SENDER (MESSAGE), SENDING_DATE (MESSAGE), SUBJECT (MESSAGE)
  FROM ELECTRONIC_MAIL
  WHERE CONTAINS (MESSAGE,
  '"performance" AND "products" AND "marketplace"') = 1
```

관련 키워드 또는 동의어를 탐색하는 메세지 내용을 분석할 수 있는 포함 UDF를 사용했습니다.

예: LOB 위치 지정자를 UDT 인스턴스 조작에 사용

전체 전자 우편을 어플리케이션 프로그램의 호스트 변수로 전송하지 않고 특정 전자 우편에 대한 정보를 얻으려는 경우를 가정합니다(전자 우편은 꽤 클수도 있습니다). 사용자 UDT는 LOB에서 정의되므로, 이러한 목적으로 LOB 위치 지정자를 사용할 수 있습니다.

```
EXEC SQL BEGIN DECLARE SECTION
  long hv_len;
  char hv_subject[200];
  char hv_sender[200];
  char hv_buf[4096];
  char hv_current_time[26];
  SQL TYPE IS BLOB_LOCATOR hv_email_locator;
EXEC SQL END DECLARE SECTION

EXEC SQL SELECT MESSAGE
  INTO :hv_email_locator
  FROM ELECTRONIC MAIL
  WHERE ARRIVAL_TIMESTAMP = :hv_current_time;

EXEC SQL VALUES (SUBJECT (E_MAIL(:hv_email_locator))
  INTO :hv_subject;
.... code that checks if the subject of the e_mail is relevant ....
.... if the e_mail is relevant, then.............................

EXEC SQL VALUES (SENDER (CAST (:hv_email_locator AS E_MAIL)))
  INTO :hv_sender;
```

호스트 변수가 BLOB 위치 지정자 유형(UDT의 소스 유형)이므로, UDT에서 정의된 UDF의 인수로 사용되었을 때마다 BLOB 위치 지정자를 사용자 UDT로 명시적으로 변환했습니다.

DataLink 사용

DataLink 자료 유형은 데이터베이스 파일에 저장될 수 있는 자료 유형을 확장하기 위한 기본 빌딩 블록 중 하나입니다. DataLink는 열에 저장된 실제 자료는 오브젝트의 포인터일 뿐이라는 데서 유래합니다. 이 오브젝트는 이미지 파일, 소리 녹음, 테스트 파일 등, 임의의 유형일 수 있습니다. 이는 표의 행이 전형적인 자료 유형의 오브젝트에 대한 정보를 포함하는데 사용될 뿐 아니라 오브젝트 자체가 DataLink 자료 유형에 의해 참조될 수 있음을 의미합니다. 사용자는 SQL 스칼라 함수를 사용하여 오브젝트와 오브젝트가 저장된 서버의 경로를 다시 가져올 수 있습니다. DataLink 자료 유형을 사용하여 행과 오브젝트 사이에 느슨한 관계가 형성됩니다. 예를 들어, 행을 삭제하면 DataLink에 의해 참조된 오브젝트와의 관계가 끊어지지만 오브젝트 자체는 삭제되지 않을 수 있습니다.

DataLink 열과 함께 작성된 SQL 표는 실제 오브젝트를 포함하지 않고서도 오브젝트에 대한 정보를 보유하기 위해 사용될 수 있습니다. 이 개념으로 인해 사용자가 SQL 표를 사용하여 관리할 수 있는 자료 유형에 대해 유연성이 생깁니다. 예를 들어, 사용자가 서버의 통합 파일 시스템에 수천 개의 비디오 클립을 저장한 경우, SQL 표를 사용하여 이 비디오 클립에 대한 정보를 저장할 수 있습니다. 그러나 사용자는 이미 디렉토리에 오브젝트를 저장해 두었으므로, SQL 표가 실제 기억장치 바이트를 포함하지 않고 단순히 오브젝트에 대한 참조만을 포함하기를 원합니다. 좋은 해결책은 DataLink를 사용하는 것입니다. SQL 표는 전형적인 SQL 자료 유형을 사용하여 제목, 길이, 자료 등과 같은 각 클립에 대한 정보를 저장합니다. 그러나 클립 자체는 DataLink 열에 의해 참조됩니다. 표의 각 행은 선택형 주석과 오브젝트에 대한 URL을 저장합니다. 그러면 클립으로 작업하는 어플리케이션은 SQL 인터페이스를 사용하여 URL을 검색한 후 브라우저나 다른 재생 소프트웨어를 사용하여 URL에 대한 작업을 하고 비디오 클립을 재생합니다.

이러한 기술을 사용하는데는 여러 가지 장점이 있습니다.

- 통합 파일 시스템은 모든 유형의 스트림 파일을 저장할 수 있습니다.
- 통합 파일 시스템은 문자 열, 또는 LOB 열에 조차 들어 가지 않는 초대형 오브젝트를 저장할 수 있습니다.
- 통합 파일 시스템의 계층적 특성은 스트림 파일 오브젝트를 구성하고 작업하는데 알맞습니다.
- 오브젝트 바이트를 데이터베이스 외부와 통합 파일 시스템 내부에 둠으로써, 어플리케이션은 SQL 실행 시간 엔진이 조회문과 보고서를 처리하도록 허용하고, 파일 시스템이 비디오 스트림, 이미지, 텍스트 등의 표시를 처리하도록 허용하여 성능을 향상시킬 수 있습니다.

DataLink를 사용하여 오브젝트가 ″링크된″ 상태에 있는 동안 오브젝트를 제어할 수도 있습니다. 오브젝트를 참조하는 SQL 표에 행이 있는 동안 참조된 오브젝트가 삭제, 이동, 또는 이름 변경될 수 없도록 DataLink 열이 작성될 수 있습니다. 이 오브젝트는 링크되었다고 간주됩니다. 일단 참조를 포함하는 행이 삭제되면, 오브젝트는 링크되지 않게 됩니다. 이 개념을 완벽하게 이해하려면, DataLink 열을 작성할 때 지정될 수 있는 제어 레벨을 알아야 합니다. DataLink 열을 작성할 때 사용된 정확한 구문은 SQL 참조서를 참조하십시오.

DataLink에 대한 자세한 내용은 다음의 절을 참조하십시오.

- 254 페이지의『NO LINK CONTROL』
- 254 페이지의『FILE LINK CONTROL(파일 시스템 허가와 함께)』
- 254 페이지의『FILE LINK CONTROL(데이터베이스 허가와 함께)』
- 254 페이지의『DataLink로 작업하기 위해 사용된 명령』

NO LINK CONTROL

열이 NO LINK CONTROL로 작성되면, 행이 SQL 표에 추가될 때 발생하는 링크는 없습니다. URL은 구문적으로 올바르다고 검증되지만, 서버에 액세스할 수 있는지 또는 파일이 아직도 존재하는지 확인하는 검사는 없습니다.

FILE LINK CONTROL(파일 시스템 허가와 함께)

자료 링크 열이 파일 시스템(FS) 허가와 함께 FILE LINK CONTROL로 작성되면 시스템은 자료 링크 값이 유효한 서버 이름과 파일명을 가진 유효한 URL임을 검증합니다. 파일은 행이 SQL 표에 삽입될 때 존재해야 합니다. 오브젝트가 발견되면, 링크되었다고 표시됩니다. 이는 오브젝트가 링크되어 있는 동안 이동, 삭제, 또는 이름 변경될 수 없음을 의미합니다. 또한, 오브젝트는 한번 이상 링크될 수도 없습니다. URL의 서버 이름 부분이 리모트 시스템을 지정하는 경우, 해당 시스템에 액세스할 수 있어야 합니다. DataLink가 들어 있는 행이 삭제되면, 오브젝트는 링크되지 않게 됩니다. DataLink 값이 다른 값으로 갱신되면 이전 오브젝트는 링크되지 않게 되며 새로운 오브젝트가 링크됩니다.

통합 파일 시스템은 아직도 링크된 오브젝트에 대한 허가를 관리할 책임이 있습니다. 허가는 링크 또는 링크 해제 처리 동안 수정되지 않습니다. 이 옵션으로 오브젝트가 링크되는 동안 오브젝트 존재를 제어할 수 있습니다.

FILE LINK CONTROL(데이터베이스 허가와 함께)

DataLink 열이 데이터베이스 허가와 함께 FILE LINK CONTROL로 작성되면, URL이 검증되고 오브젝트에 대한 모든 기존 허가는 제거됩니다. 오브젝트의 소유권은 특별한 시스템 제공 사용자 프로파일로 변경됩니다. 오브젝트가 링크되어 있는 동안 오브젝트에 대한 유일한 액세스는 링크된 오브젝트를 가진 SQL 표에서 URL을 가져오는 것입니다. 이는 SQL에 의해 리턴된 URL에 추가된 특별한 액세스 토큰을 사용하여 처리됩니다. 액세스 토큰이 없으면 오브젝트에 액세스하려는 모든 시도는 권한 위반으로 실패합니다. 액세스 토큰이 있는 URL이 일반적인 방법(FETCH, SELECT INTO 등)에 의해 SQL 표에서 검색되면, 파일 시스템 필터는 액세스 토큰을 유효화하고 오브젝트에 대한 액세스를 허용합니다.

이 옵션은 직접적인 방법으로 오브젝트에 액세스하려는 사용자에 대해 링크된 오브젝트의 갱신을 방지할 수 있는 제어를 제공합니다. 오브젝트 액세스할 수 있는 유일한 방법은 SQL 조작에서 액세스 토큰을 얻는 것이므로, 관리자는 DataLink 열을 포함하는 SQL 표에 대한 데이터베이스 허가를 사용하여 효율적으로 링크된 오브젝트에 대한 액세스를 제어할 수 있습니다.

DataLink로 작업하기 위해 사용된 명령

DataLink 자료 유형에 대한 지원은 3 개의 다른 구성요소로 분해할 수 있습니다.

1. DB2 데이터베이스 지원에는 DATALINK라고 불리는 자료 유형이 있습니다. 이는 CREATE TABLE과 ALTER TABLE과 같은 SQL문에서 지정될 수 있습니다. 열은 널(null) 이외의 디폴트 값을 가질 수 없습니다. 자료에 대한 액세스는 SQL 인터페이스를 사용해야 합니다. 이는 DATALINK 자체가 다른 호스트 변수 유형과 호환될 수 없기 때문입니다. SQL 스칼라 함수를 사용하여 문자 형식의 DATALINK 값을 검색할 수 있습니다. 열의 값을 INSERT하고 UPDATE하기 위해 SQL에서 사용되어야 하는 DLVALUE 스칼라 함수가 있습니다.
2. 자료 링크 파일 관리자(DLFM)는 서버의 파일에 대한 링크 상태를 관리하고, 각 파일에 대한 메타 자료를 추적합니다. 이 코드는 링크, 링크 해제, 확약 제어 문제를 처리합니다. 자료 링크의 중요한 면은 DLFM이 자료 링크 열을 포함하는 SQL 표와 같은 실제 시스템에 있을 필요가 없다는 것입니다. 따라서, SQL 표는 동일한 시스템의 통합 파일 시스템이나 원격 서버의 통합 파일 시스템에 상주하는 오브젝트를 링크할 수 있습니다.
3. 링크된 오브젝트를 포함하도록 지정된 디렉토리에 있는 파일에 대해 파일 시스템이 조작을 시도할 때 DataLink 필터가 실행되어야 합니다. 이 구성요소는 파일이 링크되었는지, 그리고 선택적으로, 사용자가 파일에 액세스하도록 권한을 부여받았는지를 결정합니다. 파일명에 액세스 토큰이 포함되어 있으면, 토큰이 검증됩니다. 이 필터 처리에는 별도의 오버헤드가 있으므로, 액세스된 오브젝트가 자료 링크 '접두부' 내의 디렉토리 중 하나에 존재할 때만 실행됩니다. 접두부에 대한 아래 설명을 참조하십시오.

DataLink로 작업할 때 시스템을 적절히 구성하기 위해 취해야 할 몇 가지 단계가 있습니다.

- TCP/IP는 DataLink로 작업할 때 사용되는 모든 시스템에 구성되어야 합니다. 여기에는 링크될 오브젝트를 포함할 시스템 뿐만 아니라, DataLink 열과 함께 SQL 표가 작성될 시스템이 포함됩니다. 대부분의 경우 이는 같은 시스템입니다. 오브젝트를 참조하기 위해 사용되는 URL은 TCP/IP 서버 이름을 포함하므로 이 이름은 DataLink를 포함할 시스템에 의해 인식되어야 합니다. CFGTCP 명령은 TCP/IP 이름을 구성하거나 TCP/IP 이름 서버를 등록하기 위해 사용될 수 있습니다.
- SQL 표를 포함하는 시스템은 로컬 데이터베이스 시스템과 생략가능한 리모트 시스템을 반영하여 갱신된 관계형 데이터베이스(RDB) 디렉토리를 가져야 합니다. WRKRDBDIRE 명령은 이 디렉토리의 정보를 수정하거나 추가하기 위해 사용될 수 있습니다. 일치시키기 위해 같은 이름을 TCP/IP 서버 이름과 관계형 데이터베이스(RDB) 이름으로 사용하도록 권장됩니다.
- DLFM 서버는 링크될 오브젝트를 포함할 모든 시스템에서 시작되어야 합니다. STRTCPSVR *DLFM 명령은 DLFM 서버를 시작하기 위해 사용될 수 있습니다. DLFM 서버는 CL 명령 ENDTCPSVR *DLFM을 사용하여 종료될 수 있습니다.

일단 DLFM이 시작되면 DLFM을 구성하기 위해 필요한 몇 가지 단계가 있습니다. 이 DLFM 기능은 QShell 인터페이스에서 입력될 수 있는 실행가능한 스크립트를 통해 사

용할 수 있습니다. 대화식 쉘 인터페이스를 사용하려면 CL 명령 QSH를 사용합니다. 이로 인해 DLFM 스크립트 명령을 입력할 수 있는 명령 입력 화면이 나타납니다. 도움말 텍스트와 구문 다이어그램을 표시하기 위해 스크립트 명령 dfmadmin-help를 사용할 수 있습니다. 가장 자주 사용되는 기능의 경우 CL 명령도 제공됩니다. CL 명령을 사용하여 스크립트 인터페이스를 사용하지 않고 대부분 또는 모든 DLFM 구성을 할 수 있습니다. 사용자가 원하는대로 QSH 명령 입력 화면에서 스크립트 명령을 사용하거나 또는 CL 명령 입력 화면에서 CL 명령을 사용할 수 있습니다.

이 기능들은 시스템 관리자 또는 데이터베이스 관리자를 위한 것이므로, 모두 *IOSYSCFG 특수 권한이 필요합니다.

접두부 추가 - 접두부는 링크될 오브젝트를 포함하는 경로 또는 디렉토리입니다. 시스템에서 DLFM을 설정할 때 관리자는 DataLink에 사용될 접두부를 추가해야 합니다. 접두부를 추가하기 위해 스크립트 명령 dfmadmin -add_prefix가 사용됩니다. 접두부를 추가하는 CL 명령은 ADDPFXDLFM입니다.

예를 들면, 링크될 오브젝트를 포함하는 /mydir/datalinks/ 디렉토리는 서버 TESTSYS1에 있습니다. 관리자는 접두부를 추가하기 위해 ADDPFXDLFM PREFIX((‘/mydir/datalinks/’)) 명령을 사용합니다. URL에 대한 다음과 같은 링크

```
http://TESTSYS1/mydir/datalinks/videos/file1.mpg
```

또는

```
file://TESTSYS1/mydir/datalinks/text/story1.txt
```

의 경우 경로가 유효한 접두부로 시작하기 때문에 유효합니다.

스크립트 명령 dfmadmin -del_prefix를 사용하여 접두부를 제거하는 것도 가능합니다. 접두부 이름 내에 포함된 디렉토리 구조에 링크된 오브젝트가 없는 경우에만 실행될 수 있으므로 이는 일반적으로 사용되는 기능이 아닙니다.

호스트 데이터베이스 추가 - 호스트 데이터베이스는 링크 요구를 한 관계형 데이터베이스(RDB) 시스템입니다. DLFM이 자료 구조를 포함할 SQL 표와 같은 시스템에 있는 경우 로컬 데이터베이스 이름만이 추가되어야 합니다. 리모트 시스템으로부터 DLFM으로 링크 요구가 있는 경우 모든 이름이 DLFM과 함께 등록되어야 합니다. 호스트 데이터베이스를 추가하는 스크립트 명령은 dfmadmin -add_db이며 CL 명령은 ADDHDBDLFM입니다. 이 기능은 또한 SQL 표가 있는 라이브러리도 등록될 것을 요구합니다.

예를 들면, 이미 /mydir/datalinks/ 접두부를 추가한 서버 TESTSYS1에서 사용자는 라이브러리 TESTDB 또는 PRODDB의 로컬 시스템의 SQL 표가 이 시스템의 오브젝트 링크를 허용하기를 원합니다. 다음 명령을 사용합니다. **ADDHDBDLFM HOSTDBLIB((TESTDB) (PRODDB)) HOSTDB(TESTSYS1)**

일단 DLFM이 시작되고 접두부와 호스트 데이터베이스 이름이 등록되면, 파일 시스템 오브젝트를 링크할 수 있습니다.

제 13 장 사용자 정의 기능(UDF) 작성

사용자 정의 기능(UDF)은 소스, 외부, SQL의 세 가지 유형으로 이루어집니다. 소스 기능 UDF는 조작을 수행하기 위해 다른 기능을 호출합니다. SQL과 외부 기능 UDF는 사용자가 별도의 코드를 작성하여 실행해야 합니다. 이 장은 SQL과 외부 기능 작성에 대해 설명합니다. 외부와 SQL 기능을 작성하려면 다음을 수행해야 합니다.

- 『UDF 실행 시간 환경』을 이해하십시오.
- 데이터베이스에 알리기 위해 UDF를 등록하십시오.
- 기능을 수행하고 적합한 매개변수를 전달하기 위해 기능 코드를 작성하십시오.
- 기능을 디버그하고 테스트하십시오.

기능 기록에 대한 예는 다음을 참조하십시오.

- 272 페이지의 『예: 수의 제곱 UDF』
- 274 페이지의 『예: 카운터』

UDF 실행 시간 환경

UDF가 실행되는 환경과 그 환경의 제한에 대해 고려해야 할 몇 가지 사항이 있습니다. UDF에 대해 복합 기능을 작성하려면 이 요소를 주의깊게 고려해야 합니다.

요소는 다음과 같습니다.

- 『UDF가 실행되는 시간 길이』
- 260 페이지의 『스레드 고려사항』
- 260 페이지의 『병렬 처리』

UDF가 실행되는 시간 길이

UDF는 SQL문 실행에서 호출되는데, 이 SQL문은 일반적으로 표의 수천 행에 잠재적으로 실행되는 조회 조작입니다. 따라서 UDF는 데이터베이스의 행 레벨에서 호출되어야 합니다.

이렇게 낮은 레벨에서 호출됨에 따라, UDF가 호출될 때와 UDF가 실행되는 동안 보유되는 특정 자원(잠금과 점유)이 있습니다. 이 자원은 일차적으로 UDF를 호출하고 있는 SQL문과 관련된 색인과 표의 잠금입니다. 이 보유된 자원 때문에, UDF가 연장된 시간(분 또는 시간)이 걸리는 조작을 수행하지 않는 것이 중요합니다. 긴 시간 동안 자원을 보유하는 중요한 특성 때문에, 데이터베이스는 일정 시간 동안만 UDF가 끝나기를 기다립니다. UDF가 할당된 시간 내에 끝나지 않으면, SQL문이 실패하므로 일반 사용자에게는 상당히 치명적일 수 있습니다.

데이터베이스에 의해 사용된 디폴트 UDF 대기 시간은 정상 UDF가 끝까지 실행될 수 있는 충분한 시간보다 길어야 합니다. 그러나, 실행시간이 긴 UDF가 있어서 대기 시간을 늘리려면, 조회 INI 파일의 UDF_TIME_OUT 옵션을 사용하면 됩니다. INI 파일에 대한 자세한 내용은 *데이터베이스 성능 및 조회 최적화* 정보에서 조회 옵션 파일 QAQQINI를 참조하십시오. 그러나, UDF_TIME_OUT에 대해 지정된 값에 상관없이 데이터베이스가 초과할 수 없는 최대한의 시간 제한이 있습니다.

UDF가 실행되는 동안 자원이 보류되므로, UDF가 원래의 SQL문에 대해 할당된 표나 색인과 같은 곳에서 조작되지 않거나, 또는 조작된다면 SQL문에서 수행되고 있는 조작과 충돌하는 조작을 수행하지 않는 것이 중요합니다. 특히, UDF는 해당 표에서 행 삽입, 갱신, 또는 삭제 조작을 수행하려고 시도해서는 안됩니다.

스레드 고려사항

FENCED로 정의된 UDF는 작업을 호출한 SQL문과 같은 작업에서 실행됩니다. 그러나, UDF는 SQL문을 실행하는 스레드와는 분리되어 시스템 스레드에서 실행됩니다. 스레드에 대한 자세한 정보는 Information Center의 프로그래밍 범주에서 복수 스레드 프로그래밍에 대한 데이터베이스 고려사항을 참조하십시오.

UDF는 SQL문과 같은 작업에서 실행되므로, SQL문과 같은 환경을 많이 공유합니다. 그러나, 분리된 스레드에서 실행되므로 다음 스레드 고려사항이 적용됩니다.

- UDF는 SQL문의 스레드가 보유한 스레드 레벨 자원과 상충됩니다. 일차적으로 위에서 논의된 표 자원이 충돌합니다.
- UDF는 SQL문이 호출되었을 때 사용중인, 프로그램이 허용한 권한을 계승하지 않습니다. UDF 권한은 UDF 프로그램 자체와 연관된 권한, 또는 SQL문을 실행하는 사용자의 권한에서 발생합니다.
- UDF는 2차 스레드에서 실행되지 못하도록 블록화된 조작을 수행할 수 없습니다.
- UDF 프로그램은 명명된 활성 그룹 아래에서, 또는 호출자의 활성 그룹(ACTGRP 매개변수)에서 실행되도록 작성되어야 합니다. ACTGRP(*NEW)를 지정하는 프로그램은 UDF로 실행되도록 허용되지 않습니다.

UNFENCED로서 기능 정의에 대한 정보는 272 페이지의 『기능을 펜스 또는 펜스되지 않게 설정』을 참조하십시오.

병렬 처리

UDF는 병렬 처리를 허용하도록 정의될 수 있습니다. 이는 같은 UDF 프로그램이 동시에 복수 스레드에서 실행될 수 있음을 의미합니다. 그러므로, UDF에 대해 ALLOW PARALLEL이 지정되면 스레드 수준에서 보호되는지 확인합니다. 스레드에 대한 자세한 내용은 iSeries Information Center의 프로그래밍 범주에서 복수 스레드 프로그래밍에 대한 데이터베이스 고려사항을 참조하십시오.

사용자 정의 표 기능은 병렬로 실행될 수 없으므로, 기능을 작성할 때 DISALLOW PARALLEL을 지정해야 합니다.

기능 코드 작성

기능 코드 작성은 기능을 수행하는 SQL 또는 외부 기능을 작성하는 방법을 아는 것과 관련이 있습니다. 기능을 정확하게 정의하기 위해 기능 코드와 데이터베이스 사이의 인터페이스를 이해하는 것과, 실행가능한 프로그램을 작성할 때 패키지 옵션을 결정하는 것과도 관련됩니다.

SQL 기능 또는 외부 기능으로서 UDF를 기록할 수 있습니다. 자세한 내용은 다음을 참조하십시오.

- 『SQL 기능으로 UDF 작성』
- 262 페이지의 『외부 기능으로 UDF 작성』

또한 270 페이지의 『표 기능 고려사항』을 참조하십시오.

SQL 기능으로 UDF 작성

SQL 기능은 CREATE FUNCTION문을 사용하여 사용자가 정의, 작성, 등록한 UDF입니다. 따라서, SQL 언어를 사용해서만 작성되며, 기능 정의는 하나의(잠재적으로 큰) CREATE FUNCTION문 내에 완전히 포함됩니다. SQL 기능을 작성하면 UDF가 등록되고, 기능에 대한 실행가능한 코드가 생성되고, 매개변수가 실제로 전달되는 방법에 대한 세부사항이 데이터베이스에 정의됩니다. 따라서, 이들 기능을 작성하는 것은 분명하며 기능에 오류를 일으킬 가능성이 별로 없습니다.

스칼라 UDF

SQL 스칼라 기능에 대한 CREATE FUNCTION문은 다음과 같은 일반적인 흐름을 따릅니다.

```
CREATE FUNCTION function-name(parameters) RETURNS return-value
LANGUAGE SQL
BEGIN
   sql-statements
END
```

예를 들면, 날짜를 기준으로 우선순위를 리턴하는 기능입니다.

```
CREATE FUNCTION PRIORITY(indate DATE) RETURNS CHAR(7)
LANGUAGE SQL
BEGIN
RETURN(
      CASE   WHEN indate>CURRENT DATE-3 DAYS THEN 'HIGH'
         WHEN indate>CURRENT DATE-7 DAYS THEN 'MEDIUM'
         ELSE 'LOW'
      END
      );
END
```

그런 후에 기능은 다음과 같이 호출될 수 있습니다.

```
SELECT ORDERNBR, PRIORITY(ORDERDUEDATE) FROM ORDERS
```

표 UDF

SQL 표 기능에 대한 CREATE FUNCTION문은 다음과 같은 일반적인 흐름을 따릅니다.

```
CREATE FUNCTION function-name(parameters) RETURNS TABLE return-columns
LANGUAGE SQL
BEGIN
   sql-statements
RETURN
   select-statement
END
```

예를 들어, 날짜를 기준으로 날짜를 리턴하는 기능입니다.

```
CREATE FUNCTION PROJFUNC(indate DATE)
  RETURNS TABLE (PROJNO CHAR(6), ACTNO SMALLINT, ACTSTAFF DECIMAL(5,2),
           ACSTDATE DATE, ACENDATE DATE)
  LANGUAGE SQL
BEGIN
   RETURN SELECT * FROM PROJACT
     WHERE ACSTDATE<=indate;
  END
```

그런 후에 기능은 다음과 같이 호출될 수 있습니다.

```
SELECT * FROM TABLE(PROJFUNC(:datehv)) X
```

단 하나의 RETURN문을 가지려면 SQL 표 기능이 필요합니다.

외부 기능으로 UDF 작성

SQL 이외의 언어로 실행가능한 UDF 코드를 작성할 수도 있습니다. 이 방법은 SQL 기능보다 다소 많은 작업이 필요하지만, 사용자가 가장 효율적인 언어를 사용할 수 있는 유연성을 제공합니다. 실행가능한 코드는 프로그램 또는 서비스 프로그램에 포함될 수 있습니다.

또한 외부 기능은 Java에 기록될 수도 있습니다. 매개변수에 대한 설명은 IBM Developer Kit for Java의 Java SQL 루틴 주제를 참조하십시오.

DB2에서 외부 기능으로 인수 전달

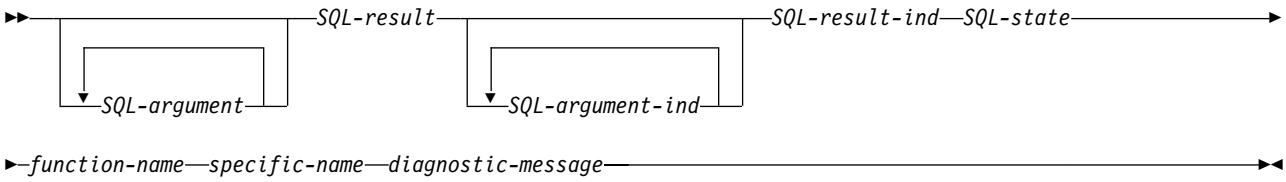
DB2는 UDF에 전달된 모든 매개변수에 대한 기억장치를 제공합니다. 그러므로 매개변수가 주소에 의해 외부 기능에 전달됩니다. 이는 프로그램에 대해 매개변수를 전달하는 일반적인 방법입니다. 서비스 프로그램의 경우 기능 코드에서 매개변수가 정확하게 정의되었는지 확인합니다.

UDF에서 매개변수를 정의하고 사용할 때, 해당 매개변수에 대해 정의된 기억장치보다 많은 기억장치가 주어진 매개변수에 대해 참조되지 않도록 주의해야 합니다. 매개변수

는 모두 같은 공간에 저장되며, 주어진 매개변수의 기억장치 공간을 초과하면 다른 매개변수 값을 겹쳐쓸 수 있습니다. 이로 인해 유효하지 않은 입력 자료를 기능이 보거나, 데이터베이스에 리턴된 값이 유효하지 않을 수 있습니다.

다음은 외부 UDF에 제공된, 지원되는 여러 가지 매개변수 방법입니다. 대부분, 외부 프로그램 또는 서비스 프로그램에 전달되는 매개변수 수가 방법마다 다릅니다.

매개변수 방법 SQL: 매개변수 방법 SQL은 산업 표준 SQL(구조화된 조회 언어)을 따릅니다. 이 매개변수 양식은 스칼라 UDF에만 사용될 수 있습니다. 매개변수 방법 SQL을 사용하여 다음과 같이 매개변수가 지정된 순서로 외부 프로그램에 전달됩니다.



SQL-argument

이 인수는 UDF를 호출하기 전에 DB2에 의해 설정됩니다. 이 값은 *n*번 반복되는데, 여기서 *n*은 기능 참조에 지정된 인수 수입니다. 이들 각 인수의 값은 기능 호출에 지정된 표현식에서 가져옵니다. 값은 기능 작성 명령문에 정의된 매개변수의 자료 유형에 표현됩니다. 주: 이 매개변수는 입력으로만 취급됩니다. UDF에 의한 매개변수 값 변경은 DB2에 의해 무시됩니다.

SQL-result

DB2에 리턴하기 전에 이 인수는 UDF에 의해 설정됩니다. 데이터베이스는 리턴 값에 대한 기억장치를 제공합니다. 매개변수가 주소에 의해 전달되므로 주소는 리턴 값이 위치할 기억장치의 주소입니다. 데이터베이스는 CREATE FUNCTION문에 정의된대로 리턴 값에 대해 필요한만큼 기억장치를 제공합니다. CAST FROM절이 CREATE FUNCTION문에 사용되면, DB2는 UDF가 CAST FROM절에 정의된 값을 리턴한다고 가정하며, 그렇지 않을 경우 DB2는 UDF가 RETURNS절에 정의된대로 값을 리턴한다고 가정합니다.

SQL-argument-ind

이 인수는 UDF를 호출하기 전에 DB2에 의해 설정됩니다. 대응하는 *SQL-argument*가 널(null)인지 아닌지를 판별하기 위해 UDF에 의해 사용될 수 있습니다. *n*번째 *SQL-argument-ind*는 앞에서 기술한 *n*번째 *SQL-argument*에 대응합니다. 각 인디케이터는 부호가 있는 두 바이트 정수로 정의됩니다. 다음 값 중 하나로 설정됩니다.

0 인수가 존재하며 널이 아닙니다.

-1 인수가 널입니다.

RETURNS NULL ON NULL INPUT으로 기능이 정의되면, UDF는 널 값을 검사할 필요가 없습니다. 그러나, CALLS ON NULL INPUT으로 기능이 정의되면, 인수가 널이 될 수 있으므로 UDF는 널 입력이 있는지 검사해야 합니다. 주: 이 매개변수는 입력으로만 취급됩니다. UDF에 의한 매개변수 값 변경은 DB2에 의해 무시됩니다.

SQL-result-ind

DB2에 리턴하기 전에 이 인수는 UDF에 의해 설정됩니다. 데이터베이스는 리턴 값에 대한 기억장치를 제공합니다. 인수는 부호가 있는 두 바이트 정수로 정의됩니다. 음수 값으로 설정되면 데이터베이스는 기능 결과를 널로 해석합니다. 0 또는 양수 값으로 설정되면, 데이터베이스는 *SQL-result*에 리턴된 값을 사용합니다. 데이터베이스는 리턴 값 인디케이터에 대한 기억장치를 제공합니다. 매개변수가 주소에 의해 전달되므로 주소는 인디케이터 값이 위치할 기억장치의 주소입니다.

SQL-state

이 인수는 SQLSTATE를 표시하는 CHAR(5) 값입니다.

이 매개변수는 '00000'로 설정된 데이터베이스에서 전달되며 기능에 대한 결과 상태로 기능에 의해 설정될 수 있습니다. 일반적으로 SQLSTATE는 기능에 의해 설정되지 않지만, 다음과 같이 오류 또는 경고를 데이터베이스에 알려주기 위해 사용될 수 있습니다.

01Hxx 기능 코드가 경고 상황을 감지했습니다. 이로 인해 SQL 경고가 발생합니다. *xx*는 가능한 스트링 중 하나입니다.

38xxx 기능 코드가 오류 상황을 감지했습니다. SQL 오류가 발생합니다. *xxx*는 가능한 스트링 중 하나입니다.

기능이 사용할 수 있는 유효한 SQLSTATE에 대한 자세한 내용은 SQL 메세지 및 코드를 참조하십시오.

function-name

이 인수는 UDF를 호출하기 전에 DB2에 의해 설정됩니다. 기능 이름을 대신하여 기능 코드가 호출될 때 그 기능 이름을 포함하는 VARCHAR(139) 값입니다.

전달되는 기능 이름 형식은 다음과 같습니다.

```
<schema-name>.<function-name>
```

호출되고 있는 정의를 기능 코드가 구분하기 위해 코드가 복수 UDF 정의에 의해 사용될 때 이 매개변수가 유용합니다. 주: 이 매개변수는 입력으로만 취급됩니다. UDF에 의한 매개변수 값 변경은 DB2에 의해 무시됩니다.

specific-name

이 인수는 UDF를 호출하기 전에 DB2에 의해 설정됩니다. 기능 이름을 대신하여 기능 코드가 호출될 때 그 기능 이름을 포함하는 VARCHAR(128) 값입니다.

function-name과 마찬가지로, 호출되고 있는 정의를 기능 코드가 구분하기 위해 코드가 복수 UDF 정의에 의해 사용될 때 이 매개변수가 유용합니다. *specific-name*에 대한 자세한 정보는 CREATE FUNCTION을 참조하십시오. 주: 이 매개변수는 입력으로만 취급됩니다. UDF에 의한 매개변수 값 변경은 DB2에 의해 무시됩니다.

diagnostic-message

이 인수는 UDF를 호출하기 전에 DB2에 의해 설정됩니다. SQLSTATE 경고 또는 오류가 UDF에 의해 통지될 때 메세지 텍스트를 다시 송신하기 위해 UDF에 의해 사용될 수 있는 VARCHAR(70) 값입니다.

이 값은 UDF로 입력시 데이터베이스에 의해 초기화되고 설명 정보와 함께 UDF에 의해 설정될 수 있습니다. SQL-state 매개변수가 UDF에 의해 설정되지 않으면 메세지 텍스트는 DB2에 의해 무시됩니다.

매개변수 방법 DB2SQL: DB2SQL 매개변수 방법을 사용하여, 매개변수 방법 SQL에 대해 전달되는 것과 같은 매개변수와 같은 순서의 매개변수가 외부 프로그램이나 서비스 프로그램으로 전달됩니다. 그러나, DB2SQL을 사용하여 추가적인 선택형 매개변수도 전달됩니다. 그러나 아래의 선택형 매개변수 중 하나 이상이 UDF 정의에 지정되면, 이 매개변수가 아래 정의된 순서로 UDF에 전달됩니다. 공통 매개변수에 대해서는 매개변수 방법 SQL을 참조하십시오. 이 매개변수는 스칼라와 표 UDF 모두에 사용될 수 있습니다.

스칼라 기능의 경우:

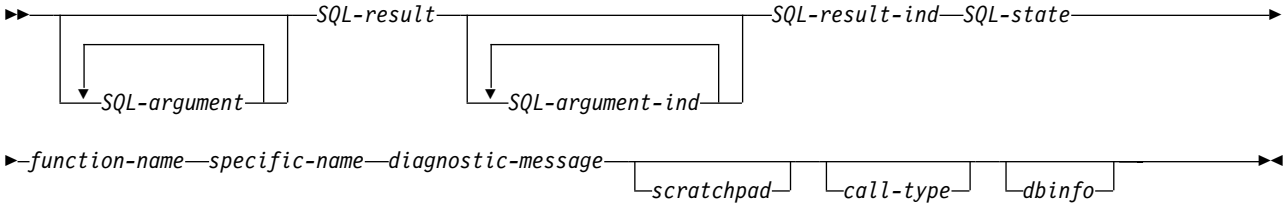
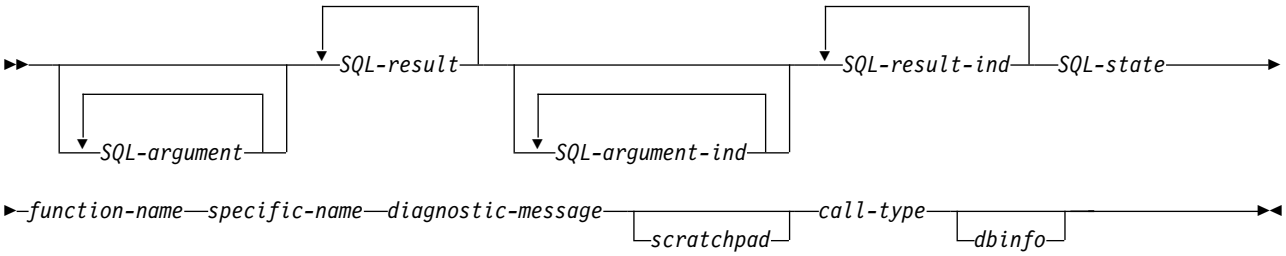


표 기능의 경우:



scratchpad

이 인수는 UDF를 호출하기 전에 DB2에 의해 설정됩니다. UDF에 대한 CREATE FUNCTION문이 SCRATCHPAD 키워드를 지정한 경우에만 이 인수가 존재합니다. 이 인수는 다음 요소가 있는 구조입니다.

- scratchpad 길이가 들어 있는 INTEGER
- 처음 UDF를 호출하기 전에 DB2에 의해 모든 2진 0으로 초기화된 실제 scratchpad

scratchpad는 UDF 호출을 통해 유지보수되므로 UDF에 의해 작업 기억장치 또는 지속적인 기억장치로 사용될 수 있습니다.

표 기능의 경우 CREATE FUNCTION에 FINAL CALL이 지정된 경우 UDF에 대한 FIRST 호출 전에 위에서와 같이 scratchpad가 초기화됩니다. 이 호출 후에 scratchpad 내용은 완전히 표 기능이 제어하게 됩니다. 그 이후 DB2는 scratchpad의 내용을 검사하거나 변경하지 않습니다. scratchpad는 각 호출에서 기능에 전달됩니다. 기능은 다시 입력할 수 있고 DB2는 scratchpad에 상태 정보를 보관합니다.

NO FINAL CALL이 지정되었거나 표 기능의 기본값으로 설정된 경우, scratchpad는 각 OPEN 호출에 대하여 위에서와 같이 초기화되며 scratchpad 내용은 OPEN 호출 간의 표 기능이 완전히 제어하게 됩니다. 이것은 결합이나 부속 조회에 사용되는 표 기능에 매우 중요할 수 있습니다. OPEN 호출에서 scratchpad의 내용을 유지보수할 필요가 있다면 CREATE FUNCTION문에 FINAL CALL을 지정해야 합니다. FINAL CALL을 지정하면 일반적인 OPEN, FETCH 및 CLOSE 호출과 함께 scratchpad 유지보수 및 자원 해제를 목적으로 표 기능도 FIRST 및 FINAL 호출을 수신하게 됩니다.

call-type

이 인수는 UDF를 호출하기 전에 DB2에 의해 설정됩니다. 스칼라 함수의 경우, UDF에 대한 CREATE FUNCTION문이 FINAL CALL 키워드를 지정한 경우에만 이 인수가 존재합니다. 그러나 표 기능의 경우 항상 존재합니다.

scratchpad 인수 다음에 나오거나 scratchpad 인수가 없는 경우 *diagnostic-message* 인수 다음에 나옵니다. 이 인수는 INTEGER 형식의 값을 받습니다.

스칼라 함수의 경우:

-1 이 명령문에 대한 UDF를 *처음* 호출하는 것입니다. 첫 호출은 모든 SQL 인수 값이 전달된다는 점에서 *정상* 호출입니다.

0 *정상* 호출입니다(모든 정상 입력 인수 값이 전달됩니다).

1 *마지막* 호출입니다. 전달되는 *SQL-argument* 또는 *SQL-argument-ind* 값이 없습니다. UDF는 SQL-result, SQL-result-ind 인수, SQL-state 또는 diagnostic-message 인수를 사용하여 응답을 리턴해서는 안됩니다. 이들 인수는 UDF에서 리턴될 때 DB2에 의해 무시됩니다.

표 기능의 경우:

-2 이 명령문에 대한 UDF를 *처음* 호출하는 것입니다. 첫 호출은 모든 SQL 인수 값이 전달된다는 점에서 *정상* 호출입니다.

-1 이것은 이 명령문에 대한 UDF에 *호출을 여는* 것입니다. NO FINAL CALL이 지정되면 scratchpad가 *초기화*되는데 반드시 그런 것은 아닙니다. All SQL 인수가 전달됩니다.

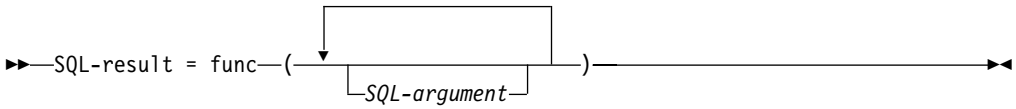
0 이것은 *페치* 호출입니다. DB2는 표 기능이 리턴 값 세트로 구성된 행을 리턴하거나 SQLSTATE 값 '02000'으로 표시되는 표의 끝 조건을 리턴할 것으로 기대합니다.

1 이것은 *닫기* 호출입니다. 이 호출은 OPEN 호출의 균형 조절하며 외부 CLOSE 처리 및 자원 해제를 수행하는 데 사용됩니다.

2 *마지막* 호출입니다. 전달되는 *SQL-argument* 또는 *SQL-argument-ind* 값이 없습니다. UDF는 SQL-result, SQL-result-ind 인수, SQL-state 또는 diagnostic-message 인수를 사용하여 응답을 리턴해서는 안됩니다. 이들 인수는 UDF에서 리턴될 때 DB2에 의해 무시됩니다.

dbinfo 이 인수는 UDF를 호출하기 전에 DB2에 의해 설정됩니다. UDF에 대한 CREATE FUNCTION문이 DBINFO 키워드를 지정한 경우에만 이 인수가 존재합니다. 인수는 그 정의가 `sqludf` include에 포함된 구조입니다.

매개변수 방법 GENERAL(또는 SIMPLE CALL): 매개변수 방법 GENERAL을 사용하여, 매개변수가 CREATE FUNCTION문에서 지정된 것처럼 외부 서비스 프로그램에 전달됩니다. 이 매개변수 양식은 스칼라 UDF에만 사용될 수 있습니다. 형식은 다음과 같습니다.



SQL-argument

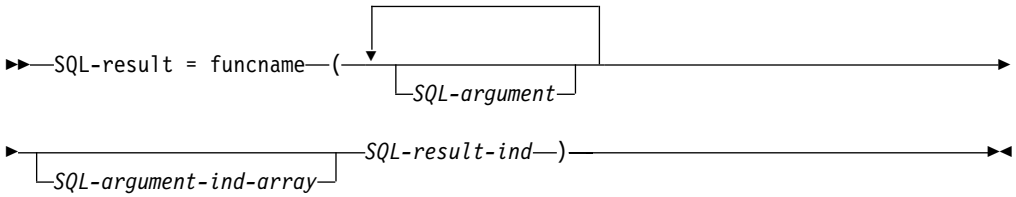
이 인수는 UDF를 호출하기 전에 DB2에 의해 설정됩니다. 이 값은 *n*번 반복되는데, 여기서 *n*은 기능 참조에 지정된 인수 수입니다. 이들 각 인수의 값은 기능 호출에 지정된 표현식에서 가져옵니다. CREATE FUNCTION문의 정의된 매개변수의 자료 유형에 표현됩니다. 주: 이 매개변수는 입력으로만 취급됩니다. UDF에 의한 매개변수 값 변경은 DB2에 의해 무시됩니다.

SQL-result

이 값은 UDF에 의해 리턴됩니다. DB2는 값을 데이터베이스 기억장치에 복사합니다. 값을 정확하게 리턴하려면 기능 코드는 값을 리턴하는 기능이어야 합니다. 데이터베이스는 CREATE FUNCTION문에 정의된대로 리턴 값에 대해 정의된만큼의 값만 복사합니다. CAST FROM절이 CREATE FUNCTION문에 사용되면, DB2는 UDF가 CAST FROM절에 정의된 값을 리턴한다고 가정하며, 그렇지 않을 경우 DB2는 UDF가 RETURNS절에 정의된대로 값을 리턴한다고 가정합니다.

기능 코드가 값을 리턴하는 기능이어야 한다는 요구사항 때문에, 매개변수 방법 GENERAL에 대해 사용된 기능 코드는 서비스 프로그램에 작성되어야 합니다.

매개변수 방법 GENERAL WITH NULLS: 매개변수 양식 GENERAL WITH NULLS는 스칼라 UDF에만 사용될 수 있습니다. 이 매개변수 양식의 경우, 다음과 같이 매개변수가 (지정된 순서대로) 서비스 프로그램에 전달됩니다.



SQL-argument

이 인수는 UDF를 호출하기 전에 DB2에 의해 설정됩니다. 이 값은 *n*번 반복되는데, 여기서 *n*은 기능 참조에 지정된 인수 수입니다. 이들 각 인수의 값은 기능 호출에 지정된 표현식에서 가져옵니다. CREATE FUNCTION문의 정의된 매개변수의 자료 유형에 표현됩니다. 주: 이 매개변수는 입력으로만 취급됩니다. UDF에 의한 매개변수 값 변경은 DB2에 의해 무시됩니다.

SQL-argument-ind-array

이 인수는 UDF를 호출하기 전에 DB2에 의해 설정됩니다. 하나 이상의

*SQL-arguments*가 널(null)인지 아닌지를 판별하기 위해 UDF에 의해 사용될 수 있습니다. 이 값은 부호가 있는 두 바이트 정수(인디케이터) 배열입니다. *n* 번째 배열 인수는 *n*번째 *SQL-argument*에 대응합니다. 각 배열 항목은 다음 값 중 하나로 설정됩니다.

0 인수가 존재하며 널이 아닙니다.

-1 인수가 널입니다.

UDF는 널 입력이 있는지 검사해야 합니다. 주: 이 매개변수는 입력으로만 취급됩니다. UDF에 의한 매개변수 값 변경은 DB2에 의해 무시됩니다.

SQL-result-ind

DB2에 리턴하기 전에 이 인수는 UDF에 의해 설정됩니다. 데이터베이스는 리턴 값에 대한 기억장치를 제공합니다. 인수는 부호가 있는 두 바이트 정수로 정의됩니다. 음수 값으로 설정되면 데이터베이스는 기능 결과를 널로 해석합니다. 0 또는 양수 값으로 설정되면, 데이터베이스는 *SQL-result*에 리턴된 값을 사용합니다. 데이터베이스는 리턴 값 인디케이터에 대한 기억장치를 제공합니다. 매개변수가 주소에 의해 전달되므로 주소는 인디케이터 값이 위치할 기억장치의 주소입니다.

SQL-result

이 값은 UDF에 의해 리턴됩니다. DB2는 값을 데이터베이스 기억장치에 복사합니다. 값을 정확하게 리턴하려면 기능 코드는 값을 리턴하는 기능이어야 합니다. 데이터베이스는 CREATE FUNCTION문에 정의된대로 리턴 값에 대해 정의된만큼의 값만 복사합니다. CAST FROM절이 CREATE FUNCTION문에 사용되면, DB2는 UDF가 CAST FROM절에 정의된 값을 리턴한다고 가정하며, 그렇지 않을 경우 DB2는 UDF가 RETURNS절에 정의된대로 값을 리턴한다고 가정합니다.

기능 코드가 값을 리턴하는 기능이어야 한다는 요구사항 때문에, 매개변수 방법 GENERAL WITH NULLS에 대해 사용된 기능 코드는 서비스 프로그램에 작성되어야 합니다.

주:

1. CREATE FUNCTION문에 지정된 외부 이름은 인용되거나 인용되지 않고 지정될 수 있습니다. 이름은 인용되지 않은 경우 대문자로 된 다음에 저장되며, 인용된 경우 지정되어 있는 대로 저장됩니다. 이는 데이터베이스가 기능 정의에 따라 저장된 이름과 정확하게 일치하는 이름을 가지고 있는 프로그램을 탐색하기 때문에 실제 프로그램을 명명할 때 중요합니다. 예를 들면, 기능이 다음과 같이 작성되었을 경우

```
CREATE FUNCTION X(INT) RETURNS INT
  LANGUAGE C
  EXTERNAL NAME 'MYLIB/MYPGM(MYENTRY)'
```

및 프로그램의 소스가 다음과 같은 경우

```
void myentry(
       int*in
       int*out,
       .
       .
       . .
```

데이터베이스는 항목이 소문자 *myentry*로 되어 있고 대문자 *MYENTRY*를 찾도록 지시를 받았기 때문에 항목을 찾지 못합니다.

2. C++ 모듈을 포함하는 서비스 프로그램의 경우, C++ 소스 코드에서 *extern "C"*가 프로그램 기능 정의에 선행하는지 확인하십시오. 그렇지 않으면, C++ 컴파일러는 기능명의 '이름 지우기'를 수행하여 데이터베이스가 찾을 수 없게 됩니다.

매개변수 방법 DB2GENERAL: 매개변수 방법 DB2GENERAL은 Java UDF에서 사용합니다. 이 매개변수에 대한 설명은 IBM Developer Kit for Java의 Java SQL 루틴 주제를 참조하십시오.

매개변수 양식 Java: Java 매개변수 방법은 SQLJ 제1부에 있는 SQL 루틴 표준에 의해 지정된 방법입니다. 이 매개변수에 대한 설명은 IBM Developer Kit for Java의 Java SQL 루틴 주제를 참조하십시오.

표 기능 고려사항

외부 표는 표를 참조하는 SQL로 표를 전달하는 UDF입니다. 표 기능 참조는 SELECT의 FROM 절에서만 유효합니다. 표 기능을 사용하는 경우, 다음을 관찰하십시오.

- 표 기능이 표를 전달한다고 하더라도 DB2와 UDF 간의 실제 인터페이스는 한 번에 한 행씩 이루어집니다.OPEN, FETCH, CLOSE, FIRST 및 FINAL의 다섯 종류의 호출이 표 기능에 이루어집니다. FIRST 및 FINAL의 존재는 UDF를 정의하는 방법에 따라 호출합니다. 스칼라 함수에 사용될 수 있는 동일한 *호출 유형* 메카니즘이 이 호출들을 구분하는 데 사용됩니다.
- DB2와 사용자 정의 스칼라 함수 사이에 사용되는 표준 인터페이스가 확장되어 표 기능을 수용할 수 있습니다. 표 기능에 대하여 *SQL-result* 인수가 반복됩니다. 열에 대응하는 각 인스턴스가 CREATE FUNCTION문의 RETURNS TABLE 절에 정의된 것처럼 리턴됩니다. *SQL-result-ind* 인수도 마찬가지로 반복되며 대응하는 *SQL-result* 인스턴스에 각 인스턴스가 관련됩니다.
- 표 기능에 대한 CREATE FUNCION문의 RETURNS절에 정의된 모든 결과 열이 리턴되어야 하는 것은 아닙니다. CREATE FUNCTION의 DBINFO 키워드와 그에 대응하는 *dbinfo* 인수를 사용하여 특정 표 기능 참조에 필요한 열만 리턴되도록 하는 최적화가 가능합니다.
- 리턴된 개별 열 값은 스칼라 함수에 의해 리턴된 값과 형식에 있어 일치합니다.

- 표 기능에 대한 CREATE FUNCTION문에는 CARDINALITY *n* 스펙이 있습니다. 이 스펙을 사용하여 Optimizer에게 결과의 대략의 *크기*를 알려 줌으로써 기능이 참조될 때 Optimizer가 최선의 선택을 할 수 있도록 합니다. 표 기능의 CARDINALITY로 지정된 것에 상관없이 무한 cardinality를 사용하는 기능 즉 FETCH 호출에서 항상 행을 리턴하는 기능을 작성할 때는 주의해야 합니다. DB2는 조회 처리에서의 촉매로 *표의 끝* 조건을 기다립니다. 그러므로 표의 끝 조건(SQL 상태 값 '02000')을 리턴하지 않는 표 기능은 무한 처리 루프를 발생시킬 수 있습니다.

표 기능 오류 처리

표 기능 호출에 대한 오류 처리 모델은 다음과 같습니다.

1. FIRST 호출에 실패하면 더 이상의 호출이 이루어지지 않습니다.
2. FIRST 호출이 성공하면 중첩된 OPEN, FETCH 및 CLOSE 호출이 이루어지고 FINAL 호출은 항상 이루어집니다.
3. OPEN 호출이 실패하면 FETCH 나 CLOSE 호출은 이루어지지 않습니다.
4. OPEN 호출이 성공하면 FETCH 및 CLOSE 호출이 이루어집니다.
5. FETCH 호출이 실패하면 더 이상의 FETCH 호출은 이루어지지 않지만 CLOSE 호출이 이루어집니다.

주: 이 모델은 표 UDF에 대한 일반적인 오류 처리를 설명합니다. 시스템 장애나 통신 문제가 발생하는 경우 오류 처리 모델이 나타내는 호출은 수행되지 않을 수도 있습니다.

스칼라 함수 오류 처리

FINAL CALL 스펙을 사용하여 정의된 스칼라 UDF에 대한 오류 처리 모델은 다음과 같습니다.

1. FIRST 호출에 실패하면 더 이상의 호출이 이루어지지 않습니다.
2. FIRST 호출이 성공하면 명령문 처리로 보장된 보다 많은 NORMAL 호출이 이루어지고 FINAL 호출은 항상 이루어집니다.
3. NORMAL 호출이 실패하면 더 이상의 NORMAL 호출은 이루어지지 않지만 FINAL 호출이 이루어집니다(FINAL CALL을 지정한 경우). 이것은 FIRST 호출에서 오류가 리턴된 경우 FINAL 호출이 이루어지지 않으므로 리턴하기 전에 UDF가 정리해야 한다는 것을 의미합니다.

주: 이 모델은 스칼라 UDF에 대한 일반적인 오류 처리를 설명합니다. 시스템 장애나 통신 문제가 발생하는 경우 오류 처리 모델이 나타내는 호출은 수행되지 않을 수도 있습니다.

기능을 펜스 또는 펜스되지 않게 설정

사용자 정의 기능(UDF)을 작성할 때 UDF를 펜스된 UDF로 만들 것인지를 고려해야 합니다.기본적으로 UDF는 펜스된 UDF로 작성됩니다. 펜스되었다는 것은 데이터베이스가 별도의 스레드에서 UDF를 실행해야 함을 의미합니다. 복잡한 UDF의 경우 고유한 SQL 커서명 생성 과 같은 잠재적인 문제점을 방지할 수 있으므로 이러한 구별은 의미가 있습니다. 자원 충돌에 신경을 쓰지 않아도 된다는 점 역시 기본값을 사용하여 펜스된 UDF로 설정하는 이유 중 하나입니다. 'NOT FENCED' 옵션을 사용하여 작성된 UDF는 데이터베이스에게 UDF를 시작한 같은 스레드 내에서 UDF가 실행되도록 사용자가 요구함을 나타냅니다. 펜스되지 않았다는 것은 펜스된 UDF와 같은 방식으로 UDF를 실행시키기로 결정할 수 있었던 데이터베이스에 대한 제안입니다.

```
CREATE FUNCTION QGPL.FENCED (parameter1 INTEGER)
RETURNS INTEGER LANGUAGE SQL
BEGIN
RETURN parameter1 * 3;
END;
CREATE FUNCTION QGPL.UNFENCED1 (parameter1 INTEGER)
RETURNS INTEGER LANGUAGE SQL NOT FENCED
-- Build the UDF to request faster execution via the NOT FENCED option
BEGIN
RETURN parameter1 * 3;
END;
```

UDF 코드 예

다음의 예는 SQL 기능 및 외부 기능을 사용하여 UDF를 구현하는 방법을 보여줍니다.

- 『예: 수의 제곱 UDF』
- 274 페이지의 『예: 카운터』
- 275 페이지의 『예: 날씨 표 기능』

예: 수의 제곱 UDF

주: 코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

사용자가 수의 제곱을 리턴하는 기능을 원한다고 가정합니다. 조회문은 다음과 같습니다.

```
SELECT SQUARE(myint) FROM mytable
```

다음 예는 여러 가지 방법으로 UDF를 정의하는 방법을 보여줍니다.

- **SQL 기능 사용**

  ```
  CREATE FUNCTION SQUARE( inval INT) RETURNS INT
  LANGUAGE SQL
  BEGIN
  RETURN(inval*inval);
  END
  ```

- **외부 기능 사용, 매개변수 방법 SQL:**

  CREATE FUNCTION문:

```
CREATE FUNCTION SQUARE(INT) RETURNS INT CAST FROM FLOAT
LANGUAGE C
EXTERNAL NAME 'MYLIB/MATH(SQUARE)'
DETERMINISTIC
       NO SQL
       NO EXTERNAL ACTION
PARAMETER STYLE SQL
ALLOW PARALLEL
```

  코드:

```
void SQUARE(int *inval,
double *outval,
short *inind,
short *outind,
char *sqlstate,
char *funcname,
char *specname,
char *msgtext)
{
if (*inind<0)
  *outind=-1;
else
  {
   *outval=*inval;
   *outval=(*outval)*(*outval);
   *outind=0;
  }
return;
}
```

  디버그할 수 있도록 외부 서비스 프로그램 작성:

```
CRTCMOD MODULE(mylib/square) DBGVIEW(*SOURCE)
   CRTSRVPGM SRVPGM(mylib/math) MODULE(mylib/square)
EXPORT(*ALL) ACTGRP(*CALLER)
```

- **외부 기능 사용, 매개변수 방법 GENERAL:**

  CREATE FUNCTION문:

```
CREATE FUNCTION SQUARE(INT) RETURNS INT CAST FROM FLOAT
LANGUAGE C
EXTERNAL NAME 'MYLIB/MATH(SQUARE)'
DETERMINISTIC
       NO SQL
       NO EXTERNAL ACTION
PARAMETER STYLE GENERAL
ALLOW PARALLEL
```

  코드:

```
double SQUARE(int *inval)
{
  double outval;
```

```
  outval=*inval;
  outval=outval*outval;
return(outval);
}
```

디버그할 수 있도록 외부 서비스 프로그램 작성:

```
CRTCMOD MODULE(mylib/square) DBGVIEW(*SOURCE)

   CRTSRVPGM SRVPGM(mylib/math) MODULE(mylib/square)
EXPORT(*ALL) ACTGRP(*CALLER)
```

예: 카운터

주: 코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

사용자가 단순히 SELECT문의 행에 번호를 매기기를 원하다고 가정합니다. 사용자는 카운터를 증가하여 리턴하는 UDF를 작성합니다. 이 예에서는 DB2 SQL 매개변수 방법과 scratchpad로 외부 기능을 사용합니다.

```
CREATE FUNCTION COUNTER()
       RETURNS INT
       SCRATCHPAD
       NOT DETERMINISTIC
       NO SQL
       NO EXTERNAL ACTION
       LANGUAGE C
       PARAMETER STYLE DB2SQL
       EXTERNAL NAME 'MYLIB/MATH(ctr)'
       DISALLOW PARALLEL

/* structure scr defines the passed scratchpad for the function "ctr" */
struct scr {
   long len;
   long countr;
   char not_used[96];
};

void ctr (
   long *out,                          /* output answer (counter) */
   short *outnull,                     /* output NULL indicator */
   char *sqlstate,                     /* SQL STATE */
   char *funcname,                     /* function name */
   char *specname,                     /* specific function name */
   char *mesgtext,                     /* message text insert */
   struct scr *scratchptr) {           /* scratch pad */

   *out = ++scratchptr->countr;      /* increment counter & copy out */
   *outnull = 0;
return;
}
/* end of UDF : ctr */
```

이 UDF에 대해 다음을 알 수 있습니다.

- 정의된 입력 SQL 인수는 없지만, 값을 리턴합니다.
- scratchpad 입력 인수를 네 개의 표준 추적 인수, *SQL-state*, *function-name*, *specific-name*, *message-text* 뒤에 붙입니다.
- 전달된 scratchpad를 맵핑하기 위해 구조 정의를 포함합니다.
- 정의된 입력 매개변수가 없습니다. 이는 코드와 일치합니다.

- SCRATCHPAD는 DB2가 할당하도록 코드화되고, 적절히 초기화되며 scratchpad 인수를 전달합니다.
- UDF가 SQL 입력 인수보다 더 종속적이므로 NOT DETERMINISTIC으로 지정했습니다(이 경우는 아님).
- UDF의 정확한 기능은 단일 scratchpad에 따라 달라지므로 DISALLOW PARALLEL을 정확하게 지정했습니다.

예: 날씨 표 기능

주: 코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

다음은 미국의 다양한 도시에서 날씨 정보를 리턴하는 표 기능의 예입니다. 이 도시들에 대한 날씨 날짜는 예제 프로그램의 주석에 설명되어 있듯이 외부 파일에서 읽어 옵니다. 이 데이터에는 도시 이름이 나오고 그 뒤에 날씨 정보가 뒤따릅니다. 이 패턴은 다른 도시에 대해서도 반복됩니다.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sqludf.h> /* for use in compiling User Defined Function */


#define   SQL_NOTNULL   0   /* Nulls Allowed - Value is not Null */
#define   SQL_ISNULL   -1   /* Nulls Allowed - Value is Null */
#define   SQL_TYP_VARCHAR 448
#define   SQL_TYP_INTEGER 496
#define   SQL_TYP_FLOAT 480

/* Short and long city name structure */
typedef struct {
  char * city_short ;
  char * city_long ;
} city_area ;

/* Scratchpad data */ 1
/* Preserve information from one function call to the next call */
typedef struct {
  /* FILE * file_ptr; if you use weather data text file */
  int file_pos ;  /* if you use a weather data buffer */
} scratch_area ;

/* Field descriptor structure */
typedef struct {
  char fld_field[31] ;                    /* Field data */
  int  fld_ind ;            /* Field null indicator data */
  int  fld_type ;                         /* Field type */
  int  fld_length ;  /* Field length in the weather data */
  int  fld_offset ;  /* Field offset in the weather data */
} fld_desc ;

/* Short and long city name data */
city_area cities[] = {
```

```
  { "alb", "Albany, NY"                 },
  { "atl", "Atlanta, GA"                },
  .
  .
  .
  { "wbc", "Washington DC, DC"          },
  /* You may want to add more cities here */

  /* Do not forget a null termination */
  { ( char * ) 0, ( char * ) 0          }
} ;

/* Field descriptor data */
fld_desc fields[] = {
  { "", SQL_ISNULL, SQL_TYP_VARCHAR, 30,  0 }, /* city          */
  { "", SQL_ISNULL, SQL_TYP_INTEGER,  3,  2 }, /* temp_in_f     */
  { "", SQL_ISNULL, SQL_TYP_INTEGER,  3,  7 }, /* humidity      */
  { "", SQL_ISNULL, SQL_TYP_VARCHAR,  5, 13 }, /* wind          */
  { "", SQL_ISNULL, SQL_TYP_INTEGER,  3, 19 }, /* wind_velocity */
  { "", SQL_ISNULL, SQL_TYP_FLOAT,    5, 24 }, /* barometer     */
  { "", SQL_ISNULL, SQL_TYP_VARCHAR, 25, 30 }, /* forecast      */
  /* You may want to add more fields here */

  /* Do not forget a null termination */
  { ( char ) 0, 0, 0, 0, 0 }
} ;

/* Following is the weather data buffer for this example. You */
/* may want to keep the weather data in a separate text file. */
/* Uncomment the following fopen() statement.  Note that you  */
/* have to specify the full path name for this file.          */
char * weather_data[] = {
   "alb.forecast",
   "   34   28%    wnw   3  30.53 clear",
   "atl.forecast",
   "   46   89%   east  11  30.03 fog",
   .
   .
   .
   "wbc.forecast",
   "   38   96%    ene  16  30.31 light rain",
   /* You may want to add more weather data here */

   /* Do not forget a null termination */
   ( char * ) 0
} ;

#ifdef __cplusplus
extern "C"
  #endif
/* This is a subroutine. */
/* Find a full city name using a short name */
int get_name( char * short_name, char * long_name ) {

    int name_pos = 0 ;

    while ( cities[name_pos].city_short != ( char * ) 0 ) {
       if (strcmp(short_name, cities[name_pos].city_short) == 0) {
```

```
            strcpy( long_name, cities[name_pos].city_long ) ;
            /* A full city name found */
            return( 0 ) ;
          }
          name_pos++ ;
        }
        /* Could not find such city in the city data */
        strcpy( long_name, "Unknown City" ) ;
        return( -1 ) ;

}

#ifdef __cplusplus
extern "C"
  #endif
/* This is a subroutine. */
/* Clean all field data and field null indicator data */
int clean_fields( int field_pos ) {

    while (fields[field_pos].fld_length !=0 ) {
       memset( fields[field_pos].fld_field, '\0', 31 ) ;
       fields[field_pos].fld_ind = SQL_ISNULL ;
       field_pos++ ;
    }
    return( 0 ) ;

}

#ifdef __cplusplus
extern "C"
  #endif
/* This is a subroutine. */
/* Fills all field data and field null indicator data ... */
/* ... from text weather data */
int get_value( char * value, int field_pos ) {

    fld_desc * field ;
    char field_buf[31] ;
    double * double_ptr ;
    int * int_ptr, buf_pos ;

    while ( fields[field_pos].fld_length != 0 ) {
       field = &fields[field_pos] ;
       memset( field_buf, '\0', 31 ) ;
       memcpy( field_buf,
               ( value + field->fld_offset ),
               field->fld_length ) ;
       buf_pos = field->fld_length ;
       while ( ( buf_pos > 0 ) &&
               ( field_buf[buf_pos] == ' ' ) )
          field_buf[buf_pos--] = '\0' ;
       buf_pos = 0 ;
       while ( ( buf_pos < field->fld_length ) &&
               ( field_buf[buf_pos] == ' ' ) )
          buf_pos++ ;
       if ( strlen( ( char * ) ( field_buf + buf_pos ) ) > 0 ||
            strcmp( ( char * ) ( field_buf + buf_pos ), "n/a") != 0 ) {
          field->fld_ind = SQL_NOTNULL ;
```

```
                /* Text to SQL type conversion */
                switch( field->fld_type ) {
                  case SQL_TYP_VARCHAR:
                       strcpy( field->fld_field,
                               ( char * ) ( field_buf + buf_pos ) ) ;
                       break ;
                  case SQL_TYP_INTEGER:
                       int_ptr = ( int * ) field->fld_field ;
                       *int_ptr = atoi( ( char * ) ( field_buf + buf_pos ) ) ;
                       break ;
                  case SQL_TYP_FLOAT:
                       double_ptr = ( double * ) field->fld_field ;
                       *double_ptr = atof( ( char * ) ( field_buf + buf_pos ) ) ;
                       break ;
                  /* You may want to add more text to SQL type conversion here */
                }

             }
             field_pos++ ;
          }
          return( 0 ) ;

}

#ifdef __cplusplus
extern "C"
  #endif
void SQL_API_FN weather( /* Return row fields */
                    SQLUDF_VARCHAR * city,
                    SQLUDF_INTEGER * temp_in_f,
                    SQLUDF_INTEGER * humidity,
                    SQLUDF_VARCHAR * wind,
                    SQLUDF_INTEGER * wind_velocity,
                    SQLUDF_DOUBLE  * barometer,
                    SQLUDF_VARCHAR * forecast,
                    /* You may want to add more fields here */

                    /* Return row field null indicators */
                    SQLUDF_NULLIND * city_ind,
                    SQLUDF_NULLIND * temp_in_f_ind,
                    SQLUDF_NULLIND * humidity_ind,
                    SQLUDF_NULLIND * wind_ind,
                    SQLUDF_NULLIND * wind_velocity_ind,
                    SQLUDF_NULLIND * barometer_ind,
                    SQLUDF_NULLIND * forecast_ind,
                    /* You may want to add more field indicators here */

                    /* UDF always-present (trailing) input arguments */
                    SQLUDF_TRAIL_ARGS_ALL
                  ) {

  scratch_area * save_area ;
  char line_buf[81] ;
  int line_buf_pos ;

  /* SQLUDF_SCRAT is part of SQLUDF_TRAIL_ARGS_ALL */
  /* Preserve information from one function call to the next call */
```

```
save_area = ( scratch_area * ) ( SQLUDF_SCRAT->data ) ;

/* SQLUDF_CALLT is part of SQLUDF_TRAIL_ARGS_ALL */
switch( SQLUDF_CALLT ) {

  /* First call UDF: Open table and fetch first row */
  case SQL_TF_OPEN:
       /* If you use a weather data text file specify full path */
       /* save_area->file_ptr = fopen("tblsrv.dat","r"); */
       save_area->file_pos = 0 ;
       break ;

  /* Normal call UDF: Fetch next row */ 2
  case SQL_TF_FETCH:
       /* If you use a weather data text file */
       /* memset(line_buf, '\0', 81); */
       /* if (fgets(line_buf, 80, save_area->file_ptr) == NULL) { */
       if ( weather_data[save_area->file_pos] == ( char * ) 0 ) {

          /* SQLUDF_STATE is part of SQLUDF_TRAIL_ARGS_ALL */
          strcpy( SQLUDF_STATE, "02000" ) ;

          break ;
       }
       memset( line_buf, '\0', 81 ) ;
       strcpy( line_buf, weather_data[save_area->file_pos] ) ;
       line_buf[3] = '\0' ;

       /* Clean all field data and field null indicator data */
       clean_fields( 0 ) ;

       /* Fills city field null indicator data */
       fields[0].fld_ind = SQL_NOTNULL ;

       /* Find a full city name using a short name */
       /* Fills city field data */
       if ( get_name( line_buf, fields[0].fld_field ) == 0 ) {
          save_area->file_pos++ ;
          /* If you use a weather data text file */
          /* memset(line_buf, '\0', 81); */
          /* if (fgets(line_buf, 80, save_area->file_ptr) == NULL) { */
          if ( weather_data[save_area->file_pos] == ( char * ) 0 ) {
             /* SQLUDF_STATE is part of SQLUDF_TRAIL_ARGS_ALL */
             strcpy( SQLUDF_STATE, "02000" ) ;
             break ;
          }
          memset( line_buf, '\0', 81 ) ;
          strcpy( line_buf, weather_data[save_area->file_pos] ) ;
          line_buf_pos = strlen( line_buf ) ;
          while ( line_buf_pos > 0 ) {
             if ( line_buf[line_buf_pos] >= ' ' )
                line_buf_pos = 0 ;
else {
                line_buf[line_buf_pos] = '\0' ;
                line_buf_pos-- ;
             }
          }
       }
```

```
                    /* Fills field data and field null indicator data ... */
                    /* ... for selected city from text weather data */
                    get_value( line_buf, 1 ) ;  /* Skips city field */

                    /* Builds return row fields */
                    strcpy( city, fields[0].fld_field ) ;
                    memcpy( (void *) temp_in_f,
                            fields[1].fld_field,
                            sizeof( SQLUDF_INTEGER ) ) ;
                    memcpy( (void *) humidity,
                            fields[2].fld_field,
                            sizeof( SQLUDF_INTEGER ) ) ;
                    strcpy( wind, fields[3].fld_field ) ;
                    memcpy( (void *) wind_velocity,
                            fields[4].fld_field,
                            sizeof( SQLUDF_INTEGER ) ) ;
                    memcpy( (void *) barometer,
                            fields[5].fld_field,
                            sizeof( SQLUDF_DOUBLE ) ) ;
                    strcpy( forecast, fields[6].fld_field ) ;

                    /* Builds return row field null indicators */
                    memcpy( (void *) city_ind,
                            &(fields[0].fld_ind),
                            sizeof( SQLUDF_NULLIND ) ) ;
                    memcpy( (void *) temp_in_f_ind,
                            &(fields[1].fld_ind),
                            sizeof( SQLUDF_NULLIND ) ) ;
                    memcpy( (void *) humidity_ind,
                            &(fields[2].fld_ind),
                            sizeof( SQLUDF_NULLIND ) ) ;
                    memcpy( (void *) wind_ind,
                            &(fields[3].fld_ind),
                            sizeof( SQLUDF_NULLIND ) ) ;
                    memcpy( (void *) wind_velocity_ind,
                            &(fields[4].fld_ind),
                            sizeof( SQLUDF_NULLIND ) ) ;
                    memcpy( (void *) barometer_ind,
                            &(fields[5].fld_ind),
                            sizeof( SQLUDF_NULLIND ) ) ;
                    memcpy( (void *) forecast_ind,
                            &(fields[6].fld_ind),
                            sizeof( SQLUDF_NULLIND ) ) ;

                    /* Next city weather data */
                    save_area->file_pos++ ;

                    break ;

                /* Special last call UDF for cleanup (no real args!): Close table */ 3
                case SQL_TF_CLOSE:
                    /* If you use a weather data text file */
                    /* fclose(save_area->file_ptr); */
                    /* save_area->file_ptr = NULL; */
                    save_area->file_pos = 0 ;
                    break ;
```

```
  }
}
```

이 UDF 코드에 들어 있는 숫자를 참조하여 다음을 살펴 보십시오.

1. scratchpad가 정의되었습니다. row 변수가 OPEN 호출 시 초기화 되었으며 iptr 배열 및 nbr_rows 변수가 열릴 때 *mystery* 기능에 의해 채워집니다.
2. FETCH는 iptr 배열에서 행을 색인으로 사용하여 현재 요소인 iptr에서 out_c1, out_c2, and out_c3 결과 값 포인터에 의해 가리켜진 위치로 관련된 값을 이동시킵니다.
3. 마지막으로 CLOSE는 OPEN에 의해 확보되고 scratchpad에서 앵커된 기억장치를 해제합니다.

다음은 이 UDF에 대한 CREATE FUNCTION문입니다.

```
CREATE FUNCTION tfweather_u()
  RETURNS TABLE (CITY VARCHAR(25),
                 TEMP_IN_F INTEGER,
                 HUMIDITY INTEGER,
                 WIND VARCHAR(5),
                 WIND_VELOCITY INTEGER,
                 BAROMETER FLOAT,
                 FORECAST VARCHAR(25))
  SPECIFIC tfweather_u
  DISALLOW PARALLEL
  NOT FENCED
  DETERMINISTIC
  NO SQL
  NO EXTERNAL ACTION
  SCRATCHPAD
  NO FINAL CALL
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  EXTERNAL NAME 'LIB1/WEATHER(weather)';
```

이 명령문에 대해 다음을 알 수 있습니다.

- 아무런 입력을 받지 않으며 7개의 출력 열을 리턴합니다.
- SCRATCHPAD가 지정되어 DB2는 할당하고 적절하게 초기화하여 scratchpad 인수를 전달합니다.
- NO FINAL CALL이 지정됩니다.
- SQL 입력 인수보다 더 종속적이므로 NOT DETERMINISTIC으로 지정되었습니다. 즉 이것은 mystery 기능에 따라 달라지므로 실행할 때마다 내용이 달라질 것으로 간주합니다.
- 표 기능에 DISALLOW PARALLEL이 필요합니다.
- CARDINALITY 100은 DB2 optimizer가 제공하는 예상되는 리턴되는 행 수를 추정한 것입니다.

- DBINFO는 사용되지 않았으며 해당 기능을 참조하는 특정 명령문에 필요한 열만을 리턴하기 위한 최적화도 구현되지 않았습니다.
- NOT NULL CALL이 지정되어 입력 SQL 인수가 NULL이고 이 조건을 점검할 필요가 없는 경우 UDF가 호출되지 않습니다.

이 표 기능이 생성한 모든 행을 선택하려면 다음 조회를 사용하십시오.

```
SELECT *
    FROM TABLE (tfweather_u())x
```

제 14 장 동적 SQL 어플리케이션

동적 SQL을 사용하면 프로그램 실행시 어플리케이션을 정의하고 SQL문을 수행할 수 있습니다. 동적 SQL을 제공하는 어플리케이션은 문자 스트링 양식의 SQL문 입력(또는 빌드)으로서 받아들여집니다. 어플리케이션은 실행할 SQL문의 유형에 대해 알 필요가 없습니다. 어플리케이션은 다음을 수행합니다.

- SQL문을 빌드 또는 허용합니다.
- 수행할 SQL문을 준비합니다.
- 명령문을 수행합니다.
- SQL 리턴 코드를 처리합니다.

대화식 SQL(제 17 장『대화식 SQL 사용』에서 설명되는)은 동적 SQL 프로그램의 예입니다. SQL문은 대화식 SQL에 의해 처리되고 동적으로 실행됩니다.

주:

1. 런타임 오버헤드는 정적 SQL문보다 동적 SQL을 사용하여 처리된 명령문의 경우 더 큽니다. 추가 처리는 프로그램을 단지 수행하는 것이 아니고 프로그램을 사전컴파일하고 바인딩한 후 수행하는데 필요한 처리와 유사합니다. 그러므로 동적 SQL의 융통성이 필요한 어플리케이션에만 사용해야 합니다. 다른 어플리케이션은 일반(정적) SQL문을 사용하여 데이터베이스의 자료에 액세스해야 합니다.
2. EXECUTE 또는 EXECUTE IMMEDIATE문이 들어 있고 FOR READ ONLY 절을 사용한 프로그램은 블록화가 커서에 대해 행을 검색하기 위하여 사용되므로 커서를 읽기 전용으로 만들어 성능을 향상시킬 수 있습니다.

   ALWBLK(*ALLREAD) CRTSQLxxx 옵션은 FOR UPDATE OF를 명시적으로 작성하지 않거나 커서를 가리키는 삭제 내용이나 갱신 내용이 위치된 모든 커서에 대해 FOR READ ONLY 선언 부분을 내포합니다. 내포된 FOR READ ONLY가 있는 커서는 위의 두 번째 항목에 대한 이점을 얻게 됩니다.

일부 동적 SQL문에는 주소 변수가 사용됩니다. iSeries용 RPG 프로그램은 주소 변수를 관리하기 위해 PL/I, COBOL, C 또는 iSeries용 ILE RPG 프로그램의 지원을 필요로 합니다.

다음 표는 iSeries용 DB2 UDB에서 지원하는 모든 명령문을 보여 주고 동적 어플리케이션에서 사용될 수 있는지를 나타냅니다.

주: 다음 표에서 *동적 SQL* 열의 숫자는 다음 페이지에 있는 주에 해당됩니다.

표 28. 동적 어플리케이션에 허용되는 SQL문의 리스트

SQL문	**정적 SQL**	**동적 SQL**
ALTER TABLE	Y	Y
BEGIN DECLARE SECTION	Y	N
CALL	Y	Y
CLOSE	Y	N
COMMENT ON	Y	Y
COMMIT	Y	Y
CONNECT	Y	N
CREATE ALIAS	Y	Y
CREATE DISTINCT TYPE	Y	Y
CREATE FUNCTION	Y	Y
CREATE INDEX	Y	Y
CREATE PROCEDURE	Y	Y
CREATE SCHEMA	Y	Y
CREATE TABLE	Y	Y
CREATE TRIGGER	Y	Y
CREATE VIEW	Y	Y
DECLARE CURSOR	Y	주 1 참조
DECLARE GLOBAL TEMPORARY TABLE	Y	Y
DECLARE PROCEDURE	Y	N
DECLARE STATEMENT	Y	N
DECLARE VARIABLE	Y	N
DELETE	Y	Y
DESCRIBE	Y	주 2 참조
DESCRIBE TABLE	Y	N
DISCONNECT	Y	N
DROP	Y	Y
END DECLARE SECTION	Y	N
EXECUTE	Y	주 3 참조
EXECUTE IMMEDIATE	Y	주 4 참조
FETCH	Y	N
FREE LOCATOR	Y	Y
GRANT	Y	Y
HOLD LOCATOR	Y	Y
INCLUDE	Y	N
INSERT	Y	Y
LABEL ON	Y	Y
LOCK TABLE	Y	Y
OPEN	Y	N

표 28. 동적 어플리케이션에 허용되는 SQL문의 리스트 (계속)

SQL문	정적 SQL	동적 SQL
PREPARE	Y	주 5 참조
RELEASE	Y	N
RELEASE SAVEPOINT	Y	Y
RENAME	Y	Y
REVOKE	Y	Y
ROLLBACK	Y	Y
SAVEPOINT	Y	Y
SELECT INTO	Y	주 6 참조
SELECT문	Y	주 7 참조
SET CONNECTION	Y	N
SET OPTION	Y	주 8 참조
SET PATH	Y	Y
SET RESULT SETS	Y	N
SET SCHEMA	Y	Y
SET TRANSACTION	Y	Y
SET 변수	Y	N
UPDATE	Y	Y
VALUES INTO	Y	N
WHENEVER	Y	N

주:

1. 준비될 수 없으나, 실행 전에 연관된 동적 SELECT문에 대한 커서를 정의하기 위하여 사용됩니다.
2. 준비될 수 없으나, 준비된 명령문의 설명을 리턴하는데 사용됩니다.
3. 준비될 수 없으나 이미 준비된 SQL문을 수행하는데 사용됩니다. SQL문은 EXECUTE문을 사용하기 전에 PREPARE문에 의해 미리 준비되어야 합니다. 287 페이지의 『PREPARE문과 EXECUTE문 사용』에 있는 PREPARE의 예를 참조하십시오.
4. 준비될 수 없지만 매개변수 마커가 하나도 없는 동적 명령문 스트링으로 사용될 수 있습니다. EXECUTE IMMEDIATE문으로 명령문 스트링이 준비되고 프로그램 실행시 동적으로 실행됩니다. 286 페이지의 『비SELECT문 처리』에 있는 EXECUTE IMMEDIATE의 예를 참조하십시오.
5. 준비될 수 없으나, 실행하기 전에 동적 SELECT문을 분석, 최적화 및 설정하는데 사용됩니다. 286 페이지의 『비SELECT문 처리』에 있는 PREPARE의 예를 참조하십시오.
6. SELECT INTO문이 준비되어 EXECUTE IMMEDIATE에서 사용될 수 없습니다.
7. EXECUTE나 EXECUTE IMMEDIATE와 함께 사용될 수 없으나 OPEN과 함께 준비되고 사용될 수 있습니다.

8. REXX 프로시듀어를 실행할 때에나 사전컴파일된 프로그램에서만 사용할 수 있습니다.

주: 코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

동적 SQL 어플리케이션 설계 및 실행

동적 SQL문을 발행하려면 동적 SQL문이 사전컴파일시에 준비되지 않고 실행시에 준비되어야 하므로 EXECUTE문이나 EXECUTE IMMEDIATE문과 함께 사용해야 합니다. EXECUTE IMMEDIATE문은 SQL문이 프로그램 실행시에 동적으로 준비되고 수행되도록 합니다.

동적 SQL문에는 SELECT문 및 비SELECT문의 두 가지 기본 유형이 있습니다. 비SELECT문에는 DELETE, INSERT 및 UPDATE와 같은 문이 포함됩니다.

ODBC와 같은 인터페이스를 사용하는 클라이언트 서버 어플리케이션은 일반적으로 동적 SQL을 사용하여 데이터베이스를 액세스합니다. iSeries Access를 사용하는 클라이언트 서버 어플리케이션 개발에 대한 자세한 정보는 iSeries Access Express에 대한 프로그래밍을 참조하십시오.

비SELECT문 처리

동적 SQL 비SELECT문을 빌드하려면 다음을 수행하십시오.

1. 빌드하려는 SQL문이 동적으로 수행될 수 있는 명령문인지 확인하십시오(284 페이지의 표 28 참조).
2. SQL문을 빌드하십시오(SQL문을 빌드, 검증 및 실행하려면 제 17 장 『대화식 SQL 사용』을 참조하십시오).

동적 SQL 비SELECT문을 수행하려면 다음을 수행하십시오.

1. EXECUTE IMMEDIATE를 사용하여 SQL문을 수행하거나 SQL문을 준비한 후 준비된 명령문을 실행하십시오.
2. 결과로 나온 SQL 리턴 코드를 처리하십시오.

다음은 동적 SQL 비SELECT문(stmtstrg)을 수행하는 어플리케이션의 예입니다.

```
  EXEC SQL
EXECUTE IMMEDIATE :stmtstrg;
```

동적 SQL문의 CCSID

SQL문은 주로 호스트 변수입니다. 호스트 변수의 CCSID는 명령문 텍스트의 CCSID로 사용됩니다. PL/I에서 또한 열 표현식이 될 수 있습니다. 이 경우에는 작업 CCSID가 명령문 텍스트의 CCSID로 사용됩니다.

동적 SQL문은 명령문 텍스트의 CCSID를 사용해 처리됩니다. 이는 대부분 가변 문자에 영향을 미칩니다. 예를 들어, 비부호(not sign)(¬)는 CCSID 500의 'BA'X에 위치합니다. 이는 사용자 명령문 텍스트의 CCSID가 500인 경우, SQL은 비부호(¬)를 'BA'X에 위치시킬 것으로 예상합니다.

명령문 텍스트 CCSID가 65535이면 SQL이 CCSID 37을 가지고 있는 것처럼 가변 문자를 처리합니다. 이는 SQL이 비부호(¬)를 '5F'X에서 찾음을 의미합니다.

PREPARE문과 EXECUTE문 사용

비SELECT문에 매개변수 표시가 없다면 비SELECT문은 자동으로 EXECUTE IMMEDIATE문을 사용하여 수행될 수 있습니다. 비SELECT문에 매개변수 표시가 있다면 비SELECT는 PREPARE 및 EXECUTE를 사용하여 수행되어야 합니다.

PREPARE문은 비SELECT문(예를 들어 DELETE문)을 준비하고 사용자가 선택한 이름을 부여합니다. CRTSQLxxx 명령에 DLYPRP(*YES)가 지정되면 USING절이 PREPARE문에 지정된 경우를 제외하고는 명령문이 EXECUTE 또는 DESCRIBE문에 처음 사용될 때까지 준비가 지연됩니다. 여기서는 S1이라고 부르겠습니다. 명령문이 준비된 후에는 매개변수 마커에 다른 값을 사용하여 동일한 프로그램 내에서 여러 번 수행될 수 있습니다. 다음은 준비된 명령문이 여러 번 수행되는 한 예입니다.

```
DSTRING = 'DELETE FROM CORPDATA.EMPLOYEE WHERE EMPNO = ?';

/*The ? is a parameter marker which denotes
  that this value is a host variable that is
  to be substituted each time the statement is run.*/

EXEC SQL PREPARE S1 FROM :DSTRING;

/*DSTRING is the delete statement that the PREPARE statement is
  naming S1.*/

 DO UNTIL (EMP =0);
/*The application program reads a value for EMP from the
  display station.*/
   EXEC SQL
     EXECUTE S1 USING :EMP;


END;
```

위의 예와 유사한 루틴에서는 프로그램이 작성될 때 입력 자료를 제공하는 호스트 변수가 선언되므로 매개변수 마커 수와 자료 유형을 알아야 합니다.

주: 어플리케이션 서버 종료에 연결될 때마다 어플리케이션 서버와 연결된 모든 준비된 명령문이 파괴됩니다. 연결은 성공적인 COMMIT의 뒤에 오는 CONNECT(유형 1)문, DISCONNECT문 또는 RELEASE문에 의해 종료됩니다.

SELECT문 처리 및 SQLDA 사용

기본 SELECT문에는 **고정 리스트**와 **가변 리스트**의 두 가지 유형이 있습니다.

고정 리스트 SELECT문을 처리하려는 경우, SQLDA는 필요하지 않습니다.

가변 리스트 SELECT문을 처리하기 위해서는 먼저 SQLDA 구조를 선언해야 합니다. SQLDA는 호스트 변수 입력 값을 어플리케이션 프로그램에서 SQL로 전달하고 SQL로부터 출력 값을 받는데 사용되는 제어 블록입니다. 또한 SELECT 리스트 표현식에 대한 정보가 PREPARE 또는 DESCRIBE문으로 리턴될 수 있습니다.

고정 리스트 SELECT문

동적 SQL에서 고정 리스트 SELECT문은 예측 가능 번호 및 유형의 자료를 검색하기 위하여 설계된 문입니다. 이러한 명령문을 사용할 때는 검색된 자료를 수용할 호스트 변수를 예측하여 정의할 수 있으므로 SQLDA가 필요하지 않습니다. 각 후속 FETCH는 최종 FETCH와 동일한 수의 값을 리턴시키고 이러한 값은 최종 FETCH에 리턴된 값과 동일한 자료 형식을 가집니다. 다른 SQL 어플리케이션에서와 마찬가지로 호스트 변수를 지정할 수 있습니다.

모든 SQL 지원 어플리케이션 프로그램으로 고정 리스트 동적 SELECT문을 사용할 수 있습니다.

고정 리스트 SELECT문을 동적으로 수행하기 위하여 어플리케이션은 다음 사항을 만족시켜야만 합니다.

1. 호스트 변수에 입력 SQL문을 위치시켜야 합니다.
2. PREPARE문을 발행하여 동적 SQL문의 유효성을 검사하고 수행 가능한 형식으로 기록해야 합니다. CRTSQLxxx 명령에 DLYPRP(*YES)가 지정되면 USING절이 PREPARE문에 지정된 경우를 제외하고는 명령문이 EXECUTE 또는 DESCRIBE 문에 처음 사용될 때까지 준비가 지연됩니다.
3. 명령문 이름에 대한 커서를 선언해야 합니다.
4. 커서를 열어야 합니다.
5. 행을 변수의 고정 리스트(다음 섹션의 가변 리스트 SELECT문에 설명되어 있는 대로 가변 리스트 SELECT문을 사용하고 있을 경우 설명자 영역보다는)에 FETCH 하십시오.
6. 자료의 끝이 발생하면 커서를 닫아야 합니다.
7. 결과로 나온 SQL 리턴 코드를 처리해야 합니다.

예를 들면 다음과 같습니다.

```
MOVE 'SELECT EMPNO, LASTNAME FROM CORPDATA.EMPLOYEE WHERE EMPNO>?'
TO DSTRING.
   EXEC SQL
  PREPARE S2 FROM :DSTRING END-EXEC.
```

```
    EXEC SQL
  DECLARE C2 CURSOR FOR S2 END-EXEC.

    EXEC SQL
  OPEN C2 USING :EMP END-EXEC.

 PERFORM FETCH-ROW UNTIL SQLCODE NOT=0.

    EXEC SQL
  CLOSE C2 END-EXEC.
 STOP-RUN.
 FETCH-ROW.
    EXEC SQL
  FETCH C2 INTO :EMP, :EMPNAME END-EXEC.
```

주: 이 경우 SELECT문은 항상 이전에 수행된 고정 리스트 SELECT문과 같은 숫자와 유형의 자료 항목을 리턴하므로 사용자는 SQL 설명자 영역(SQLDA)을 사용할 필요가 없습니다.

가변 리스트 SELECT문

동적 SQL에서 가변 리스트 SELECT문은 리턴될 결과 열의 수와 형식을 예측할 수 없는 문입니다. 즉, 필요한 변수의 수와 자료 유형에 대해 알지 못합니다. 그러므로 리턴된 결과 열을 수용하기 위해 사전에 호스트 변수를 정의할 수 없습니다.

주: REXX에서, 단계 5.b, 6 및 7은 적용되지 않습니다.

어플리케이션이 가변 리스트 SELECT문을 허용할 경우, 프로그램은 다음 사항을 만족시켜야만 합니다.

1. 호스트 변수에 입력 SQL문을 위치시켜야 합니다.
2. PREPARE문을 발행하여 동적 SQL문의 유효성을 검사하고 수행 가능한 형식으로 기록해야 합니다. CRTSQLxxx 명령에 DLYPRP(*YES)가 지정되면 USING 절이 PREPARE문에 지정된 경우를 제외하고는 명령문이 EXECUTE 또는 DESCRIBE문에 처음 사용될 때까지 준비가 지연됩니다.
3. 명령문 이름에 대한 커서를 선언해야 합니다.
4. 동적 SELECT문의 이름이 있는 커서(단계 3에서 명시됨)를 열어야 합니다.
5. DESCRIBE문을 발행하여 SQL로부터 결과 표의 각 열 유형 및 크기에 관한 정보를 요구해야 합니다.

   주:

   a. 또한 PREPARE문을 INTO절에 사용하여 단일 명령문으로 PREPARE 및 DESCRIBE의 기능을 수행할 수 있습니다.

   b. 검색된 각 열에 대한 설명이 들어갈 만큼 SQLDA가 크지 않으면 프로그램은 필요한 공간의 양을 결정하고 공간을 위한 기억영역을 확보하여 새로운 SQLDA를 빌드하고 DESCRIBE문을 재발행해야 합니다.

6. 검색된 자료의 행을 포함하기 위해 필요한 기억영역의 양을 할당하십시오.
7. SQLDA(SQL 설명자 영역)에 기억영역 주소를 기록하여 SQL에 검색된 자료의 각 항목이 기록될 위치를 알려야 합니다.
8. 행을 페치해야 합니다.
9. 자료의 끝이 발생하면 커서를 닫아야 합니다.
10. 결과로 나온 SQL 리턴 코드를 처리하십시오.

이들 단계 수행 방법에 대한 세부사항은 296 페이지의 『예: SQLDA에 기억장치 할당을 위한 SELECT문』을 참조하십시오.

SQL 설명자 영역(SQLDA)

동적 SQL은 SQL 설명자 영역(SQLDA)이라는 변수 구조를 사용하여 SQL과 어플리케이션 사이에 SQL문에 대한 정보를 전달합니다. SQLDA는 DESCRIBE 및 DESCRIBE TABLE문 실행에 필요하고 또한 PREPARE, OPEN, FETCH, CALL 및 EXECUTE문에 사용할 수도 있습니다.

SQLDA에 있는 정보의 의미는 그 사용에 따라 다릅니다. PREPARE와 DESCRIBE에서는 SQLDA가 준비된 명령문에 대한 정보를 어플리케이션 프로그램에 제공합니다. DESCRIBE에서 SQLDA는 표 또는 보기에 있는 열에 대한 정보를 어플리케이션 프로그램에 제공합니다. OPEN, EXECUTE, CALL 및 FETCH에서는 SQLDA가 호스트 변수에 대한 정보를 제공합니다. 예를 들어, DESCRIBE문을 사용하여 SQLDA로 값을 읽고 호스트 변수의 주소로 변경한 다음, FETCH문에서 다시 사용할 수 있습니다.

어플리케이션이 동시에 여러 개의 커서를 열도록 하는 경우, 각 동적 SELECT문에 하나씩 여러 개의 SQLDA를 코딩할 수 있습니다. 자세한 정보는 SQL 참조서의 SQLDA 및 SQLCA를 참조하십시오.

SQLDA는 C, C++, COBOL, PL/I, REXX 및 RPG에서 사용할 수 있습니다. iSeries용 RPG는 포인터 설정 방법을 제공하지 않기 때문에, SQLDA를 PL/I, C, C++, COBOL 또는 iSeries용 ILE RPG 프로그램이 iSeries용 ILE RPG 프로그램 외부에서 설정해야 합니다. 그런 다음, 해당 프로그램은 iSeries용 RPG 프로그램을 호출해야 합니다.

SQLDA 형식

SQLDA는 4개의 변수와 그 뒤에 집합적으로 SQLVAR라 불리는 6개의 변수 중 임의 수의 변수로 구성됩니다.

주: REXX의 SQLDA와는 다릅니다. 자세한 내용은 *호스트 언어에 의한 SQL 프로그래밍* 정보에서 주제, REXX 어플리케이션에서 SQL문 코딩을 참조하십시오.

SQLDA가 OPEN, FETCH, CALL 및 EXECUTE에 사용될 경우, 발생되는 각 SQLVAR는 호스트 변수를 설명합니다.

SQLDA 필드는 다음과 같습니다.

SQLDAID

SQLDAID는 기억장치 덤프에 ‘eyecatcher’를 사용합니다. 이것은 PREPARE 또는 DESCRIBE문에 사용되는 SQLDA 다음에 값 SQLDA가 있는 8자의 열입니다. 이 변수는 FETCH, OPEN, CALL 또는 EXECUTE에 사용되지 않습니다.

7번째 바이트는 각 열에 대해 하나 이상의 SQLVAR 항목이 필요한지 판별하기 위해 사용됩니다. LOB 또는 고유한 유형 열이 있는 경우 다중 SQLVAR 항목이 필요할 수 있습니다. LOB 또는 고유한 유형이 없는 경우 이 플래그는 공백으로 설정됩니다.

SQLDAID는 REXX에 적용되지 않습니다.

SQLDABC

SQLDABC는 SQLDA의 길이를 표시합니다. 이것은 PREPARE 또는 DESCRIBE문에 사용되는 SQLDA 다음에 값 SQLN*LENGTH(SQLVAR) + 16이 있는 4바이트 정수입니다. SQLDABC는 FETCH, OPEN, CALL 또는 EXECUTE에 의해 사용되기 전에 SQLN*LENGTH(SQLVAR) + 16보다 크거나 같은 값으로 설정되어야 합니다.

SQLABC는 REXX에 적용되지 않습니다.

SQLN SQLN은 SQLVAR의 총 어커런스 수를 지정하는 2바이트 정수입니다. 이는 SQL문에 사용되기 전에 0보다 크거나 같은 값으로 설정되어야 합니다.

SQLN은 REXX에서는 적용되지 않습니다.

SQLD SQLD는 SQLVAR의 어커런스 수를 지정하는 2바이트 정수로 SQLDA에 의해 설명되는 호스트 변수 또는 열의 수입니다.이 필드는 SQL에 의해 DESCRIBE 또는 PREPARE문에 설정됩니다. 다른 명령문에서는 이 필드를 사용하기 전에 0보다 크거나 같고 SQLN보다 작거나 같은 값으로 설정해야 합니다.

SQLVAR

이 그룹은 각 호스트 변수 또는 열에 대해 한 번씩 반복됩니다. 이러한 변수는 SQL에 의해 DESCRIBE 또는 PREPARE문에 설정됩니다. 다른 명령문에서는 사용하기 전에 설정되어야 합니다. 이러한 변수는 다음과 같이 정의됩니다.

SQLTYPE

SQLTYPE는 293 페이지의 표 29에 표시된 호스트 변수 또는 열의 자료 유형을 지정하는 2바이트 정수입니다. SQLTYPE의 홀수 값은 호스트 변수에 SQLIND에 의해 주소 지정된 연관 인디케이터 변수가 있음을 표시합니다.

SQLLEN

SQLLEN는 호스트 변수 또는 열의 길이 속성을 지정하는 2바이트 정수 변수입니다.

SQLRES

SQLRES는 경계 정렬을 위해 확보된 12바이트 영역입니다. OS/400에서 포인터는 1/4 단어 경계에 *있어야* 합니다.

SQLRES는 REXX에서 적용되지 않습니다.

SQLDATA

SQLDATA는 SQLDA가 OPEN, FETCH, CALL 및 EXECUTE에 사용될 때, 호스트 변수의 주소를 지정하는 16바이트 포인터 변수입니다.

SQLDA가 PREPARE와 DESCRIBE에서 사용될 때, 이 영역에 다음 정보가 채워집니다.

문자 또는 그래픽 필드의 CCSID는 SQLDATA의 세 번째 및 네 번째 바이트에 저장됩니다. BIT 자료의 경우 CCSID는 65535입니다. REXX에서 CCSID는 변수 SQLCCSID에 리턴됩니다.

SQLIND

SQLIND는 SQLDA가 OPEN, FETCH, CALL 및 EXECUTE에 사용될 때 널인지의 여부를 나타내는데 사용되는 작은 정수 호스트 변수의 주소를 지정하는 16바이트 포인터입니다. 음의 값은 널을 나타내고 음수 이외의 값은 널이 아님을 나타냅니다. 이 포인터는 SQLTYPE에 홀수 값이 들어 있는 경우에만 사용됩니다.

SQLDA가 PREPARE와 DESCRIBE에 사용될 때는 이 영역이 나중에 사용되기 위해 예약되어 있습니다.

SQLNAME

SQLNAME은 길이가 30인 가변 길이 문자 변수입니다. PREPARE 또는 DESCRIBE 다음 이 변수에는 선택된 열 이름, 레이블명 또는 시스템 열 이름이 들어 있습니다. OPEN, FETCH, EXECUTE 또는 CALL에서, 이 변수는 문자 스트링의 CCSID(코드화 문자 세트 ID)를 전달하는 데 사용될 수 있습니다. CCSID는 문자 및 그래픽 호스트 그래픽에 전달될 수 있습니다.

입력 SQLDA의 SQLVAR 배열 항목의 SQLNAME 필드를 설정하여 CCSID를 지정할 수 있습니다.

자료 유형	부속 유형	SQLNAME의 길이	SQLNAME 바이트 1 & 2	SQLNAME 바이트 3 & 4
문자	SBCS	8	X'0000'	CCSID
문자	MIXED	8	X'0000'	CCSID

자료 유형	부속 유형	SQLNAME의 길이	SQLNAME 바이트 1 & 2	SQLNAME 바이트 3 & 4
문자	BIT	8	X'0000'	X'FFFF'
GRAPHIC	적용되지 않음	8	X'0000'	CCSID
기타 자료 유형	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음

주: SQLNAME 필드는 CCSID 대체용임을 명심해야 합니다. 디폴트를 사용하는 어플리케이션은 CCSID 정보를 전달할 필요가 없습니다. CCSID가 전달되지 않으면 작업에 대한 디폴트 CCSID가 사용됩니다.

그래픽 호스트 변수에 대한 디폴트는 작업 CCSID에 대해 연관된 2바이트 CCSID입니다. 연관된 2바이트 CCSID가 존재하지 않으면 65535가 사용됩니다.

표 29. PREPARE, DESCRIBE, FETCH, OPEN, CALL 또는 EXECUTE에 대한 SQLTYPE 및 SQLLEN 값

	PREPARE 및 DESCRIBE의 경우		FETCH, OPEN, CALL 및 EXECUTE의 경우	
SQLTYPE	열 자료 유형	SQLLEN	HOST VARIABLE DATA TYPE	SQLLEN
384/385	날짜	10	날짜의 고정 길이 문자 스트링 표시	호스트 변수의 길이 속성
388/389	시간	8	시간의 고정 길이 문자 스트링 표시	호스트 변수의 길이 속성
392/393	시간소인	26	시간소인의 고정 길이 문자 스트링 표시	호스트 변수의 길이 속성
396/397	자료 링크 주 #1	열의 길이 속성	N/A	N/A
400/401	N/A	N/A	널 종료 그래픽 스트링	호스트 변수의 길이 속성
392/393	시간소인	26	시간소인의 고정 길이 문자 스트링 표시	호스트 변수의 길이 속성
404/405	BLOB	0(주 #2 참조)	BLOB	사용되지 않음(주 #2 참조)
408/409	CLOB	0(주 #2 참조)	CLOB	사용되지 않음(주 #2 참조)
412/413	DBCLOB	0(주 #2 참조)	DBCLOB	사용되지 않음(주 #2 참조)
452/453	고정 길이 문자 스트링	열의 길이 속성	고정 길이 문자 스트링	호스트 변수의 길이 속성
456/457	긴 가변 길이 문자 스트링	열의 길이 속성	긴 가변 길이 문자 스트링	호스트 변수의 길이 속성
460/461	N/A	N/A	널 종료 문자 스트링	호스트 변수의 길이 속성
464/465	가변 길이 그래픽 스트링	열의 길이 속성	가변 길이 그래픽 스트링	호스트 변수의 길이 속성

표 29. PREPARE, DESCRIBE, FETCH, OPEN, CALL 또는 EXECUTE에 대한 SQLTYPE 및 SQLLEN 값 (계속)

	PREPARE 및 DESCRIBE의 경우		FETCH, OPEN, CALL 및 EXECUTE의 경우	
SQLTYPE	**열 자료 유형**	**SQLLEN**	**HOST VARIABLE DATA TYPE**	**SQLLEN**
468/469	고정 길이 그래픽 스트링	열의 길이 속성	고정 길이 그래픽 스트링	호스트 변수의 길이 속성
472/473	긴 가변 길이 그래픽 스트링	열의 길이 속성	긴 그래픽 스트링	호스트 변수의 길이 속성
476/477	N/A	N/A	PASCAL L-스트링	호스트 변수의 길이 속성
480/481	부동 소수점	단정밀도의 경우 4, 배정밀도의 경우 8	부동 소수점	단정밀도의 경우 4, 배정밀도의 경우 8
484/485	팩 10진수	바이트 1에서 정밀도, 2에서 소수 자릿수	팩 10진수	바이트 1에서 정밀도, 2에서 소수 자릿수
488/489	존(zoned) 10진수	바이트 1에서 정밀도, 2에서 소수 자릿수	존(zoned) 10진수	바이트 1에서 정밀도, 2에서 소수 자릿수
492/493	큰 정수	8	큰 정수	8
496/497	큰 정수	4(주 #3 참조)	큰 정수	4
500/501	작은 정수	2(주 #3 참조)	작은 정수	2
504/505	N/A	N/A	DISPLAY SIGN LEADING SEPARATE	바이트 1에서 정밀도, 2에서 소수 자릿수
904/905	N/A	N/A	ROWID	40
916/917	N/A	N/A	BLOB 파일 참조 변수	267
920/921	N/A	N/A	CLOB 파일 참조 변수	267
924/925	N/A	N/A	DBCLOB 파일 참조 변수	267
960/961	N/A	N/A	BLOB 위치 지정자	4
964/965	N/A	N/A	CLOB 위치 지정자	4
968/969	N/A	N/A	DBCLOB 위치 지정자	4

주:

1. DataLink 자료 유형은 DESCRIBE TABLE에서만 리턴됩니다.
2. 2차 SQLVAR의 len.sqllonglen 필드에는 열의 길이 속성이 들어 있습니다.
3. 크고 작은 2진 숫자는 길이 2 또는 4로 SQL 설명자 영역(SQLDA)에서 표시될 수 있습니다. 바이트 1에는 정밀도로, 바이트 2에는 스케일로 표시될 수도 있습니다. 첫 번째 바이트가 X'00'보다 큰 경우, 정밀도와 스케일을 나타냅니다. 큰 정수 숫자는 정밀도와 스케일을 허용하지 않습니다. SQLDA는 큰 정수 숫자를 길이 5로 정의합니다.

SQLVAR2

이것은 세 개의 필드를 포함하는 확장 SQLVAR 구조입니다. 결과가 고유한 유형 또는 LOB 열을 포함한 경우, 모든 열에 확장 SQLVAR이 필요합니다. 고

유한 유형의 경우, 고유한 유형 이름이 들어 있습니다. LOB의 경우, 호스트 변수의 길이 속성과 실제 길이가 들어 있는 버퍼를 가리키는 포인터가 들어 있습니다. LOB를 표시하기 위해 위치 지정자가 사용되는 경우, 이 항목은 필요하지 않습니다. 필요한 확장 SQLVAR 발생 수는 SQLDA가 제공된 명령문과 서술되고 있는 매개변수 또는 열의 자료 유형에 따라 달라집니다. SQLDAID의 7번째 바이트는 항상 필요한 SQLVAR 집합수로 설정됩니다.

SQLD가 충분한 SQLVAR 발생수로 설정되지 않은 경우

- SQLD는 모든 집합에 대해 필요한 총 SQLVAR 발생수로 설정됩니다.
- 기본 SQLVAR 항목에 대해 최소한 충분하게 지정되지 않았으면 SQLCA의 SQLCODE 필드에 +237 경고가 리턴됩니다. 기본 SQLVAR 항목은 리턴되지만 확장 SQLVAR는 리턴되지 않습니다.
- 기본 SQLVAR 항목에 대해서도 충분한 SQLVAR가 지정되지 않았으면 SQLCA의 SQLCODE 필드에 +239 경고가 리턴됩니다. SQLVAR 항목은 리턴되지 않습니다.

SQLLONGLEN

SQLLONGLEN은 LOB(BLOB, CLOB 또는 DBCLOB) 호스트 변수 또는 열의 길이 속성을 지정하는 4바이트 정수 변수입니다.

SQLDATALEN

SQLDATALEN은 호스트 변수의 길이 주소를 지정하는 16바이트 포인터 변수입니다. 이 변수는 LOB(BLOB, CLOB 및 DBCLOB) 호스트 변수에만 사용됩니다. 이는 DESCRIBE 또는 PREPARE에는 사용되지 않습니다.

이 필드가 NULL이면, 자료의 실제 길이는 자료 시작 부분 바로 앞 4바이트에 저장되며 SQLDATA는 필드 길이의 첫 바이트를 가리킵니다. 길이는 BLOB 또는 CLOB에 대한 바이트 수와 DBCLOB에 대한 문자 수를 나타냅니다.

이 필드가 NULL이 아니면, 일치하는 기본 SQLVAR의 SQLDATA 필드가 가리키는 버퍼 자료의 바이트 단위로 실제 길이(DBCLOB에 대해서도)가 들어 있는 4바이트 길이의 버퍼에 대한 포인터가 들어 있습니다.

SQLDATATYPE_NAME

SQLDATATYPE_NAME은 길이가 30인 가변 길이 문자 변수입니다. 이는 DESCRIBE 또는 PREPARE에만 사용됩니다. 이 변수는 다음 중 하나로 설정됩니다.

- 고유한 유형의 열에 대해, 데이터베이스 관리자는 이 값을 완전히 규정된 고유한 유형 이름으로 설정합니다. 규정된 이름이 30바이트보다 길면, 이 값은 절단됩니다.

- 레이블에 대해 데이터베이스 관리자는 이 값을 레이블의 처음 20바이트로 설정합니다.
- 열 이름에 대해, 데이터베이스 관리자는 이 값을 열 이름으로 설정합니다.

예: SQLDA에 기억장치 할당을 위한 SELECT문

사용자 어플리케이션이 사용할 때마다 달라지는 동적 SELECT문을 처리해야 한다고 가정합니다.이 명령문은 표시장치에서 읽히거나 다른 응용프로그램에서 전달되거나 응용프로그램에 의해 빌드될 수 있습니다. 즉 사용자는 매번 이 명령문이 무엇을 리턴할 것인지 정확히 알 수 없습니다. 사용자 어플리케이션은 미리 알 수 없는 데이터 유형을 갖는 다양한 수의 결과 열을 처리할 수 있어야 합니다.

예를 들어, 다음 명령문을 처리해야 합니다.

```
   SELECT WORKDEPT, PHONENO
        FROM CORPDATA.EMPLOYEE
WHERE LASTNAME = 'PARKER'
```

주: 이 SELECT문에는 INTO절이 없습니다. 동적 SELECT문은 한 행만을 리턴하는 경우에도 INTO절을 *가져서는 안됩니다*.

명령문은 호스트 변수에 지정됩니다. 이 경우 이름이 DSTRING인 호스트 변수는 다음과 같이 PREPARE문을 사용하여 처리됩니다.

```
EXEC SQL
PREPARE S1 FROM :DSTRING;
```

다음으로 결과 열의 수와 데이터 유형을 판별해야 합니다. 이를 수행하기 위해 SQLDA가 필요합니다.

SQLDA를 정의하는 첫 번째 단계는 기억장치를 할당하는 것입니다(REXX에서는 기억영역 할당이 필요하지 않습니다). 기억영역을 확보하는 기법은 언어에 따라 다릅니다. SQLDA는 16바이트 경계에 할당되어야 합니다. SQLDA는 16바이트 길이의 고정 길이 헤더로 구성됩니다. 헤더 다음에 가변 길이 배열 섹션(SQLVAR)이 있고 각 요소의 길이는 80바이트입니다.

할당에 필요한 기억장치의 *크기*는 SQLVAR 배열에 지정하려는 요소 수에 따라 다릅니다. 선택한 각 열은 해당 SQLVAR 배열 요소를 가져야 합니다. 그러므로 SELECT문에 작성된 열의 수가 할당할 SQLVAR 배열 요소 수를 결정합니다. 이 SELECT문은 런타임 시에 지정되므로, 액세스될 열 수를 알 수 없습니다. 그러므로 사용자가 열 수를 평가해야 합니다. 이 예는 20열 이상의 열이 단일 SELECT문에 의해 액세스될 수 없음을 암시합니다. 이 경우 SQLVAR 배열에 20 차원이 있어야 하고 각 선택 목록은 SQLVAR에 대응 항목을 가지고 있어야 합니다. 그러면 전체 SQLDA *크기*는 20 x 80 즉 1600에 16을 더해 전체 1616바이트가 됩니다.

SQLN 필드에 SQLDA로 평가한 충분한 공간을 할당한 후, SQLDA의 SQLN 필드를 SQLVAR 배열 요소 수와 동일하게 설정해야 합니다(이 경우 20).

기억장치를 할당한 후 DESCRIBE문을 발행할 수 있습니다.

```
EXEC SQL
DESCRIBE S1 INTO :SQLDA;
```

DESCRIBE문이 수행될 때 SQL은 명령문의 선택 리스트에 대한 정보를 제공하는 값을 SQLDA에 위치시킵니다. 다음 표에서는 DESCRIBE가 실행된 후의 SQLDA 내용을 보여줍니다. 이 문맥에서 의미있는 항목만 표시됩니다.

SQLDA 헤더는 다음을 포함해야 합니다.

표 30. SQLDA 헤더

설명	값
SQLAID	'SQLDA'
SQLDABC	1616
SQLN	20
SQLD	2

SQLDAID는 DESCRIBE가 수행될 때 SQL에 의해 초기설정된 식별자 필드입니다. SQLDABC는 SQLDA의 바이트 수 또는 크기입니다. SQLDA 헤더 다음에는 SQLVAR 구조가 두 번 나오는데, 설명된 SELECT문의 결과 표의 각 열에 대한 것입니다.

표 31. SQLVAR 요소 1

설명	값
SQLTYPE	453
SQLLEN	3
SQLDATA(3:4)	37
SQLNAME	8 WORKDEPT

표 32. SQLVAR 요소 2

설명	값
SQLTYPE	453
SQLLEN	4
SQLDATA(3:4)	37
SQLNAME	7 PHONENO

SQLDA가 설명된 SQLVAR 요소가 들어갈 만큼 충분히 크지 않으면 프로그램이 SQLN 값을 변경해야 합니다. 예를 들어 예상 최대값인 20 열 대신 SELECT문은 실제 27을 리턴한다고 합시다. SQLVAR이 할당된 공간에서 허용되는 것보다 더 많은 요소를 필요로하기 때문에 SQL이 이 선택 목록을 설명할 수 없습니다. 대신, SQL은 SQLD를

SELECT문에 의해 지정된 실제 열 수로 설정하고 구조의 나머지를 무시합니다. 그러므로 DESCRIBE 뒤의 SQLN 값을 SQLD 값과 비교해야 합니다. SQLD의 값이 SQLN의 값보다 클 경우, 다음과 같이 SQLD 값에 따라 더 큰 SQLDA를 할당하고 DESCRIBE를 다시 수행하십시오.

```
EXEC SQL
    DESCRIBE S1 INTO :SQLDA;
IF SQLN <= SQLD THEN
DO;

/*Allocate a larger SQLDA using the value of SQLD.*/
/*Reset SQLN to the larger value.*/

EXEC SQL
    DESCRIBE S1 INTO :SQLDA;
END;
```

비SELECT문에서 DESCRIBE를 사용하는 경우, SQL은 SQLD를 0으로 설정합니다. 따라서 프로그램이 SELECT와 비SELECT문을 모두 처리하도록 설계된 경우, SELECT문인지 아닌지를 판별하도록 준비된 후 각 명령문을 서술할 수 있습니다. 이 예는 SELECT문만을 처리하도록 설계되었습니다. SQLD 값은 검사되지 않습니다.

사용자의 프로그램이 성공적 DESCRIBE에서 리턴된 SQLVAR의 요소를 분석해야 합니다. SELECT 리스트의 첫 번째 항목은 WORKDEPT입니다. SQLTYPE 필드에서, DESCRIBE은 표현식의 자료 유형에 대한 값과 널(null) 적용 여부를 리턴합니다(293 페이지의 표 29 참조).

이 예에서, SQL은 SQLTYPE을 SQLVAR 요소 1에 있는 453으로 설정합니다. 이것는 WORKDEPT가 고정 길이 문자 스트링 열이고 열에 널(null)이 허용되지 않음을 지정합니다.

SQL은 열 길이로 SQLLEN을 설정합니다. WORKDEPT의 자료 유형이 CHAR이므로, SQL이 SQLLEN 길이를 문자 열의 길이와 같게 설정합니다. WORKDEPT의 경우, 그 길이는 3입니다. 그러므로 SELECT문이 후에 수행될 때 CHAR(3) 스트링이 들어갈 만큼 큰 기억영역이 필요합니다.

WORKDEPT의 자료 유형이 CHAR FOR SBCS DATA이므로 SQLDATA의 첫 번째 4바이트는 문자 열의 CCSID로 설정되었습니다.

SQLVAR 요소의 최종 필드는 SQLNAME이라는 가변 길이 문자 스트링입니다. SQLNAME의 처음 2바이트에는 문자 자료의 길이가 들어 있습니다. 문자 자료 자체는 대개 SELECT문에 사용된 열 이름입니다(이 경우 WORKDEPT). 예외사항으로는 함수(예: SUM(SALARY)), 표현식(예: A+B-C), 상수 등 명명되지 않은 선택 리스트 항목이 있습니다. 이 경우 SQLNAME은 빈 스트링입니다. 또한 SQLNAME에는 이름 대신 레이블이 들어 있을 수 있습니다. PREPARE와 DESCRIBE문과 연관된 매개변수 중 하나는 USING절입니다. 다음과 같이 이를 지정할 수 있습니다.

```
EXEC SQL
    DESCRIBE S1 INTO:SQLDA
        USING LABELS;
```

다음을 지정할 경우입니다.

NAMES(또는 USING 매개변수 전체를 생략할 경우)
열 이름만이 SQLNAME 필드에 위치됩니다.

SYSTEM NAMES
시스템 열 이름만이 SQLNAME 필드에 위치됩니다.

LABELS
SQL문에 나열된 열과 연관된 레이블만이 여기에 입력됩니다.

ANY 레이블은 레이블이 있는 해당 열에 대한 SQLNAME 필드에 위치됩니다. 그렇지 않으면, 열 이름이 입력됩니다.

BOTH
이름 및 레이블은 모두 해당 길이가 있는 필드에 위치됩니다. 요소 수의 두 배를 포함를 포함하므로, SQLVAR 배열의 *크기*를 두 배로 해야 함을 기억하십시오.

ALL 열 이름, 레이블 및 시스템 열 이름은 해당 길이가 있는 필드에 위치됩니다. SQLVAR 배열의 *크기*를 세 배로 해야 함을 기억하십시오.

USING 옵션에 대한 자세한 정보는 *SQL 참조서*의 DESCRIBE문 및 SQLDA 절을 참조하십시오.

이 예에서, 두 번째 SQLVAR 요소에는 PHONENO 선택에 사용된 2차 열에 대한 정보가 들어 있습니다. SQLTYPE의 453 코드는 PHONENO가 CHAR 열임을 지정합니다. SQLLEN은 4로 설정됩니다.

이제 SELECT문을 실행할 때 SQLDA를 사용하여 값을 검색하도록 설정해야 합니다.

DESCRIBE의 결과를 분석한 후 SELECT문의 결과가 들어 있는 변수에 대해 기억장치를 할당할 수 있습니다. WORKDEPT에는 길이 3의 문자 필드가 할당되어야 합니다. PHONENO에는 길이 4의 문자 필드가 할당되어야 합니다. 이 결과 모두 널(null) 값이 될 수 있으므로 인디케이터 변수는 각 필드마다 할당되어야 합니다.

기억장치가 할당된 후 할당된 기억장치 영역을 나타내기 위해 SQLDATA 및 SQLIND를 설정해야 합니다. SQLVAR 배열의 각 요소에 대해 SQLDATA는 결과 값이 들어갈 장소를 나타냅니다. SQLIND는 널(null) 인디케이터 값이 들어갈 장소를 나타냅니다. 다음 표에서는 이제 구조를 보여줍니다. 이 문맥에서 의미있는 항목만 표시됩니다.

표 33. SQLDA 헤더

설명	값
SQLAID	'SQLDA'
SQLDABC	1616
SQLN	20
SQLD	2

표 34. SQLVAR 요소 1

설명	값
SQLTYPE	453
SQLLEN	3
SQLDATA	CHAR(3) 결과 영역에 대한 포인터
SQLIND	결과 열에 대한 2바이트 정수 인디케이터에 대한 포인터

표 35. SQLVAR 요소 2

설명	값
SQLTYPE	453
SQLLEN	4
SQLDATA	CHAR(4) 결과 영역에 대한 포인터
SQLIND	결과 열에 대한 2바이트 정수 인디케이터에 대한 포인터

이제 SELECT문 결과를 검색할 준비가 되었습니다. 동적으로 정의된 SELECT문에는 INTO문이 없어야 합니다. 그러므로 동적으로 정의된 모든 SELECT문은 커서를 사용해야 합니다. DECLARE, OPEN 및 FETCH의 특별 양식은 동적으로 정의된 SELECT문을 사용합니다.

DECLARE문의 예는 다음과 같습니다.

```
EXEC SQL DECLARE C1 CURSOR FOR S1;
```

위에서 볼 수 있듯이 유일한 차이점은 준비된 SELECT문(SI)의 이름이 그 자체의 SELECT문 대신 사용되는 것입니다. 결과 행의 실제 검색은 다음과 같이 수행됩니다.

```
EXEC SQL
            OPEN C1;
EXEC SQL
    FETCH C1 USING DESCRIPTOR :SQLDA;
DO WHILE (SQLCODE = 0);
/*Process the results pointed to by SQLDATA*/
EXEC SQL
    FETCH C1 USING DESCRIPTOR :SQLDA;
END;
EXEC SQL
           CLOSE C1;
```

커서가 열립니다. SELECT로의 결과 행은 FETCH문을 사용하여 한 번에 하나씩 리턴됩니다. 이 때에는 FETCH문에 출력 호스트 변수의 리스트가 없는 대신, FETCH문은 SQL이 SQLDA에 의해 기술되는 영역으로 결과를 리턴하도록 합니다. 결과는 SQLVAR 요소의 SQLDATA와 SQLIND 필드에 의해 지시되는 기억장치로 리턴됩니다. FETCH 문이 처리된 후 WORKDEPT에 대한 SQLDATA 포인터는 참조된 값을 'E11'로 설정합니다. 널이 아닌 값이 리턴되므로 그에 대응하는 인디케이터 값은 0입니다. PHONENO에 대한 SQLDATA 포인터는 참조된 값을 '4502'에 설정합니다. 널이 아닌 값이 리턴되므로 그에 대응하는 인디케이터 값도 0입니다.

매개변수 마커

사용되는 예에서, 동적으로 실행된 SELECT문에는 WHERE절의 상수 값이 있습니다. 예를 들면 다음과 같습니다.

```
WHERE LASTNAME = 'PARKER'
```

같은 SELECT문을 여러 번 수행하려는 경우, LASTNAME에 대해 다른 값을 사용하여 다음과 같은 SQL문을 사용할 수 있습니다.

```
SELECT WORKDEPT, PHONENO
    FROM CORPDATA.EMPLOYEE
    WHERE LASTNAME = ?
```

매개변수를 예측할 수 없으면 어플리케이션은 수행 시간까지 매개변수의 수 및 유형을 알 수 없습니다. 어플리케이션 실행시 이 정보를 수신하도록 배열하고 USING DESCRIPTOR를 OPEN문에 사용하여 SELECT문의 WHERE절에 포함된 매개변수 마커의 특정 호스트 변수에 포함된 값을 대체할 수 있습니다.

이와 같이 프로그램을 코딩하기 위해서는 USING DESCRIPTOR절이 있는 OPEN문을 사용할 필요가 있습니다. 이 SQL문은 커서를 여는데만 아니라 해당되는 호스트 변수의 값으로 각각의 매개변수 마커를 대체하는데 사용됩니다. 이 명령문에 지정하는 설명자명은 그러한 호스트 변수에 대한 유효한 설명이 들어 있는 SQLDA를 식별해야 합니다. 이 SQLDA는 이전에 설명한 것과는 달리 SELECT 리스트의 일부인 자료 항목에 대한 정보를 리턴하는데 사용되지 않습니다. 즉, 이것은 DESCRIBE문의 출력으로 사용되는 것이 아니라 OPEN문에 대한 입력으로 사용됩니다. SQLDA는 SELECT문의 WHERE절에 있는 매개변수 마커를 대체하는데 사용되는 호스트 변수에 대한 정보를 제공합니다. SQLDA는 어플리케이션으로부터 이 정보를 얻으며 어플리케이션은 SQLDA의 필요한 필드에 적절한 값을 위치시키도록 설계되어야 합니다. 그런 후 SQLDA는 매개변수 마커를 호스트 변수 자료로 대체하는 처리에서 SQL에 대한 정보의 소스로서 사용될 준비가 됩니다.

SQLDA를 USING DESCRIPTOR절이 있는 OPEN CURSOR문의 입력으로서 사용할 때 필드 전체가 채워질 필요는 없습니다. 특히 SQLDAID, SQLRES 및 SQLNAME

은 공백으로 남을 수 있습니다.(특정 CCSID가 필요할 경우에는 SQLNAME이 설정될 수 있습니다.) 그러므로 이 방식을 사용하여 매개변수 마커를 호스트 변수 값으로 대체할 때 다음을 결정해야 합니다.

- 매개변수 마커의 수
- 매개변수 마커의 자료 유형 및 속성(SQLTYPE, SQLLEN 및 SQLNAME)
- 인디케이터 변수의 필요 여부

또한 루틴이 SELECT문과 비SELECT문을 처리할 경우, 사용자는 그것이 어떤 범주의 명령문인지 판별하려할 것입니다(또는 SELECT 키워드를 찾는 코드를 작성할 수 있습니다).

어플리케이션이 매개변수 마커를 사용할 경우, 프로그램은 다음을 만족시켜야 합니다.

1. 명령문을 DSTRING 가변 길이의 문자 스트링 호스트 변수로 읽어들여야 합니다.
2. 매개변수 마커의 수를 판별해야 합니다.
3. 해당 크기의 SQLDA를 할당해야 합니다.
   이는 REXX에서는 적용되지 않습니다.
4. 매개변수 마커의 수에 따라 SQLN과 SQLD를
   SQLN은 REXX에서는 적용되지 않습니다.
5. SQLDABC를 SQLN*LENGTH(SQLVAR) + 16과 동일하게 설정해야 합니다.
   이는 REXX에서는 적용되지 않습니다.
6. 각 매개변수 마커의 경우, 다음과 같습니다.
   a. 자료 유형, 길이 및 인디케이터를 결정해야 합니다.
   b. SQLTYPE과 SQLLEN을 설정해야 합니다.
   c. 입력 값(?값)을 보유하기 위한 기억영역을 할당해야 합니다.
   d. 이들 값을 설정해야 합니다.
   e. 각 매개변수 마커에 대한 SQLDATA 및 SQLIND(적용 가능할 경우)를 설정해야 합니다.
   f. 문자 변수가 사용되며 문자 변수가 작업 디폴트 CCSID 이외의 CCSID에 있을 경우, SQLNAME(REXX에서는 SQLCCSID)을 설정해야 합니다.
   g. 그래픽 변수가 사용되며 작업 CCSID에 대해 연관된 DBCS CCSID 이외의 CCSID를 가질 경우, SQLNAME(REXX에서는 SQLCCSID)을 이 CCSID로 설정해야 합니다.
   h. 커서를 열고 각 매개변수 마커의 호스트 변수값을 대체하기 위해 USING DESCRIPTOR절을 가진 OPEN문을 발행해야 합니다.

그런 다음 명령문은 정상적으로 처리될 수 있습니다.

제 15 장 클라이언트 인터페이스를 통한 동적 SQL 사용

서버에서 클라이언트 인터페이스를 통해 iSeries용 DB2 UDB 자료에 액세스할 수 있습니다. 다음 주제를 사용하여 요구된 타스크를 시작할 수 있습니다.

- 『Java로 자료 액세스』
- 『Domino로 자료 액세스』
- 『ODBC(Open Database Connectivity)를 사용하여 자료에 액세스』
- 304 페이지의 『PASE(Portable Application Solutions Environment)를 사용하여 자료에 액세스』

Java로 자료 액세스

JDBC(Developer Kit for Java Database Connectivity) 드라이버로 Java 프로그램의 iSeries용 DB2 UDB 자료에 액세스할 수 있습니다. 드라이버를 사용하여 다음의 타스크를 수행합니다.

- 데이터베이스 파일에 액세스
- Java용 삽입 구조화 조회 언어(SQL)로 JDBC 데이터베이스 기능에 액세스
- fQL문 실행 및 결과 처리

JDBC 드라이버를 사용하는 방법에 대한 자세한 내용은 iSeries Information Center에서 ″IBM Developer Kit for Java JDBC 드라이버를 사용하기 위한 설정″ 주제를 참조하십시오.

Domino로 자료 액세스

iSeries용 Domino는 iSeries용 DB2 UDB 데이터베이스와 Domino 데이터베이스의 자료를 양 방향으로 통합시키는 Domino 서버 제품입니다. 이 통합 방법을 이용하려면 인증서가 두 유형의 데이터베이스 사이에서 작용하는 방법을 이해하고 관리해야 합니다. 자세한 내용은 iSeries Information Center의 iSeries용 Domino 범주를 참조하십시오.

ODBC(Open Database Connectivity)를 사용하여 자료에 액세스

Windows용 iSeries Access ODBC 드라이버를 사용하면 ODBC 클라이언트 어플리케이션들은 상호간과 서버와의 사이에 자료를 효과적으로 공유할 수 있습니다. iSeries Information Center의 Windows용 iSeries Access 범주에서 ″ODBC 관리″를 참조하십시오.

Linux용 iSeries ODBC 드라이버 주제에서 Linux를 사용하여 연결하는 것에 대한 내용을 참조할 수 있습니다. 이 주제에서는 iSeries 논리파티션에 Linux를 설치하고 Linux용 iSeries ODBC 드라이버를 설치 및 사용하여 iSeries 데이터베이스에 액세스하는 것에 대하여 설명합니다.

PASE(Portable Application Solutions Environment)를 사용하여 자료에 액세스

PASE(Portable Application Solutions Environment)는 iSeries 시스템에서 실행되고 있는 AIX(또는 다른 UNIX와 유사한) 어플리케이션을 위한 통합 실행시 환경입니다. 자세한 내용은 iSeries Information Center의 통합 운영 환경 범주에서 ″OS/400 PASE″를 참조하십시오.

제 16 장 iSeries Navigator를 사용한 확장 데이터베이스 기능

이 장에서는 iSeries Navigator를 사용하여 수행할 수 있는 데이터베이스에 대한 일부 확장 기능을 다룹니다. 개요 정보와 추가 정보가 제공됩니다. 37 페이지의 제 3 장 『iSeries Navigator 데이터베이스 시작』에는 일부 기본 기능이 들어 있습니다. 여기에 포함된 주제는 다음과 같습니다.

- 『Database Navigator를 사용하여 데이터베이스 맵핑』
- 309 페이지의 『SQL 스크립트 실행을 사용하여 데이터베이스 조회』
- 312 페이지의 『SQL 생성을 사용한 SQL문 재구성』
- 313 페이지의 『iSeries Navigator를 사용한 확장 표 기능』
- 319 페이지의 『iSeries Navigator를 사용한 SQL 오브젝트 정의』
- 321 페이지의 『SQL 패키지 작성』

SQL에 기초하지 않는 iSeries Navigator에서 수행할 수 있는 다른 기능은 데이터베이스 프로그래밍에서 다음의 주제를 참조하십시오.

- 표, 보기 및 색인 설명 표시
- 표 재구성
- 잠긴 행 표시

성능 관련 타스크는 데이터베이스 성능 및 조회 최적화를 참조하십시오.

Database Navigator를 사용하여 데이터베이스 맵핑

Database Navigator를 사용하여 시스템에서 데이터베이스 오브젝트들 간의 관계를 시각적으로 설명할 수 있습니다. 데이터베이스에 대해 작성한 시각적 설명을 Database Navigator 맵이라고 합니다. 원래 Database Navigator 맵은 데이터베이스의 스냅샷이며 맵의 모든 오브젝트들 간에 존재하는 관계입니다.

Database Navigator를 사용하면, 데이터베이스의 표, 표들 간의 관계 및 표에 첨부된 색인과 제한사항을 제시하는 그래픽 표시를 사용하여 데이터베이스 오브젝트의 복잡한 관계를 탐색할 수 있습니다. 시스템에 연결한 후에 Database Navigator를 사용하여 다음을 수행할 수 있습니다.

- Database Navigator 맵 작성
- 맵에 새로운 오브젝트 추가
- 맵에 포함시킬 오브젝트 변경
- 사용자 정의 관계 작성

Database Navigator의 주요 작업 공간은 여러 개의 기본 영역으로 분할된 창입니다. 이러한 영역들을 통해 맵에 포함시킬 오브젝트를 찾고, 맵에서 항목을 표시하고 숨기고, 맵을 보고, 맵에 대해 지연 중인 변경사항의 상태를 검사할 수 있습니다. 다음은 Database Navigator 창의 기본 영역에 대한 설명을 제공합니다.

위치 지정자 분할 창

Database Navigator 창의 왼쪽에 있는 위치 지정자 분할 창은 새로운 맵에 포함시키려는 오브젝트를 찾거나 열린 맵의 일부인 오브젝트를 찾는 데 사용됩니다. 상단 위치 지정자 분할 창은 맵에 포함시키려는 오브젝트의 이름, 유형 및 라이브러리를 지정하는 데 사용할 수 있는 탐색 기능입니다. 탐색의 결과는 라이브러리 트리 및 라이브러리 표 탭 아래에 있는 하단 위치 지정자 분할 창에 표시됩니다. 결과가 이러한 탭들 아래에 표시되면, 해당 오브젝트에서 마우스 오른쪽 버튼을 클릭하고 맵에 추가를 선택하거나 오브젝트명을 두 번 클릭하여 맵에 오브젝트를 추가할 수 있습니다. 그런 다음, 맵이 작성되면 맵의 오브젝트 탭을 클릭하여 맵에 있는 오브젝트의 리스트를 볼 수 있습니다.

맵 분할 창

Database Navigator 창의 오른쪽에 있는 맵 분할 창은 데이터베이스 오브젝트들과 이들 간의 관계를 그래픽으로 표시합니다. 맵 분할 창에서 다음을 수행할 수 있습니다.

- 시스템에 존재하지만 원래 맵의 현재 인스턴스에 포함되지 않았던 표와 보기 추가
- 맵에서 오브젝트 제거
- 오브젝트 위치 변경
- 오브젝트에서 확대 및 축소
- 맵에서 오브젝트 변경
- 맵의 모든 오브젝트에 대해 SQL 생성

상태 표시줄

오브젝트 상태 표시줄

Database Navigator 창의 왼쪽 하단에 위치한 오브젝트 상태 표시줄은 맵에서 볼 수 있으며 자격이 있는 오브젝트의 수를 표시합니다.

조치 상태 표시줄

Database Navigator 창의 하단 중앙에 위치한 조치 상태 표시줄은 맵에서 어떠한 상태가 발생했으며 수정의 지연 여부에 대한 확실한 설명을 제공합니다.

수정 상태 표시줄

수정 상태 표시줄은 수정이 완료되었는지 아니면 지연되었는지를 나타냅니다.

Database Navigator 사용 추가 정보

- 창의 한 쪽 *크기*를 변경하려면, 양쪽을 분리하는 막대(분리자)를 끌어주십시오.

- 창의 왼쪽과 오른쪽 모두에서 오브젝트를 마우스 오른쪽 버튼으로 클릭하십시오. 마우스 오른쪽 버튼을 클릭하여 나타나는 메뉴는 공통 기능에 빠르게 액세스하게 합니다.
- 라이브러리를 빨리 열고 여기에 오브젝트를 표시하려면, 라이브러리를 두 번 클릭하십시오.
- 다양한 Database Navigator 명령에 액세스하려면, **메뉴** 바나 **툴바**를 사용하십시오.

Database Navigator 맵 작성

데이터베이스에 대해 작성하는 시각적 설명을 Database Navigator 맵이라고 합니다. 맵을 작성하거나 기존의 맵을 화면정리하도록 선택할 때 맵은 실제로 데이터베이스 자료의 스냅샷입니다. 이렇게 데이터베이스를 그림으로 표현하는 것은 기존의 복잡한 데이터베이스를 이해하고 새로운 데이터베이스를 작성하고 데이터베이스와 대화하고 데이터베이스의 오브젝트를 관리하는 능력을 제공하기 위한 것입니다. Database Navigator 맵을 작성하려면, 다음과 같이 하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **Database Navigator**를 펼치십시오.
2. **Database Navigator**에서 마우스 오른쪽 버튼을 클릭하고 **신규**를 선택하십시오.
3. **Database Navigator 위치 지정자 분할 창**에서 **라이브러리 트리** 탭을 클릭한 다음 맵을 작성하는 데 사용하려는 표, 보기 또는 색인이 들어 있는 라이브러리를 선택하십시오.
4. 오브젝트 유형(표, 색인 또는 보기)을 펼치십시오.
5. 맵을 작성하려는 오브젝트에서 마우스 오른쪽 버튼을 클릭하고 **맵에 추가**를 선택하거나 오브젝트를 두 번 클릭하십시오.

   주: iSeries Navigator 창의 하단에 있는 타스크 패드에서 **데이터베이스 맵핑** 타스크를 클릭하여 맵을 작성할 수도 있습니다.

파일 메뉴에서 **나감**을 선택하여 이 맵을 저장할 수 있습니다. 그런 다음, 변경이 지연되는 경우, **변경사항 저장 위치** 대화 상자에서 **예**를 선택하십시오. 이 맵은 나중에 다시 열 수 있습니다.

데이터베이스에 대해 이 맵을 작성했으면 다음을 수행할 수 있습니다.

- 맵에 새로운 오브젝트 추가
- 맵에 포함시킬 오브젝트 변경
- 사용자 정의 관계 작성

맵에 새로운 오브젝트 추가

Database Navigator를 사용하면 맵에 추가할 새로운 SQL 오브젝트를 작성할 수 있습니다. 작성할 수 있는 오브젝트는 다음과 같습니다.

- 표
- 저널
- 보기

맵에 표시할 새로운 SQL 오브젝트를 작성하려면, 다음과 같이 하십시오.

1. Database Navigator 맵을 작성하거나 여십시오.
2. 맵 분할 창 안에서 마우스 오른쪽 버튼을 클릭하고 **작성**을 선택하십시오.
3. 작성하려는 오브젝트의 유형을 선택하십시오.

맵에 포함시킬 오브젝트 변경

디폴트로 Database Navigator는 맵의 모든 오브젝트를 탐색하고 포함시킵니다. 탐색되는 오브젝트의 수를 제한하기 위해 사용자 기본설정을 변경할 수 있습니다.

맵에 포함시킬 오브젝트를 변경하려면, 다음과 같이 수행하십시오.

1. Database Navigator 맵을 작성하거나 여십시오.
2. **옵션** 메뉴에서 **사용자 기본설정**을 선택하십시오.
3. **사용자 기본설정** 대화 상자의 **맵에 오브젝트 추가 시 다음의 관련 오브젝트 찾기** 그룹 상자에서 포함시키려는 오브젝트를 선택하거나 포함시키지 않으려는 오브젝트를 선택 취소하십시오.
4. **확인**을 클릭하십시오.
5. 맵을 새로운 기본설정으로 화면정리하려면, 정보 상자에서 **예**를 클릭하십시오.

사용자 정의 관계 작성

프로그램에 의해 정의된 관계를 가지고 있으면, 관계가 맵에 표시되도록 Database Navigator에서 사용자 정의 관계를 작성할 수 있습니다. 이에 대한 예는 프로그래머에게 두 표들 사이의 중요한 결합을 상기시키기 위해 사용자 정의 관계를 작성할 수 있습니다.

맵에 사용자 정의 관계를 추가하려면, 다음을 수행하십시오.

1. Database Navigator 맵을 작성하거나 여십시오.
2. 맵에서 마우스 오른쪽 버튼을 클릭하고 **작성**을 선택하십시오.
3. **사용자 정의 관계**를 선택하십시오.
4. 사용자 정의 관계에 대한 **이름**과 **설명**을 지정하십시오. 설명이 선택적인 일부 iSeries Navigator 기능과 달리, 사용자 정의 관계에 대한 의미있는 설명을 제공하는 것이 중요합니다. 이것은 사용자 정의 관계가 나타내는 바를 표시할 수 있는 유일한 방법이기 때문입니다.
5. 오브젝트 리스트에서 선택하여 관계에 포함시키려는 오브젝트를 선택하십시오.
6. 오브젝트에 대해 원하는 모양과 색상을 선택하십시오.

SQL 스크립트 실행을 사용하여 데이터베이스 조회

iSeries Navigator에서 **SQL 스크립트 실행** 창을 통해 SQL문의 스크립트를 작성, 편집, 실행 및 문제점 해결할 수 있습니다. 스크립트에 대한 작업을 완료했으면 PC에 저장할 수 있습니다. SQL 스크립트 실행을 사용하면 다음의 작업이 가능합니다.

- SQL 스크립트 작성
- SQL 스크립트 실행
- 저장 프로시듀어 세트 결과 보기
- 작업 기록부 보기
- SQL 스크립트 실행에 대한 옵션 변경
- 데이터베이스 오브젝트에 대한 SQL 생성

SQL 스크립트 실행 창에는 몇 개의 키 영역이 있습니다.

입력

SQL 스크립트 실행 창의 상단 부분은 **입력** 분할 창입니다. 이 영역은 실행하려는 SQL문을 작성하고 편집하는 데 사용됩니다. 명령문을 수동으로 작성하거나 **예** 리스트에서 선택할 수 있습니다. **SQL 생성** 기능을 사용하여 현재 커서 위치에서 생성된 SQL을 삽입할 수도 있습니다.

예 **예** 리스트 상자는 SQL문과 제어 언어(CL) 명령의 예들을 나열합니다. 명령문을 선택하고 삽입을 클릭하여 입력 분할 창의 현재 커서 위치에 예를 배치하십시오.

주: 각 명령문은 세미콜론으로 분리해야 합니다.

출력

SQL 스크립트 실행 창의 하단 부분은 **출력** 분할 창으로 **메세지** 탭과 실행되는 SQL문의 출력을 표시하는 추가 탭으로 구성되어 있습니다. **메세지** 탭은 실행되는 SQL문의 실행에 기초하여 피드백을 제공합니다.

SQL 스크립트 실행에 대한 추가 정보 창

- iSeries Navigator를 시작하지 않고 SQL 스크립트 기능을 사용할 수 있습니다. 스크립트 파일을 저장했으면 스크립트 파일을 두 번 클릭하여 iSeries Navigator를 시작하지 않고 이를 사용할 수 있습니다.
- 세미콜론(;)을 사용하여 명령문들을 분리하십시오.
- 두 개의 대시(--)를 사용하여 행의 나머지 부분을 주석으로 만드십시오.
- 보다 긴 주석을 작성하려면, ″/*″로 주석을 시작하고 ″*/″를 사용하여 끝내십시오. 이러한 방식으로 작성된 주석은 길이에 상관없습니다.
- 명령문에 커서를 놓으면 실행 시에 이 명령문이 자동으로 선택됩니다.
- 제어 언어(CL) 명령은 명령문의 시작 부분에 CL:을 배치하여 실행할 수 있습니다. 일괄처리 작업에서 제출할 수 있는 어떠한 CL 명령이든지 사용할 수 있습니다.

SQL 스크립트 작성

SQL 스크립트를 작성하려면, 다음과 같이 하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스를 펼치십시오.
2. 작업하려는 데이터베이스에서 마우스 오른쪽 버튼을 클릭하고 **SQL 스크립트 실행**을 선택하십시오.
3. **SQL 스크립트 실행** 창에서 **신규**를 선택하십시오.
4. 명령문을 수동으로 작성하거나 **예** 리스트에서 예를 삽입하거나 SQL 생성 기능을 사용하여 기존의 오브젝트에 대해 SQL을 검색하십시오.
5. 명령문 작성을 완료한 후에 **실행** 메뉴에서 **구문 검사**를 선택하여 명령문의 구문을 검사할 수 있습니다.

구문 검사가 완료되면, **파일** 메뉴에서 **저장**을 선택하여 스크립트를 저장할 수 있습니다. 스크립트의 위치와 이름을 묻는 프롬트가 표시됩니다.

스크립트를 작성했으면, 다음의 작업이 가능합니다.

- SQL 스크립트 실행
- 작업 기록부 보기
- SQL 스크립트 실행에 대한 옵션 변경
- 데이터베이스 오브젝트에 대한 SQL 생성

SQL 스크립트 실행

SQL 스크립트를 실행하려면, **실행** 메뉴에서 다음의 옵션 중 하나를 선택하십시오.

- **모두** - SQL 스크립트를 처음부터 끝까지 실행합니다. 오류가 발생하고 **오류 시 중단** 옵션이 작동되는 경우, 프로그램이 중단되고 오류가 발생된 명령문은 계속 선택되어 있습니다.
- **선택된 것부터** - 선택된 첫 번째 명령문이나 현재 커서 위치에서 SQL 스크립트를 시작하고 스크립트의 끝까지 계속합니다.
- **선택됨** - 선택된 명령문을 실행합니다.

결과가 **메세지** 탭의 끝에 추가됩니다. **옵션** 메뉴의 **스마트 명령문 선택** 옵션이 체크되지 않은 경우, 선택된 텍스트가 단일 SQL문으로 실행됩니다.

SQL 스크립트 실행에 대한 옵션 변경

SQL 스크립트 실행을 위한 옵션을 변경하려면, **옵션** 메뉴에서 다음의 옵션 중 하나를 선택하십시오.

오류 시 중단

오류 시 중단을 작동시키거나 작동 중지시킵니다. 이 옵션이 선택되었을 때 오류가 발생한 경우, SQL 스크립트는 실행을 중단하고 오류를 발생시킨 명령문은 계속 선택되어 있습니다.

스마트 명령문 선택

스마트 명령문 선택을 작동시키거나 작동 중지시킵니다. 이 옵션이 선택되면, **실행** 메뉴의 **선택됨** 명령이 선택되었을 때 강조표시된 모든 명령문이 순서대로 실행됩니다. 이 옵션이 선택되지 않으면, **선택됨** 명령은 강조표시된 텍스트를 단일 SQL문으로 실행합니다. 또한, **스마트 명령문 선택**을 선택하면 하나 이상의 명령문이 부분적으로만 강조표시되었더라도 전체 명령문이 실행됩니다.

별도의 창에 결과 표시

SELECT문의 결과가 **출력** 분할 창 대신 별도의 창에 표시됩니다.

작업 기록부에 디버그 메세지 포함

SQL 스크립트 실행 창에서 실행되는 명령문에 대한 디버깅을 작동시키거나 작동 중지시킵니다. 이 옵션을 작동시키고 명령문을 실행하고 **작업 기록부** 창을 화면정리하여 조회 Optimizer와 기타 데이터베이스 디버깅 메세지를 볼 수 있습니다.

저장 프로시듀어에 대한 결과 세트 보기

CALL문을 SQL문으로 입력하여 SQL 실행 스크립트에서 저장된 프로시듀어에 대한 결과 세트를 볼 수 있습니다. 결과는 일반 조회에서와 같이 결과 창에서 볼 수 있습니다. 결과 세트가 여러 개 있는 경우 별도의 결과 탭으로 각각 표시될 것입니다. 추가로, **메세지** 탭의 출력 매개변수를 볼 수 있습니다. CALL 사용에 대한 자세한 정보는 SQL 참조서 책의 CALL을 참조하시오.

작업 기록부 보기

작업 기록부는 작업과 관련된 메세지를 표시합니다. 조회 Optimizer와 기타 데이터베이스 디버깅 메세지를 보려면, **옵션** 메뉴에서 **작업 기록부에 디버그 메세지 포함**을 선택하고 다시 명령문을 실행하십시오. 이 작업을 수행할 때 **작업 기록부** 대화 상자가 열려 있으면, 새로운 메세지를 보기 위해 보기를 화면정리하십시오. 작업 기록부를 보려면, 다음과 같이 하십시오.

보기 메뉴에서 **작업 기록부**를 선택하십시오.

주: 실행 이력 지우기가 사용될 경우 작업 기록부는 지워지지 않으므로 작업 기록부를 사용하여 **출력** 분할 창에 더 이상 없는 메세지를 볼 수 있습니다.

실행되는 SQL 스크립트를 중단하거나 취소하려면, 실행 메뉴에서 다음의 옵션 중 하나를 선택하십시오.

현재 이후 중단

현재 실행 중인 명령문이 종료된 후에 SQL 스크립트 실행을 중단합니다.

취소 요구

시스템이 현재 SQL문을 취소하도록 요구합니다. 그러나, 모든 SQL문을 취소할 수 있는 것은 아니므로 이 옵션이 사용된 후에도 SQL문이 계속 완료될 수 있습니다. **취소 요구**를 누르기 전에 이미 호스트 처리를 완료한 SQL문도 계속해서 완료됩니다. 예를 들어, 조회 처리를 이미 완료했지만 아직 클라이언트에 결과를 리턴하지 않은 SELECT문은 대개 취소할 수 없습니다.

SQL 생성을 사용한 SQL문 재구성

SQL 생성을 통해 기존의 데이터베이스 오브젝트를 작성하는 데 사용되는 SQL을 재구성할 수 있습니다. 이 프로세스는 종종 역공학이라고 합니다. 스키마, 표, 유형, 보기, 프로시듀어, 기능, 별명 및 색인에 대한 SQL을 생성할 수 있습니다. 추가로, 제한사항이나 트리거가 연관된 표에 대해 SQL을 생성하는 경우, 이들에 대한 SQL도 생성됩니다. 한 번에 하나 또는 여러 개의 오브젝트에 대한 SQL을 생성할 수 있습니다. 실행 또는 편집을 위해 SQL 스크립트 실행 창에 생성된 SQL을 송신하는 옵션도 있지만 생성된 SQL을 데이터베이스나 PC 파일에 직접 기록할 수 있습니다.

SQL 생성에 대한 자세한 내용은 다음의 주제를 참조하십시오.

- 데이터베이스 오브젝트에 대한 SQL 생성
- 오브젝트 리스트 편집

데이터베이스 오브젝트에 대한 SQL 생성

기존의 데이터베이스 오브젝트를 작성하는 데 사용되는 SQL을 생성하려면, 다음과 같이 하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. 라이브러리에 대한 SQL을 생성하려면, 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 **SQL 생성**을 선택하십시오.
3. 라이브러리 내에 포함된 오브젝트에 대해 SQL을 생성하려면, SQL을 생성하려는 대상 오브젝트가 포함된 라이브러리를 클릭하십시오.
4. SQL을 생성하려는 대상 오브젝트에서 마우스 오른쪽 버튼을 클릭하고 **SQL 생성**을 선택하십시오.

탭의 값을 변경하여 SQL 생성에 대한 디폴트 옵션을 변경할 수 있습니다. **출력** 탭에서 생성된 SQL에 대한 목적지를 선택할 수 있습니다. 생성된 SQL을 **SQL 스크립트 실행**으로 송신하거나 파일, PC 또는 시스템에 직접 저장할 수 있습니다. **옵션** 탭에서 생성된 SQL이 따라야 하는 **표준**을 선택할 수 있으며 추가로 정보용 메세지를 포함시키고, 드롭 메세지를 포함시키고, 레이블을 생성하고, 읽기 가능성을 위해 포맷하도록 선택할 수 있습니다. **포맷** 탭에서 포맷 옵션을 선택할 수 있습니다. 이러한 옵션들은 표준 옵션을 따라야 합니다.

SQL을 생성할 대상 오브젝트 리스트 편집

SQL을 생성할 대상 오브젝트 리스트를 편집할 수 있습니다. 오브젝트를 추가하려면, 다음과 같이 하십시오.

1. **추가**를 클릭하십시오.
2. **오브젝트 리스트 편집** 대화 상자에서 포함시키려는 오브젝트를 탐색하고 **추가**를 선택하십시오.
3. 기본 SQL 생성 대화 상자로 되돌아가도록 **확인**을 클릭하십시오.

리스트에서 오브젝트를 제거하려면, 다음과 같이 하십시오.

1. **SQL을 생성할 대상 오브젝트**에서 제거하려는 오브젝트를 선택하십시오.
2. **제거**를 클릭하십시오.

---

iSeries Navigator를 사용한 확장 표 기능

37 페이지의 제 3 장『iSeries Navigator 데이터베이스 시작』에서는 수행할 수 있는 기본 표 기능의 일부를 설명합니다. 그러나, iSeries Navigator를 사용하여 다음의 작업도 가능합니다.

- 별명 작성
- 색인 추가
- 키 제한사항 추가
- 검사 제한사항 추가
- 참조 제한사항 추가
- 트리거 추가
- 트리거 작동 및 작동 불가능
- 제한사항 또는 트리거 제거

iSeries Navigator를 사용한 별명 작성

별명은 표나 보기의 대체명입니다. 기존 표나 보기를 참조할 수 있는 경우에 별명을 사용하여 표 또는 보기를 참조할 수 있습니다. 표나 보기 또는 표나 보기의 멤버에 대해 별명을 작성할 수 있습니다.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. 새로운 별명을 작성하려는 라이브러리에서 마우스 오른쪽 버튼을 클릭하십시오.
3. 팝업 메뉴에서 **신규**를 선택한 다음 **별명**을 선택하십시오.
4. **새 별명** 대화 상자의 **별명** 필드에 작성할 별명의 이름을 지정하십시오. 이 이름은 서버에 이미 있는 색인, 표, 보기, 파일 또는 별명과 동일할 수 없습니다.

5. **설명** 필드에 새로운 별명의 설명을 지정하십시오. 이 설명의 길이는 최대 50자까지 가능합니다. 이 필드는 선택적입니다.
6. **표/보기** 필드에 별명이 가리키게 하려는 표나 보기를 지정하십시오.
7. **라이브러리** 필드에 별명이 가리키게 하려는 표나 보기가 들어 있는 라이브러리를 지정하십시오.
8. **확장**을 클릭하십시오.
9. **확장** 대화 상자에서, 표 또는 보기에 대한 별명을 작성하려면, **표 또는 보기에 대한 별명 작성**을 클릭하십시오. 이것은 디폴트 옵션입니다.
10. 표나 보기의 멤버에 대해 별명을 작성하려면, **표 또는 보기의 멤버에 대해 별명 작성**을 클릭하십시오. 콤보 박스에서 별명이 가리키게 하려는 멤버를 입력하거나 선택하십시오.
11. **확인**을 클릭하여 **새 별명** 대화 상자로 되돌아가십시오.
12. **확인**을 클릭하여 별명을 작성하십시오.

주: 표나 보기에서 마우스 오른쪽 버튼을 클릭하고 **별명 작성**을 선택하여 별명을 작성할 수도 있습니다.

별명에 대한 자세한 내용은 62 페이지의 『ALIAS 이름 작성 및 사용』을 참조하십시오.

iSeries Navigator를 사용한 색인 추가

색인을 사용하여 자료를 정렬하고 선택할 수 있습니다. 또한, 색인을 사용하면 자료를 빨리 검색할 수 있어서 조회 성능이 더 좋아집니다.

색인을 임의의 수만큼 작성할 수 있습니다. 그러나 색인이 시스템으로 관리되기 때문에 많은 색인은 시스템 성능 측면에 있어서 부적절한 영향을 미칠 수 있습니다. 색인 및 조회 성능에 대한 자세한 내용은 *데이터베이스 성능 및 조회 최적화* 정보에서 SQL 색인의 효과적인 사용을 참조하십시오.

한 색인 유형, 코드화된 벡터 색인은 사용하면 병렬로 더 쉽게 프로세스할 수 있는 빠른 스캔이 가능하게 됩니다. 코드화 벡터 색인을 사용하여 조회를 가속화  하는 것에 대한 내용은 iSeries용 DB2 웹 페이지를 참조하십시오.

새로운 표나 기존의 표에 대해 색인을 작성할 수 있습니다. iSeries Navigator를 사용하여 기수나 코드화 벡터 색인을 작성할 수 있습니다.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. 색인을 추가하려는 표가 들어 있는 라이브러리를 클릭하십시오.
3. 세부사항 분할 창에서, 색인을 추가하려는 표를 마우스 오른쪽 버튼으로 클릭하고 **등록정보**를 선택하십시오.

4. **표 등록정보** 또는 **새 표** 대화 상자에서 색인 탭을 선택하십시오.
5. **색인** 탭에서 **신규**를 클릭하십시오.
6. **색인** 필드에 새로운 색인의 이름을 지정하십시오.
7. **라이브러리** 필드에서 색인이 상주할 라이브러리를 선택하십시오.
8. 색인을 구성할 표의 열들을 선택하십시오. 열을 추가하려면, 해당 열을 클릭하십시오. 왼쪽에 숫자가 나타납니다. 이 숫자는 색인에 있는 열의 키 위치를 판별합니다. 색인에서 열을 제거하려면, 이를 다시 클릭하십시오.
9. 키 필드의 순서를 오름차순에서 내림차순으로(또는 내림차순에서 오름차순으로) 변경하려면, 두 번째 열을 클릭하십시오.
10. **색인 유형**을 선택하십시오.
11. 코드화 벡터 색인을 작성하는 경우, 고유한 값의 수를 선택하십시오.
12. **확인**을 클릭하여 색인을 작성하십시오.

주: 새 표가 있는 위치 대화 상자에서 새로운 표에 색인을 추가할 수도 있습니다.

현재 표 편집 세션 중에 정의된 경우에만 제한사항을 수정할 수 있습니다. 제한사항을 추가한 다음 **새 표** 대화 상자나 **표 등록정보** 대화 상자에서 **확인**을 클릭한 경우, 제한사항에 대한 읽기 전용 액세스 권한이 있습니다. 제한사항 등록정보를 변경하려면, 제한사항을 드롭한 다음 적합한 변경사항으로 다시 작성해야 합니다.

색인 작성에 대한 자세한 내용은 65 페이지의 『색인 추가』를 참조하십시오.

iSeries Navigator를 사용한 키 제한사항 추가

제한조건은 데이터베이스 관리자에 의해 강제되는 규칙입니다. iSeries용 DB2 UDB는 두 가지 유형의 키 제한사항을 지원합니다.

- 고유 키 제한사항은 키 값이 고유한 경우에만 유효하다는 규칙입니다. 고유 제한조건은 INSERT 및 UPDATE문의 실행시 강제됩니다.
- 1차 키 제한사항은 고유한 제한사항의 한 형태입니다. 차이점은 1차 키에 널 가능 열이 포함될 수 없다는 점입니다.

키 제한사항을 작성하려면, 다음을 수행하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. 키 제한사항을 추가하려는 표가 포함된 라이브러리를 두 번 클릭하십시오.
3. 키를 추가하려는 표에서 마우스 오른쪽 버튼을 클릭하고 **등록정보**를 선택하십시오.
4. **표 등록정보** 대화 상자에서 **키 제한사항** 탭을 선택하십시오.
5. **키 제한사항** 탭에서 **신규**를 클릭하십시오.
6. **새 키 제한사항** 대화 상자에서 이름 텍스트 상자에 이름을 지정하십시오. 이름이 지정되지 않은 경우, 시스템이 자동으로 이름을 생성합니다.

7. 키를 추가하려는 열을 선택하십시오.
8. 1차 키를 작성하려면 **1차**를 선택하고 고유 키를 작성하려면 **고유**를 선택하십시오.
9. **확인**을 클릭하여 **표 등록정보** 대화 상자로 되돌아가십시오.
10. **확인**을 클릭하여 키를 작성하십시오.

주: 새 표가 있는 위치 대화 상자에서 새로운 표에 키 제한사항을 추가할 수도 있습니다.

현재 표 편집 세션 중에 정의된 경우에만 제한사항을 수정할 수 있습니다. 제한사항을 추가한 다음 **새 표** 대화 상자나 **표 등록정보** 대화 상자에서 **확인**을 클릭한 경우, 제한사항에 대한 읽기 전용 액세스 권한이 있습니다. 제한사항 등록정보를 변경하려면, 제한사항을 드롭한 다음 적합한 변경사항으로 다시 작성해야 합니다.

키 제한사항 추가에 대한 자세한 내용은 151 페이지의 제 10 장『자료 무결성』을 참조하십시오.

iSeries Navigator를 사용한 검사 제한사항 추가

검사 제한사항은 열이나 열 그룹에서 허용되는 값을 제한하여 삽입과 갱신 시에 자료의 유효성을 보장합니다.

검사 제한사항을 작성하려면, 다음을 수행하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. 검사 제한사항을 추가하려는 표가 포함된 라이브러리를 클릭하십시오.
3. 검사 제한사항을 추가하려는 표에서 마우스 오른쪽 버튼을 클릭하고 **등록정보**를 선택하십시오.
4. **표 등록정보** 대화 상자에서 **검사 제한사항** 탭을 클릭하십시오.
5. **검사 제한사항** 탭에서 **신규**를 클릭하십시오.
6. **검사 제한사항 탐색 조건** 대화 상자에서 이름 텍스트 상자에 이름을 지정하십시오. 이름이 지정되지 않은 경우, 시스템이 자동으로 이름을 생성합니다.
7. **열** 리스트에서 제한사항을 추가하려는 열을 두 번 클릭하십시오. 절 상자에 해당 열이 나타납니다.
8. 리스트에서 연산자를 삽입하려면 해당 연산자를 두 번 클릭하십시오. 연산자가 절 상자에 나타납니다.
9. 리스트에서 기능을 삽입하려면, 해당 기능을 두 번 클릭하십시오. 드롭 다운 리스트에서 기능 유형을 선택하여 리스트를 수정할 수 있습니다. 기능을 두 번 클릭하면 절 상자에 나타납니다.
10. 표현식이 올바를 때까지 수정하십시오.
11. **확인**을 클릭하여 **표 등록정보** 대화 상자로 되돌아가십시오.

12. **확인**을 클릭하여 검사 제한사항을 작성하십시오.

주: 새 표가 있는 위치 대화 상자에서 새로운 표에 검사 제한사항을 추가할 수도 있습니다.

현재 표 편집 세션 중에 정의된 경우에만 제한사항을 수정할 수 있습니다. 제한사항을 추가한 다음 **새 표** 대화 상자나 **표 등록정보** 대화 상자에서 **확인**을 클릭한 경우, 제한사항에 대한 읽기 전용 액세스 권한이 있습니다. 제한사항 등록정보를 변경하려면, 제한사항을 드롭한 다음 적합한 변경사항으로 다시 작성해야 합니다.

검사 제한사항에 대한 자세한 내용은 151 페이지의 『체크 제한조건 추가 및 사용』을 참조하십시오.

iSeries Navigator를 사용한 참조 제한사항 추가

참조 무결성은 한 표에서 다른 표로의 모든 참조가 유효한 데이터베이스 표 집합의 조건입니다. 참조 제한사항을 추가하여 데이터베이스에서 참조 무결성을 보장할 수 있습니다.

참조 제한사항을 작성하려면, 다음을 수행하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. 종속 표가 들어 있는 라이브러리를 두 번 클릭하십시오.
3. 종속 표에서 마우스 오른쪽 버튼을 클릭하고 **등록정보**를 선택하십시오.
4. **표 등록정보** 대화 상자에서 **참조 제한사항** 탭을 선택하십시오.
5. **참조 제한사항** 탭에서 **신규**를 클릭하십시오.
6. **새 참조 제한사항** 대화 상자에서 제한사항의 이름을 지정하십시오. 이름이 지정되지 않은 경우, 시스템이 자동으로 이름을 생성합니다.
7. 상위 표에서 상위 키 값에 종속시키려는 열을 선택하십시오.
8. 상위 표가 들어 있는 라이브러리를 선택하십시오.
9. 상위 키가 들어 있는 표를 선택하십시오.
10. 참조할 상위 키를 선택하십시오.
11. 삭제 조치를 선택하십시오.
12. 갱신 조치를 선택하십시오(삽입 조치가 디폴트).
13. **확인**을 클릭하여 **표 등록정보** 대화 상자로 되돌아가십시오.
14. **확인**을 클릭하여 참조 제한사항을 작성하십시오.

주: 새 표가 있는 위치 대화 상자에서 새로운 표에 참조 제한사항을 추가할 수도 있습니다.

현재 표 편집 세션 중에 정의된 경우에만 제한사항을 수정할 수 있습니다. 제한사항을 추가한 다음 **새 표** 대화 상자나 **표 등록정보** 대화 상자에서 **확인**을 클릭한 경우, 제한사항에 대한 읽기 전용 액세스 권한이 있습니다. 제한사항 등록정보를 변경하려면, 제한사항을 드롭한 다음 적합한 변경사항으로 다시 작성해야 합니다.

참조 제한사항에 대한 자세한 내용은 153 페이지의 『참조 제한조건 추가 또는 제거』를 참조하십시오.

iSeries Navigator를 사용한 트리거 추가

트리거는 지정된 변경 조작이 지정된 실제 데이터베이스 파일에 대해 수행될 때 자동으로 실행되는 조치 세트입니다. 이 설명에서 표는 실제 파일입니다. 변경 조작은 어플리케이션 프로그램 내의 고급 레벨 언어문을 삽입, 갱신 또는 삭제하거나 SQL INSERT, UPDATE 또는 DELETE문일 수 있습니다. 트리거는 업무 규칙 강제, 입력 자료 확인 및 감사 추적 보유 등의 업무에 유용합니다.

iSeries Navigator를 사용하면, 시스템 트리거와 SQL 트리거를 정의할 수 있습니다. 추가로, 트리거를 작동시키거나 작동 불가능하게 할 수 있습니다.

트리거를 추가하려면, 다음을 수행하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. 트리거를 추가하려는 표가 포함된 라이브러리를 클릭하십시오.
3. 트리거를 추가하려는 표에서 마우스 오른쪽 버튼을 클릭하고 **등록정보**를 선택하십시오. **표 등록정보** 대화 상자에서 **트리거** 탭을 클릭하십시오.
4. 시스템 트리거를 추가하려면 **시스템 트리거 추가**를 선택하십시오.
5. SQL 트리거를 추가하려면 **SQL 트리거 추가**를 선택하십시오.

시스템 트리거에 대한 자세한 정보는 *데이터베이스 프로그래밍* 책에서 데이터베이스에서 자동 이벤트 트리거를 참조하십시오.

SQL 트리거에 대한 자세한 내용은 166 페이지의 『SQL 트리거』를 참조하십시오.

트리거 작동 및 작동 불가능

트리거를 실행하려면 작동시켜야 합니다. 그러나, 트리거를 작동 불가능하게 하면 트리거를 실행하지 않고 표에 대해 작업할 수 있습니다.

트리거를 작동시키거나 작동 불가능하게 하려면, 다음을 수행하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. 트리거를 작동시키거나 작동 불가능하게 하려는 표가 포함된 라이브러리를 클릭하십시오.

3. 표에서 마우스 오른쪽 버튼을 클릭하고 **등록정보**를 선택하십시오.
4. **표 등록정보** 대화 상자에서 **트리거** 탭을 클릭하십시오.
5. 작동시키거나 작동 불가능하게 하려는 트리거를 선택하고 트리거를 작동시키려면 **작동**을 클릭하고 트리거를 작동 불가능하게 하려면 **작동 불가능**을 클릭하십시오.

CHGPFTRG CL 명령을 사용한 트리거 작동 및 작동 불가능에 대한 정보는 *데이터베이스 프로그래밍* 책에서 트리거 작동 및 작동 불가능을 참조하십시오.

제한사항 및 트리거 제거

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. 제한사항이나 트리거를 제거하려는 표가 포함된 라이브러리를 클릭하십시오.
3. 제한사항이나 트리거를 제거하려는 표에서 마우스 오른쪽 버튼을 클릭하고 **등록정보**를 선택하십시오.
4. **표 등록정보** 대화 상자에서 제거하려는 제한사항이나 트리거의 유형에 대한 탭을 클릭하십시오.
5. 제거하려는 제한사항을 선택하고 **삭제**를 클릭하십시오.

iSeries Navigator를 사용한 SQL 오브젝트 정의

iSeries Navigator는 시스템에서 일부 SQL 오브젝트를 정의하는 간단한 방법을 제공합니다. 예를 들어, 다음을 정의할 수 있습니다.

- 프로시듀어: 프로시듀어(종종 저장 프로시듀어라고 함)는 호스트 언어 명령문과 SQL 문이 둘 다 포함될 수 있는 조작을 수행하기 위해 호출할 수 있는 프로그램입니다. SQL의 프로시듀어는 호스트 언어의 프로시듀어와 비슷한 장점을 제공합니다. 즉 공통된 코드 부분은 한 번만 기록되고 유지보수되어 여러 프로그램에서 호출될 수 있습니다. 프로시듀어에 대한 자세한 내용은 177 페이지의 제 11 장 『저장 프로시듀어』를 참조하십시오.
- 사용자 정의 기능: 사용자 정의 기능(UDF)은 소스, 외부 및 SQL의 3가지 유형으로 구성되어 있습니다. 소스 기능 UDF는 조작을 수행하기 위해 다른 기능을 호출합니다. SQL과 외부 기능 UDF는 사용자가 별도의 코드를 작성하여 실행해야 합니다. 스칼라, 열 및 표 함수를 작성할 수 있습니다. 기능에 대한 자세한 내용은 221 페이지의 『사용자 정의 기능(UDF)』을 참조하십시오.
- 사용자 정의 유형: 사용자 정의 고유 유형은 DB2 기능을 내장 자료 유형의 사용가능한 범위를 넘어 확장할 수 있게 하는 메카니즘입니다. 사용자 정의된 고유한 유형을 사용하면 업무를 모델링하고 자료의 의미 규칙을 파악하기 위해 더 이상 시스템이 제공하는 내장 자료 유형만 사용하도록 제한하지 않기 때문에 사용자에게 상당한 권한을 주는 DB2에 대한 새로운 자료 유형을 정의할 수 있습니다. 고유한 자료 유

형을 사용하면 현재의 데이터베이스 유형에 대해 1대 1의 기초로 맵할 수 있습니다. 유형에 대한 자세한 내용은 240 페이지의 『사용자 정의된 고유한 유형(UDT)』을 참조하십시오.

iSeries Navigator를 사용한 이러한 오브젝트 정의에 대한 자세한 내용은 다음의 주제를 참조하십시오.

- 『iSeries Navigator를 사용한 저장 프로시듀어 정의』
- 『iSeries Navigator를 사용한 사용자 정의 기능 정의』
- 321 페이지의 『iSeries Navigator를 사용한 사용자 정의 유형 정의』

iSeries Navigator를 사용한 저장 프로시듀어 정의

SQL 프로그램에서 프로시듀어로 프로그램을 호출하려면, 먼저 외부 프로시듀어로 프로그램을 정의해야 합니다. 프로시듀어가 정의되는 프로그램은 프로시듀어 정의 시에 존재할 필요가 없습니다.

프로시듀어로 프로그램을 정의하려면, 다음을 수행하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. 기능을 정의하려는 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 **신규**를 선택하십시오.
3. **프로시듀어**를 선택하십시오.
4. 외부 저장 프로시듀어를 작성하려면 **외부**를 선택하십시오.
5. SQL 저장 프로시듀어를 작성하려면 **SQL**을 선택하십시오.

외부 저장 프로시듀어 작성 및 정의에 대한 자세한 내용은 178 페이지의 『외부 프로시듀어 정의』를 참조하십시오.

SQL 저장 프로시듀어 작성 및 정의에 대한 자세한 내용은 179 페이지의 『SQL 프로시듀어 정의』를 참조하십시오.

iSeries Navigator를 사용한 사용자 정의 기능 정의

사용자 정의 기능으로 프로그램을 정의하려면, 다음을 수행하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. 기능을 정의하려는 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 **신규**를 선택하십시오.
3. **기능**을 선택하십시오.
4. 외부 사용자 정의 기능을 작성하려면 **외부**를 선택하십시오.
5. SQL 사용자 정의 기능을 작성하려면 **SQL**을 선택하십시오.

6. 다른 기능에 기초하여 기능을 작성하려면 **소스**를 선택하십시오.

외부 기능 작성 및 정의에 대한 자세한 내용은 221 페이지의 『사용자 정의 기능(UDF)』을 참조하십시오.

iSeries Navigator를 사용한 사용자 정의 유형 정의

기존의 자료 유형에 기초하여 새로운 사용자 정의 자료 유형을 작성하면 사용자의 자료를 보다 잘 제어할 수 있게 됩니다.

사용자 정의 유형을 작성하려면, 다음을 수행하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. 유형을 정의하려는 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 **신규**를 선택하십시오.
3. **유형**을 선택하십시오.
4. **새 유형** 대화 상자에서 유형 필드에 새로운 유형에 부여하려는 이름을 지정하십시오.
5. **유형** 필드의 **소스 자료 유형** 섹션에 새로운 유형의 기반이 되는 자료 유형을 지정하십시오.
6. **확인**을 클릭하십시오.

사용자 정의 유형 작성 및 정의에 대한 자세한 내용은 240 페이지의 『사용자 정의된 고유한 유형(UDT)』을 참조하십시오.

SQL 패키지 작성

SQL 패키지는 준비된 SQL문과 관련된 정보를 저장하는 데 사용되는 영구 오브젝트입니다. SQL 패키지는 데이터 소스에서 ″확장 동적″ 상자를 선택하면 ODBC 지원에 의해 사용됩니다.

SQL 패키지를 작성하려면 다음을 수행하십시오.

1. **iSeries Navigator** 창에서, 서버 → **데이터베이스**를 펼치십시오.
2. 마우스 오른쪽 버튼으로 사용하려는 데이터베이스를 클릭하고 **새 SQL 패키지**를 선택하십시오.
3. **SQL 패키지 작성** 대화 상자에서, 필요에 따라 매개변수를 지정하십시오.
4. **확인**을 클릭하십시오.

매개변수의 전체 리스트는 417 페이지의 『CRTSQLPKG(구조화 조회 언어 패키지 작성) 명령』을 참조하십시오.

제 17 장 대화식 SQL 사용

이 장에서는 SQL문을 수행하기 위해 대화식 SQL을 사용하는 방법 및 프롬트 기능을 사용하는 방법에 대해 설명합니다. 대화식 SQL 사용에 관한 개요와 조언도 제공됩니다. SQL 사용법에 대해 알려면 제 2 장『SQL 시작』을 참조하십시오. 리모트 연결로 대화식 SQL을 사용하기 위한 특별한 고려사항은 334 페이지의『대화식 SQL로 리모트 데이터베이스 액세스』에서 다루어집니다.

자세한 내용은『대화식 SQL의 기본 기능』을 참조하십시오.

주:

1. 용어 *콜렉션*은 *스키마*와 동의어로 사용됩니다.
2. 코드 예에 대한 정보는 x 페이지의『코드 면책사항 정보』를 참조하십시오.

대화식 SQL의 기본 기능

대화식 SQL로 프로그래머와 데이터베이스 관리자는 테스트, 문제점 분석 및 데이터베이스 유지보수를 위해 자료를 신속하고 용이하게 정의, 분석, 삭제 또는 참조할 수 있습니다. 대화식 SQL을 사용하는 프로그래머는 행을 표에 삽입할 수 있으며 어플리케이션 프로그램에서 실행하기 전에 SQL문을 테스트할 수 있습니다. 데이터베이스 관리자는 대화식 SQL을 사용하여 권한을 부여하거나 취소할 수 있으며, 스키마, 표 또는 보기를 작성하거나 제거할 수 있고 시스템 카탈로그 표에서 정보를 선택할 수 있습니다.

대화식 SQL문이 처리되고 나면 완료 메세지나 오류 메세지가 표시됩니다. 또한 명령문이 장시간 수행되면 대개는 상태 메세지가 표시됩니다.

커서를 메세지에 위치시키고 F1(도움말) 키를 누르면 메세지상에서 도움말을 볼 수 있습니다.

대화식 SQL이 제공하는 기본 기능은 다음과 같습니다.

- **명령문 입력** 기능으로 수행할 수 있는 작업은 다음과 같습니다.
  - 대화식 SQL문의 입력 및 수행
  - 명령문의 검색 및 편집
  - SQL문에 대한 프롬트
  - 이전 명령문과 메세지를 통한 페이징
  - 세션 서비스 호출
  - 리스트 선택 기능 호출

  - 대화식 SQL 나감
- **프롬트** 기능으로 완전한 SQL문 또는 부분적인 SQL문을 입력한 후에는 F4=프롬트를 누를 수 있으며 그러면 명령문 구문에 대한 프롬트가 제공됩니다. 또한 F4 키를 눌러 모든 SQL문의 메뉴를 구할 수도 있습니다. 이 메뉴에서는 명령문을 선택할 수 있으며 명령문에 대한 구문이 프롬트됩니다.
- **리스트 선택 기능**을 사용하면 권한이 있는 관계형 데이터베이스, 스키마, 표, 보기, 열, 제한사항 또는 SQL 패키지의 리스트에서 선택할 수 있습니다.

  리스트로부터의 선택은 커서 위치에서 SQL문에 삽입될 수 있습니다.
- **세션 서비스** 기능으로 수행할 수 있는 작업은 다음과 같습니다.
  - 세션 속성 변경
  - 현재 세션 인쇄
  - 현재 세션에서 모든 항목 제거
  - 소스 파일에 세션 저장

자세한 내용은 다음의 절을 참조하십시오.

- 『대화식 SQL 시작』
- 326 페이지의 『명령문 항목 기능 사용』
- 326 페이지의 『프롬트』
- 329 페이지의 『리스트 선택 기능 사용』
- 332 페이지의 『세션 서비스 설명』
- 333 페이지의 『대화식 SQL 나가기』
- 334 페이지의 『기존 SQL 세션 사용』
- 334 페이지의 『SQL 세션 회복』
- 334 페이지의 『대화식 SQL로 리모트 데이터베이스 액세스』

대화식 SQL 시작

OS/400 명령행에 STRSQL을 입력하여 대화식 SQL 사용을 시작할 수 있습니다. 명령 및 매개변수에 대한 완전한 설명을 보려면 부록 B 『iSeries용 DB2 UDB CL 명령 설명』을 참조하십시오

SQL 입력 화면이 나타납니다. 이는 기본 대화식 SQL 화면입니다. 이 화면에서, SQL 문을 입력하고 다음을 사용할 수 있습니다.

- F4=프롬트
- F13=세션 서비스
- F16=콜렉션 선택
- F17=표 선택
- F18=열 선택

```
                                SQL문 입력

SQL문을 입력한 후 Enter 키를 누르십시오.
    현재 관계형 데이터베이스 rdjacque에 연결되어 있습니다.
===>_____________________________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________

                                                                  맨 아래
F3=나감    F4=프롬트    F6=행 삽입       F9=검색       F10=행 복사
F12=취소               F13=서비스       F24=추가 키
```

나머지 기능 키를 보려면 F24(추가) 키를 누르십시오.

```
     _____________________________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________

                                                                  맨 아래

F14=행 삭제       F15=행 분할       F16=콜렉션(라이브러리) 선택
F17=표 선택                         F18=열 선택          F24=추가 키
     (파일)                              (필드)
```

주: 시스템 명명 규칙을 사용할 경우, 위에 나오는 이름 대신에 소괄호 내의 이름이 나옵니다.

대화식 세션은 다음과 같이 구성되어 있습니다.

- STRSQL 명령에 지정한 매개변수의 값
- 각 SQL문에 후속되는 해당 메세지와 함께 세션에 입력한 SQL문
- 세션 서비스 기능을 사용하여 변경한 모든 매개변수의 값
- 사용자가 수행한 리스트 선택

대화식 SQL은 사용자 ID와 현재 워크스테이션 ID로 구성되는 고유 세션 ID를 제공합니다. 이 세션 ID 개념으로 동일한 사용자 ID를 가진 복수 사용자가 동시에 두 개 이상의 워크스테이션에서 대화식 SQL을 사용할 수 있습니다. 또한 둘 이상의 대화식 SQL 세션이 같은 사용자 ID로 동시에 같은 워크스테이션에서 수행될 수 있습니다.

SQL 세션이 존재하며 재입력되고 있을 경우, STRSQL 명령에 지정된 모든 매개변수는 무시됩니다. 기존 SQL 세션의 매개변수가 사용됩니다.

명령문 항목 기능 사용

명령문 입력 기능은 사용자가 대화식 SQL을 선택할 때 처음 들어갈 수 있도록 하는 기능입니다. 사용자는 각 대화식 SQL문을 처리한 후 명령문 입력 기능으로 리턴합니다.

명령문 입력 기능에서 사용자는 전체 SQL문을 입력하거나 프롬트하고 Enter 키를 눌러 SQL문을 제출합니다.

명령문 입력

명령 행에 입력한 명령문은 단일 행에 있거나 복수 행에 걸쳐 있을 수 있습니다. 대화식 SQL에서 SQL문에 대한 주석을 입력할 수는 없습니다. 명령문이 처리되었을 때 그 명령문과 결과 메세지는 화면에서 위로 이동합니다. 그런 후 다른 명령문을 입력할 수 있습니다.

명령문이 SQL에 의해 인식되었으나 구문 오류가 들어 있으면 명령문과 결과 텍스트 메세지(구문 오류)는 화면 위쪽으로 이동합니다. 입력영역에서 구문 오류에 커서가 위치한 재명령문의 사본이 나타납니다. 오류에 대해 알려면 커서를 메세지에 위치시키고 F1(도움말) 키를 누르십시오.

이전 명령문, 명령 및 메세지를 페이지 이동할 수 있습니다. 입력영역에 있는 명령문의 사본을 위치시키려면 이전 명령문에 커서를 놓고 F9(검색) 키를 누르십시오. 사용자가 SQL문 입력을 위한 공간이 필요할 경우, 화면에서 페이지를 아래로 이동하면 됩니다.

프롬트

프롬트 기능은 사용하려는 명령문의 구문에 대해 필요한 정보를 제공하는데 도움을 줍니다. 프롬트 기능은 *RUN, *VLD, *SYN의 세 가지 명령문 처리 모드로 사용될 수 있습니다.

프롬터를 사용할 때에는 다음의 두 가지 옵션이 있습니다.

- F4(프롬트) 키를 누르기 전에 명령문의 명령어를 입력합니다.

  명령문이 분석되고 완료된 절이 프롬트 화면에 채워집니다.

  SELECT를 입력하고 F4=프롬트를 누르면 다음 화면이 나타납니다.

```
                                SELECT문 지정

SELECT문 정보를 입력한 후 리스트를 보려면 F4를 누르십시오.

  FROM 표 . . . . . .  _____________________________________________
  SELECT 열 . . . . . .  _____________________________________________
  WHERE 조건. . . . . .  _____________________________________________
  GROUP BY 열 . . . . . .  _____________________________________________
  HAVING 조건  . . . . . .  _____________________________________________
  ORDER BY 열 . . . . . .  _____________________________________________
  FOR UPDATE OF 열  . . . . .  _____________________________________________

                                                                  맨 아래
선택사항을 입력한 후 Enter 키를 누르십시오.

  결과 표의 DISTINCT 행  . . . . . .  N    Y=예, N=아니오
  다른 SELECT와 UNION  . . . . . .  N    Y=예, N=아니오
  추가 옵션 지정 . . . . . .  N    Y=예, N=아니오



F3=나감          F4=프롬트   F5=화면정리    F6=행 삽입     F9=부속 조회 지정
F10=행 복사      F12=취소    F14=행 삭제    F15=행 분할    F24=추가 키
```

- SQL 입력 화면에 어떤 내용을 입력하기 전에 F4(프롬트) 키를 누릅니다. 명령문의 리스트가 나타납니다. 명령문의 리스트는 다양하며 현재의 대화식 SQL문의 처리 모드에 따라 달라집니다. *NONE 이외의 언어에 대한 구문 체크 모드의 경우, 리스트는 모든 SQL문을 포함합니다. 실행 및 유효성 검사 모드의 경우, 대화식 SQL에서 수행될 수 있는 명령문만이 나타납니다. 사용자는 사용하려는 명령문의 번호를 선택할 수 있습니다. 시스템은 사용자가 선택한 명령문에 대해 프롬트합니다.

  아무 것도 입력하지 않고 F4=프롬트를 누르면 다음 화면이 나타납니다.

```
SQL문 선택

다음 중 하나를 선택하십시오.

     1. ALTER TABLE
     2. CALL
     3. COMMENT ON
     4. COMMIT
     5. CONNECT
     6. CREATE ALIAS
     7. CREATE COLLECTION
     8. CREATE INDEX
     9. CREATE PROCEDURE
    10. CREATE TABLE
    11. CREATE VIEW
    12. DELETE
    13. DISCONNECT
    14. DROP ALIAS
                                                          계속...
선택
     —
F3=나감   F12=취소
```

프롬트 화면에서 F21(명령문 표시) 키를 누르면 프롬트는 이 지점에서 채워진 대로 형식화된 SQL문을 표시합니다.

프롬트 내에서 Enter 키를 누르면 프롬트 화면을 통해 작성된 명령문은 세션으로 삽입됩니다. 명령문 처리 모드가 *RUN일 경우에는 명령문이 수행됩니다. 오류가 발생하면 프롬트는 제어 상태에 있게 됩니다.

구문 검사

명령문이 프롬터에 입력되면 SQL문의 구문이 체크됩니다. 프롬터는 구문적으로 틀린 명령문은 받아들이지 않습니다. 사용자는 구문을 정정하거나 명령문의 틀린 부분을 제거해야 합니다. 그렇지 않으면 프롬트가 허용되지 않습니다.

명령문 처리 모드

명령문 처리 모드는 세션 속성 변경 화면에서 선택될 수 있습니다. *RUN(실행) 또는 *VLD(유효성 검사) 모드에서는 대화식 SQL로 실행되도록 허용된 명령문만이 프롬트될 수 있습니다. *SYN(구문 체크) 모드에서는 모든 SQL문이 허용됩니다. 그 명령문은 실제로 *SYN 또는 *VLD 모드로 실행되지 않습니다. 구문과 오브젝트의 존재 여부만이 체크됩니다.

부속 조회

부속 조회는 WHERE 또는 HAVING절을 가진 모든 화면에서 선택될 수 있습니다. 부속 조회 화면을 보려면 커서가 WHERE 또는 HAVING 입력 행에 있을 때 F9(부속 조회 지정) 키를 누르십시오. 그러면 부속 선택 정보를 입력할 수 있는 화면이 나타납니다. F9 키를 누를 때 부속 조회의 괄호안에 커서가 있으면 부속 조회 정보는 다음 화면에 채워집니다. 커서가 부속 조회 괄호 밖에 있으면 다음 화면은 공백입니다. 부속 조회에 대한 자세한 내용은 117 페이지의 『SELECT 명령문의 부속 조회』를 참조하십시오.

CREATE TABLE 프롬트

CREATE TABLE에 대한 프롬트시 개별적인 열 정의 입력에 대한 지원이 가능합니다. 커서를 화면의 열 정의 섹션에 두고 F4(프롬트) 키를 누르십시오. 하나의 열 정의에 대한 모든 정보를 입력할 수 있는 공간을 제공하는 화면이 표시됩니다.

18자보다 긴 열 이름을 입력하려면 F20(전체명 표시) 키를 누르십시오. 30자의 이름을 입력할 수 있는 충분한 공간이 있는 창이 표시됩니다.

편집 키, F6(행 삽입) 키, F10(행 복사) 키 및 F14(행 삭제) 키가 열 정의 리스트에서 항목을 추가하고 삭제하는데 사용될 수 있습니다.

DBCS 자료 입력

복수 행에 걸쳐 DBCS 자료를 처리하는 규칙은 SQL 입력 화면이나 SQL 프롬터에서와 동일합니다. 각 행에는 같은 수의 SI 문자와 SO 문자가 들어 있어야 합니다. 입력을 위해 하나 이상의 행을 필요로 하는 DBCS 자료 스트링을 처리할 때 여분의 SI 문자 및 SO 문자는 제거됩니다. 행의 마지막 열에 SI가 들어 있고 다음 행의 첫 번째 열에 SO가 들어 있을 경우, SI 및 SO 문자는 두 행이 합쳐질 때 프롬터에 의해 제거

됩니다. 행의 마지막 두 열의 1바이트 공백 앞에 SI가 오고 다음 행의 첫 번째 열에 SO가 올 경우, SI, 공백, SO 순서는 두 행이 합쳐질 때 제거됩니다. 제거로 인해 DBCS 정보는 하나의 연속 문자 스트링으로 읽혀지게 됩니다.

한 예로, 다음의 WHERE 조건이 입력된다고 가정합니다. 시프트 문자가 두 행 각각의 스트링 시작과 끝에 나타납니다.

```
                          SELECT문 지정

SELECT문 정보를 입력한 후 리스트를 보려면 F4를 누르십시오.

  FROM 표 . . . . . .  TABLE1_____________________________________
  SELECT 열 . . . . . .  *_____________________________________
  WHERE 조건  . . . . . .  COL1 = '<AABBCCDDEEFFGGHHIIJJKKLLMMNNOOPPQQ>
                           <RRSS>'_____________________________________
  GROUP BY 열 . . . . . .  _____________________________________
  HAVING 조건  . . . . . .  _____________________________________
  ORDER BY 열 . . . . . .  _____________________________________
  FOR UPDATE OF 열  . . . . .  _____________________________________
```

Enter 키를 누르면 여분의 시프트 문자(SO/SI)를 제거하면서 문자 스트링이 함께 입력됩니다. 명령문은 SQL 입력 화면에서 다음과 같이 나타납니다.

```
SELECT * FROM TABLE1 WHERE COL1 = '<AABBCCDDEEFFGGHHIIJJKKLLMMNNOOPPQQRRSS>'
```

리스트 선택 기능 사용

특정 프롬트 화면에서 F4 키를 누르거나 SQL 입력 화면에서 F16, F17 또는 F18 키를 눌러 리스트 선택 기능을 사용할 수 있습니다. 기능 키를 누르면 선택해야 할 권한이 있는 관계형 데이터베이스, 스키마, 표, 보기, 별명, 열, 제한사항, 프로시듀어, 매개변수 또는 패키지의 리스트가 제공됩니다. 표의 리스트를 요구했으나 이전에 스키마를 선택하지 않았으면, 스키마를 먼저 선택하도록 요구됩니다.

명령문에서 나타나는 순서를 숫자로 지정하여 리스트상에 하나 이상의 항목을 선택할 수 있습니다. 리스트 기능에서 나갈 때 선택된 것은 들어온 화면의 커서 위치에 삽입됩니다.

항상 가장 관심있는 리스트를 선택하십시오. 예를 들어 열의 리스트를 원하지만 원하는 열이 현재 선택되지 않은 표에 있다고 생각할 경우, F18(열 선택) 키를 누르십시오. 그런 후 F17 키를 눌러 표를 변경하십시오. 표 리스트가 먼저 선택될 경우, 표 이름이 명령문에 삽입됩니다. 사용자에게는 열 선택을 위한 선택사항이 없습니다.

SQL 입력 화면에서 SQL문을 입력하는 동안 언제나 리스트를 요구할 수 있습니다. 리스트에서 선택한 것이 SQL 입력 화면에 삽입됩니다. 리스트 화면에서 지정한 숫자 순서로 커서가 위치되어 있는 곳에 그것들이 삽입됩니다. 선택된 리스트 정보가 추가될지라도 명령문의 키워드를 입력해야 합니다.

리스트 기능은 선택열, 표 및 SQL 패키지에 필요한 규정을 제공합니다. 그러나 때로 리스트 기능이 SQL문의 목적을 판별할 수 없을 때도 있습니다. SQL문을 검토하고 선택한 열, 표 및 SQL 패키지가 적절히 규정되었는지 확인할 필요가 있습니다.

예: 리스트 선택 기능 사용

다음 예는 SELECT문을 빌드하기 위한 리스트 기능 사용법을 보여줍니다.

다음과 같은 작업을 한다고 가정합니다.

- OS/400 명령행에 STRSQL을 입력하여 대화식 SQL을 방금 입력했습니다.
- 어떤 리스트 선택이나 항목도 만들지 않았습니다.
- 명명 규칙을 위해 *SQL을 선택했습니다.

주: 예에서는 서버에 없는 리스트를 보여줍니다. 이는 예로서만 사용됩니다.

SQL문을 사용하여 다음을 시작하십시오.

1. 첫 번째 명령문 입력 행에 SELECT를 입력하십시오.
2. 두 번째 명령문 입력 행에 FROM을 입력하십시오.
3. 커서를 FROM 뒤에 두십시오.

```
                                SQL문 입력

SQL문을 입력한 후 Enter 키를 누르십시오.
===> SELECT
     FROM _
```

4. 표 이름은 FROM 뒤에 와야 하므로 F17(표 선택) 키를 눌러 표의 리스트를 살펴보십시오.

   예상대로 나타나는 표 리스트 대신 콜렉션 리스트가 나타납니다(콜렉션 선택 및 순서지정 화면). 방금 SQL 세션에 들어갔으며 작업할 스키마를 선택하지 않았습니다.

5. YOURCOLL2 스키마 옆의 *Seq* 열에 1을 입력하십시오.

```
콜렉션의 선택 및 순서 지정


콜렉션을 선택하려면 순번(1-999)을 입력하고 Enter 키를 누르십시오.

Seq   콜렉션          유형   텍스트
      YOURCOLL1       SYS    회사 손익
1     YOURCOLL2       SYS    직원 개인 자료
      YOURCOLL3       SYS    작업 분류/요구사항
      YOURCOLL4       SYS    회사 보험
```

6. Enter 키를 누르십시오.

표의 선택 및 순서지정 표 화면이 나타나며 YOURCOLL2 스키마에 있는 표를 보여줍니다.

7. 1을 PEOPLE 표의 옆에 있는 *Seq* 열에 입력하십시오.

```
                          표의 선택 및 순서 지정

표를 선택하려면 순서 번호(1-999)를 입력한 후 Enter 키를 누르십시오.

Seq  표                        콜렉션         유형  텍스트
     EMPLCO                    YOURCOLL2      TAB   사원 회사 자료
1    PEOPLE                    YOURCOLL2      TAB   사원 개인 자료
     EMPLEXP                   YOURCOLL2      TAB   사원 경험
     EMPLEVL                   YOURCOLL2      TAB   사원 평가 보고서
     EMPLBEN                   YOURCOLL2      TAB   사원 손익 레코드
     EMPLMED                   YOURCOLL2      TAB   사원 의료 레코드
     EMPLINVST                 YOURCOLL2      TAB   사원 투자 레코드
```

8. Enter 키를 누르십시오.

   SQL 입력 화면이 FROM 다음에 삽입된 표 이름 YOURCOLL2.PEOPLE과 함께 다시 나타납니다. 표 이름은 *SQL 명명 규칙의 스키마명에 의해 규정됩니다.

```
                              SQL문 입력

SQL문을 입력한 후 Enter 키를 누르십시오.
===> SELECT
     FROM YOURCOLL2.PEOPLE _
```

9. 커서를 SELECT 뒤에 두십시오.
10. 열 이름은 SELECT 뒤에 와야 하므로 F18(열 선택) 키를 눌러 열의 리스트를 살펴보십시오.

    PEOPLE 표의 열을 보여주는 열 선택 및 순서 지정 화면이 나타납니다.

11. 2를 *NAME* 열 옆에 있는 *Seq* 열에 입력하십시오.
12. 1을 *SOCSEC* 열 옆에 있는 *Seq* 열에 입력하십시오.

```
                          열 선택 및 순서 지정

열을 선택하려면 순서 번호(1-999)를 입력한 후 Enter 키를 누르십시오.

Seq  열                   표                   유형                  자리수
2    NAME                 PEOPLE               CHARACTER               6
     EMPLNO               PEOPLE               CHARACTER              30
1    SOCSEC               PEOPLE               CHARACTER              11
     STRADDR              PEOPLE               CHARACTER              30
     CITY                 PEOPLE               CHARACTER              20
     ZIP                  PEOPLE               CHARACTER               9
     PHONE                PEOPLE               CHARACTER              20
```

13. Enter 키를 누르십시오.

    SQL 입력 화면은 SELECT 다음에 나타나는 SOCSEC, NAME과 함께 다시 표시됩니다.

```
                                SQL문 입력

SQL문을 입력한 후 Enter 키를 누르십시오.
===> SELECT SOCSEC, NAME
     FROM YOURCOLL2.PEOPLE
```

14. Enter 키를 누르십시오.

    이제 작성한 명령문이 수행됩니다.

일단 리스트 기능을 사용하면 세션 속성 변경 화면에서 선택한 값을 사용자가 변경할 때까지 또는 스키마의 리스트를 변경할 때까지 효력을 유지합니다.

세션 서비스 설명

SQL문 입력 화면에서 F13 키를 누르면 대화식 SQL 세션 서비스 화면이 요청됩니다.

이 화면에서는 세션 속성을 변경하고 세션의 인쇄, 지움 또는 소스 파일로의 저장을 수행할 수 있습니다.

옵션 1(세션 속성 변경)은 세션 속성 변경 화면을 표시하며 이는 대화식 SQL 세션에 적용되는 현재의 값을 선택할 수 있게 해줍니다. 이 화면에서 나타난 옵션은 선택된 명령문 처리 옵션에 기초하여 변경됩니다.

다음 세션 속성을 변경할 수 있습니다.

- 확약 제어 속성
- 명령문 처리 제어
- SELECT 출력 장치
- 스키마의 리스트
- 모든 시스템과 SQL 오브젝트 또는 SQL 오브젝트만을 선택하기 위한 리스트 유형
- 자료를 표시할 경우 화면정리 옵션
- 자료 복사 허용 옵션
- 명명 옵션
- 프로그램 작성 언어
- 날짜 형식
- 시간 형식
- 날짜 분리 문자
- 시간 분리자
- 소수점 표시
- SQL 스트링 분리문자

- 정렬 순서
- 언어 식별자

옵션 2(현재 세션 인쇄)는 즉시 현재 세션을 인쇄한 다음 작업을 계속할 수 있는 프린터 변경 화면에 액세스합니다. 프린터 정보를 입력하도록 프롬트가 제공됩니다. 입력한 모든 SQL문과 표시된 모든 메세지는 SQL 입력 화면에 나온 대로 인쇄됩니다.

옵션 3(현재 세션에서 모든 항목 제거)을 사용하여 SQL문 입력 화면과 세션 이력에서 모든 SQL문과 메세지를 제거할 수 있습니다. 정보를 삭제하려는지를 확인하기 위해 사용자에게 프롬트가 제공됩니다.

옵션 4(소스 파일에 세션 저장)는 사용자가 소스 파일에 세션을 저장할 수 있는 소스 파일 변경 화면에 액세스합니다. 소스 파일명을 입력하도록 프롬트가 제공됩니다. 이 기능으로 사용자는 소스 입력 유틸리티(SEU)를 통해 호스트 언어 프로그램에 소스 파일을 삽입할 수 있습니다.

주: 옵션 4로 SQL을 사용하는 고급 언어(HLL) 프로그램의 표준 SQL문을 삽입할 수 있습니다. 옵션 4에 의해 작성된 소스 파일은 RUNSQLSTM(SQL문 실행)에 대한 입력 소스 파일로 편집되고 사용될 수 있습니다.

대화식 SQL 나가기

SQL 입력 화면에서 F3(나감)을 누르면 대화식 SQL 환경에서 나가 다음 중 하나를 수행할 수 있습니다.

1. 세션 저장 후 나감. 대화식 SQL에서 나갑니다. 현재의 세션이 저장되어 다음에 대화식 SQL을 시작할 때 사용됩니다.
2. 세션을 저장하지 않고 나감. 세션을 저장하지 않고 대화식 SQL에서 나갑니다.
3. 세션 재개. 대화식 SQL에 있으면서 SQL 입력 화면으로 리턴합니다. 현재의 세션 매개변수가 계속 적용됩니다.
4. 소스 파일에 세션 저장. 소스 파일에 현재의 세션을 저장합니다. 세션을 저장할 장소를 선택할 수 있도록 '소스 파일 변경' 화면이 나타납니다. 대화식 SQL에서는 이 세션을 회복할 수 없으며 다시 이 세션에 대해 작업할 수도 없습니다.

주:

1. 옵션 4로는 SQL을 사용한 고급 언어(HLL) 프로그램에 원래 SQL문을 삽입할 수 있습니다. 프로그램에 명령문을 복사하려면 소스 입력 유틸리티(SEU)를 사용하십시오. 소스 파일이 편집되어 RUNSQLSTM(SQL문 실행) 명령에 대한 입력 소스 파일로 사용될 수 있습니다.
2. 행이 변경되고 현재 이 작업 단위에 대한 잠금이 보유된 상태에서 대화식 SQL을 나가려고 하면 경고 메세지가 표시됩니다.

기존 SQL 세션 사용

대화식 SQL 나감 화면에서 옵션 1(세션 저장 및 나감)을 사용하여 하나의 대화식 SQL 세션만을 저장한 경우, 어떤 워크스테이션에서도 해당 세션을 재개할 수 있습니다. 그러나 옵션 1을 사용하여 다른 워크스테이션에 둘 이상의 세션을 저장하는 경우, 대화식 SQL은 먼저 워크스테이션과 일치하는 세션을 재개하려 합니다. 일치하는 세션을 사용할 수 없는 경우, 대화식 SQL이 탐색 범위를 증가시켜 사용자 ID에 속하는 모든 세션을 포함시킵니다. 사용자 ID에 대한 세션을 사용할 수 없는 경우, 시스템은 사용자 ID와 현재 워크스테이션에 대한 새로운 세션을 작성합니다.

예를 들어 한 세션을 워크스테이션 1에 저장하고 또 다른 세션을 워크스테이션 2에 저장했으며 현재 워크스테이션 1에서 작업중인 경우를 가정할 수 있습니다. 대화식 SQL은 먼저 워크스테이션 1에 대해 저장된 세션을 재개하려 합니다. 해당 세션이 현재 사용중이라면 대화식 SQL은 워크스테이션 2에 대해 저장된 세션을 재개하려 합니다. 해당 세션도 사용중이라면 시스템은 워크스테이션 1에 대한 두 번째 세션을 작성합니다.

그러나 워크스테이션 3에서 작업중이고 워크스테이션 2와 연관된 ISQL 세션을 사용하려 한다고 가정합니다. 이 경우에는 먼저 대화식 SQL 나감 화면의 옵션 2(세션을 저장하지 않고 나감)를 사용함으로써 워크스테이션 1에서 세션을 삭제해야 할 수도 있습니다.

SQL 세션 회복

이전 세션이 이상 종료되었으면 대화식 SQL은 다음 세션 시작시(다음 STRSQL 명령이 입력될 때) SQL 세션 회복 화면을 표시합니다. 이 화면에서, 다음 중 하나를 수행할 수 있습니다.

- 옵션 1(기존의 SQL 세션 재개 시도)을 선택함으로써 기존의 세션을 회복합니다.
- 옵션 2(기존의 SQL 세션 삭제 및 새로운 세션 시작)를 선택함으로써 기존의 세션을 삭제하고 새로운 세션을 시작합니다.

기존 세션을 삭제하고 새로운 세션으로 계속하도록 선택하면 STRSQL을 입력했을 때 지정한 매개변수가 사용됩니다. 기존 세션을 회복하기 위해 선택하거나 이전에 저장된 세션에 들어가고 있으면 STRSQL을 입력할 때 지정한 매개변수는 무시되고 기존 세션의 매개변수가 사용됩니다. 매개변수가 지정된 값에서 기존 세션 값으로 변경되었음을 표시하도록 메세지가 리턴됩니다.

대화식 SQL로 리모트 데이터베이스 액세스

대화식 SQL에서 SQL CONNECT문을 사용하면 리모트 관계형 데이터베이스와 통신할 수 있습니다. 대화식 SQL은 CONNECT문에 대해 CONNECT(유형 2) 의미(분배된 작업 단위)를 사용합니다. 대화식 SQL은 SQL 세션을 시작할 때 로컬 RDB에 내재적 연결을 수행합니다. CONNECT문이 완료될 때 메세지는 설정된 관계형 데이터베이스 연결을 나타냅니다. 새로운 세션의 시작 및 COMMIT(*NONE)가 지정되지 않았

거나 저장된 세션의 복원이 *NONE이 아니고 세션과 함께 저장된 확약 레벨도 *NONE이 아니면 연결이 확약 제어로 등록됩니다. 이러한 내재 연결 및 가능한 확약 제어 등록은 리모트 데이터베이스에 대한 후속 연결에 영향을 줄 수도 있습니다. 자세한 정보를 보려면 382 페이지의 『연결 유형 판별』을 참조하십시오. 다음은 리모트 시스템과의 연결에 앞서 권장되는 사항입니다.

- 분산 작업 단위를 지원하지 않는 어플리케이션 서버와 연결할 때에는 로컬과의 연결을 포함하여 이전의 모든 연결을 종료하도록 RELEASE ALL 앞에 COMMIT를 사용하십시오.
- 비iSeries용 DB2 UDB 어플리케이션 서버와 연결할 때에는 로컬과의 내재적 연결을 포함하여 이전의 연결을 종료하도록 RELEASE ALL 및 COMMIT를 발행하고 최소한 *CHG로 확약 제어 레벨을 변경하십시오.

비iSeries용 DB2 UDB 어플리케이션 서버와 연결할 때 몇몇 세션 속성은 해당 어플리케이션 서버가 지원하는 속성으로 변경됩니다. 다음 표는 변경되는 속성을 보여줍니다.

표 36. 수치표

세션 속성	원래 값	새로운 값
날짜 형식	*YMD *DMY *MDY *JUL	*ISO *EUR *USA *USA
시간 형식	하나의 : 분리자가 있는 *HMS다른 분리자가 있는 *HMS	*JIS *EUR
확약 제어	*CHG, *NONE *ALL	*CS 반복 가능 읽기
명명 규칙	*SYS	*SQL
자료 복사 허용	*NO, *YES	*OPTIMIZE
자료 정리	*ALWAYS	*FORWARD
소수점	*SYSVAL	*PERIOD
정렬 순서	*HEX 외의 다른 수	*HEX

주:

1. 버전 2 릴리스 3 이전의 릴리스를 실행 중인 서버에 연결하는 경우, 정렬 순서 값은 *HEX로 변경됩니다.
2. DB2/2 또는 DB2/6000 어플리케이션 서버에 연결될 때 지정된 날짜 및 시간 형식은 서로 동일해야 합니다.

연결이 완료된 후 세션 속성이 변경되었음을 알리는 메세지가 송신됩니다. 변경된 세션 속성은 세션 서비스 화면을 사용하여 표시될 수 있습니다. 대화식 SQL이 수행되는 동안에는 어떤 연결도 디폴트 활성 그룹에 대해 설정되지 않습니다.

대화식 SQL로 리모트 시스템에 연결될 때 구문만으로 된 명령문 처리 모드는 리모트 시스템 대신 로컬 시스템에 의해 지원된 구문에 대해 명령문의 구문을 체크합니다. 이와 유사하게 SQL 프롬터 및 리스트 지원은 명령문 구문과 로컬 시스템에서 지원하는 명명 규칙을 사용합니다. 그러나 명령문은 리모트 시스템에서 수행됩니다. 두 시스템 사이의 SQL 지원 레벨의 차이 때문에 수행시 리모트 시스템의 명령문에서 구문 오류가 발견될 수 있습니다.

로컬 관계형 데이터베이스에 연결되어 있으면 스키마와 표의 리스트를 사용할 수 있습니다. 열의 리스트는 DESCRIBE TABLE문을 지원하는 관계형 데이터베이스 관리자에 사용자가 연결되어 있을 경우에만 사용이 가능합니다.

보호 대화를 사용하는 지연중인 변경 또는 연결 상태에서 대화식 SQL을 나갈 때에는 연결이 그대로 유지됩니다. 연결에 대한 추가 작업을 수행하지 않을 경우, 연결은 다음 COMMIT 또는 ROLLBACK 조작중에 종료됩니다. 대화식 SQL을 나가기 전에 RELEASE ALL 및 COMMIT를 수행하면 연결을 종료시킬 수 있습니다.

비iSeries용 DB2 UDB 어플리케이션 서버의 리모트 액세스에 대한 대화식 SQL을 사용하려면 몇 가지 설정이 필요합니다. 자세한 내용은 분산 데이터베이스 프로그래밍 책을 참조하십시오.

주: 통신 추적의 출력에는 ‘CREATE TABLE XXX’문에 대한 참조가 있을 수 있습니다. 이것은 패키지의 존재를 판별하는데 사용되며 정상적인 처리의 일부이므로 무시해서는 안됩니다.

제 18 장 SQL문 프로세서 사용

이 섹션에서는 SQL문 프로세서를 설명합니다. 이 프로세서는 SQL문 수행 (RUNSQLSTM) 명령에 사용될 수 있습니다.

SQL문 프로세서는 SQL문이 소스 멤버로부터 처리되도록 합니다. 소스 멤버의 명령문은 소스 컴파일 없이 계속적으로 수행되고 변경될 수 있습니다. 이것은 데이터베이스 환경을 더 쉽게 설정할 수 있도록 해 줍니다. SQL문 프로세서를 사용할 수 있는 명령문은 다음과 같습니다.

- ALTER TABLE
- CALL
- COMMENT ON
- COMMIT
- CREATE ALIAS
- CREATE DISTINCT TYPE
- CREATE FUNCTION
- CREATE INDEX
- CREATE PROCEDURE
- CREATE SCHEMA
- CREATE TABLE
- CREATE TRIGGER
- CREATE VIEW
- DECLARE GLOBAL TEMPORARY TABLE
- DELETE
- DROP
- GRANT(기능 또는 프로시듀어 권한)
- GRANT(패키지 권한)
- GRANT(표 권한)
- GRANT(사용자 정의 유형 특권)
- INSERT
- LABEL ON
- LOCK TABLE
- RELEASE SAVEPOINT

- RENAME
- REVOKE(기능 또는 프로시듀어 권한)
- REVOKE(패키지 권한)
- REVOKE(표 권한)
- REVOKE(사용자 정의 유형 권한)
- ROLLBACK
- SAVEPOINT
- SET PATH
- SET SCHEMA
- SET TRANSACTION
- UPDATE

소스 멤버에서 명령문은 세미콜론으로 끝나고 **EXEC SQL**로 시작되지 않습니다. 소스 멤버의 레코드 길이가 80 이하인 경우, 처음 80자만이 읽힙니다. 소스 멤버의 주석은 행 주석 또는 블록 주석일 수 있습니다. 행 주석은 이중 하이픈(--)으로 시작되고 행의 끝에서 종료합니다. 블록 주석은 /*로 시작되고 다음의 */에 도달할 때까지 여러 행에 걸쳐 계속될 수 있습니다. 블록 주석은 중첩될 수 있습니다. SQL문과 주석만이 소스 파일에서 허용됩니다. SQL문에 대한 출력 리스트 및 결과 메세지는 인쇄 파일로 보내집니다. 디폴트 인쇄 파일은 QSYSPRT입니다.

소스 멤버의 모든 명령문에서만 구문 체크하려면 RUNSQLSTM 명령에 PROCESS(*SYN) 매개변수를 지정하십시오.

자세한 내용은 다음의 절을 참조하십시오.

- 『오류 발생 후 명령문 실행』
- 339 페이지의 『SQL문 프로세서에서 확약 제어』
- 339 페이지의 『SQL문 프로세서의 스키마』
- 340 페이지의 『SQL문 프로세서에 대한 소스 멤버 리스트』

오류 발생 후 명령문 실행

명령문이 RUNSQLSTM 명령의 오류 레벨(ERRLVL) 매개변수에 대해 지정된 값보다 높은 심각한 오류를 리턴할 경우, 명령문은 실패합니다. 소스의 나머지 명령문이 구문 오류를 체크하기 위하여 기술될 수 있으나 이들 명령문은 실행될 수 없습니다. 대부분 SQL 오류는 30의 심각도를 가집니다. SQL문 실패 후 처리를 계속하려면 RUNSQLSTM 명령의 ERRLVL 매개변수를 30 이상으로 설정하십시오.

SQL문 프로세서에서 확약 제어

확약 제어 레벨은 RUNSQLSTM 명령에 지정됩니다. *NONE 이외의 확약 제어 레벨이 지정된다면 SQL문은 확약 제어 아래에서 수행됩니다. 모든 명령문이 제대로 실행되었다면 COMMIT는 SQL문 처리 프로그램 종료시에 수행됩니다. 그렇지 않다면 ROLLBACK이 수행됩니다. 명령문의 리턴 코드 심각도가 RUNSQLSTM 명령의 ERRLVL 매개변수에 지정된 값 이하이면 명령문은 제대로 작성된 것입니다.

SET TRANSACTION문은 소스 멤버 내의 RUNSQLSTM 명령에서 지정된 확약 제어 레벨을 대체하는데 사용될 수 있습니다.

주: 확약 제어를 갖는 SQL문 처리 프로그램을 사용하려면 작업이 작업 단위 경계에 있어야 합니다.

SQL문 프로세서의 스키마

SQL문 처리프로그램은 CREATE SCHEMA문을 지원합니다. 이는 두 개의 고유한 섹션을 가진 복합 명령문입니다. 첫 번째 섹션은 스키마에 대한 콜렉션을 정의합니다. 두 번째 섹션에는 콜렉션에서 오브젝트를 정의하는 DDL문이 있습니다.

첫 번째 섹션은 다음 두 가지 방식 중 하나로 작성할 수 있습니다.

- CREATE SCHEMA *collection-name*

  작성된 콜렉션은 지정 콜렉션명을 사용합니다.
- CREATE SCHEMA AUTHORZATION 권한명

  작성된 콜렉션은 콜렉션명으로 권한명을 사용합니다. 스키마 수행시, 사용자는 **권한명**으로 명명된 사용자 프로파일에 대한 권한을 가져야 합니다.

  명령문의 권한명에 보유된 권한에는 반드시 다음을 포함시켜야 합니다.
  - CREATE COLLECTION문을 실행할 권한
  - CREATE SCHEMA 내에 있는 각 SQL문을 실행할 권한

CREATE SCHEMA문의 두 번째 섹션에는 개수에 관계없이 다음 명령문을 포함시킬 수 있습니다.

- COMMENT ON
- CREATE ALIAS
- CREATE DISTINCT TYPE
- CREATE INDEX
- CREATE TABLE
- CREATE VIEW
- GRANT(표 권한)

- GRANT(사용자 정의 유형 특권)
- LABEL ON

이들 명령문은 명령문의 첫 번째 섹션 바로 뒤에 나옵니다. 명령문과 섹션은 세미콜론으로 분리**되지** 않습니다. 다른 SQL문이 이 스키마 정의 뒤에 온다면 스키마의 최종 명령문은 세미콜론으로 종료해야 합니다.

스키마 명령문의 두 번째 부분에서 작성되거나 참조된 모든 오브젝트는 스키마에 대해 작성된 콜렉션에 있어야 합니다. 규정되지 않은 모든 참조는 내재적으로 작성된 콜렉션에 의해 규정되어야 합니다. 규정된 모든 참조는 작성된 콜렉션에 의해 규정되어야 합니다.

SQL문 프로세서에 대한 소스 멤버 리스트

코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

```
5722ST1 V5R2M0 020719              Run SQL Statements            SCHEMA                          08/06/02 15:35:18   Page   1
Source file..............CORPDATA/SRC
Member..................SCHEMA
Commit..................*NONE
Naming..................*SYS
Generation level.........10
Date format..............*JOB
Date separator...........*JOB
Time format..............*HMS
Time separator ..........*JOB
Default Collection.......*NONE
IBM SQL flagging.........*NOFLAG
ANS flagging.............*NONE
Decimal point............*JOB
Sort Sequence............*JOB
Language ID..............*JOB
Printer file.............*LIBL/QSYSPRT
Source file CCSID.........65535
Job CCSID.................0
Statement processing......*RUN
Allow copy of data........*OPTIMIZE
Allow blocking...........*READ
Source member changed on 04/01/98  11:54:10
```

그림 8. SQL문 프로세서에 대한 QSYSPRT 리스트 (1/3)

```
5722ST1 V5R2M0 020719              Run SQL Statements            SCHEMA                          08/06/02 15:35:18   Page   2
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
    1
    2     DROP COLLECTION DEPT;
    3     DROP COLLECTION MANAGER;
    4
    5     CREATE SCHEMA DEPT
    6          CREATE TABLE EMP (EMPNAME CHAR(50), EMPNBR INT)
    7                           -- EMP will be created in collection DEPT
    8          CREATE INDEX EMPIND ON EMP(EMPNBR)
    9                           -- EMPIND will be created in DEPT
   10          GRANT SELECT ON EMP TO PUBLIC;  -- grant authority
   11
   12     INSERT INTO DEPT/EMP VALUES('JOHN SMITH', 1234);
   13                            /* table must be qualified since no
   14                               longer in the schema */
   15
   16     CREATE SCHEMA AUTHORIZATION MANAGER
   17                             -- this schema will use MANAGER's
   18                             -- user profile
   19           CREATE TABLE EMP_SALARY (EMPNBR INT, SALARY DECIMAL(7,2),
   20                                    LEVEL CHAR(10))
   21           CREATE VIEW LEVEL AS SELECT EMPNBR, LEVEL
   22                FROM EMP_SALARY
   23           CREATE INDEX SALARYIND ON EMP_SALARY(EMPNBR,SALARY)
   24
   25           GRANT ALL ON LEVEL TO JONES GRANT SELECT ON EMP_SALARY TO CLERK
   26                   -- Two statements can be on the same line
* * * * *  E N D  O F  S O U R C E  * * * * *
```

그림 8. SQL문 프로세서에 대한 QSYSPRT 리스트 (2/3)

```
5722ST1 V5R2M0 020719              Run SQL Statements            SCHEMA                          08/06/02 15:35:18   Page     3
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
MSG ID  SEV  RECORD  TEXT
SQL7953   0       1  Position 1 Drop of DEPT in QSYS complete.
SQL7953   0       3  Position 3 Drop of MANAGER in QSYS complete.
SQL7952   0       5  Position 3 Collection DEPT created.
SQL7950   0       6  Position 8 Table EMP created in DEPT.
SQL7954   0       8  Position 8 Index EMPIND created in DEPT on table EMP in
                     DEPT.
SQL7966   0      10  Position 8 GRANT of authority to EMP in DEPT completed.
SQL7956   0      10  Position 40 1 rows inserted in EMP in DEPT.
SQL7952   0      13  Position 28 Collection
                     MANAGER created.
SQL7950   0      19  Position 9 Table EMP_SALARY created in collection
                     MANAGER.
SQL7951   0      21  Position 9 View LEVEL created in MANAGER.
SQL7954   0      23  Position 9 Index SALARYIND created in MANAGER on table
                     EMP_SALARY in MANAGER.
SQL7966   0      25  Position 9 GRANT of authority to LEVEL in MANAGER
                     completed.
SQL7966   0      25  Position 37 GRANT of authority to EMP_SALARY in MANAGER
                     completed.

Message Summary
Total    Info    Warning    Error    Severe    Terminal
13     13         0         0         0          0
00 level severity errors found in source
                            * * * * *  E N D  O F  L I S T I N G  * * * * *
```

그림 8. SQL문 프로세서에 대한 QSYSPRT 리스트 (3/3)

제 19 장 iSeries용 DB2 UDB 자료 보호

이 장에서는 권한이 없는 사용자로부터 SQL 자료를 보호하는 보안 계획과 자료 무결성을 보장하기 위한 방법을 설명합니다. 자세한 내용은 다음의 주제를 참조하십시오.

- 『SQL 오브젝트에 대한 보안』
- 345 페이지의 『iSeries Navigator를 사용한 자료 보안』
- 347 페이지의 『자료 무결성』

코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

SQL 오브젝트에 대한 보안

SQL 오브젝트를 비롯하여 서버의 모든 오브젝트는 시스템 보안 기능이 관리합니다. 사용자는 SQL 오브젝트에 대해 SQL GRANT나 REVOKE문 또는 CL 명령인 오브젝트 권한 편집(EDTOBJAUT), 오브젝트 권한부여(GRTOBJAUT), 오브젝트 권한 취소(RVKOBJAUT) 명령으로 권한을 부여받을 수 있습니다. 시스템 보안과 GRTOBJAUT 및 RVKOBJAUT 명령 사용에 대한 자세한 정보는 iSeries 보안 참조서 를 참조하십시오.

SQL GRANT 및 REVOKE문은 SQL 패키지, SQL 프로시듀어, 표, 보기 및 표와 보기의 개별적인 열에 대해 조작합니다. 또한 SQL GRANT와 REVOKE문은 개인용 권한과 공용 권한만을 부여합니다. 몇몇 경우에는 사용자에게 명령 및 프로그램과 같은 다른 오브젝트에 대한 권한을 부여하기 위해 EDOBJAUT, GRTOBJAUT 및 RVKOBJAUT를 사용해야 합니다.

GRANT문 및 REVOKE문에 대한 자세한 내용은 SQL 참조서를 참조하십시오.

SQL문에 대해 체크되는 권한은 명령문이 정적 실행, 동적 실행, 대화식 실행인지에 따라 다릅니다.

정적 SQL문의 경우:

- USRPRF 값이 *USER인 경우, SQL문을 실행하기 위한 권한은 프로그램을 실행하는 사용자 프로파일을 사용하여 로컬로 체크됩니다. 리모트로 SQL문을 실행하는 권한은 어플리케이션 서버에서 사용자 프로파일을 사용하여 체크됩니다. *USER는 시스템(*SYS)명에 대한 디폴트입니다.
- USRPRF 값이 *OWNER인 경우, SQL문을 실행하기 위한 권한은 프로그램을 실행하는 사용자 및 프로그램 소유자의 사용자 프로파일을 사용하여 로컬로 체크됩니다. 리모트로 SQL문을 실행하는 권한은 어플리케이션 서버 작업 및 SQL 패키지 소

유자의 사용자 프로파일을 사용하여 체크됩니다. 최상위 권한이 사용됩니다. *OWNER는 SQL(*SQL)명에 대한 디폴트입니다.

동적 SQL문의 경우:

- USRPRF 값이 *USER인 경우, SQL문을 실행하기 위한 권한은 프로그램을 실행하는 사용자 프로파일을 사용하여 로컬로 체크됩니다. 리모트로 SQL문을 실행하는 권한은 어플리케이션 서버 작업의 사용자 프로파일을 사용하여 체크됩니다.
- USRPRF 값이 *OWNER이고 DYNUSRPRF가 *USER인 경우, SQL문을 수행하기 위한 권한은 프로그램을 실행하는 사용자가 사용자 프로파일을 사용하여 로컬로 체크됩니다. 리모트로 SQL문을 실행하는 권한은 어플리케이션 서버 작업의 사용자 프로파일을 사용하여 체크됩니다.
- USRPRF 값이 *OWNER이고 DYNUSRPRF가 *OWNER인 경우, SQL문을 실행하기 위한 권한은 프로그램을 실행하는 사용자 및 프로그램 소유자의 사용자 프로파일을 사용하여 로컬로 체크됩니다. 리모트로 SQL문을 실행하는 권한은 어플리케이션 서버 작업 및 SQL 패키지 소유자의 사용자 프로파일을 사용하여 체크됩니다. 최상위 권한이 사용됩니다. 보안 관계상 DYNUSRPRF에 대한 *OWNER 매개변수 값을 주의해서 사용해야 합니다. 이 옵션은 프로그램을 실행할 사용자에게 프로그램 소유자의 액세스 권한을 부여합니다.

대화식 SQL문의 경우, 권한은 명령문을 처리하는 사용자의 권한과 대조되어 체크됩니다. 허용된 권한은 대화식 SQL문에 대해서는 사용되지 않습니다.

권한부여 ID

권한부여 ID는 고유 사용자를 식별하며 서버에서는 사용자 프로파일 오브젝트입니다. 권한부여 ID는 시스템 CRTUSRPRF(사용자 프로파일 작성) 명령을 사용하여 작성됩니다.

보기

보기는 권한이 없는 사용자가 중요한 자료에 액세스하는 것을 방지할 수 있습니다. 어플리케이션 프로그램은 표에 있는 중요한 자료 또는 제한된 자료에 액세스하지 않고 표에서 필요로 하는 자료에 액세스할 수 있습니다. 보기는 SELECT 리스트(예: 사원 급여)에서 이러한 열들을 지정하지 않음으로써 특정 열에 대한 액세스를 제한할 수 있습니다. 보기는 WHERE절을 지정함으로써 표에서 특정 행에 대한 액세스를 제한할 수도 있습니다(예를 들어 특정 부서 번호와 연관된 행에 대한 액세스만을 허용함).

감사(auditing)

iSeries용 DB2 UDB는 미정부의 C2 보안 레벨에 맞게 설계되었습니다. 이 레벨의 기본 피처는 시스템에서 조치에 대한 감사 기능입니다. iSeries용 DB2 UDB에서는 시스템 보안 기능에 의해 관리되는 감사 기능을 사용합니다. 감사는 오브젝트 레벨, 사용자 또는 시스템 레벨에서 수행될 수 있습니다. 시스템 값 QAUDCTL은 감사가 오브젝트

에서 수행되는지 또는 사용자 레벨에서 수행되는지를 제어합니다. CHGUSRAUD(사용자 감사 변경) 명령 및 CHGOBJAUD(오브젝트 감사 변경) 명령은 어느 사용자와 오브젝트가 감사될 것인지를 지정합니다. 시스템 값 QAUDLVL은 어떤 유형의 조치가 감사될 것인지를 제어합니다(예를 들어 권한부여 실패, 작성, 삭제, 부여, 취소 등). 감사에 대한 자세한 정보는 iSeries 보안 참조서 를 참조하십시오.

iSeries용 DB2 UDB에서는 iSeries용 DB2 UDB 저널 지원을 사용하여 행 변경사항을 감사할 수 있습니다.

저널 감사 항목은 종종 발생 순서와 같지 않은 경우도 있습니다. 예를 들어 확약 제어하에서 수행중인 작업은 표를 삭제하고 이 표와 동일한 이름으로 새로운 표를 작성한 후 확약합니다. 이 경우에 감사 저널에는 작성 후 삭제로 기록될 것입니다. 이는 작성된 오브젝트가 즉시 저널링되기 때문입니다. 확약 제어하에서 삭제된 오브젝트는 확약이 수행될 때까지 단지 숨겨질 뿐이고 실제로 삭제되는 것은 아닙니다. 일단 확약되면 조치가 저널링됩니다.

iSeries Navigator를 사용한 자료 보안

iSeries Navigator를 사용하면 다음과 같이 수행하여 자료를 보안할 수 있습니다.

- 『오브젝트에 대한 공용 권한 정의』
- 346 페이지의 『새로운 오브젝트에 대한 디폴트 공용 권한 설정』
- 346 페이지의 『오브젝트에 대해 사용자 또는 그룹 권한부여』

오브젝트에 대한 공용 권한 정의

공용 권한은 오브젝트에 대한 특정 액세스 권한이 없는 사용자에게 액세스 유형을 설명하기 위해 시스템의 모든 오브젝트에 대해 정의됩니다. 공용 권한을 정의하려면, 다음을 수행하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. 권한을 편집하려는 오브젝트가 보일 때까지 탐색하십시오.
3. 권한을 추가하려는 오브젝트에서 마우스 오른쪽 버튼을 클릭하고 **권한**을 선택하십시오.
4. **권한** 대화 상자의 그룹 리스트에서 **공용**을 선택하십시오.
5. 세부 권한을 구현하기 위해 **세부사항** 버튼을 클릭하십시오.
6. 적합한 선택란 옆의 상자를 체크표시하여 공용으로 원하는 권한을 적용하십시오.
7. **확인**을 클릭하십시오.

새로운 오브젝트에 대한 디폴트 공용 권한 설정

디폴트 공용 권한 설정을 통해 라이브러리에서 새로운 오브젝트가 작성될 때 이러한 새로운 모든 오브젝트에 지정되는 공통 권한을 가질 수 있습니다. 다른 레벨의 보안이 필요한 개별 오브젝트에 대해 권한을 편집할 수 있습니다. 디폴트 공용 권한을 설정하려면, 다음을 수행하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. 공용 권한을 설정하려는 라이브러리에서 마우스 오른쪽 버튼을 클릭하고 **권한**을 선택하십시오.
3. **권한** 대화 상자에서 **새 오브젝트**를 클릭하십시오.
4. **새 오브젝트** 대화 상자에서 디폴트 공용 권한을 선택하십시오. 권한부여 리스트를 지정하려면, 권한부여 리스트의 이름을 입력하거나 **찾아보기**로 찾을 수 있습니다. 권한부여 리스트 등록정보를 보려면, **열기**를 선택하십시오.
5. **확인**을 클릭하십시오.

오브젝트에 대해 사용자 또는 그룹 권한부여

일부 사용자들은 공용 권한에서 허용하는 권한과 다른 권한을 오브젝트에 대해 요구할 수 있습니다. 오브젝트에 대해 사용자 또는 그룹을 권한부여하려면, 다음을 수행하십시오.

1. **iSeries Navigator** 창에서 서버 → **데이터베이스** → 작업하려는 데이터베이스 → **라이브러리**를 펼치십시오.
2. 권한을 편집하려는 오브젝트가 보일 때까지 탐색하십시오.
3. 권한을 추가하려는 오브젝트에서 마우스 오른쪽 버튼을 클릭하고 **권한**을 선택하십시오.
4. **권한** 대화 상자에서 **추가**를 클릭하십시오.
5. **추가** 대화 상자에서 하나 이상의 사용자 및 그룹을 선택하거나 사용자 또는 그룹 이름 필드에 사용자 또는 그룹의 이름을 입력하십시오.
6. **확인**을 클릭하십시오.

그러면, 리스트의 맨 위에 사용자 또는 그룹이 추가됩니다.

주: 사용자 또는 그룹에는 오브젝트에 대한 디폴트 권한이 부여됩니다. 사용자의 권한을 시스템에 의해 정의된 유형 중 하나로 변경하거나 권한을 사용자 정의할 수 있습니다.

오브젝트에서 사용자 권한을 제거하려면, 다음을 수행하십시오.

1. 제거하려는 사용자나 그룹을 선택하십시오.
2. **제거**를 클릭하십시오.

자료 무결성

자료 무결성은 권한이 없는 사람에 의한 시스템 조작이나 하드웨어 장애(디스크에 물리적 훼손), 프로그래밍 오류, 작업이 완료되기 전의 인터럽션(전원 장애와 같은) 또는 동시에 수행중인 어플리케이션의 간섭(순번 지정 문제 등)으로 인한 훼손이나 변경으로부터 자료를 보호합니다. 자료 무결성은 다음 기능에 의해서 보장됩니다.

- 『동시성』
- 349 페이지의 『저널링』
- 350 페이지의 『확약 제어』
- 356 페이지의 『아토믹 조작』
- 358 페이지의 『제한조건』
- 359 페이지의 『저장/복원』
- 360 페이지의 『손상 허용 한계』
- 360 페이지의 『색인 회복』
- 361 페이지의 『카탈로그 무결성』
- 362 페이지의 『사용자 보조 기억장치 풀(ASP)』
- 362 페이지의 『독립 보조 기억장치 풀(IASP)』

확약 제어, 저널 관리, 데이터베이스 프로그래밍 및 백업 및 회복  등에서 각 기능에 대한 자세한 정보를 참조할 수 있습니다.

동시성

동시성은 여러 사용자가 자료 무결성의 손상 없이 동시에 같은 표나 보기의 자료에 액세스하고 변경할 수 있는 기능입니다. 이러한 기능은 iSeries용 DB2 UDB 데이터베이스 관리자에 의해 자동으로 제공됩니다. 잠금은 사용자가 동시에 같은 자료를 변경하는 것을 보호하기 위해 표와 행에 내재적으로 사용이 지정됩니다.

일반적으로 iSeries용 DB2 UDB는 무결성을 보장하기 위해 행에 대한 잠금을 확보합니다. 그러나 몇몇 상황에서는 iSeries용 DB2 UDB가 행 잠금 대신 보다 배타적인 표 잠금을 확보해야 합니다. 추가 정보는 350 페이지의 『확약 제어』를 참조하십시오.

예를 들어 현재 한 커서에 의해 보유되는 행에 대한 갱신(배타적) 잠금은 같은 프로그램의 또 다른 커서에 의해 확보될 수 있습니다(또는 DELETE나 UPDATE문이 해당 커서와 연관되지 않음). 이로써 다른 FETCH가 수행될 때까지 첫 번째 커서를 참조하는 지정된 UPDATE와 지정된 DELETE문을 막습니다. 현재 커서에 의해 보유된 행에 대한 읽기(공유된 비갱신) 잠금은 같은 프로그램 내의 또 다른 커서(또는 DELETE나 UPDATE문)가 같은 행에 대한 잠금을 확보하는 것을 막지 않습니다.

디폴트 및 사용자가 지정할 수 있는 잠금 대기 시간종료 값이 지원됩니다. iSeries용 DB2 UDB는 디폴트 레코드 대기 시간(60초)과 디폴트 파일 대기 시간(*IMMED)으로 표, 보기 및 색인을 작성합니다. 이러한 잠금 대기 시간은 DML문에 사용됩니다. CL 명령인 CHGPF(실제 파일 변경), CHGLF(논리 파일 변경) 및 OVRDBF(데이터베이스 파일 대체)를 사용하여 이 값들을 변경할 수 있습니다.

모든 DDL문과 LOCK TABLE문에 사용되는 잠금 대기 시간은 작업 디폴트 대기 시간(DFTWAIT)입니다. 이 값은 CL 명령 작업 변경(CHGJOB) 또는 클래스 변경(CHGCLS)을 사용하여 변경할 수 있습니다.

하나의 대형 레코드 대기 시간이 지정되는 경우, 교착 상태 검출이 제공됩니다. 예를 들어 하나의 작업이 행 1에 대한 배타적 잠금을 가지며 또 다른 작업이 행 2에 대한 배타적 잠금을 가진다고 가정합니다. 첫 번째 작업이 행 2를 잠그려는 경우, 이는 두 번째 작업이 잠금을 보류중이므로 대기합니다. 두 번째 작업이 행 1을 잠그려 할 경우, iSeries용 DB2 UDB는 두 개의 작업이 교착 상태에 있음을 발견하고 두 번째 작업으로 오류를 리턴시킵니다.

SQL LOCK TABLE문을 사용하여 다른 사용자가 동시에 표를 사용하지 못하도록 명시적으로 방지할 수 있으며 SQL 참조서에 설명됩니다. COMMIT(*RR)을 사용하면 작업 단위 동안에 다른 사용자가 표를 사용하지 못하도록 할 수도 있습니다.

성능을 향상시키기 위해 iSeries용 DB2 UDB는 ODP(open data path)를 자주 열어 놓습니다(자세한 내용은 데이터베이스 성능 및 조회 최적화 정보를 참조하십시오). 이러한 성능 피처는 ODP에 의해 참조되는 표에 대한 잠금은 그대로 두지만 행에 대한 모든 잠금은 그대로 두지 않습니다. 표에 남아 있는 잠금으로 인해 또 다른 작업이 해당 표에 대한 조작을 수행할 수 없게 되는 경우도 있습니다. 그러나 대부분의 경우, iSeries용 DB2 UDB는 다른 작업들이 잠금 보류중이고 이벤트가 이러한 작업에 신호를 보냄을 발견합니다. 이 이벤트로 인해 iSeries용 DB2 UDB는 해당 표와 연관되고 현재 성능상의 이유로 인해 열려 있는 모든 ODP(및 표가 잠그는 릴리스)를 닫습니다. 잠금 대기 시간 종료는 이벤트가 신호되고 다른 작업이 ODP를 닫을 만큼 충분히 길어야 하며 그렇지 않으면 오류가 리턴됩니다.

LOCK TABLE문을 사용하여 표 잠금을 확보하지 않거나 COMMIT(*ALL)나 COMMIT(*RR)를 사용하지 않으면 한 작업에 의해 이미 읽혀진 자료가 다른 작업에 의해 바로 변경될 수 있습니다. 대개 SQL문이 실행될 때 읽히는 자료는 즉시 사용할 수 있습니다(예를 들면, FETCH에서). 그러나 다음과 같은 경우에 자료는 SQL문이 실행되기 전에 읽히게 되어 자료를 현재 사용하지 못할 수 있습니다(예를 들면, OPEN 동안).

- ALWCPYDTA(*OPTIMIZE)가 지정되고 최적자가 자료를 복사하는 것이 복사하지 않는 것보다 더 잘 수행되는지를 판별합니다.

- 일부 조회에서는 데이터베이스 관리자가 임시 결과 표를 작성할 것을 요구합니다. 임시 결과 표에 있는 자료는 커서가 열린 후 변경된 내용에는 영향을 주지 않습니다. 임시 결과 표는 다음 경우에 필요합니다.
  - ORDER BY절에 지정된 열에 대한 기억영역의 총 길이가 2000바이트를 초과하는 경우
  - ORDER BY 및 GROUP BY절이 다른 순서의 다른 열을 지정하는 경우
  - UNION 또는 DISTINCT절이 지정되는 경우
  - ORDER BY 또는 GROUP BY절 모두가 동일한 표에 없는 열을 지정하는 경우
  - JOINDFT 자료 정의 스펙(DDS) 키워드에 의해 다른 파일에 대해 정의되는 논리 파일을 결합하는 경우
  - 복수의 데이터베이스 파일 멤버에 기초한 논리 파일에서 GROUP BY를 결합하거나 지정하는 경우
  - 조회에 파일 중 적어도 하나가 GROUP BY절이 포함된 보기로서 결합이 들어 있는 경우
  - 조회에 GROUP BY절이 포함된 보기를 참조하는 GROUP BY절이 들어 있는 경우
- 기본 부속 조회는 조회가 열릴 때 평가됩니다.

저널링

iSeries용 DB2 UDB 저널 지원은 감사 추적 및 정방향과 역방향 회복을 제공합니다. 정방향 회복은 표의 구버전을 취하고 저널에 기록된 변경 내용을 표에 적용하는데 사용될 수 있습니다. 역방향 회복은 표에서 저널에 기록된 변경 내용을 제거하는데 사용될 수 있습니다.

SQL 스키마가 작성될 때 저널과 저널 리시버는 스키마에 작성됩니다. SQL이 저널 및 저널 리시버를 작성할 때 ASP절이 CREATE COLLECTION 또는 CREATE SCHEMA문에서 지정되면 그것들은 사용자 보조 기억장치 풀에서만 작성됩니다. 그러나 저널 리시버를 자체 ASP에 두면 성능이 향상될 수 있으므로 저널을 관리하는 사용자가 앞으로 모든 저널 리시버를 별도의 ASP에 작성하려할 수도 있습니다.

표가 스키마에 작성되는 경우, 자동으로 스키마(QSQJRN)에 작성된 저널 iSeries용 DB2 UDB에 저널됩니다. 비스킨마에서 작성된 표 역시 QSQJRN으로 명명된 저널이 라이브러리에 존재할 경우 저널링이 시작되게 합니다. 이 시점 이후 저널, 저널 리시버 및 저널에 대한 표의 저널링을 관리하기 위해 저널 기능을 사용하는 것은 사용자의 책임입니다. 예를 들어, 표가 스키마로 이동되면 저널링 상태에 대한 자동 변경이 발생하지 않습니다. 표가 복원되면 정상의 저널 규칙이 적용됩니다. 즉, 표가 저장시에 저널되었으면 복원시 동일한 저널에 저널되고 저장시에 저널되지 않았으면 복원시에도 저널되지 않습니다.

SQL 콜렉션에서 작성된 저널은 대개 모든 변경내용을 SQL 표에 기록하는데 사용되는 저널입니다. 그러나 시스템 저널 기능을 사용하여 SQL 표를 다른 저널에 저널할 수 있습니다. 이는 한 스키마의 표가 다른 스키마의 표에 대한 모 그룹일 경우 필요할 수도 있습니다. 이것은 iSeries용 DB2 UDB에서 갱신이나 삭제가 상위 표에 대해 이루어질 때 참조 제한조건 안의 상위 파일과 종속 파일이 같은 저널로 저널되도록 요구하기 때문입니다.

사용자는 저널 기능을 사용하여 어떤 표상에서나 저널링을 중단할 수 있으나 그렇게 하면 확약 제어하에서 어플리케이션 실행이 방해를 받습니다. 저널링이 NO ACTION, CASCADE, SET NULL 또는 SET DEFAULT의 삭제 규칙이 있는 참조 제한조건의 상위 표에서 중단될 경우, 모든 갱신과 삭제 조작이 방해를 받습니다. 그렇지 않고 사용자가 COMMIT(*NONE)를 지정할 경우에는 어플리케이션에서 계속 기능을 수행할 수 있습니다. 그러나 이 경우, 저널링 및 확약 제어가 제공하는 같은 레벨의 무결성은 제공하지 않습니다.

저널링에 대한 자세한 정보는 저널링 주제를 참조하십시오.

확약 제어

iSeries용 DB2 UDB 확약 제어 지원은 데이터베이스 변경 그룹(갱신, 삽입, DDL 조작 또는 삭제)을 단일 작업 단위(트랜잭션)로 처리하는 수단을 제공합니다. 확약 조작은 조작 그룹의 완료를 보장합니다. 롤백 조작은 조작 그룹의 복구를 보장합니다. 저장점을 사용하여 트랜잭션을 롤백할 수 있는 더 작은 단위로 나눌 수 있습니다. 확약 조작은 몇 가지 다른 인터페이스를 통해 발행될 수 있습니다. 예를 들면 다음과 같습니다.

- SQL COMMIT문
- CL COMMIT 명령
- 언어 확약문(RPG COMMIT문과 같은)

롤백 조작은 몇 가지 다른 인터페이스를 통해 발행될 수 있습니다. 예를 들면 다음과 같습니다.

- SQL ROLLBACK문
- A CL ROLLBACK 명령
- 언어 롤백문(RPG ROLBK문과 같은)

확약이나 롤백될 수 없는 SQL문들은 다음과 같습니다.

- DROP COLLECTION
- 권한 보유자가 지정된 오브젝트에 존재하는 경우 GRANT 또는 REVOKE

확약 제어가 이미 SQL문이 COMMIT(*NONE) 또는 RELEASE문이 아닌 분리 레벨로 수행될 때 시작되지 않았다면 iSeries용 DB2 UDB가 명시적으로 CL 명령 확약 제어 시작(STRCMTCTL) 명령을 호출하여 확약 제어 환경을 설정합니다. iSeries용 DB2

UDB는 NFYOBJ(*NONE), CMTSCOPE(*ACTGRP) 매개변수를 LCKLVL과 함께 STRCMTCTL 명령에 지정합니다. 지정된 LCKLVL은 CRTSQLxxx, STRSQL 또는 RUNSQLSTM 명령의 COMMIT 매개변수에서 잠금 레벨입니다. REXX에서, 지정된 LCKLVL은 SET OPTION문에서 잠금 레벨입니다. 다른 CMTSCOPE, NFYOBJ 또는 LCKLVL을 지정하기 위해서는 STRCMTCTL 명령을 사용할 수 있습니다. CMTSCOPE(*JOB)을 지정하여 작업 레벨 확약 정의를 시작하는 경우, iSeries용 DB2 UDB는 활성 그룹 내 프로그램에 대한 작업 레벨 확약 정의를 사용합니다.

주:

1. 확약 제어 사용시 자료 조작어 명령문에 의해 어플리케이션 프로그램에서 참조된 표는 저널되어야 합니다.
2. 지정된 LCKLVL은 단지 디폴트 잠금 레벨임에 유의하십시오. 확약 제어가 시작되면 SET TRANSACTION SQL문 및 CRTSQLxxx, STRSQL 또는 RUNSQLSTM 명령의 COMMIT 매개변수에 지정된 잠금 레벨이 디폴트 잠금 레벨을 대체할 것입니다.

확약 제어하에서 수행중인 GROUP BY 또는 HAVING 열 함수를 사용하는 커서의 경우, ROLLBACK HOLD는 커서의 위치에 영향을 주지 않습니다. 추가로 다음 커서는 확약 제어하에서 발생합니다.

- COMMIT(*CHG) 및 (ALWBLK(*NO) 또는 (ALWBLK(*READ))가 커서 중 하나에 대해 지정될 경우, COMMIT(*CHG)가 요구되었으나 허용되지 않음을 알리는 메세지(CPI430B)가 송신됩니다.
- COMMIT(*ALL), COMMIT(*RR) 또는 COMMIT(*CS)가 KEEP LOCKS절과 함께 커서 중 하나에 대해 지정될 경우, iSeries용 DB2 UDB는 공유 모드로 참조된 모든 표를 잠급니다(*SHRNUP). 잠금은 어플리케이션의 동시 처리가 명명된 표에서 읽기 전용 조작을 제외한 모든 조작을 수행하지 못하도록 합니다. KEEP LOCKS절과 함께 COMMIT(*ALL), COMMIT(*RR) 또는 COMMIT(*CS)가 요청되었으나 허용되지 않은 커서 중 하나에 대해 지정되었음을 알리는 메세지(SQL7902 또는 CPI430A)가 송신됩니다. 메세지 SQL0595 역시 송신됩니다.

COMMIT(*ALL), COMMIT(*RR) 또는 COMMIT(*CS)가 KEEP LOCKS절과 함께 지정되고 카탈로그 파일이 사용되었거나 임시 결과 표가 요청된 커서의 경우, iSeries용 DB2 UDB는 공유 모드에서 모든 참조된 표를 잠급니다(*SHRNUP). 이는 동시 프로세스가 표에 대한 읽기 전용을 제외한 모든 조작을 수행할 수 없도록 합니다. COMMIT(*ALL)이 요청되었으나 허용되지 않았음을 나타내는 메세지(SQL7902 또는 CPI430A)가 송신됩니다. 메세지 SQL0595 역시 송신됩니다.

프로그램이 사전컴파일되었을 때 ALWBLK(*ALLREAD) 및 COMMIT(*CHG)가 지정되면 모든 읽기 전용 커서는 행의 블록화를 허용하며 ROLLBACK HOLD만이 커서 위치를 롤백시키지 않습니다.

COMMIT(*RR)가 요구되면 조회가 닫힐 때까지 표가 잠깁니다. 커서가 읽기 전용이면 표가 잠깁니다(*SHRNUP). 커서가 갱신 모드에 있으면 표가 잠깁니다(*EXCLRD). 다른 사용자가 표를 잠그면 반복 가능 읽기의 실행으로 인해 표를 동시에 액세스할 수 없습니다.

COMMIT(*NONE) 이외의 개별 레벨이 지정되고 어플리케이션이 ROLLBACK을 발행하거나 활성 그룹이 정상적으로 종료될 경우(및 확약 정의가 *JOB이 아닐 경우), 작업 단위 내에서 이루어진 모든 갱신, 삽입, 삭제 및 DDL 조작이 복구됩니다. 어플리케이션이 COMMIT를 발행하거나 활성 그룹이 정상 종료될 경우, 작업 단위 내에서 이루어진 모든 갱신, 삽입, 삭제 및 DDL 조작이 확약됩니다.

iSeries용 DB2 UDB는 행에 대한 잠금을 사용하여 작업 단위 완료 전에 변경된 자료를 다른 작업이 액세스하지 못하도록 합니다. COMMIT(*ALL)가 지정되면 페치된 행에 대한 읽기 잠금도 작업 단위가 완료되기 전에 읽힌 자료를 다른 작업이 변경하는 것을 방지하는데 사용됩니다. 이로 인해 다른 작업이 변경되지 않은 행을 읽는 것이 방해받지는 않습니다. 이는 동일한 작업 단위로 행을 다시 읽을 경우, 같은 결과를 얻게 합니다. 읽기 잠금으로 인해 다른 작업이 동일한 행을 페치하는 것이 방해받지는 않습니다.

확약 제어는 하나의 작업 단위에서 최대 4백만번의 고유한 행 변경을 처리합니다. COMMIT(*ALL) 또는 COMMIT(*RR)가 지정되면 읽힌 모든 행도 32,768 범위에 포함됩니다(행이 작업 단위에서 두 번 이상 변경되거나 읽히면, 그것은 32,768 범위에 한 번만 계산됩니다). 다수의 잠금을 보유하면 시스템 성능에 부정적인 영향을 미치며 작업 단위의 종료까지 동시 사용자가 작업 단위에서 잠긴 행에 액세스할 수 없게 됩니다. 작업 단위에서 처리되는 행의 수를 적게 유지하는 것이 사용자의 최대 관심사입니다.

확약 제어는 확약 제어하에서 열리거나 작업 단위에서 보류중인 변경과 함께 닫힌 파일을 각 저널에 대해 512개까지 허용합니다.

COMMIT HOLD 및 ROLLBACK HOLD는 커서를 항상 열린 상태로 유지하며 OPEN을 다시 발행하지 않고 다른 작업 단위를 시작하도록 허용합니다. iSeries 시스템에 없는 리모트 데이터베이스에 연결되어 있는 경우, HOLD 값을 사용할 수 없습니다. 그러나 DECLARE CURSOR에서 WITH HOLD 옵션은 COMMIT 이후에 커서를 열린 상태로 유지하는데 사용될 수 있습니다. 이 커서 유형은 iSeries 시스템에 없는 리모트 데이터베이스에 연결되어 있는 경우에 지원됩니다. 그러한 커서는 롤백에서 닫힙니다.

표 37. 레코드 잠금 기간

SQL문	**COMMIT 매개변수(주 6 참조)**	**행 잠금 기간**	**잠금 유형**
SELECT INTO	*NONE	잠금 없음	
SET 변수	*CHG	잠금 없음	
VALUES INTO	*CS(주 8 참조)	읽기 및 해제시 잠긴 행	READ
	*ALL(주 2 참조)	읽기부터 ROLLBACK 또는 COMMIT까지	READ

표 37. 레코드 잠금 기간 (계속)

SQL문	COMMIT 매개변수(주 6 참조)	행 잠금 기간	잠금 유형
FETCH(읽기 전용 커서)	*NONE	잠금 없음	
	*CHG	잠금 없음	
	*CS(주 8 참조)	읽기부터 다음 FETCH까지	READ
	*ALL(주 2 참조)	읽기부터 ROLLBACK 또는 COMMIT까지	READ
FETCH(갱신 또는 삭제 가능한 커서)(주 1 참조)	*NONE	행이 갱신되거나 식제되지 않는 경우	UPDATE
		읽기에서 다음 FETCH까지 갱신되거나 삭제되지 않은 경우	
		행이 갱신되거나 삭제되는 경우	
		읽기에서 UPDATE 또는 DELETE까지 갱신 또는 삭제된 경우	
	*CHG	행이 갱신되거나 식제되지 않는 경우	UPDATE
		읽기에서 다음 FETCH까지 갱신되거나 삭제되지 않은 경우	
		행이 갱신되거나 삭제되는 경우	
		읽기에서 COMMIT 또는 ROLLBACK까지 갱신 또는 삭제된 경우	
	*CS	행이 갱신되거나 식제되지 않는 경우	UPDATE
		읽기에서 다음 FETCH까지 갱신되거나 삭제되지 않은 경우	
		행이 갱신되거나 삭제되는 경우	
		읽기에서 COMMIT 또는 ROLLBACK까지 갱신 또는 삭제된 경우	
	*ALL	읽기부터 ROLLBACK 또는 COMMIT까지	UPDATE[3]
INSERT(목표 표)	*NONE	잠금 없음	
	*CHG	삽입에서 ROLLBACK 또는 COMMIT까지	UPDATE
	*CS	삽입에서 ROLLBACK 또는 COMMIT까지	UPDATE
	*ALL	삽입에서 ROLLBACK 또는 COMMIT까지	UPDATE[4]
INSERT(부속 선택 내의 표)	*NONE	잠금 없음	
	*CHG	잠금 없음	
	*CS	각 행을 읽는 동안 잠김	READ
	*ALL	읽기부터 ROLLBACK 또는 COMMIT까지	READ
UPDATE(비커서)	*NONE	각 행을 갱신하는 동안 잠김	UPDATE
	*CHG	읽기부터 ROLLBACK 또는 COMMIT까지	UPDATE
	*CS	읽기부터 ROLLBACK 또는 COMMIT까지	UPDATE
	*ALL	읽기부터 ROLLBACK 또는 COMMIT까지	UPDATE
DELETE(비커서)	*NONE	각 행을 삭제하는 동안 잠김	UPDATE
	*CHG	읽기부터 ROLLBACK 또는 COMMIT까지	UPDATE
	*CS	읽기부터 ROLLBACK 또는 COMMIT까지	UPDATE
	*ALL	읽기부터 ROLLBACK 또는 COMMIT까지	UPDATE
UPDATE(커서로)	*NONE	잠금은 행 갱신시 해제됨	UPDATE
	*CHG	읽기부터 ROLLBACK 또는 COMMIT까지	UPDATE
	*CS	읽기부터 ROLLBACK 또는 COMMIT까지	UPDATE
	*ALL	읽기부터 ROLLBACK 또는 COMMIT까지	UPDATE
DELETE(커서로)	*NONE	잠금은 행 삭제시 해제됨	UPDATE
	*CHG	읽기부터 ROLLBACK 또는 COMMIT까지	UPDATE
	*CS	읽기부터 ROLLBACK 또는 COMMIT까지	UPDATE
	*ALL	읽기부터 ROLLBACK 또는 COMMIT까지	UPDATE
부속 조회(갱신 또는 삭제 가능한 커서 또는 UPDATE 또는 DELETE 비커서)	*NONE	읽기에서 다음 FETCH까지	READ
	*CHG	읽기에서 다음 FETCH까지	READ
	*CS	읽기에서 다음 FETCH까지	READ
	*ALL(주 2 참조)	읽기부터 ROLLBACK 또는 COMMIT까지	READ

표 37. 레코드 잠금 기간 (계속)

SQL문	COMMIT 매개변수(주 6 참조)	행 잠금 기간	잠금 유형
부속 조회(읽기 전용 커서 또는 SELECT INTO)	*NONE *CHG *CS *ALL	잠금 없음 잠금 없음 각 행을 읽는 동안 잠김 읽기부터 ROLLBACK 또는 COMMIT까지	 READ READ

주:

1. 결과 표가 읽기 전용이 아니고(SQL 참조서에서 DECLARE CURSOR 설명 참조) 다음 중 하나에 해당하는 경우 커서는 UPDATE 또는 DELETE 기능으로 열립니다.
   - 커서가 FOR UPDATE절로 정의됩니다.
   - 커서가 FOR UPDATE, FOR READ ONLY 또는 ORDER BY절 없이 정의되면 프로그램에는 다음 중 최소한 하나가 포함됩니다.
     - 동일한 커서명을 참조하는 커서 UPDATE
     - 동일한 커서명을 참조하는 커서 DELETE
     - CRTSQLxxx 명령에 지정된 EXECUTE 또는 EXECUTE IMMEDIATE문과 ALWBLK(*READ) 또는 ALWBLK(*NONE)
2. 표나 보기는 COMMIT(*ALL)를 만족시키기 위해 배타적으로 잠길 수 있습니다. UNION을 포함한 부속 선택이 처리되거나 조회 처리가 임시 결과 사용을 필요로 할 경우, 확약되지 않은 변경을 볼 수 없도록 배타적 잠금이 확보됩니다.
3. 행이 갱신 또는 삭제되지 않으면 잠금은 *READ로 축소됩니다.
4. UPDATE는 목표 표의 행을 잠그고 READ는 부속 선택 표의 행을 잠급니다.
5. 표나 보기는 반복 읽기 기능을 만족시키기 위해 배타적으로 잠길 수 있습니다. 행 잠금은 반복 읽기 기능에서도 계속 수행됩니다. 잠금이 지정되며 지속 기간은 *ALL과 동일합니다.
6. 반복 가능 읽기(*RR) 행 잠금은 *ALL에 대해 표시된 잠금과 동일합니다.
7. 분리 레벨 및 잠금에 대한 자세한 설명은 SQL 참조서의 분리 레벨을 참조하십시오.
8. KEEP LOCKS절이 *CS로 지정되는 경우, 모든 읽기 잠금은 커서가 닫히거나 COMMIT 또는 ROLLBACK이 수행될 때까지 보류됩니다. 분리 절과 연관된 커서가 없는 경우, 잠금은 SQL문이 완료되기까지 보류됩니다.

자세한 내용은 확약 제어 주를 참조하십시오.

저장점

저장점은 트랜잭션 내의 이정표를 작성하게 합니다. 트랜잭션이 롤백되면 트랜잭션의 맨 처음이 아닌 지정된 저장점까지 변경된 내용을 원래대로 되돌립니다. 저장점은 SAVEPOINT SQL문을 사용하여 설정됩니다. 예를 들어, STOP_HERE 저장점을 작성합니다.

```
SAVEPOINT STOP_HERE
ON ROLLBACK RETAIN CURSORS
```

어플리케이션의 프로그램 논리에서 어플리케이션이 진행됨에 따라 저장점 이름을 다시 사용할 것인지 또는 다시 사용되어서는 안되는 어플리케이션의 고유한 이정표를 저장점이 의미하고 있지 않은지 등을 나타냅니다.

저장점이 또다른 SAVEPOINT문을 사용하여 이동되지 않아야 하는 고유 이정표를 표시하는 경우, UNIQUE 키워드를 지정하십시오. 이것은 SAVEPOIN문에 동일한 저장점 이름을 사용하는 저장된 프로시듀어를 호출하여 우연하게 해당 이름을 다시 사용하는 일이 발생하지 않도록 방지합니다. 그러나 SAVEPOINT문이 루프에 사용되면, UNIQUE 키워드를 사용하지 않아야 합니다. 다음 SQL문은 START_OVER 고유 저장점을 설정합니다.

```
SAVEPOINT START_OVER UNIQUE
       ON ROLLBACK RETAIN CURSORS
```

저장점을 롤백하려면, TO SAVEPOINT절이 있는 ROLLBACK문을 사용하십시오. 다음 예에서는 SAVEPOINT 및 ROLLBACK TO SAVEPOINT문 사용을 설명합니다.

이 어플리케이션 논리에서는 원하는 날짜에 비행기 예약을 하고 호텔을 예약합니다.호텔을 사용할 수 없으면 비행기 예약을 롤백한 후 다른 날짜에 프로세스를 반복합니다. 최대 세번까지 시도됩니다.

```
got_reservations =0;
EXEC SQL SAVEPOINT START_OVER UNIQUE ON ROLLBACK RETAIN CURSORS;

if (SQLCODE != 0) return;

for (i=0; i<3 & got_reservations == 0; ++i)
{
  Book_Air(dates(i), ok);
  if (ok)
  {
    Book_Hotel(dates(i), ok);
    if (ok) got_reservations = 1;
  else
    {
      EXEC SQL ROLLBACK TO SAVEPOINT START_OVER;
      if (SQLCODE != 0) return;
    }
  }
}

EXEC SQL RELEASE SAVEPOINT START_OVER;
```

저장점은 RELEASE SAVEPOINT문을 사용하여 해제됩니다. RELEASE SAVEPOINT문을 사용하여 명시적으로 저장점을 해제하지 않은 경우 현재의 저장점 레벨 끝에서 해제되거나 트랜잭션 끝에서 해제됩니다. 다음 명령문은 START_OVER 저장점을 해제합니다.

```
RELEASE SAVEPOINT START_OVER
```

트랜잭션이 확약되거나 롤백될 때 저장점이 해제됩니다. 일단 저장점 이름이 해제되면 저장점 이름까지의 롤백은 더 이상 불가능합니다. COMMIT 또는 ROLLBACK문은 트랜잭션 내에 설정된 모든 저장점 이름을 해제합니다. 모든 저장점 이름이 트랜잭션 내에서 해제되었으므로, 모든 저장점 이름이 다음 확약 또는 롤백을 재사용할 수 있습니다.

저장점 레벨

단일 명령문은 사용자 정의 기능, 트리거 또는 저장된 프로시듀어 등을 암시적으로 또는 명시적으로 호출할 수 있습니다. 이것은 네스팅으로 알려져 있습니다. 일부 경우에, 새로운 네스팅 레벨이 시작되면 새로운 저장점 레벨도 시작됩니다. 새로운 저장점 레벨은 더 낮은 레벨 루틴이나 트리거로 저장점 활동으로부터 어플리케이션 호출을 분리합니다.

저장점은 저장점이 정의된 동일한 저장점 레벨(또는 범위) 내에서만 참조될 수 있습니다. A ROLLBACK TO SAVEPOINT 문은 현재 저장점 레벨 외부에서 설정된 저장점으로 롤백하는 데 사용할 수 없습니다. 마찬가지로 RELEASE SAVEPOINT 문도 현재 저장점 레벨 외부에서 설정된 저장점을 해제하는 데 사용할 수 없습니다. 다음 표는 저장점 레벨이 시작되고 종료되는 시기를 요약한 것입니다.

새로운 저장점 레베은 다음 경우에 시작됩니다.	해당 저장점 레벨은 다음 경우에 종료됩니다.
작업 단위가 시작됩니다.	COMMIT 또는 ROLLBACK이 발행됩니다.
트리거가 호출됩니다.	트리거가 완료합니다.
사용자 정의 기능이 호출됩니다.	사용자 정의 기능을 요청자로 리턴합니다.
저장 프로시듀어가 호출되고, 해당 저장 프로시듀어가 NEW SAVEPOINT LEVEL절을 사용하여 작성됩니다.	저장 프로시듀어를 호출자로 리턴합니다.
ATOMIC 복합 SQL문에 대하여 BEGIN이 있습니다.	ATOMIC 복합문에 대하여 END가 있습니다.

저장점 레벨에 설정된 저장점은 그 저장점이 종료할 때 암시적으로 해제됩니다.

분산 데이터베이스 환경 내의 저장점에 대한 고려사항

저장점의 범위는 단일 연결에만 국한됩니다. 일단 저장점이 설정되면 어플리케이션이 연결되어 있는 모든 리모트 데이터베이스로 분배되지 않습니다. 저장점은 저장점이 설정될 때 어플리케이션이 연결되어 있던 현재의 데이터베이스에만 적용됩니다.

아토믹 조작

COMMIT(*CHG), COMMIT(*CS) 또는 COMMIT(*ALL)하에서 수행할 때 모든 조작이 아토믹으로 이루어지도록 보장됩니다. 즉, 완료되거나 시작되지 않은 것처럼 보입니다. 이는 기능이 종료되거나 인터럽트되는(전원 중단, 비정상적인 작업 종료 또는 작업 취소 등) 시기와 방법에 관계없이 해당되는 것입니다.

그러나 COMMIT(*NONE)가 지정될 경우, 일부 기초 데이터베이스 자료 정의 기능은 아토믹이 아닙니다. 다음 SQL 자료 정의문은 아토믹으로 보장되는 것입니다.

ALTER TABLE(주 1 참조)

COMMENT ON(주 2 참조)

LABEL ON(주 2 참조)

GRANT(주 3 참조)

REVOKE(주 3 참조)

DROP TABLE(주 4 참조)

DROP VIEW(주 4 참조)

DROP INDEX

DROP PACKAGE

주:

1. 열 정의 변경뿐만 아니라 제한조건이 추가되거나 제거되어야 할 경우에는 조작들이 한번에 하나씩 처리되므로 전체 SQL문은 아토믹이 아닙니다. 조작 순서는 다음과 같습니다.
   - 제한조거 제거
   - RESTRICT 옵션이 지정된 열들을 드롭
   - 기타 모든 열 정의 변경사항(DROP COLUMN CASCADE, ALTER COLUMN, ADD COLUMN)
   - 제한조건의 추가
2. COMMENT ON 또는 LABEL ON문에 대해 복수 열이 지정될 경우, 열은 한번에 하나씩 처리되므로 전체 SQL문이 아토믹은 아니지만 각각의 개별 열이나 오브젝트에 대한 COMMENT ON 또는 LABEL ON은 아토믹입니다.
3. 여러 개의 표, SQL 패키지 또는 사용자가 GRANT 또는 REVOKE문에 지정되면 표는 한번에 하나씩 처리됩니다. 따라서 전체 SQL문은 아토믹이 아니지만 각 개별 표에 대한 GRANT 또는 REVOKE문은 아토믹입니다.
4. DROP TABLE 또는 DROP VIEW문을 수행하는 동안 종속 보기가 제거되어야 하는 경우, 각 종속 보기는 한번에 하나씩 처리되므로 전체 SQL문은 아토믹이 아닙니다.

다음의 자료 정의 명령문은 둘 이상의 iSeries용 DB2 UDB 데이터베이스 조작을 포함하므로 아토믹이 아닙니다.

CREATE ALIAS

CREATE DISTINCT TYPE

CREATE FUNCTION

CREATE INDEX

CREATE PROCEDURE

CREATE SCHEMA

CREATE TABLE

CREATE TRIGGER

CREATE VIEW

DROP ALIAS

DROP DISTINCT TYPE

DROP FUNCTION

DROP PROCEDURE

DROP SCHEMA

DROP TRIGGER

RENAME(주 1 참조)

주:

1. RENAME은 이름 또는 시스템 이름이 변경된 경우에만 아토믹입니다. 둘 모두 변경된 경우 RENAME은 아토믹이 아닙니다.

예를 들어 CREATE TABLE은 iSeries용 DB2 UDB 실제 파일(PF) 작성 후 그러나 멤버 추가 전에 인터럽트될 수 있습니다. 그러므로 CREATE문의 경우, 조작이 이상 종료되면 오브젝트를 제거한 후 다시 작성해야 하는 수도 있습니다. DROP COLLECTION 문의 경우, 콜렉션을 다시 제거하거나 CL 명령 라이브러리 삭제(DLTLIB)를 사용하여 콜렉션의 나머지 부분을 제거할 수 있습니다.

제한조건

iSeries용 DB2 UDB는 고유, 참조 제한조건 및 체크 제한조건을 지원합니다. 고유 제한조건은 키의 값이 고유하도록 보장하는 규칙입니다. 참조 제한조건은 종속 표 내 외부 키의 모든 널이 아닌 값이 상위 표에서 맵핑하는 상위 키를 갖도록 보장하는 규칙입니다. 체크 제한조건은 하나의 열 또는 열들의 그룹에서 허용되는 값을 제한하는 규칙입니다.

iSeries용 DB2 UDB는 DML(자료 조작 언어) 명령문 중에 제한조건의 유효성을 강제 시행합니다. 그러나 어떤 조작(종속 표의 복원과 같은)은 제한조건의 유효성이 알려지지 않도록 합니다. 이 경우 DML 명령문은 iSeries용 DB2 UDB가 제한조건의 유효성을 확인할 때까지 방해받을 수 있습니다.

- 고유 제한조건은 색인과 함께 수행됩니다. 고유 제한조건을 수행하는 색인이 유효하지 않을 경우, 현재 리빌드를 요구하는 색인을 표시하기 위해 EDTRBDAP(액세스 경로의 리빌드 편집) 명령이 사용될 수 있습니다.
- iSeries용 DB2 UDB가 현재 참조 제한조건 또는 체크 제한조건이 유효한지 여부를 알지 못하는 경우, 이 제한조건은 체크 지연중 상태로 간주됩니다. 현재 리빌드를 요구하는 색인을 표시하기 위해서는 EDTCPCST(체크 보류 제한조건 편집) 명령이 사용될 수 있습니다.

제한조건에 대한 자세한 정보는 151 페이지의 제 10 장『자료 무결성』및 데이터베이스 프로그래밍을 참조하십시오.

저장/복원

OS/400 저장/복원 기능은 표, 보기, 색인, 저널, 저널 리시버, SQL 패키지, SQL 프로시듀어, SQL 트리거, 사용자 정의 기능, 사용자 정의 유형 및 콜렉션을 디스크(저장 파일) 또는 외부 매체(테이프 또는 디스켓)에 저장하는데 사용됩니다. 저장된 버전은 추후에 모든 iSeries 시스템에 복원될 수 있습니다. 저장/복원 기능은 전체 콜렉션, 선택된 오브젝트 또는 저장 예정 날짜와 시간 이후에 변경된 오브젝트만을 허용합니다. 이전 상태로 오브젝트를 복원하는데 필요한 모든 정보가 저장됩니다. 이 기능은 표나 전체 콜렉션의 이전 버전으로 자료를 복원하여 개별표 손상을 회복하는데 사용될 수 있습니다.

SQL 프로시듀어용으로 작성된 프로그램이나 SQL 함수 또는 소스 함수용으로 작성된 서비스 프로그램이 복원되는 경우, 이는 이미 같은 서명의 프로시듀어나 함수가 존재하지 않는 한 SYSROUTINES와 SYSPARMS 카탈로그에 자동으로 추가됩니다. QSYS에 작성된 SQL 프로그램은 복원시 SQL 프로시듀어로서 작성되지 않습니다. 또한, CREATE PROCEDURE 또는 CREATE FUNCTION문에 언급된 외부 프로그램이나 서비스 프로그램은 SYSROUTINES의 루틴을 등록하는데 필요한 정보를 포함하고 있을 수 있습니다. 정보가 존재하고 그 서명이 고유할 경우, 함수 또는 프로시듀어는 복원될 때 SYSROUTINES 및 SYSPARMS에도 추가됩니다.

SQL 표가 복원되면, 이 표에 대해 정의된 SQL 트리거도 복원됩니다. SQL 트리거 정의는 SYSTRIGGERS, SYSTRIGDEP, SYSTRIGCOL 및 SYSTRIGUPD 카탈로그에 자동으로 추가됩니다. SQL 표가 저장 및 복원되면 SQL CREATE TRIGGER문에서 작성된 프로그램 오브젝트도 저장 및 복원됩니다. 프로그램 오브젝트의 저장 및 복원은 데이터베이스 관리자에 의해 자동화되지 않습니다. SQL 표를 새로운 라이브러리에 복원할 때에는 자체 참조 트리거에 대한 예방책을 검토해야 합니다. SQL 참조서의 CREATE TRIGGER문 절에 대한 주의사항에서 작동되지 않는 트리거를 참조하십시오.

사용자 정의 유형에 대해 *SQLUDT 오브젝트가 복원될 경우, 사용자 유형 정의는 자동으로 SYSTYPES 카탈로그에 추가됩니다. 사용자 정의 유형과 소스 유형 사이에 캐스트하는데 필요한 적절한 함수들이 작성됩니다(유형 및 함수가 아직 존재하지 않을 경우).

분산 SQL 프로그램이나 그와 연관된 SQL 패키지를 여러 시스템에 저장하고 복원할 수 있습니다. 이로써 다른 시스템에 있는 SQL 프로그램의 모든 사본이 동일한 어플리케이션 서버에서 동일한 SQL 패키지에 액세스할 수 있습니다. 이로써 또한 단일 분산 SQL 프로그램이 복원된 SQL 패키지를 갖는(CRTSQLPKG도 사용될 수 있음) 모든 어플리케이션 서버에 연결될 수 있습니다. SQL 패키지는 다른 라이브러리로 복원될 수 없습니다.

경고: 스키마를 기존 라이브러리 또는 다른 이름이 있는 스키마로 복원하면 저널, 저널 리시버 또는 IDDU 사전(존재할 경우)을 복원할 수 없습니다. 스키마가 다른 이름을 가진 스키마로 복원될 경우, 이 스키마의 카탈로그 보기는 기존 스키마의 오브젝트만 반영합니다. 그러나 QSYS2의 카탈로그 보기는 모든 오브젝트를 적절하게 반영합니다.

손상 허용 한계

서버는 디스크 오류로 인한 손상을 감소시키거나 제거하기 위한 몇 가지 메카니즘을 제공합니다. 예를 들어 이중 복사, 체크섬 및 RAID 디스크는 모든 디스크 문제의 발생을 줄일 수 있습니다. iSeries용 DB2 UDB 기능은 디스크 오류 또는 시스템 오류로 인한 손상에 대한 일정 허용 한계치를 가집니다.

DROP 조작은 손상과는 관계없이 정상적으로 수행됩니다. 이는 손상이 발생할 경우, 최소한의 표, 보기, SQL 패키지, 색인, 프로시듀어, 기능 또는 고유한 유형이 삭제되고 복원되거나 다시 작성될 수 있음을 확실히 하는 것입니다.

디스크 오류로 표 내 행의 작은 부분이 손상된 경우에는 iSeries용 DB2 UDB 데이터베이스 관리자를 통해 아직 액세스가 가능한 행을 읽을 수 있습니다.

색인 회복

iSeries용 DB2 UDB는 색인 회복을 다루는 여러 기능을 제공합니다.

- 시스템 관리 색인 보호

  EDTRCYAP CL 명령으로 사용자는 iSeries용 DB2 UDB로 하여금 시스템 또는 전원 장애시 시스템상의 모든 색인을 회복하는데 필요한 시간의 양을 지정된 시간 이하로 유지하도록 보장할 것을 지시할 수 있습니다. 시스템은 지정된 회복 시간의 범위 내에서 자동으로 시스템 저널에 충분한 정보를 저널합니다.

- 색인 저널링

  iSeries용 DB2 UDB는 전원 또는 시스템 장애로 인해 색인 전체를 리빌드할 필요가 없도록 하는 색인 저널링 기능을 제공합니다. 색인이 저널되면 시스템 데이터베이스 지원은 자동으로 색인이 스크래치에서 리빌드되지 않고 표 내의 자료로 동기화되도록 합니다. SQL 색인은 자동으로 *저널되지 않습니다*. 그러나 사용자는 CL 명령 저널 액세스 경로 시작(STRJRNAP) 명령을 사용하여 iSeries용 DB2 UDB에서 작성한 색인을 저널할 수 있습니다.

- 색인 리빌드

  시스템에 있는 모든 색인은 색인이 유지보수될 때 지정하는 유지보수 옵션이 있습니다. SQL 색인은 *IMMED 유지보수 속성으로 작성됩니다.

  전원 공급 중단 또는 이상 시스템 장애의 경우, 변경 처리중인 색인이 실제 자료와 일치하는지 확인하기 위해 데이터베이스 관리자에 의해 재구성될 수도 있습니다. 시스템에 있는 모든 색인에는 필요한 경우 색인이 재구성될 때 지정하는 회복 옵션이

있습니다. UNIQUE 속성이 있는 모든 SQL 색인은 *IPL의 회복 속성으로 작성됩니다(이는 OS/400이 시작되기 전에 이 색인이 재구성됨을 의미합니다). 다른 모든 SQL 색인은 *AFTIPL 회복 옵션으로 작성됩니다(이는 오퍼레이팅 시스템이 시작된 후 색인이 비동기적으로 재구성됨을 의미합니다). IPL시 오퍼레이터는 재구성되어야 하는 색인과 그 회복 옵션을 표시하는 화면을 볼 수 있습니다. 오퍼레이터는 회복 옵션을 대체할 수 있습니다.

- 색인의 저장 및 복원

  저장/복원 기능으로 CL 명령 SAVOBJ(오브젝트 저장) 또는 SAVLIB(라이브러리 저장)에서 ACCPTH(*YES)를 사용하여 표를 저장할 때 색인을 저장할 수 있습니다. 색인이 이미 저장된 상태에서 복원하는 경우에는 색인을 리빌드할 필요가 없습니다. 이전에 저장 및 복원되지 않은 색인은 자동적으로 또한 비동기적으로 데이터베이스 관리자에 의해 재구성됩니다.

카탈로그 무결성

카탈로그에는 스키마에 있는 표, 보기, SQL 패키지, 색인, 프로시듀어, 기능, 트리거 및 매개변수에 대한 정보가 들어 있습니다. 데이터베이스 관리자는 카탈로그에 있는 정보가 항상 정확하도록 확인합니다. 이는 일반 사용자가 카탈로그에 있는 모든 정보를 명시적으로 변경하는 것을 방지하고 카탈로그에 설명된 표, 보기, SQL 패키지, 색인, 유형, 프로시듀어, 기능, 트리거 및 매개변수에 변경이 발생할 때 카탈로그 내의 정보를 내재적으로 유지보수함으로써 이루어집니다.

카탈로그의 무결성은 스키마의 오브젝트가 SQL문, OS/400 CL 명령, System/38 환경 CL 명령, System/36 환경 기능 또는 iSeries 시스템에 있는 다른 제품 또는 유틸리티에 의해 변경되어도 유지보수됩니다. 예를 들어, 표 삭제는 SQL DROP문을 실행하거나 OS/400 DLTF CL 명령을 발행하거나 System/38 DLTF CL 명령을 발행하거나 WRKF 또는 WRKOBJ 화면에 옵션 4를 입력하여 수행할 수 있습니다. 표를 삭제하는데 사용된 인터페이스와 상관없이 데이터베이스 관리자는 삭제가 수행될 때 카탈로그에서 표의 설명을 제거합니다. 다음은 카탈로그에 대한 기능과 이들이 카탈로그에 미치는 영향에 관한 리스트입니다.

표 38. 목록에 대한 여러 기능의 영향

기능	카탈로그에 대한 영향
표로 제한조건 추가	카탈로그에 정보가 추가됨
표로부터 제한조건 제거	카탈로그에서 관련 정보가 제거됨
스키마로 오브젝트 작성	카탈로그에 정보가 추가됨
스키마로부터 오브젝트 삭제	카탈로그에서 관련 정보가 제거됨
스키마로 오브젝트 복원	카탈로그에 정보가 추가됨
오브젝트의 긴 주석 변경	카탈로그에 주석이 갱신됨
오브젝트 레이블(텍스트) 변경	카탈로그에서 레이블이 갱신됨
오브젝트 소유자 변경	카탈로그에서 소유자가 갱신됨
스키마에서 오브젝트 이동	카탈로그에서 관련 정보가 제거됨

표 38. 목록에 대한 여러 기능의 영향 (계속)

기능	카탈로그에 대한 영향
스키마로 오브젝트 이동	카탈로그에 정보가 추가됨
오브젝트 이름 변경	카탈로그에서 오브젝트명이 갱신됨

사용자 보조 기억장치 풀(ASP)

CREATE COLLECTION 및 CREATE SCHEMA문에서 ASP절을 사용하여 스키마가 사용자 ASP에서 작성될 수 있습니다. 사용자 ASP에서 라이브러리를 작성하기 위해 CRTLIB 명령을 사용할 수도 있습니다. 그 라이브러리는 SQL 표, 보기 및 색인을 수신하는데 사용될 수 있습니다. 보조 기억장치 풀에 대한 자세한 정보는

백업 및 회복 을 참조하십시오.

독립 보조 기억장치 풀(IASP)

독립 디스크 풀은 iSeries 서버에서 사용자 데이터베이스를 설정하는 데 사용됩니다. 독립 디스크 풀의 세 가지 유형은 1차, 2차 및 사용자 정의 파일 시스템(UDFS)입니다. 데이터베이스는 독립 디스크 풀을 사용하여 설정됩니다.

iSeries 서버의 경우, 복수 데이터베이스에 대해 작업할 수 있습니다. iSeries 서버는 시스템 데이터베이스(SYSBAS) 및 하나 이상의 사용자 데이터베이스에 대하여 작업할 수 있는 기능을 제공합니다. 사용자 데이터베이스는 독립된 디스크 풀을 사용하여 iSeries 서버에서 구현되는데, 독립된 디스크 풀은 iSeries Navigator의 디스크 관리 기능에서 설정됩니다.일단 독립된 디스크 풀이 설정되면 iSeries Navigator의 데이터베이스 기능 아래에 또 다른 데이터베이스가 있는 것처럼 보입니다.

제 20 장 어플리케이션 프로그램에 있는 SQL문 테스트

이 장에서는 어플리케이션 프로그램에 있는 SQL문에 대한 테스트 환경을 설정하는 방법과 이 프로그램의 오류를 디버그하는 방법을 설명합니다.

자세한 내용은 다음의 절을 참조하십시오.

- 『테스트 환경 수립』
- 364 페이지의『SQL 어플리케이션 프로그램 테스트』

테스트 환경 수립

프로그램을 테스트하는데 필요한 내용은 다음과 같습니다.

- **권한부여.** 표와 보기를 작성하고 SQL 자료에 액세스하며 프로그램을 작성하고 수행하려면 권한이 있어야 합니다.
- **테스트 자료 구조.** 프로그램이 표와 보기의 자료를 갱신, 삽입 또는 삭제하는 경우, *프로그램 수행을 확인하기 위해서는 테스트 자료*를 사용해야 합니다. 프로그램이 표와 보기에서 자료를 검색만 할 경우, 프로그램을 테스트할 때 실행 레벨 자료를 사용하도록 고려할 수도 있습니다. 그러나 CL 명령인 디버그 시작(STRDBG)에 UPDPROD(*NO)를 사용하여 제품 레벨 자료가 부주의로 인하여 변경되지 않도록 확실히 하는 것이 바람직합니다. 디버그에 대한 자세한 정보는 CL 프로그래밍의 테스트에 대한 장을 참조하십시오.
- **테스트 입력 자료.** 테스트 중 프로그램에서 사용되는 입력 자료는 가능한 모든 입력 조건을 표시하는 유효한 자료여야 합니다. 유효한 입력 자료를 사용하지 않는 한 출력 자료가 유효하다고 확신할 수 없습니다.

  프로그램이 입력 자료의 유효성을 확인하는 경우, 유효한 자료는 처리되고 유효하지 않은 자료는 검출되는지를 알 수 있도록 유효한 자료와 유효하지 않은 자료 모두를 포함시키십시오.

  후속 테스트를 위해서는 자료를 화면정리해야 합니다.

프로그램을 철저히 테스트하려면 프로그램 전체의 모든 경로를 가능한 한 많이 테스트하십시오. 예를 들면 다음과 같습니다.

- 입력 자료를 사용하여 프로그램이 각 분기마다 수행되도록 하십시오.
- 결과를 체크하십시오. 예를 들어 프로그램이 행을 갱신하면 행이 올바르게 갱신되었는지 볼 수 있도록 행을 선택하십시오.
- 프로그램 오류 루틴을 반드시 테스트하십시오. 가능하면 프로그램이 예상된 많은 오류 조건을 통과하도록 입력 자료를 사용하십시오.

- 프로그램이 사용한 편집 및 유효성 검사 루틴을 테스트하십시오. 프로그램이 자료를 올바르게 편집하고 유효성 검사를 했는지 확인하기 위해 가능하면 많은 입력 자료의 조합을 프로그램으로 제공하십시오.

자세한 내용은 『테스트 자료 구조 설계』를 참조하십시오.

테스트 자료 구조 설계

SQL 자료에 액세스하는 어플리케이션을 테스트하려면 테스트 표 및 보기를 작성해야 합니다.

- **기존 표의 테스트 보기.** 어플리케이션이 자료를 변경하지 않고 자료가 하나 또는 그 이상의 제품 레벨 표에 존재하면, 기존 표의 보기를 사용하도록 고려할 수도 있습니다. 제품 레벨 자료가 예기치 않게 변경되지 않도록 하려면 UPDPROD(*NO)와 함께 STRDBG 명령을 사용하는 것이 바람직합니다. 디버그에 대한 자세한 정보는 CL 프로그래밍 의 테스트에 대한 장을 참조하십시오.

- **테스트 표.** 어플리케이션이 자료를 작성, 변경 또는 삭제할 때 테스트 자료가 들어 있는 표를 사용하여 어플리케이션을 테스트할 수 있습니다. 표와 보기 작성에 대한 설명은 제 2 장 『SQL 시작』을 참조하십시오.

  또한, CL 명령인 CRTDUPOBJ(복제 오브젝트 작성)를 사용하여 테스트 표, 보기 또는 색인을 작성할 수도 있습니다.

권한부여

표를 작성하기 전에 표를 작성하고 표가 상주할 스키마를 사용하기 위해서는 권한이 있어야 합니다. 또한 테스트하려는 프로그램을 작성하고 실행하는 권한이 있어야 합니다.

기존 표나 보기(직접적으로 또는 보기에 대한 기준으로서)를 사용하려면, 사용자에게 표나 보기에 액세스할 수 있는 권한이 있어야 합니다.

보기를 작성하려면, 사용자에게 보기를 작성할 수 있는 권한이 있어야 하고 보기가 기초한 각 표와 보기에 대한 권한이 있어야 합니다. 특정 SQL문에 필요한 특정 권한에 대한 자세한 내용은 SQL 참조서를 참조하십시오.

SQL 어플리케이션 프로그램 테스트

iSeries용 DB2 UDB SQL 어플리케이션을 테스트하는 두 단계가 있는데, 프로그램 디버그 단계와 성능 확인 단계입니다. 이러한 단계는 365 페이지의 『프로그램 디버그 단계』와 365 페이지의 『성능 검증 단계』입니다.

프로그램 디버그 단계

이 테스트 단계는 SQL 조회가 올바르게 지정되고 프로그램이 올바른 결과를 생성하는지를 확인하기 위해서 수행됩니다.

SQL문이 있는 프로그램의 오류를 수정하는 것은 SQL문이 없는 프로그램을 수정하는 것과 동일합니다. 그러나 SQL문이 디버그 모드의 작업에서 수행될 때 데이터베이스 관리자는 각 SQL문이 수행되는 방법에 관한 메세지를 작업 기록부에 기록합니다. 이 메세지는 SQL문에 대한 SQLCODE의 표시입니다. 명령문이 정상적으로 수행되면 SQLCODE 값은 0이 되며 완료 메세지가 발행됩니다. 음의 SQLCODE는 진단 메세지를 표시합니다. 양의 SQLCODE는 정보용 메세지를 표시합니다.

메세지는 접두어로 **SQL**이 붙은 4자리 코드 또는 **SQ**가 붙은 5자리 코드입니다. 예를 들어, SQLCODE -204의 결과 SQL0204 메세지가 나오고 SQLCODE 30000의 결과 SQ30000 메세지가 나옵니다.

SQLCODE와 연관된 것은 SQLSTATE입니다. SQLSTATE는 다른 IBM 관계형 데이터베이스 제품 중에 공통 오류 조건을 식별하는 SQLCA에 제공되는 추가의 리턴 코드입니다. 다른 관계형 데이터베이스 제품에서 같은 오류 조건이 같은 SQLSTATE를 생성합니다. 같은 오류 조건이 같은 SQLCODE를 생성하지는 않습니다. 이 리턴 코드는 특히 비iSeries용 DB2 UDB 시스템에서 수행되는 관계형 데이터베이스 조작에서 리턴된 오류의 원인을 판별할 때 유용합니다.

비ILE 프로그램 디버깅의 경우, 디버그 모드로 된 고급 언어문 번호에 대한 참조가 컴파일 리스팅으로부터 이루어져야 합니다. ILE 프로그램 디버깅의 경우,
DBGVIEW(*SOURCE)를 지정하는 프로그램을 사전컴파일한 후 소스 레벨 디버거를 사용하십시오.

SQL은 항상 디버그 모드에 있는지의 여부와 관계없이 음의 SQLCODE와 +100이 아닌 양의 값 코드에 대한 메세지를 작업 기록부에 기록합니다.

성능 검증 단계

이 테스트 단계는 해당 색인이 사용 가능한지와 데이터베이스 관리자가 원하는 응답 시간 내에 조회를 해결할 수 있는 방식으로 조회가 작성되는지를 확인합니다. SQL 어플리케이션의 성능은 액세스되는 표의 속성에 따릅니다. 소형 표를 사용하면 조회의 응답 시간은 색인의 사용 가능성에 의해 영향을 받지 않습니다. 그러나 동일한 조회를 대형 표가 있는 데이터베이스에서 수행하고 해당 색인이 존재하지 않는 경우에는 조회에 대한 응답 시간이 매우 길어집니다.

테스트 환경은 가능한 한 실행 환경과 유사해야 합니다. 테스트 스키마에는 제품 스키마와 이름 및 구성이 동일한 표가 있어야 합니다. 동일한 색인을 양쪽 스키마에 있는 표에서 사용할 수 있어야 합니다. 표 내 행의 수와 값의 분배가 유사해야 합니다.

성능을 검증하기 위해 사용할 수 있는 툴과 명령에 대한 설명은 데이터베이스 성능 및 조회 최적화 책을 참조하십시오.

제 21 장 분산 관계형 데이터베이스 기능

*분산 관계형 데이터베이스*는 상호연결된 컴퓨터 시스템에 흩어져 있는 SQL 오브젝트의 세트로 구성됩니다. 이러한 관계형 데이터베이스는 같은 유형(예를 들어, iSeries용 DB2 UDB) 또는 다른 유형(OS/390용 DB2 Universal Database, VSE 및 VM용 DB2, DB2 Universal Database(UDB) 또는 DRDA를 지원하는 타사 데이터베이스 관리 시스템)일 수 있습니다. 각 관계형 데이터베이스에는 그 환경에서 표를 관리하는 관계형 데이터베이스 관리자가 있습니다. 데이터베이스 관리는 다른 시스템의 관계형 데이터베이스에 있는 SQL문을 수행하기 위해 주어진 데이터베이스 관리자 액세스가 허용되는 방법으로 서로 통신하고 상호 작용합니다.

어플리케이션 리퀘스터는 연결 가운데 어플리케이션 측을 지원합니다. 어플리케이션 서버는 어플리케이션 리퀘스터가 연결되어 있는 로컬이나 리모트 데이터베이스입니다. iSeries용 DB2 UDB는 어플리케이션 리퀘스터가 어플리케이션 서버와 통신할 수 있게 하는 분산 관계형 데이터베이스 구조(DRDA)에 대한 지원을 제공합니다. 또한 iSeries용 DB2 UDB는 나감 프로그램을 호출하여 DRDA를 지원하지 않는 다른 데이터베이스 관리 시스템에 있는 자료에 액세스할 수 있도록 해줍니다. 이 나감 프로그램을 어플리케이션 리퀘스터 드라이버(ARD) 프로그램이라고 합니다.

iSeries용 DB2 UDB는 두 가지 레벨의 분산 관계형 데이터베이스를 지원합니다.

- 리모트 작업 단위(RUW)

  리모트 작업 단위에서는 하나의 작업 단위 동안 SQL 명령을 준비하거나 실행하는 것이 오직 하나의 어플리케이션 서버에서 발생합니다. iSeries용 DB2 UDB는 APPC 또는 TCP/IP상에서 RUW를 지원합니다.
- 분산된 작업 단위(DUW)

  분산된 작업 단위에서는 하나의 작업 단위 동안 SQL문의 준비나 실행이 여러 어플리케이션 서버에서 발생할 수 있습니다. 그러나, 단일 SQL문은 단일 어플리케이션 서버에 위치한 오브젝트만을 참조할 수 있습니다. iSeries용 DB2 UDB는 APPC를 통한 DUW를 지원하며 V5R1에서부터 TCP/IP를 통한 DUW에 대한 지원을 도입했습니다.

분산 관계형 데이터베이스에 대한 자세한 내용은 분산 데이터베이스 프로그래밍 책을 참조하십시오.

자세한 내용은 다음을 참조하십시오.

- 368 페이지의『iSeries용 DB2 UDB 분산 관계형 데이터베이스 지원』
- 369 페이지의『iSeries용 DB2 UDB 분산 관계형 데이터베이스 예 프로그램』
- 370 페이지의『SQL 패키지 지원』

- 374 페이지의 『SQL에 대한 CCSID 고려사항』
- 375 페이지의 『연결 관리와 활성 그룹』
- 381 페이지의 『분산 지원』
- 389 페이지의 『분산 작업 단위』
- 392 페이지의 『어플리케이션 리퀘스터 드라이버 프로그램』
- 393 페이지의 『문제점 처리』
- 394 페이지의 『DRDA 저장 프로시듀어 고려사항』

iSeries용 DB2 UDB 분산 관계형 데이터베이스 지원

DB2 UDB 조회 관리자 및 SQL 개발 킷 사용권 프로그램은 다음 SQL문으로 분산 데이터베이스에 대한 대화식 액세스를 지원합니다.

- CONNECT
- SET CONNECTION
- DISCONNECT
- RELEASE
- DROP PACKAGE
- GRANT PACKAGE
- REVOKE PACKAGE

이러한 명령문에 대한 자세한 설명은 SQL 참조서를 참조하십시오.

추가 지원은 SQL 사전컴파일러 명령의 매개변수를 통해 개발 키트로 제공됩니다.

CRTSQLCI(SQL ILE C 오브젝트 작성) 명령

CRTSQLCPPI(SQL ILE C++ 오브젝트 작성) 명령

CRTSQLCBL(SQL COBOL 프로그램 작성) 명령

CRTSQLCBLI(SQL ILE COBOL 오브젝트 작성) 명령

CRTSQLPLI(SQL PL/I 프로그램 작성) 명령

CRTSQLRPG(SQL RPG 프로그램 작성) 명령

CRTSQLRPGI(SQL ILE RPG 오브젝트 작성) 명령

SQL 사전컴파일러 명령에 대한 자세한 내용은 *호스트 언어를 사용한 SQL 프로그래밍* 정보에서 주제 SQL문을 사용한 프로그램 준비 및 실행을 참조하십시오. SQL 패키지 작성(CRTSQLPKG) 명령은 분산 프로그램으로 작성된 SQL 프로그램으로부터 SQL 패키지를 작성할 수 있도록 합니다. CRTSQLPKG 및 CRTSQLxxx 명령에 대한 구문과 매개변수 정의는 부록 B 『iSeries용 DB2 UDB CL 명령 설명』에 제공됩니다.

코드 예에 대한 정보는 x 페이지의 『코드 면책사항 정보』를 참조하십시오.

iSeries용 DB2 UDB 분산 관계형 데이터베이스 예 프로그램

SQL 제품에는 리모트 작업 단위의 관계형 데이터베이스 샘플 프로그램이 함께 제공됩니다. 또한 QSQL 라이브러리 내에 분산 iSeries용 DB2 UDB 샘플 프로그램을 수행할 환경 설정에 도움이 되는 파일과 멤버가 여러 개 있습니다.

이들 파일과 멤버를 사용하려면 파일 QSQL/QSQSAMP에 있는 SETUP 일괄처리 작업을 수행해야 합니다. SETUP 일괄처리 작업은 다음을 수행하기 위해 예를 사용자 정의할 수 있도록 합니다.

- 리모트와 로컬 위치에서 QSQSAMP 라이브러리 작성
- 리모트와 로컬 위치에서 관계형 데이터베이스 디렉토리 항목 설정
- 로컬 위치에서 어플리케이션 패널 작성
- 분산 샘플 어플리케이션 스키마, 표, 색인 및 보기를 작성하기 위한 프로그램 사전컴파일, 컴파일 및 실행
- 로컬 및 리모트 위치의 표로 자료 로드
- 프로그램 사전컴파일 및 컴파일
- 리모트 위치의 어플리케이션 프로그램에 대한 SQL 패키지 작성
- 부서 표의 위치 열을 갱신하기 위해 프로그램을 사전컴파일, 컴파일, 실행

SETUP 수행 전에 QSQL/QSQSAMP 파일의 SETUP 멤버를 편집할 필요가 있을 수 있습니다. 명령어는 주석으로서 멤버에 포함이 됩니다. SETUP을 실행하려면, 시스템 명령행에 다음의 명령을 지정하십시오.

```
========> SBMDBJOB QSQL/QSQSAMP SETUP
```

일괄처리 작업이 끝날 때까지 기다리십시오.

샘플 프로그램을 사용하려면 명령 행에 다음 명령을 지정하십시오.

```
========> ADDLIBLE QSQSAMP
```

샘플 프로그램의 사용자 정의 가능한 첫 번째 화면을 호출하려면 명령 행에 다음 명령을 지정하십시오.

```
========> CALL QSQ8HC3
```

다음 화면이 표시됩니다. 이 화면에서 데이터베이스 샘플 프로그램을 사용자 정의할 수 있습니다.

```
DB2 for OS/400 조직 어플리케이션


조치.............:   _     A  (추가)               E (삭제)
D  (표시)             U (갱신)

오브젝트.........:   __    DE (부서)               EM (관리자명)
DS (부서 구조)

탐색 기준........:   __    DI (부서 ID)          MN (관리자명)
DN (부서명)           EI (사원 ID)
MI (관리자 ID)        EN (사원명)

위치.............:   _____________  (공백은 로컬임)

자료.............:   ____________________________


                                                              맨 아래

F3=나감
                                   (C) COPYRIGHT IBM CORP. 1982, 1991
```

SQL 패키지 지원

OS/400 프로그램은 SQL 패키지라고 불리는 오브젝트를 지원합니다(OS/400 오브젝트 유형은 *SQLPKG입니다). SQL 패키지에는 분산 프로그램을 실행시킬 때 어플리케이션 서버에서 SQL문을 실행시키기 위한 제어 구조와 액세스 계획이 있습니다. 다음의 경우 SQL 패키지가 작성될 수 있습니다.

- RDB 매개변수가 CRTSQLxxx 명령문에 지정되고 프로그램 오브젝트가 성공적으로 생성되어야 합니다. SQL 패키지는 RDB 매개변수에서 지정한 시스템에서 생성될 것입니다.

  컴파일이 성공적이지 않았거나 모듈 오브젝트만 생성한다면 SQL 패키지가 생성되지 않습니다.
- CRTSQLPKG문을 사용해야 합니다. CRTSQLPKG는 사전컴파일에서 패키지가 생성되지 않았거나 해당 패키지가 사전컴파일문에 지정되지 않은 RDB에서 필요로 할 경우 패키지를 생성하기 위하여 사용됩니다.

SQL 삭제 명령(DLTSQLPKG)을 통해 사용자는 로컬 시스템의 SQL 패키지를 삭제할 수 있습니다.

SQL 패키지의 작성과 연관된 권한부여 ID에 의해 부여된 특권에 리모트 시스템(어플리케이션 서버) 패키지 작성에 관한 적절한 권한이 없는 한 SQL 패키지는 작성되지 않습니다. 프로그램을 수행하기 위해서는 권한부여 ID에 SQL 패키지의 EXECUTE 특권이 있어야 합니다. iSeries 시스템에서 EXECUTE 권한에는 *OBJOPR 및 *EXECUTE의 시스템 권한이 포함됩니다.

CRTSQLPKG(SQL 패키지 작성) 명령의 구문은 부록 B『iSeries용 DB2 UDB CL 명령 설명』을 참조하십시오

SQL 패키지의 유효한 SQL문

다른 서버에 연결되는 프로그램은 SET TRANSACTION문을 제외하고 SQL 참조서에 설명되어 있는 대로 SQL문 중 하나를 사용할 수 있습니다. iSeries용 DB2 UDB가 아닌 시스템을 참조하는 iSeries용 DB2 UDB를 사용하여 컴파일된 프로그램은 리모트 시스템이 지원하는 실행 가능한 SQL문을 사용할 수 있습니다. 사전컴파일러는 이해할 수 없는 명령문에 대한 진단 메세지를 계속해서 발행합니다. 이러한 명령문들은 SQL패키지가 작성되는 동안에 리모트 시스템으로 보내집니다. 명령문이 현재 어플리케이션 서버에서 수행될 수 없으면 수행시 지원에 의해 SQLCODE 84 또는 525가 표시됩니다. 예를 들어, 복수 행 FETCH, 블록화된 INSERT 그리고 화면 이동 가능 커서 지원은 어플리케이션 리퀘스터와 어플리케이션 서버 모두 버전 2 릴리스 2 이상의 OS/400인 분산 프로그램에서만 허용됩니다. 자세한 정보는 SQL 참조서의 분산 관계형 데이터베이스 사용에 대한 고려사항을 참조하십시오.

SQL 패키지 작성에 대한 고려사항

SQL 패키지를 작성할 때에는 여러 가지 사항을 고려해야 합니다. 고려사항 중 몇 가지를 아래에 제시하였습니다.

CRTSQLPKG 권한부여

iSeries 시스템에서 SQL 패키지를 작성할 때 사용된 권한부여 ID에는 CRTSQLPKG 명령에 대한 *USE 권한이 있어야 합니다.

비iSeries용 DB2 UDB에 패키지 작성

비iSeries용 DB2 UDB에 대한 프로그램과 SQL 패키지 작성시 관계형 데이터베이스에 고유한 SQL문을 사용하려 할 때에는 CRTSQLxxx GENLVL 매개변수가 30으로 설정되어야 합니다. 심각도 레벨 30이 넘는 메세지가 발행되지 않는 한 프로그램은 작성됩니다. 심각도 레벨 30을 넘는 메세지가 발행되면, 명령문은 관계형 데이터베이스에 대해 유효하지 않게 됩니다. 예를 들어, 정의되지 않았거나 사용할 수 없는 변수나 상수는 심각도가 30 이상인 메세지를 발행합니다.

11 이상의 GENLVL로 수행될 때에는 예상치 못한 메세지에 대해 사전컴파일러 리스팅이 체크되어야 합니다. DB2 UDB에 대한 패키지를 작성중이라면, GENLVL 매개변수를 20 미만의 값으로 설정해야 합니다.

RDB 매개변수가 iSeries용 DB2 UDB 시스템이 아닌 시스템을 지정하는 경우, 다음의 옵션이 CRTSQLxxx 명령에 사용되어서는 안됩니다.

- COMMIT(*NONE)
- OPTION(*SYS)
- DATFMT(*MDY)

- DATFMT(*DMY)
- DATFMT(*JUL)
- DATFMT(*YMD)
- DATFMT(*JOB)
- DYNUSRPRF(*OWNER)
- TIMSEP(*BLANK) 또는 TIMSEP(‘,’)이 지정된 경우, TIMFMT(*HMS)
- SRTSEQ(*JOBRUN)
- SRTSEQ(*LANGIDUNQ)
- SRTSEQ(*LANGIDSHR)
- SRTSEQ(라이브러리명/표 이름)

주: DB2 Universal Database 서버에 연결할 때, 다음의 추가 규칙이 적용됩니다.

- 지정된 날짜와 시간 형식이 반드시 동일해야 합니다.
- TEXT 매개변수에는 반드시 *BLANK 값을 사용해야 합니다.
- 디폴트 스키마(DFTRDBCOL)는 지원되지 않습니다.
- 패키지가 작성되고 있는 소스 프로그램의 CCSID는 65535일 수 없으나, 65535를 사용하면 빈 패키지가 작성됩니다.

목표 릴리스(TGTRLS)

패키지 작성중 어느 릴리스가 해당 함수를 지원할 수 있는지 판별하기 위해 SQL문이 체크됩니다. 이 릴리스는 패키지의 복원 레벨로 설정됩니다. 예를 들어, 표에 FORIGN KEY 제한조건을 추가하는 CREATE TABLE문이 패키지에 있으면, 패키지의 복원 레벨은 FOREIGN KEY 제한조건이 이전 릴리스에 지원되지 않으므로 버전 3 릴리스 1일 것입니다. TGTRLS 메세지는 TGTRLS 매개변수가 *CURRENT인 경우 표시되지 않습니다.

SQL문 크기

SQL 패키지 작성 기능이 사전컴파일러가 처리할 수 있는 것과 동일한 크기의 SQL문을 처리하지 못할 수 있습니다. SQL 프로그램의 사전컴파일시 SQL문은 프로그램의 연관된 영역에 놓이게 됩니다. 이 때, 각 토큰이 공백으로 분리됩니다. 또한 RDB 매개변수가 지정되면 소스문의 호스트 변수가 ‘H’로 대체됩니다. SQL 패키지 작성 기능은 명령문에 대한 호스트 변수와 함께 명령문을 어플리케이션 서버로 전달합니다. 토큰 사이의 공백 추가와 호스트 변수 대체로 명령문이 최대 SQL문의 크기를 초과할 수 있습니다(SQL0101 이유 5).

패키지를 요구하지 않는 명령문

어떤 경우에는 SQL 패키지의 작성을 시도하는데 SQL 패키지는 작성되지 않은 채 프로그램이 계속 수행됩니다. 이 상황은 프로그램에 SQL 패키지가 수행되기를 요구하지 않는 SQL문만 있을 때 일어납니다. 예를 들어 SQL문 DESCRIBE TABLE만을 포

함하는 프로그램은 SQL 패키지 생성중 메세지 SQL5041을 생성할 것입니다. SQL 패키지가 필요하지 않은 SQL문은 다음과 같습니다.

- COMMIT
- CONNECT
- DESCRIBE TABLE
- DISCONNECT
- RELEASE
- RELEASE SAVEPOINT
- ROLLBACK
- SAVEPOINT
- SET CONNECTION

패키지 오브젝트 유형

SQL 패키지는 항상 비ILE 오브젝트로 작성되며 디폴트 활성 그룹에서 수행됩니다.

ILE 프로그램과 서비스 프로그램

ILE 프로그램과 SQL문이 포함된 몇 개의 모듈을 바인드하는 서비스 프로그램에는 각 모듈에 대한 개별 SQL 패키지가 있어야 합니다.

패키지 작성 연결

패키지 작성을 위해 수행된 연결 유형은 RDBCNNMTH 매개변수를 사용하여 요구된 연결 유형에 기초합니다. RDBCNNMTH(*DUW)가 지정되면 확약 제어가 사용되고 해당 연결은 읽기 전용 연결이 됩니다. 해당 연결이 읽기 전용이면 패키지 작성이 실패합니다.

작업 단위

패키지 작성이 내재적으로 확약 또는 롤백을 수행하므로, 확약 정의는 패키지 작성 시도에 앞서 작업 단위 경계에 있어야 합니다. 작업 단위 경계에 있어야 할 확약 정의에 대해서는 아래 조건이 모두 만족되어야 합니다.

- SQL이 작업 단위 경계에 있어야 합니다.
- 확약 제어를 사용한 로컬 또는 DDM 파일 열기가 없으며, 변경 지연중인 채로 닫힌 로컬 또는 DDM 파일이 없어야 합니다.
- 등록된 API 자원이 없어야 합니다.
- DRDA나 DDM과 연관되지 않은 등록 LU 6.2 자원이 없어야 합니다.

로컬 패키지 작성

RDB 매개변수에 지정된 이름은 로컬 시스템명일 수 있습니다. 이것이 로컬 시스템명이면 SQL 패키지가 로컬 시스템에서 작성됩니다. SQL 패키지를 다른 서버에 저장(SAVOBJ 명령)한 후에 복원(RSTOBJ 명령)할 수 있습니다. 로컬 시스템과 연결된 프

로그램을 수행할 때는 SQL 패키지가 사용되지 않습니다. *LOCAL을 RDB 매개변수에 지정하는 경우, *SQLPKG 오브젝트가 작성되지 않으나 패키지 정보는 *PGM 오브젝트에 저장됩니다.

레이블

SQL 패키지에 대한 설명을 작성하려는 경우 LABEL ON문을 사용할 수 있습니다.

일관성 토큰

프로그램과 그에 연관된 SQL 패키지에는 SQL 패키지에 대한 호출이 있을 때 체크되는 일관성 토큰이 들어 있습니다. 일관성 토큰은 일치되어야 하며 그렇지 않으면 패키지를 사용할 수 없습니다. 프로그램과 SQL 패키지가 조정되지 않는 것으로 표시할 수도 있습니다. 프로그램이 iSeries 시스템에 있고 어플리케이션 서버는 다른 iSeries 시스템에 있다고 가정합니다. 프로그램이 세션 A에서 수행되고 세션 B(SQL 패키지 또한 재작성되는 세션)에서 재작성됩니다. 이 경우, 세션 A에서는 프로그램에 대한 다음 호출에서 일관성 토큰 오류가 초래될 수 있습니다. 각 호출마다 SQL 패키지가 위치되지 않도록 하기 위해 SQL은 각 세션에 의해 사용된 SQL 패키지에 대한 주소 리스트를 유지보수합니다. 세션 B가 SQL 패키지를 재사용할 때, 기존 SQL 패키지는 QRPLOBJ 라이브러리로 이동됩니다. 세션 A의 SQL 패키지 주소는 여전히 유효합니다(이 상황은 프로그램이 수행되고 있는 세션에서 프로그램과 SQL 패키지를 작성하거나 프로그램 작성 이전에 기존 SQL 패키지를 삭제하는 리모트 명령을 제출함으로써 피할 수 있습니다).

새로운 SQL 패키지를 사용하려면 리모트 시스템과의 연결을 종료해야 합니다. 세션을 사인 오프한 다음 다시 사인 온하거나 대화식 SQL(STRSQL) 명령을 사용하여 비보호 네트워크 연결에 대해 DISCONNECT 또는 보호 연결에 대해 RELEASE와 COMMIT를 차례로 발행할 수 있습니다. 그런 다음 RCLDDMCNV를 네트워크 연결 종료에 사용해야 합니다. 프로그램을 다시 호출하십시오.

SQL 및 순환

사전컴파일시 어텐션 키로부터 SQL을 이미 호출한 경우에는 예기치 않은 결과가 나옵니다.

CRTSQLxxx, CRTSQLPKG, STRSQL 명령과 SQL 런타임 환경은 순환적이지 않습니다. 순환이 시도되면 예기치 않은 결과가 나올 수 있습니다. 명령 중 하나가 수행되는 중에는 (또는 삽입 SQL문으로 프로그램이 수행되는 중) 명령이 완료하기도 전에 작업이 인터럽트되고 또 다른 SQL 기능이 시작되는 경우에 순환이 발생합니다.

SQL에 대한 CCSID 고려사항

분산 어플리케이션을 실행 중이며 시스템 중 하나가 iSeries 시스템이 아닌 경우, iSeries 서버에서 작업 CCSID 값을 65535로 설정할 수 없습니다.

리모트 시스템이 SQL 패키지 작성을 요구하기 전에, 어플리케이션 리퀘스터는 항상 RDB 매개변수, SQL 패키지명, 라이브러리명 및 SQL 패키지의 텍스트를 작업의 CCSID에서 CCSID 500으로 변환합니다. 이는 DRDA에 의해 요구됩니다. 리모트 관계형 데이터베이스가 iSeries 시스템일 때 이름은 CCSID 500에서 작업 CCSID로 변환되지 않습니다

분리 ID를 표, 보기, 색인, 스키마, 라이브러리 또는 SQL 패키지 이름에 사용하지 않는 것이 바람직합니다. 서로 다른 CCSID를 가진 시스템들간에는 이름 변환이 발생하지 않습니다. 다음은 CCSID가 37로 실행중인 시스템 A와 CCSID가 500으로 실행중인 시스템 B에 대한 예입니다.

- 시스템 A에서 이름이 ″a¬b|c″인 표를 작성하는 프로그램을 작성하십시오.
- 시스템 A에 프로그램 ″a¬b|c″를 저장한 다음 이를 시스템 B로 복원하십시오.
- CCSID의 ¬에 대한 코드점은 x'5F'인 반면 CCSID 500에서 코드점은 x'BA'입니다.
- 시스템 B에서 이름은 ″a[b]c″로 표시됩니다. 이름이 ″a¬b|c″인 표를 참조하는 프로그램을 작성했다면, 프로그램은 해당 표를 발견하지 못합니다.

at 부호(@), 파운드 부호(#) 및 달러 부호($) 문자는 SQL 오브젝트명에 사용할 수 없습니다. 이러한 코드점은 사용된 CCSID에 따라 다릅니다. 분리된 이름이나 3개국 확장자를 사용하는 경우 이름 분석 기능은 후속 릴리스에서 실패할 수도 있습니다.

연결 관리와 활성 그룹

자세한 내용은 다음의 주제를 참조하십시오.

- 『연결 및 대화』
- 376 페이지의 『PGM1에 대한 소스 코드:』
- 377 페이지의 『PGM2에 대한 소스 코드:』
- 377 페이지의 『PGM3에 대한 소스 코드:』
- 379 페이지의 『동일한 관계형 데이터베이스로의 복수 연결』
- 380 페이지의 『디폴트 활성 그룹에 대한 내재적 연결 관리』
- 380 페이지의 『비디폴트 활성 그룹에 대한 내재적 연결 관리』

연결 및 대화

TCP/IP를 DRDA에서 사용하기 전에, '연결'이라는 용어는 명확하지 않았습니다. SQL의 관점에서 연결을 언급해왔습니다. 즉, 한 곳에서 시작한 연결은 몇몇 RDB를 CONNECT TO했고, DISCONNECT 수행시 종료되거나 성공적인 COMMIT가 뒤따라 오는 RELEASE ALL이 발생했습니다. APPC 대화는 작업의 DDMCNV 속성 값과 iSeries 또는 다른 유형의 시스템과 대화가 있는지의 여부에 따라 유지될 수도 있고 유지되지 않을 수도 있습니다.

TCP/IP 용어에는 '대화'라는 용어가 포함되지 않습니다. 그러나 유사한 개념은 존재합니다. DRDA에서 TCP/IP를 지원하게 되면, 용어 '대화'는 APPC 대화에 관해 특별히 명시되지 않는 한 이 책에서 '네트워크'라는 용어로 대체합니다. 따라서, 이제 사용자가 알아야 하는 두 가지 다른 유형의 연결은 위에서 설명한 유형의 SQL 연결과 용어 '대화'를 대체시킨 '네트워크' 연결입니다.

두 가지 유형의 연결간에 혼란 가능성이 있는 부분에서는 의도하는 의미를 잘 이해할 수 있도록 'SQL' 또는 '네트워크'로 규정했습니다.

SQL 연결은 활성 그룹 레벨에서 관리됩니다. 한 작업 내에 있는 각각의 활성 그룹은 각각의 연결을 관리하고 이 연결은 전체 활성 그룹에서 공유되지 않습니다. 디폴트 활성 그룹에서 실행된 프로그램의 경우, 버전 2 릴리스 3 이전처럼 계속 관리됩니다.

다음은 복수 활성 그룹에서 실행되는 어플리케이션의 예입니다. 이 예는 활성 그룹, 연결 관리 및 확약 제어 사이의 상호작용을 설명하기 위하여 사용됩니다. 이것이 권장되는 코딩 유형은 **아닙니다**.

PGM1에 대한 소스 코드:

```
     ....
   EXEC SQL
      CONNECT TO SYSB
END-EXEC.
   EXEC SQL
      SELECT ....
END-EXEC.
    CALL PGM2.
    ....
```

그림 9. PGM1에 대한 소스 코드:

PGM1에 대해 프로그램과 SQL 패키지를 작성하는 명령은 다음과 같습니다.

```
CRTSQLCBL PGM(PGM1) COMMIT(*NONE) RDB(SYSB)
```

PGM2에 대한 소스 코드:

```
...
  EXEC SQL
          CONNECT TO SYSC;
  EXEC SQL
          DECLARE C1 CURSOR FOR
              SELECT ....;
  EXEC SQL
          OPEN C1;
        do {
  EXEC SQL
                FETCH C1 INTO :st1;
  EXEC SQL
                UPDATE ...
                    SET COL1 = COL1+10
                  WHERE CURRENT OF C1;
              PGM3(st1);
        } while SQLCODE == 0;
  EXEC SQL
         CLOSE C1;
      EXEC SQL COMMIT;
....
```

그림 10. PGM2에 대한 소스 코드:

PGM2에 대해 프로그램과 SQL 패키지를 작성하는 명령은 다음과 같습니다.

```
CRTSQLCI OBJ(PGM2) COMMIT(*CHG) RDB(SYSC) OBJTYPE(*PGM)
```

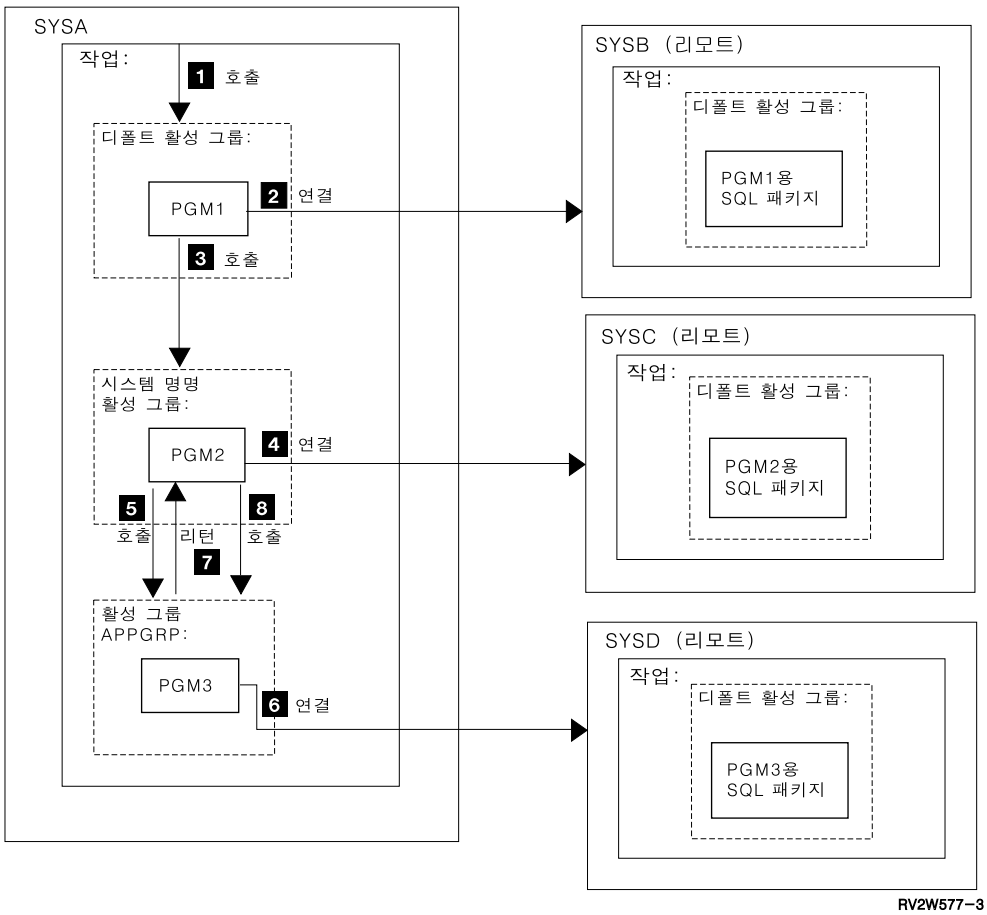
PGM3에 대한 소스 코드:

```
...
 EXEC SQL
   INSERT INTO TAB VALUES(:st1);
 EXEC SQL COMMIT;
....
```

그림 11. PGM3에 대한 소스 코드

PGM3에 대해 프로그램과 SQL 패키지를 작성하는 명령은 다음과 같습니다.

```
CRTSQLCI OBJ(PGM3) COMMIT(*CHG) RDB(SYSD) OBJTYPE(*MODULE)
CRTPGM PGM(PGM3) ACTGRP(APPGRP)
CRTSQLPKG PGM(PGM3) RDB(SYSD)
```



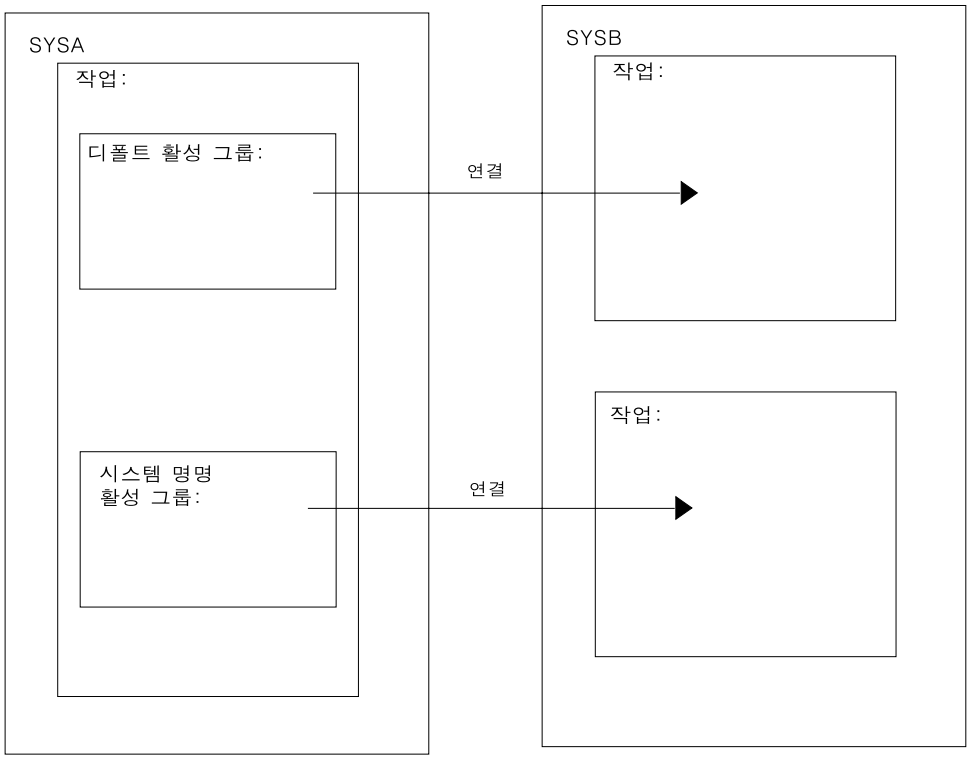
이 예에서 PGM1은 CRTSQLCBL 명령을 사용해 작성된 비ILE 프로그램입니다. 이 프로그램은 디폴트 활성 그룹에서 실행됩니다. PGM2는 CRTSQLCI 명령을 사용해 작성되고, 시스템 명명 활성 그룹에서 실행됩니다. PGM3 역시 CRTSQLCI 명령을 사용하여 작성되었으나 이는 APPGRP로 명명된 활성 그룹에서 실행됩니다. APPGRP가 ACTGRP 매개변수에 대한 디폴트 값이 아니기 때문에 CRTPGM 명령은 개별적으로 발행됩니다. CRTPGM 명령은 SYSD 관계형 데이터베이스상의 SQL 패키지를 작성하는 명령 뒤에 옵니다. 이 예에서 사용자는 작업 레벨 정의를 명시적으로 시작할 수 없습니다. 묵시적으로 SQL이 확약 제어를 시작합니다.

1. PGM1이 호출되고 디폴트 활성 그룹에서 실행됩니다.
2. PGM1이 관계형 데이터베이스 SYSB에 연결되고 SELECT문이 실행됩니다.
3. 그런 후 PGM1이 PGM2를 호출하고 시스템 명명 활성 그룹에서 실행됩니다.
4. PGM2는 관계형 데이터베이스 SYSC에 연결됩니다. PGM1과 PGM2가 다른 활성 그룹에 있기 때문에 시스템 명명 활성 그룹에서 PGM2로 시작되는 연결이 디폴트 활성 그룹에서 PGM1로 시작된 연결과 단절되지 않습니다. 두 연결이 활동하고 있습니다. PGM2는 커서를 열고 행을 반입하고 갱신합니다. PGM2는 확약 제어하에서 실행되며, 작업 단위의 중간에 있으며 연결 가능한 상태가 아닙니다.
5. PGM2가 PGM3을 호출하고, 활성 그룹 APPGRP에서 실행됩니다.

6. INSERT문은 활성 그룹에서 수행되는 첫 번째 문입니다. 첫 번째 SQL문은 내재 연결이 관계형 데이터베이스 SYSD가 되게 합니다. 행은 관계형 데이터베이스 SYSD에 위치한 표 TAB로 삽입됩니다. 삽입이 확약됩니다. 시스템 명명 활성 그룹에서 지연되고 있는 변경은 확약 제어가 활성 그룹의 확약 범위를 갖는 SQL로 시작되지 않았기 때문에 확약되지 않습니다.
7. PGM3이 나가고 제어는 PGM2로 리턴됩니다. PGM2가 다른 행을 페치하고 갱신합니다.
8. PGM3이 행을 삽입하기 위해 다시 호출됩니다. 내재적 연결이 PGM3의 첫 번째 호출에서 수행됩니다. 활성 그룹이 PGM3 호출 사이에서 종료되지 않았기 때문에 후속 호출이 수행되지 않습니다. 마지막으로 PGM2에 의해 모든 행이 처리되고, 시스템 명명 활성 그룹과 연관된 작업 단위가 확약됩니다.

동일한 관계형 데이터베이스로의 복수 연결

다른 활성 그룹이 같은 관계형 데이터베이스에 연결되는 경우, 각각의 SQL 연결은 자체의 네트워크 연결과 어플리케이션 서버 작업을 가집니다. 활성 그룹이 확약 제어로 실행되면, 하나의 활성 그룹에서 확약된 변경은 작업 레벨 확약 정의가 사용되지 않는 한, 다른 활성 그룹에서 확약 변경을 수행하지 않습니다.



RV2W578-2

디폴트 활성 그룹에 대한 내재적 연결 관리

어플리케이션 리퀘스터는 어플리케이션 서버에 내재적으로 연결될 수 있습니다. 내재적 SQL 연결은 어플리케이션 리퀘스터가 첫 번째 SQL문이 디폴트 활성 그룹에 대해 첫 번째로 사용되고 있는 SQL 프로그램에 의해 발행중이고 다음 조건이 만족될 때 이루어집니다.

- 발행되는 SQL문이 매개변수가 있는 CONNECT문이 아닙니다.
- SQL이 디폴트 활성 그룹에서 활동하지 않습니다.

분산 프로그램에서, 내재적 SQL 연결은 RDB 매개변수에 지정된 관계형 데이터베이스로 이루어집니다. 비분산 프로그램에서는 내재적 SQL 연결이 로컬 관계형 데이터베이스로 이루어집니다.

SQL은 비활동 상태가 되면서 디폴트 활성 그룹의 모든 활성 연결을 종료시킵니다. 다음의 경우에 SQL은 활성화되지 않습니다.

- 어플리케이션 리퀘스터는 프로세스가 종료되고 다음의 조건이 만족될 경우에 첫 번째로 활동하고 있는 SQL 프로그램을 검출합니다.
  - 지연중인 SQL 변경이 없습니다.
  - 보호된 연결을 사용하는 연결이 없습니다.
  - SET TRANSACTION문이 활동 상태가 아닙니다.
  - CLOSQLCSR(*ENDJOB)로 사전컴파일된 프로그램이 실행되지 않습니다.

  지연중인 변경, 보호된 연결 또는 사용중인 SET TRANSACTION문이 있다면, SQL이 나감 상태에 놓입니다. CLOSQLCSR(*ENDJOB)로 사전컴파일된 프로그램이 실행되었다면, SQL은 작업이 종료될 때까지 디폴트 활성 그룹에 대해 활동 상태로 남아 있습니다.
- SQL이 종료된 상태에 있는 경우, 작업 단위 끝에 있습니다. 이는 SQL 프로그램 외부에서 COMMIT나 ROLLBACK 명령을 발행할 때 일어납니다.
- 작업 종료시

비디폴트 활성 그룹에 대한 내재적 연결 관리

어플리케이션 리퀘스터는 어플리케이션 서버에 내재적으로 연결될 수 있습니다. 내재적 SQL 연결은 어플리케이션 리퀘스터가 활성 그룹에 대해 발행된 첫 번째 SQL문이 매개변수가 있는 CONNECT문이 아님을 발견하는 경우에 일어납니다.

분산 프로그램의 경우, 내재적 SQL 연결은 RDB 매개변수에 지정된 관계형 데이터베이스로 이루어집니다. 비분산 프로그램의 경우, 내재적 SQL 연결은 로컬 관계형 데이터베이스로 이루어집니다.

내재적 단절은 프로세스의 다음 상황에서 발생합니다.

- 활성 그룹이 종료될 때, 확약 제어가 활동 상태가 아니거나 활성 그룹 레벨의 확약 제어가 활동 상태에 있거나 작업 레벨 확약 정의가 작업 단위 경계에 있을 때.

  작업 레벨 확약이 활동 상태에 있고 작업 단위 경계에 있지 않을 때, SQL은 종료 상태에 놓입니다.
- SQL이 종료된 상태에서 작업 레벨 확약 정의가 확약되거나 롤백중일 때
- 작업 종료시

분산 지원

iSeries용 DB2 UDB는 두 가지 레벨의 분산 관계형 데이터베이스를 지원합니다.

- 리모트 작업 단위(RUW)

  리모트 작업 단위에서는 하나의 작업 단위 동안 SQL 명령을 준비하거나 실행하는 것이 오직 하나의 어플리케이션 서버에서 발생합니다. 어플리케이션 리퀘스터에서 어플리케이션 프로세스를 갖는 활성 그룹은 어플리케이션 서버와 연결될 수도 있고, 하나 이상의 작업 단위에서 임의의 수의 해당 어플리케이션 서버의 오브젝트를 참조하는, 정적 또는 동적 SQL문을 실행할 수 있습니다. 리모트 작업 단위는 DRDA 레벨 1로 참조되기도 합니다.
- 분산된 작업 단위(DUW)

  분산된 작업 단위에서는 하나의 작업 단위 동안 SQL문의 준비나 실행이 여러 어플리케이션 서버에서 발생할 수 있습니다. 그러나, 단일 SQL문은 단일 어플리케이션 서버에 위치한 오브젝트만을 참조할 수 있습니다. 분산된 작업 단위는 DRDA 레벨 2로 참조되기도 합니다.

  분산된 작업 단위는 다음을 허용합니다.

  – 하나의 논리적 작업 단위에서 복수 어플리케이션 서버에 대한 갱신 액세스

    또는

  – 하나의 논리적 작업 단위에서 복수 어플리케이션 서버에 대한 읽기 액세스로 단일 어플리케이션 서버에 대한 갱신 액세스

  복수 어플리케이션 서버가 하나의 작업 단위에서 갱신될 수 있는지의 여부는 어플리케이션 리퀘스터에 하나의 동기점 관리자만 있는지 아니면 여러 동기점 관리자가 있는지, 어플리케이션 리퀘스터와 어플리케이션 서버간에 2단계 확약 프로토콜 지원이 있는지에 따라 다릅니다.

  동기점 관리자 사전컴파일러는 2단계 확약 프로토콜 참여자간의 확약과 롤백 조작을 조합하는 시스템 요소입니다. 분산된 갱신을 실행할 때, 서로 다른 시스템의 동기점 관리자는 자원이 일관성있는 상태를 찾도록 협동 작업을 합니다. 동기점 관리자가 사용하는 프로토콜과 흐름은 2단계 확약 프로토콜로도 참조됩니다.

  2단계 확약이 사용되는 경우, 연결은 보호 자원이며, 그렇지 않은 경우 연결은 비보호 자원입니다.

시스템간에 사용된 자료 전송 프로토콜 유형은 네트워크 연결의 보호 여부에 영향을 줍니다. V5R1 이전에는 TCP/IP 연결이 항상 보호되지 않았으므로 제한된 방법으로만 분산 작업 단위에 참여할 수 있었습니다. V5R1에서는 TCP/IP를 통한 DUW에 대해 완전한 지원이 추가되었습니다.

예를 들어, 프로그램에서 작성된 첫 번째 연결이 TCP/IP를 통해 V5R1 이전으로 이루어진 경우, 이에 대해 갱신을 수행할 수 있지만 후속 연결은 APPC를 사용하는 경우에도 읽기 전용이 됩니다.

대화식 SQL 사용시, 첫 번째 SQL 연결은 로컬 시스템으로 이루어집니다. 따라서, V5R1 이전 환경에서 TCP/IP를 사용하여 리모트 시스템을 갱신하려면 RELEASE ALL 다음에 COMMIT를 수행하여 CONNECT TO remote-tcp-system을 수행하기 전에 모든 SQL 연결을 종료해야 합니다.

연결 유형 판별

리모트 연결이 설정될 때 리모트 연결은 비보호 또는 보호 네트워크 연결을 사용합니다. 확약 가능 갱신에 있어서 이 연결은 읽기 전용, 갱신 가능 또는 연결이 설정될 때 갱신 가능 여부를 모를 수 있습니다. 확약 가능 갱신은 임의의 삽입, 삭제, 갱신 또는 확약 제어하에서 실행되는 DDL문입니다. 연결이 읽기 전용이면, COMMIT(*NONE)를 사용한 변경이 실행될 수 있습니다. CONNECT 또는 SET CONNECTION 후에 SQLCA의 SQLERRD(4)가 연결 유형을 알려줍니다. SQLERRD(4)는 연결이 비보호 또는 보호 네트워크 연결 중 어느 것을 사용하는지도 나타냅니다. 특정 값에 대한 설명은 다음과 같습니다.

1. 확약 가능 갱신이 연결에 대해 수행될 수 있습니다. 연결이 비보호됩니다. 확약 가능 갱신은 다음 상황에서 일어납니다.
   - 연결이 리모트 작업 단위(RDBCNNMTH(*RUW))를 사용하여 설정됩니다. 여기에는 또한 로컬 연결과 리모트 작업 단위를 사용하는 어플리케이션 리퀘스터 드라이버(ARD) 연결이 포함됩니다.
   - 연결이 분산 작업 단위(RDBCNNMTH(*DUW))를 사용하여 설정될 경우, 다음 사항들을 모두 만족시킵니다.
     – 연결이 로컬이 아닙니다.
     – 어플리케이션 서버가 분산된 작업 단위를 지원하지 않습니다. 예를 들어, iSeries용 DB2 UDB 어플리케이션 서버는 버전 3 릴리스 1 이전의 OS/400 릴리스를 가집니다.
     – 연결을 발행하는 프로그램의 확약 제어 레벨이 *NONE이 아닙니다.
     – 확약 가능 갱신을 수행할 수 있는(로컬을 포함한) 다른 어플리케이션 서버의 연결이 없거나, 모든 연결이 분산 작업 단위를 지원하지 않는 어플리케이션 서버로의 읽기 전용 연결입니다.
     – 확약 정의를 위해 확약 제어하에서 열린 갱신 가능 로컬 파일이 없습니다.

- 확약 정의에 대한 확약 제어하에서 다른 연결을 사용하는 열린 갱신 가능 DDM 파일이 없습니다.
- 확약 정의를 위한 API 확약 제어 자원이 없습니다.
- 확약 정의를 위해 등록된 보호 연결이 없습니다.

확약 제어하에서 실행될 때, SQL은 리모트 연결에 대해서는 1단계 갱신 가능 DRDA 자원을 등록하고, 로컬 및 ARD 연결에 대해서는 2단계 갱신 가능 DRDA 자원을 등록합니다.

2. 확약 가능 갱신이 연결에서 수행되지 않습니다. 연결은 읽기 전용입니다. 네트워크 연결이 비보호됩니다.

   이는 리모트 작업 단위 연결 관리(*RUW)로 컴파일된 어플리케이션에 대해서는 절대로 발생하지 않습니다.

   분산 작업 단위 어플리케이션의 경우, 이는 연결 설정시 다음 사항을 만족시키는 경우에만 발생합니다.

   - 연결이 로컬이 아닙니다.
   - 어플리케이션 서버가 분산 작업 단위를 지원하지 않습니다.
   - 다음 중 최소한 하나라도 만족해야 합니다.
     - 연결을 발행하는 프로그램의 확약 제어 레벨이 *NONE이어야 합니다.
     - 분산 작업 단위를 지원하지 않는 어플리케이션 서버로의 또 다른 연결이 존재해야 하며, 그 어플리케이션 서버는 확약 가능 갱신을 수행할 수 있어야 합니다.
     - 분산 작업 단위(로컬 포함)를 지원하는 어플리케이션 서버로의 또 다른 연결이 존재해야 합니다.
     - 확약 정의를 위해 확약 제어하에서 열린 갱신 가능 로컬 파일이 있어야 합니다.
     - 확약 정의를 위해 확약 제어하에서 다른 연결을 사용하는 열린 갱신 가능 DDM 파일이 있습니다.
     - 확약 정의를 위한 1단계 API 확약 제어 자원이 없습니다.
     - 확약 정의를 위해 등록된 보호 연결이 있습니다.

   확약 제어하에서 실행될 때, SQL은 1단계 읽기 전용 자원을 등록할 것입니다.

3. 확약 가능 갱신을 수행할 수 있는지 여부를 알 수 없습니다. 연결이 보호됩니다.

   이는 리모트 작업 단위 연결 관리(*RUW)로 컴파일된 어플리케이션에 대해서는 절대로 발생하지 않습니다.

   분산 작업 단위의 경우, 이는 연결 설정시 다음이 모두 만족되는 경우에만 발생합니다.

   - 연결이 로컬이 아닙니다.

- 연결을 발행하는 프로그램의 확약 제어 레벨이 *NONE이 아닙니다.
- 어플리케이션 서버가 분산 작업 단위와 2단계 확약 프로토콜(보호 연결)을 지원합니다.

확약 제어하에서 실행될 때, SQL은 2단계 미결 자원을 등록합니다.

4. 확약 가능 갱신을 수행할 수 있는지 여부를 알 수 없습니다. 연결이 보호되지 않습니다.

   이는 리모트 작업 단위 연결 관리(*RUW)로 컴파일된 어플리케이션에 대해서는 절대로 발생하지 않습니다.

   분산 작업 단위의 경우, 확약 가능 갱신은 연결이 설정될 때 다음 사항을 모두 만족시키는 경우에만 발생합니다.
   - 연결이 로컬이 아닙니다.
   - 어플리케이션 서버가 분산 작업 단위를 지원합니다.
   - 어플리케이션 서버가 2단계 확약 프로토콜(보호 연결)을 지원하지 않거나 연결을 발행중인 프로그램의 확약 제어 레벨이 *NONE입니다.

   확약 제어하에서 실행될 때, SQL은 1단계 DRDA 미결 자원을 등록합니다.
5. 확약 가능한 갱신이 수행될 수 있는지 여부는 알 수 없으며 연결은 분산 작업 단위를 사용하는 로컬 연결이거나 분산 작업 단위를 사용하는 ARD 연결입니다.

   확약 제어하에서 실행될 때, SQL은 2단계 DRDA 미결 자원을 등록합니다.

1단계 및 2단계 자원에 대한 자세한 정보는 백업 및 회복 의 확약 제어 주제를 참조하십시오.

다음 표는 분산 리모트 작업 단위 연결을 야기시키는 연결 유형을 요약한 것입니다. SQLERRD(4)는 성공적인 CONNECT문과 SET CONNECTION문에 설정됩니다.

표 39. 연결 유형 요약

확약 제어하의 연결	어플리케이션 서버의 2단계 확약 지원 여부	어플리케이션 서버의 분산 작업 단위 지원 여부	기타 갱신 가능 1단계 자원 등록 여부	SQLERRD(4)
아니오	아니오	아니오	아니오	2
아니오	아니오	아니오	예	2
아니오	아니오	예	아니오	4
아니오	아니오	예	예	4
아니오	예	아니오	아니오	2
아니오	예	아니오	예	2
아니오	예	예	아니오	4
아니오	예	예	예	4
예	아니오	아니오	아니오	1
예	아니오	아니오	예	2

표 39. 연결 유형 요약 (계속)

확약 제어하의 연결	어플리케이션 서버의 2단계 확약 지원 여부	어플리케이션 서버의 분산 작업 단위 지원 여부	기타 갱신 가능 1단계 자원 등록 여부	SQLERRD(4)
예	아니오	예	아니오	4
예	아니오	예	예	4
예	예	아니오	아니오	N/A *
예	예	아니오	예	N/A *
예	예	예	아니오	3
예	예	예	예	3

*DRDA는 리모트 작업 단위(DRDA1)를 지원하는 어플리케이션 서버에서는 보호 연결을 허용하지 않습니다. 여기에는 모든 iSeries용 DB2 TCP/IP 연결이 포함됩니다.

연결과 확약 제어 제한사항

확약 제어를 사용하여 연결할 수 있는 시기에 대해서는 몇 가지 제한사항이 있습니다. 이러한 제한사항은 연결이 COMMIT(*NONE)를 사용하여 설정되었지만 확약 제어를 사용하여 명령문을 실행할 경우에도 적용됩니다.

2단계 미결 자원 또는 갱신 가능한 자원이 등록되었거나 1단계 갱신 가능 자원이 등록된 경우, 또 다른 갱신 가능 자원을 등록할 수 없습니다.

더 나아가서, 보호 연결이 비활동 상태이고 DDMCNV 작업 속성이 *KEEP이면, 사용되지 않은 DDM 연결은 RUW 연결 관리자를 사용하여 컴파일된 프로그램의 CONNECT문을 실패하게 만듭니다.

RUW 연결 관리자를 사용하여 실행되고 작업 레벨 확약 정의가 사용될 경우에는 약간의 제한이 있습니다.

- 작업 레벨 확약 정의가 하나 이상의 활성 그룹에서 사용되면, 모든 RUW 연결은 반드시 로컬 관계형 데이터베이스에 대한 것이어야 합니다.
- 연결이 리모트이면, 단 하나의 활성 그룹만이 RUW 연결에 대한 작업 레벨 확약 정의를 사용할 수 있습니다.

연결 상태 판별

매개변수가 없는 CONNECT문을 사용하면 현재의 연결이 갱신 가능한지 아니면 현재 작업 단위에 대해 읽기 전용인지를 판별할 수 있습니다. 이때 SQLERRD(3)에는 값 1 또는 2가 리턴됩니다. SQLERRD(3)의 값은 다음과 같이 판별됩니다.

1. 확약 가능 갱신이 작업 단위에 대한 연결에서 수행될 수 있습니다.

   이는 다음 중 하나를 만족시키는 경우에 발생합니다.

   - SQLERRD(4)가 값 1을 갖습니다.
   - 다음 사항을 모두 만족시킵니다.

- SQLERRD(4)가 값 3 또는 5를 갖습니다.
- 확약 가능 갱신이 가능한 분산 작업 단위를 지원하지 않는 어플리케이션 서버와의 연결이 존재하지 않습니다.
- 다음 사항 중 하나를 만족시킵니다.
  - 첫 번째 확약 가능 갱신이 보호 연결을 사용하는 연결에 대해 수행되고, 로컬 데이터베이스에 대해 수행되거나 ARD 프로그램과의 연결에 대해 수행됩니다.
  - 확약 제어하에서 갱신 가능 로컬 파일이 열려 있습니다.
  - 보호 연결을 사용하는 열린 갱신 가능 DDM 파일이 있습니다.
  - 2단계 API 확약 제어 자원이 있습니다.
  - 확약 가능 갱신이 수행되지 않았습니다.

• 다음 사항을 모두 만족시킵니다.
- SQLERRD(4)가 값 4를 갖습니다.
- 확약 가능 갱신이 가능한 분산 작업 단위를 지원하지 않는 어플리케이션 서버와의 다른 연결이 존재하지 않습니다.
- 첫 번째 확약 가능 갱신이 이 연결에서 수행되거나 확약 가능 갱신이 일어나지 않은 상태입니다.
- 보호 연결을 사용하는 열린 갱신 가능 DDM 파일이 없습니다.
- 확약 제어에서 갱신 가능 논리 파일이 열려 있지 않습니다.
- 2단계 API 확약 제어 자원이 있습니다.

2. 확약 가능 갱신이 이 작업 단위의 연결에서 수행될 수 없습니다.

   이는 다음 중 하나를 만족시키는 경우에 발생합니다.

   • SQLERRD(4)가 값 2를 갖습니다.

   • SQLERRD(4)가 값 3 또는 5를 갖거나 다음 중 하나를 만족시킵니다.
   - 리모트 작업 단위만을 지원하는 갱신 가능 어플리케이션 서버와의 연결이 존재합니다.
   - 첫 번째 확약 가능 갱신이 비보호 연결을 사용하는 연결에 대해 수행됩니다.

   • SQLERRD(4)가 값 4를 갖고 다음 중 하나를 만족시킵니다.
   - 리모트 작업 단위만을 지원하는 갱신 가능 어플리케이션 서버와의 연결이 존재합니다.
   - 첫 번째 확약 가능 갱신이 이 연결에서 수행되지 않습니다.
   - 보호 연결을 사용하는 열린 갱신 가능 DDM 파일이 있습니다.
   - 확약 제어에서 갱신 가능 파일이 열려 있습니다.
   - 2단계 API 확약 제어 자원이 있습니다.

다음 표는 리모트 작업 단위만을 지원하는 어플리케이션 서버에 대해 갱신 가능 연결이 있고 첫 번째 확약 가능 갱신이 발생할 때, SQLERRD(4) 값을 기본으로 SQLERRD(3) 값이 판별되는 방법을 요약한 것입니다.

표 40. SQLERRD(3) 값 판별 요약

SQLERRD(4)	갱신 가능 리모트 작업 단위 어플리케이션 서버와의 연결 존재 여부	첫 번째 확약 가능 갱신이 발생하는 곳*	SQLERRD(3)
1	--	--	1
2	--	--	2
3	예	--	2
3	아니오	갱신 없음	1
3	아니오	1 단계	2
3	아니오	이 연결	1
3	아니오	2 단계	1
4	예	--	2
4	아니오	갱신 없음	1
4	아니오	1 단계	2
4	아니오	이 연결	1
4	아니오	2 단계	2
5	예	--	2
5	아니오	갱신 없음	1
5	아니오	1 단계	2
5	아니오	이 연결	1
5	아니오	2 단계	1

* 이 열의 용어들은 다음과 같이 정의됩니다.

- *갱신 없음*은 확약 가능 갱신이 수행되지 않았고, 보호 연결을 사용하는 갱신을 위해 열린 DDM 파일이 없었으며, 갱신을 위해 열린 로컬 파일이 없고, 확약 제어 API가 등록되지 않았음을 나타냅니다.
- *1단계*는 첫 번째 확약 가능 갱신이 비보호 연결을 사용해서 수행되었거나 비보호 연결을 사용한 갱신에 대해 DDM 파일이 열렸음을 나타냅니다.
- *2단계*는 확약 가능 갱신이 2단계 분산 작업 단위 어플리케이션 서버에서 수행되었고, DDM 파일이 보호 연결을 사용하여 갱신하기 위해 열렸으며, 확약 제어 API가 등록되었거나 확약 제어하에서 갱신될 수 있도록 로컬 파일이 열렸음을 나타냅니다.

SQLERRD(4)가 값 3, 4 또는 5(ARD 프로그램에 기초하여)이고 SQLERRD(3) 값이 2일 때, 연결에 대한 확약 가능 갱신을 위한 시도가 있으면, 작업 단위가 롤백 요구 상태로 들어가게 됩니다. 작업 단위가 롤백 요구 상태에 있으면, ROLLBACK문만이 허용됩니다. 다른 명령문은 모두 SQLCODE -918을 야기시킵니다.

분산 작업 단위 연결 고려사항

분산 작업 단위 어플리케이션에서 연결할 때에는 여러 사항을 고려해야 합니다. 이 섹션에서는 몇 가지 설계상의 주의점을 제시합니다.

- 작업 단위가 하나 이상의 어플리케이션 서버에서 갱신 작업을 수행하고, 확약 제어가 사용되면 갱신이 수행될 모든 연결이 확약 제어를 통해 작성됩니다. 연결이 확약 제어를 통해 만들어지지 않고 후에 확약 가능 갱신이 수행되면, 해당 작업 단위에 대한 읽기 전용 연결이 생기기 쉽습니다.
- 로컬 파일, DDM 파일 및 확약 제어 API 자원과 같은 다른 비SQL 확약 자원은 연결의 갱신 가능성과 읽기 전용 상태에 영향을 줍니다.
- 확약 제어를 사용하여 분산 작업 단위를 지원하지 않는 어플리케이션 서버에 연결하는 경우(예를 들어, TCP/IP를 사용한 V4R5 iSeries), 이 연결은 갱신 가능하거나 읽기 전용입니다. 연결이 갱신 가능할 경우, 이는 갱신만 가능한 연결입니다.

연결 종료

리모트 연결이 자원을 사용하므로, 앞으로 사용되지 않을 연결은 가능한 한 빨리 종료시켜야 합니다. 연결은 내재적 또는 명시적으로 종료될 수 있습니다. 언제 연결이 내재적으로 종료되는지는 380 페이지의 『디폴트 활성 그룹에 대한 내재적 연결 관리』와 380 페이지의 『비디폴트 활성 그룹에 대한 내재적 연결 관리』를 참조하십시오.

DISCONNECT문을 사용하거나 성공적인 COMMIT문 다음에 RELEASE문을 위치시켜 연결을 명시적으로 종료시킬 수 있습니다. DISCONNECT문은 비보호 연결 또는 로컬 연결을 사용하는 연결과 함께일 경우에만 사용될 수 있습니다. DISCONNECT문이 실행되면 연결이 종료됩니다. RELEASE문은 보호 또는 비보호 연결과 사용될 수 있습니다. RELEASE문이 실행될 때 연결은 종료되지 않고 해제 상태에 놓이게 됩니다. 해제 상태의 연결도 여전히 사용할 수 있습니다. 해당 연결은 COMMIT문이 성공적으로 실행되어야 종료됩니다. 해제된 상태의 연결은 ROLLBACK이나 실패한 COMMIT문으로는 종료되지 않습니다.

리모트 SQL 연결이 설정되면, DDM 네트워크 연결(APPC 대화 또는 TCP/IP 연결)이 사용됩니다. SQL 연결 종료시, 네트워크 연결은 비사용 상태에 있거나 삭제되었을 수 있습니다. 네트워크 연결이 삭제될지 또는 비사용 상태에 놓일지는 DDMCNV 작업 속성에 따라 달라집니다. 작업 속성 값이 *KEEP이며 연결이 다른 iSeries 서버에 대해 이루어지면, 연결은 사용되지 않습니다. 작업 속성 값이 *DROP이고 연결이 다른 iSeries 서버에 대해 이루어지면, 연결이 제거됩니다. 연결이 iSeries가 아닌 서버에 대해 이루어지면, 연결은 항상 제거됩니다. 다음 상황에서는 *DROP이 바람직합니다.

- 비사용 연결을 유지보수하는 비용이 너무 높고 연결이 상대적으로 빨리 사용되지 않는 경우.
- 여러 프로그램을 함께 실행시킬 때, 어떤 프로그램은 RUW 연결 관리로 컴파일되고 다른 프로그램은 DUW 연결 관리로 컴파일되는 경우. 리모트 위치에 대한 RUW 연결 관리로 컴파일된 프로그램 실행 시도는 보호 연결이 존재할 때 실패합니다.
- DDM 또는 DRDA를 사용하여 보호 대화로 수행할 경우. 비사용 보호 대화에 대한 확약 및 롤백은 추가 오버헤드를 만들 수 있습니다.

RCLDDMCNV(DDM 연결 재생) 명령을 사용하여 확약 경계에 있는 사용되지 않은 모든 연결을 종료할 수 있습니다.

분산 작업 단위

분산 작업 단위(DUW)를 사용하면 동일한 작업 단위 내의 복수 어플리케이션 서버에 액세스할 수 있습니다. 각 SQL문은 단 하나의 어플리케이션 서버에 액세스할 수 있습니다. 분산 작업 단위를 사용하면, 단일 작업 단위 내의 복수 어플리케이션 서버에서 변경이 확약되거나 롤백될 수 있습니다.

분산 작업 단위 연결 관리

CONNECT, SET CONNECTION, DISCONNECT 및 RELEASE문은 DUW 환경에서 연결을 관리하는데 사용됩니다. 분산 작업 단위 CONNECT는 프로그램이 디폴트인 RDBCNNMTH(*DUW)로 사전컴파일되었을 때 실행됩니다. 이 유형의 CONNECT문은 기존 연결을 해제시키지는 않지만 대신 이전의 연결을 휴지 상태에 놓이게 합니다. CONNECT문에서 지정된 관계형 데이터베이스가 현재 연결이 됩니다. CONNECT문은 새로운 연결 시작을 위해서만 사용될 수 있습니다. 기존 연결간의 전환을 위해서는 SET CONNECTION문을 사용해야 합니다. 연결이 시스템 자원을 사용하므로 계속 필요하지 않을 때에는 종료시켜야 합니다. RELEASE나 DISCONNECT문으로 연결을 종료시킬 수 있습니다. 연결을 종료시키려면 성공적인 확약 다음에 RELEASE문이 와야 합니다.

다음은 확약 제어를 사용하는 DUW 환경에서 실행되는 C 프로그램의 예입니다.

```
....
 EXEC SQL WHENEVER SQLERROR GO TO done;
 EXEC SQL WHENEVER NOT FOUND GO TO done;
....
 EXEC SQL
   DECLARE C1 CURSOR WITH HOLD FOR
       SELECT PARTNO, PRICE
            FROM PARTS
            WHERE SITES_UPDATED = 'N'
            FOR UPDATE OF SITES_UPDATED;
/*    Connect to the systems      */
 EXEC SQL CONNECT TO LOCALSYS;
EXEC SQL CONNECT TO SYSB;
EXEC SQL CONNECT TO SYSC;
/* Make the local system the current connection */
    EXEC SQL SET CONNECTION LOCALSYS;
/* Open the cursor */
 EXEC SQL OPEN C1;
```

그림 12. 분산 작업 단위 프로그램의 예 (1/4)

```
while (SQLCODE==0)
   {
    /* Fetch the first row */
    EXEC SQL FETCH C1 INTO :partnumber,:price;
    /* Update the row which indicates that the updates have been
       propagated to the other sites */
    EXEC SQL UPDATE PARTS SET SITES_UPDATED='Y'
                WHERE CURRENT OF C1;
    /* Check if the part data is on SYSB */
    if ((partnumber > 10) && (partnumber < 100))
       {
        /* Make SYSB the current connection and update the price */
        EXEC SQL SET CONNECTION SYSB;
         EXEC SQL UPDATE PARTS
                     SET PRICE=:price
                     WHERE PARTNO=:partnumber;
        }
```

그림 12. 분산 작업 단위 프로그램의 예 (2/4)

```
     /* Check if the part data is on SYSC */
     if ((partnumber > 50) && (partnumber < 200))
        {
         /* Make SYSC the current connection and update the price */
         EXEC SQL SET CONNECTION SYSC;
         EXEC SQL UPDATE PARTS
                     SET PRICE=:price
                     WHERE PARTNO=:partnumber;
        }
     /* Commit the changes made at all 3 sites */
EXEC SQL COMMIT;
     /* Set the current connection to local so the next row
        can be fetched */
    EXEC SQL SET CONNECTION LOCALSYS;
    }
done:
```

그림 12. 분산 작업 단위 프로그램의 예 (3/4)

```
EXEC SQL WHENEVER SQLERROR CONTINUE;
/* Release the connections that are no longer being used */
EXEC SQL RELEASE SYSB;
EXEC SQL RELEASE SYSC;
/* Close the cursor */
EXEC SQL CLOSE C1;
/* Do another commit which will end the released connections.
   The local connection is still active because it was not
   released. */
EXEC SQL COMMIT;
...
```

그림 12. 분산 작업 단위 프로그램의 예 (4/4)

이 프로그램에는 3개의 활동중인 어플리케이션 서버(로컬 시스템인 LOCALSYS와 2개의 리모트 시스템 SYSB와 SYSC)가 있습니다. SYSB와 SYSC도 분산 작업 단위와 2단계 확약을 지원합니다. 초기에 각 어플리케이션 서버가 트랜잭션에 개입되므로 CONNECT문을 사용하여 모든 연결이 활동 상태가 됩니다. DUW를 사용할 때, CONNECT문은 이전 연결을 해제하지는 않지만 대신에 이전 연결을 휴지 상태에 놓이게 합니다. 모든 어플리케이션 서버가 연결되고 나면 SET CONNECTION문을 사용하여 로컬 연결이 현재 연결이 됩니다. 그런 후, 커서가 열리고 자료의 첫 번째 행이 페치(fetch)됩니다. 그런 다음 어떤 어플리케이션 서버에서 자료가 갱신될 것인지가 결정됩니다. SYSB가 갱신되어야 할 경우, SET CONNECTION문에 의해 SYSB가 현재 연결이 된 후 갱신이 실행됩니다. SYSC에 대해서도 같은 작업이 수행됩니다. 그리고 변경이 확약됩니다. 2단계 확약이 사용되므로 로컬 시스템과 두 리모트 시스템에서 변경이 확약됩니다. 커서가 WITH HOLD로 선언되었으므로 확약 이후에도 열린 상태로 있게 됩니다. 그러면 현재 연결이 로컬 시스템에 대한 것으로 변경되어 다음 행이 페치될 수 있습니다. 이러한 일련의 페치, 갱신, 확약은 자료가 모두 처리될 때까지 반복됩니다. 모든 자료가 페치되고 나면 양쪽 리모트 시스템에 대한 연결이 해제됩니다. 이들은 보호 연결을 사용하므로 단절할 수 없습니다. 연결이 해제된 후에는 확약이 발생되어 연결이 종료됩니다. 로컬 시스템은 계속 연결된 상태로 처리를 계속합니다.

연결 상태 체크

읽기 전용 연결을 가질 수 있는 환경에서 실행중일 때에는 확약 가능 갱신을 수행하기 전에 연결 상태를 체크해야 합니다. 이것은 작업 단위가 롤백 요구 상태로 들어가지 않게 해줍니다. 다음의 COBOL로 작성된 예에서는 연결 상태를 체크하는 방법을 보여줍니다.

```
      ...
    EXEC SQL
        SET CONNECTION SYS5
END-EXEC.
       ...
  * Check if the connection is updateable.
       EXEC SQL CONNECT END-EXEC.
  * If connection is updateable, update sales information otherwise
  * inform the user.
       IF SQLERRD(3) = 1 THEN
    EXEC SQL
            INSERT INTO SALES_TABLE
                VALUES(:SALES-DATA)
          END-EXEC
         ELSE
          DISPLAY 'Unable to update sales information at this time'.
  ...
```

그림 13. 연결 상태 체크의 예

커서 및 준비된 명령문

커서와 준비된 명령문은 컴파일 단위에서 연결까지로 범위가 지정됩니다. 컴파일 단위로 범위가 지정되면 따로 컴파일된 프로그램에서 호출된 프로그램은 커서 또는 호출 프로그램에 의해 열리거나 준비된 명령문을 사용할 수 없습니다. 연결로 범위가 지정되면 프로그램 내의 각 연결이 각각 고유의 커서 또는 준비된 명령문에 대한 인스턴스를 가집니다.

다음의 분산 작업 단위의 예에서는 동일한 커서명이 어떻게 두 개의 다른 연결에서 열리고 그 결과 커서 C1이 두 인스턴스를 갖는지를 보여줍니다.

```
.....
 EXEC SQL DECLARE C1 CURSOR FOR
             SELECT * FROM CORPDATA.EMPLOYEE;
 /* Connect to local and open C1 */
 EXEC SQL CONNECT TO LOCALSYS;
 EXEC SQL OPEN C1;
 /* Connect to the remote system and open C1 */
 EXEC SQL CONNECT TO SYSA;
 EXEC SQL OPEN C1;
 /* Keep processing until done */
 while (NOT_DONE) {
    /* Fetch a row of data from the local system */
 EXEC SQL SET CONNECTION LOCALSYS;
    EXEC SQL FETCH C1 INTO :local_emp_struct;
    /* Fetch a row of data from the remote system */
    EXEC SQL SET CONNECTION SYSA;
    EXEC SQL FETCH C1 INTO :rmt_emp_struct;
    /* Process the data */
    .....
 }
 /* Close the cursor on the remote system */
 EXEC SQL CLOSE C1;
 /* Close the cursor on the local system */
 EXEC SQL SET CONNECTION LOCALSYS;
 EXEC SQL CLOSE C1;
 .....
```

그림 14. DUW 프로그램 커서의 예

어플리케이션 리퀘스터 드라이버 프로그램

DRDA를 구현하는 제품에 의해 제공되는 데이터베이스 액세스를 위해 iSeries용 DB2 UDB는 SQL 요구를 처리하는 iSeries용 DB2 UDB 어플리케이션 리퀘스터에 나감 프로그램을 작성하기 위한 인터페이스를 제공합니다. 이러한 나감 프로그램을 **어플리케이션 리퀘스터 드라이버**라고 합니다. 서버는 다음의 조작 중에 ARD 프로그램을 호출합니다.

- 관계형 데이터베이스(RDB) 매개변수가 ARD 프로그램의 해당 RDB명에 일치할 때 CRTSQLPKG나 CRTSQLxxx 명령을 사용하여 패키지 작성이 수행되는 동안
- 현재 연결이 ARD 프로그램에 해당하는 RDB명에 대한 것일 때 SQL문을 처리하는 동안

이러한 호출로 ARD 프로그램은 리모트 관계형 데이터베이스에 SQL문과 해당 명령문에 대한 정보를 전달하고 결과를 다시 시스템으로 리턴시킵니다. 시스템은 어플리케이션이나 사용자에게 결과를 리턴시킵니다. ARD 프로그램에 의해 액세스되는 관계형 데이터베이스로의 액세스는 부적절한 환경에서 DRDA 어플리케이션 서버에 액세스하는 것과 비슷하게 보입니다. 그러나, 모든 DRDA 기능이 ARD 환경에서 지원되는 것은 아닙니다. 지원되지 않는 기능의 예는 대형 오브젝트(LOB)와 긴 암호(암호 문구)입니다.

어플리케이션 리퀘스터 드라이버 프로그램에 대한 자세한 내용은 OS/400 파일 API를 참조하십시오.

문제점 처리

분산 데이터베이스 기능에 대한 오류 정보를 캡처하고 보고하기 위한 주요 전략을 **FFDC(first failure data capture)**라고 합니다. FFDC 지원의 목적은 APAR(Authorized Program Analysis Report)이 작성될 수 있는 OS/400 시스템의 DDM 구성 요소에서 발견된 오류에 대해 정확한 정보를 제공하는 것입니다. 이 기능을 이용하여, 키 구조와 DDM 자료 스트림이 자동적으로 스풀 파일에 덤프됩니다. 오류 정보의 처음 1024바이트도 시스템 오류 기록부에 기록됩니다. 첫 번째 오류 발생에 대한 오류 정보의 이러한 자동 덤프는 사용자가 같은 실패를 다시 보고하도록 해서는 안된다는 것을 의미합니다. FFDC는 OS/400 DDM 구성요소의 어플리케이션 리퀘스터와 어플리케이션 서버 기능 모두에서 활동합니다. 그러나, 기록될 FFDC 자료에 대한 시스템 값 QSFWERRLOG는 *LOG로 설정됩니다.

주: 음수 SQLCODE가 모두 덤프되는 것은 아닙니다. APAR을 작성하는데 사용될 수 있는 코드만 덤프됩니다. 분산 관계형 데이터베이스 조작의 문제점 처리에 대한 자세한 내용은 *Distributed Database Problem Determination Guide*를 참조하십시오.

SQL 오류가 검출되면 해당 SQLSTATE가 있는 SQLCODE가 SQLCA에 리턴됩니다. 이러한 코드에 대한 자세한 정보는 iSeries Information Center의 SQL 메세지 및 코드 주제를 참조하십시오.

DRDA 저장 프로시듀어 고려사항

iSeries DRDA 서버는 저장 프로시듀어로부터 하나 이상의 결과 세트 리턴을 지원합니다. 그러나, V5R1에서는 이 피처를 저장 프로시듀어 결과 세트를 지원하는 iSeries가 아닌 클라이언트에서만 사용할 수 있도록 서버에서만 가능합니다.

V5R2에서 iSeries 클라이언트측 지원이 SQL용 CLI 인터페이스를 사용하는 어플리케이션에 추가되었습니다. 그러나 V5R1 iSeries 서버가 저장된 프로시듀어 결과 세트를 V5R2 iSeries 클라이언트로 리턴하도록 하려면 PTF를 적용해야 합니다. 릴리스, 날짜 또는 수정 번호별로 정렬된 안내문 목록을 참조하려면 PTF Cover Letter Database  를 참조하십시오.

결과 세트는 SQL SELECT문과 연관된 하나 이상의 SQL 커서를 열어서 저장 프로시듀어에 생성할 수 있습니다. 그 외에 최대 하나의 배열로 이루어진 결과 세트도 리턴될 수 있습니다. 결과 세트를 리턴하는 저장 프로시듀어 작성에 대한 자세한 정보는 SQL 참조서의 SET RESULT SETS, CREATE PROCEDURE(SQL) 및 CREATE PROCEDURE(External)문의 설명을 참조하십시오. DRDA로 저장 프로시듀어 사용에 대한 일반 정보는 분산 데이터베이스 프로그래밍을 참조하십시오.

부록 A. iSeries용 DB2 UDB 샘플 표

이 부록에는 이 안내서와 SQL 참조서에서 참조되고 사용된 샘플 표가 들어 있습니다. 표 외에 표를 작성하는 SQL문도 있습니다. 표 작성에 대해 알려면 18 페이지의『표 작성 및 사용』을 참조하십시오.

그룹으로서, 표에는 사원, 부서, 프로젝트 및 활동을 설명하는 정보가 포함됩니다. 이 정보는 DB2 UDB 조회 관리자 및 SQL 개발 킷 사용권 프로그램의 피처 중 일부를 보여주는 샘플 어플리케이션으로 구성됩니다. 모든 예는 표가 이름이 CORPDATA(회사 자료용)인 스키마에 있는 것으로 가정합니다.

저장 프로시듀어는 이러한 모든 표를 작성하기 위한 DDL문과 이를 수록할 INSERT 문이 들어 있는 시스템의 일부로 제공됩니다. 프로시듀어는 프로시듀어에 대한 호출에 지정된 스키마를 작성합니다. 이것은 SQL 외부 저장 프로시듀어이므로 대화식 SQL 및 iSeries Navigator를 비롯한 어느 SQL 인터페이스에서나 호출할 수 있습니다. 프로시듀어를 호출하려면 (*SAMPLE*은 작성하려는 스키마임) 다음 명령문을 발행하십시오.

```
CALL QSYS.CREATE_SQL_SAMPLE ('SAMPLE')
```

스키마명은 대문자로 지정되어야 합니다. 스키마는 아직 존재하지 않아야 합니다.

표는 다음과 같습니다.

- 396 페이지의『부서 표(DEPARTMENT)』
- 397 페이지의『사원 표(EMPLOYEE)』
- 399 페이지의『사원 사진 표(EMP_PHOTO)』
- 400 페이지의『사원 이력서 표(EMP_RESUME)』
- 401 페이지의『프로젝트에 대한 사원 활동 표(EMPPROJACT)』
- 404 페이지의『프로젝트 표(PROJECT)』
- 406 페이지의『프로젝트 활동 표(PROJACT)』
- 409 페이지의『활동 표(ACT)』
- 410 페이지의『클래스 스케줄 표(CL_SCHED)』
- 410 페이지의『인 트레이 표(IN_TRAY)』

색인, 별명 및 보기가 이러한 여러 표에 대해 작성됩니다. 보기 정의는 여기에 포함되지 않았습니다.

첫 번째 세트에 관련되지 않은 3개의 다른 표도 작성됩니다.

- 412 페이지의『구성 표(ORG)』
- 413 페이지의『직원 표(STAFF)』

- 414 페이지의『판매 표(SALES)』

주:

1. 이 샘플 표에서 물음표(?)는 널(null)값을 나타냅니다.

부서 표(DEPARTMENT)

부서 표는 회사의 각 부서를 설명하고 그 부서장과 그 부서의 보고를 받는 상위 부서를 식별합니다. 부서 표는 다음의 CREATE TABLE문과 ALTER TABLE문을 사용하여 작성됩니다.

```
CREATE TABLE DEPARTMENT
      (DEPTNO    CHAR(3)           NOT NULL,
       DEPTNAME  VARCHAR(36)       NOT NULL,
       MGRNO     CHAR(6),
       ADMRDEPT  CHAR(3)           NOT NULL
       LOCATION  CHAR(16),
       PRIMARY KEY (DEPTNO))

ALTER TABLE DEPARTMENT
      ADD FOREIGN KEY ROD (ADMRDEPT)
          REFERENCES DEPARTMENT
          ON DELETE CASCADE
```

다음의 외부 키가 나중에 추가됩니다.

```
ALTER TABLE DEPARTMENT
      ADD FOREIGN KEY RDE (MGRNO)
          REFERENCES EMPLOYEE
          ON DELETE SET NULL
```

다음의 색인이 작성됩니다.

```
CREATE UNIQUE INDEX XDEPT1
       ON DEPARTMENT (DEPTNO)

CREATE INDEX XDEPT2
       ON DEPARTMENT (MGRNO)

CREATE INDEX XDEPT3
       ON DEPARTMENT (ADMRDEPT)
```

다음의 별명이 표에 대해 작성됩니다.

```
CREATE ALIAS DEPT FOR DEPARTMENT
```

다음 표에 열의 내용이 나와 있습니다.

표 41. 부서 표의 열

열 이름	설명
DEPTNO	부서 번호 또는 ID
DEPTNAME	부서의 일반 활동을 설명하는 이름
MGRNO	부서장의 사원 번호(EMPNO)

표 41. 부서 표의 열 (계속)

열 이름	설명
ADMRDEPT	해당 부서의 보고를 받는 부서(DEPTNO). 최상위 부서는 자기 부서로 보고합니다.
LOCATION	부서의 위치.

DEPARTMENT의 전체 리스트는 『DEPARTMENT』를 참조하십시오.

DEPARTMENT

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	?
B01	PLANNING	000020	A00	?
C01	INFORMATION CENTER	000030	A00	?
D01	DEVELOPMENT CENTER	?	A00	?
D11	MANUFACTURING SYSTEMS	000060	D01	?
D21	ADMINISTRATION SYSTEMS	000070	D01	?
E01	SUPPORT SERVICES	000050	A00	?
E11	OPERATIONS	000090	E01	?
E21	SOFTWARE SUPPORT	000100	E01	?
F22	BRANCH OFFICE F2	?	E01	?
G22	BRANCH OFFICE G2	?	E01	?
H22	BRANCH OFFICE H2	?	E01	?
I22	BRANCH OFFICE I2	?	E01	?
J22	BRANCH OFFICE J2	?	E01	?

사원 표(EMPLOYEE)

사원 표에는 사원 번호로 모든 사원이 식별되며 기본적인 인사 정보가 나열됩니다. 사원 표는 다음의 CREATE TABLE문과 ALTER TABLE문을 사용하여 작성됩니다.

```
CREATE TABLE EMPLOYEE
      (EMPNO       CHAR(6)          NOT NULL,
          FIRSTNME VARCHAR(12)      NOT NULL,
          MIDINIT     CHAR(1)       NOT NULL,
       LASTNAME    VARCHAR(15)      NOT NULL,
       WORKDEPT    CHAR(3)                  ,
       PHONENO     CHAR(4)                  ,
       HIREDATE    DATE                     ,
       JOB         CHAR(8)                  ,
       EDLEVEL     SMALLINT         NOT NULL,
       SEX         CHAR(1)                  ,
       BIRTHDATE   DATE                     ,
       SALARY      DECIMAL(9,2)             ,
```

```
       BONUS       DECIMAL(9,2)            ,
       COMM        DECIMAL(9,2)
       PRIMARY KEY (EMPNO))

ALTER TABLE EMPLOYEE
      ADD FOREIGN KEY RED (WORKDEPT)
      REFERENCES DEPARTMENT
      ON DELETE SET NULL

ALTER TABLE EMPLOYEE
      ADD CONSTRAINT NUMBER
      CHECK (PHONENO >= '0000' AND PHONENO <= '9999')
```

다음의 색인이 작성됩니다.

```
CREATE UNIQUE INDEX XEMP1
       ON EMPLOYEE (EMPNO)

CREATE INDEX XEMP2
       ON EMPLOYEE (WORKDEPT)
```

다음의 별명이 표에 대해 작성됩니다.

```
CREATE ALIAS EMP FOR EMPLOYEE
```

다음 표에 열의 내용이 나와 있습니다.

열 이름	설명
EMPNO	사원 번호
FIRSTNME	사원 이름
MIDINIT	사원 이름 가운데 중간 약자
LASTNAME	사원 이름 가운데 성
WORKDEPT	사원이 일하는 부서의 ID
PHONENO	사원 전화번호
HIREDATE	입사일
JOB	사원이 맡은 직무
EDLEVEL	정규 교육 이수 기간(연 수)
SEX	사원의 성별(M 또는 F)
BIRTHDATE	생년월일
SALARY	연봉(달러)
BONUS	연간 보너스(달러)
COMM	연간 수당(달러)

EMPLOYEE의 전체 리스트는 399 페이지의 『EMPLOYEE』를 참조하십시오.

EMPLOYEE

EMP NO	FIRST NAME	MID INIT	LASTNAME	WORK DEPT	PHONE NO	HIRE DATE	JOB	ED LEVEL	SEX	BIRTH DATE	SAL-ARY	BONUS	COMM
000010	CHRISTINE	I	HAAS	A00	3978	1965-01-01	PRES	18	F	1933-08-24	52750	1000	4220
000020	MICHAEL	L	THOMPSON	B01	3476	1973-10-10	MANAGER	18	M	1948-02-02	41250	800	3300
000030	SALLY	A	KWAN	C01	4738	1975-04-05	MANAGER	20	F	1941-05-11	38250	800	3060
000050	JOHN	B	GEYER	E01	6789	1949-08-17	MANAGER	16	M	1925-09-15	40175	800	3214
000060	IRVING	F	STERN	D11	6423	1973-09-14	MANAGER	16	M	1945-07-07	32250	500	2580
000070	EVA	D	PULASKI	D21	7831	1980-09-30	MANAGER	16	F	1953-05-26	36170	700	2893
000090	EILEEN	W	HENDERSON	E11	5498	1970-08-15	MANAGER	16	F	1941-05-15	29750	600	2380
000100	THEODORE	Q	SPENSER	E21	0972	1980-06-19	MANAGER	14	M	1956-12-18	26150	500	2092
000110	VINCENZO	G	LUCCHESSI	A00	3490	1958-05-16	SALESREP	19	M	1929-11-05	46500	900	3720
000120	SEAN		O'CONNELL	A00	2167	1963-12-05	CLERK	14	M	1942-10-18	29250	600	2340
000130	DOLORES	M	QUINTANA	C01	4578	1971-07-28	ANALYST	16	F	1925-09-15	23800	500	1904
000140	HEATHER	A	NICHOLLS	C01	1793	1976-12-15	ANALYST	18	F	1946-01-19	28420	600	2274
000150	BRUCE		ADAMSON	D11	4510	1972-02-12	DESIGNER	16	M	1947-05-17	25280	500	2022
000160	ELIZABETH	R	PIANKA	D11	3782	1977-10-11	DESIGNER	17	F	1955-04-12	22250	400	1780
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1978-09-15	DESIGNER	16	M	1951-01-05	24680	500	1974
000180	MARILYN	S	SCOUTTEN	D11	1682	1973-07-07	DESIGNER	17	F	1949-02-21	21340	500	1707
000190	JAMES	H	WALKER	D11	2986	1974-07-26	DESIGNER	16	M	1952-06-25	20450	400	1636
000200	DAVID		BROWN	D11	4501	1966-03-03	DESIGNER	16	M	1941-05-29	27740	600	2217
000210	WILLIAM	T	JONES	D11	0942	1979-04-11	DESIGNER	17	M	1953-02-23	18270	400	1462
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29	DESIGNER	18	F	1948-03-19	29840	600	2387
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21	CLERK	14	M	1935-05-30	22180	400	1774
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05	CLERK	17	M	1954-03-31	28760	600	2301
000250	DANIEL	S	SMITH	D21	0961	1969-10-30	CLERK	15	M	1939-11-12	19180	400	1534
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11	CLERK	16	F	1936-10-05	17250	300	1380
000270	MARIA	L	PEREZ	D21	9001	1980-09-30	CLERK	15	F	1953-05-26	27380	500	2190
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24	OPERATOR	17	F	1936-03-28	26250	500	2100
000290	JOHN	R	PARKER	E11	4502	1980-05-30	OPERATOR	12	M	1946-07-09	15340	300	1227
000300	PHILIP	X	SMITH	E11	2095	1972-06-19	OPERATOR	14	M	1936-10-27	17750	400	1420
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12	OPERATOR	12	F	1931-04-21	15900	300	1272
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07	FILEREP	16	M	1932-08-11	19950	400	1596
000330	WING		LEE	E21	2103	1976-02-23	FILEREP	14	M	1941-07-18	25370	500	2030
000340	JASON	R	GOUNOT	E21	5698	1947-05-05	FILEREP	16	M	1926-05-17	23840	500	1907
200010	DIAN	J	HEMMINGER	A00	3978	1965-01-01	SALESREP	18	F	1933-08-14	46500	1000	4220
200120	GREG		ORLANDO	A00	2167	1972-05-05	CLERK	14	M	1942-10-18	29250	600	2340
200140	KIM	N	NATZ	C01	1793	1976-12-15	ANALYST	18	F	1946-01-19	28420	600	2274
200170	KIYOSHI		YAMAMOTO	D11	2890	1978-09-15	DESIGNER	16	M	1951-01-05	24680	500	1974
200220	REBA	K	JOHN	D11	0672	1968-08-29	DESIGNER	18	F	1948-03-19	29840	600	2387
200240	ROBERT	M	MONTEVERDE	D21	3780	1979-12-05	CLERK	17	M	1954-03-31	28760	600	2301
200280	EILEEN	R	SCHWARTZ	E11	8997	1967-03-24	OPERATOR	17	F	1936-03-28	26250	500	2100
200310	MICHELLE	F	SPRINGER	E11	3332	1964-09-12	OPERATOR	12	F	1931-04-21	15900	300	1272
200330	HELENA		WONG	E21	2103	1976-02-23	FIELDREP	14	F	1941-07-18	25370	500	2030
200340	ROY	R	ALONZO	E21	5698	1947-05-05	FIELDREP	16	M	1926-05-17	23840	500	1907

사원 사진 표(EMP_PHOTO)

사원 사진 표에는 사원 번호에 의해 저장된 사원의 사진이 들어 있습니다. 사원 사진 표는 다음의 CREATE TABLE문과 ALTER TABLE문을 사용하여 작성됩니다.

```
CREATE TABLE EMP_PHOTO
            (EMPNO CHAR(6) NOT NULL,
             PHOTO_FORMAT VARCHAR(10) NOT NULL,
             PICTURE BLOB(100K),
             EMP_ROWID CHAR(40) NOT NULL DEFAULT '',
             PRIMARY KEY (EMPNO,PHOTO_FORMAT))

ALTER TABLE EMP_PHOTO
           ADD COLUMN DL_PICTURE DATALINK(1000)
               LINKTYPE URL NO LINK CONTROL
```

```
ALTER TABLE EMP_PHOTO
             ADD FOREIGN KEY (EMPNO)
             REFERENCES EMPLOYEE
             ON DELETE RESTRICT
```

다음의 색인이 작성됩니다.

```
CREATE UNIQUE INDEX XEMP_PHOTO
             ON EMP_PHOTO (EMPNO,PHOTO_FORMAT)
```

다음 표에 열의 내용이 나와 있습니다.

열 이름	설명
EMPNO	사원 번호
PHOTO_FORMAT	PICTURE에 저장된 이미지의 형식
PICTURE	사진 이미지
EMP_ROWID	현재 사용되지 않는 고유 행 ID

EMP_PHOTO의 전체 리스트는 『EMP_PHOTO』를 참조하십시오.

EMP_PHOTO

EMPNO	PHOTO_FORMAT	PICTURE	EMP_ROWID
000130	bitmap	?	
000130	gif	?	
000140	bitmap	?	
000140	gif	?	
000150	bitmap	?	
000150	gif	?	
000190	bitmap	?	
000190	gif	?	

사원 이력서 표(EMP_RESUME)

사원 이력서 표에는 사원 번호에 의해 저장된 사원의 이력서가 들어 있습니다. 사원 이력서 표는 다음의 CREATE TABLE문과 ALTER TABLE문을 사용하여 작성됩니다.

```
CREATE TABLE EMP_RESUME
            (EMPNO CHAR(6) NOT NULL,
             RESUME_FORMAT VARCHAR(10) NOT NULL,
             RESUME CLOB(5K),
             EMP_ROWID CHAR(40) NOT NULL DEFAULT '',
             PRIMARY KEY (EMPNO,RESUME_FORMAT))

ALTER TABLE EMP_RESUME
           ADD COLUMN DL_RESUME DATALINK(1000)
               LINKTYPE URL NO LINK CONTROL
```

```
ALTER TABLE EMP_RESUME
            ADD FOREIGN KEY (EMPNO)
            REFERENCES EMPLOYEE
            ON DELETE RESTRICT
```

다음의 색인이 작성됩니다.

```
CREATE UNIQUE INDEX XEMP_RESUME
            ON EMP_RESUME (EMPNO,RESUME_FORMAT)
```

다음 표에 열의 내용이 나와 있습니다.

열 이름	설명
EMPNO	사원 번호
RESUME_FORMAT	RESUME에 저장된 텍스트의 형식
RESUME	이력서
EMP_ROWID	현재 사용되지 않는 고유 행 ID

EMP_RESUME의 전체 리스트는 『EMP_RESUME』를 참조하십시오.

EMP_RESUME

EMPNO	RESUME_FORMAT	RESUME	EMP_ROWID
000130	ascii	?	
000130	html	?	
000140	ascii	?	
000140	html	?	
000150	ascii	?	
000150	html	?	
000190	ascii	?	
000190	html	?	

프로젝트에 대한 사원 활동 표(EMPPROJACT)

프로젝트에 대한 사원 활동 표에는 각 프로젝트에 나열된 각 활동을 수행하는 사원이 식별됩니다. 활동 참여도(풀타임 또는 파트타임)에 대한 사원 레벨과 스케줄 또한 표에 있습니다. 프로젝트에 대한 사원 활동 표는 다음의 CREATE TABLE문과 ALTER TABLE문을 사용하여 작성됩니다.

```
CREATE TABLE EMPPROJACT
      (EMPNO     CHAR(6)          NOT NULL,
       PROJNO    CHAR(6)          NOT NULL,
       ACTNO     SMALLINT         NOT NULL,
       EMPTIME   DECIMAL(5,2)             ,
       EMSTDATE  DATE                     ,
       EMENDATE  DATE                     )
```

```
ALTER TABLE EMPPROJACT
              ADD FOREIGN KEY REPAPA (PROJNO, ACTNO, EMSTDATE)
              REFERENCES PROJACT
              ON DELETE RESTRICT
```

다음의 별명이 표에 대해 작성됩니다.

```
CREATE ALIAS EMPACT FOR EMPPROJACT

CREATE ALIAS EMP_ACT FOR EMPPROJACT
```

다음 표에 열의 내용이 나와 있습니다.

표 42. 프로젝트에 대한 사원 활동 표의 열

열 이름	설명
EMPNO	사원 ID 번호
PROJNO	사원에게 할당된 프로젝트의 PROJNO
ACTNO	사원에게 할당된 프로젝트 내의 활동
EMPTIME	EMSTDATE에서 EMENDATE까지 프로젝트에 소요되는 사원의 풀타임 비율(0.00에서 1.00사이)
EMSTDATE	활동 시작 날짜
EMENDATE	활동 완료 날짜

EMPPROJACT의 전체 리스트는 『EMPPROJACT』를 참조하십시오.

EMPPROJACT

EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
000010	AD3100	10	.50	1982-01-01	1982-07-01
000070	AD3110	10	1.00	1982-01-01	1983-02-01
000230	AD3111	60	1.00	1982-01-01	1982-03-15
000230	AD3111	60	.50	1982-03-15	1982-04-15
000230	AD3111	70	.50	1982-03-15	1982-10-15
000230	AD3111	80	.50	1982-04-15	1982-10-15
000230	AD3111	180	.50	1982-10-15	1983-01-01
000240	AD3111	70	1.00	1982-02-15	1982-09-15
000240	AD3111	80	1.00	1982-09-15	1983-01-01
000250	AD3112	60	1.00	1982-01-01	1982-02-01
000250	AD3112	60	.50	1982-02-01	1982-03-15
000250	AD3112	60	1.00	1983-01-01	1983-02-01
000250	AD3112	70	.50	1982-02-01	1982-03-15
000250	AD3112	70	1.00	1982-03-15	1982-08-15
000250	AD3112	70	.25	1982-08-15	1982-10-15
000250	AD3112	80	.25	1982-08-15	1982-10-15
000250	AD3112	80	.50	1982-10-15	1982-12-01
000250	AD3112	180	.50	1982-08-15	1983-01-01

EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
000260	AD3113	70	.50	1982-06-15	1982-07-01
000260	AD3113	70	1.00	1982-07-01	1983-02-01
000260	AD3113	80	1.00	1982-01-01	1982-03-01
000260	AD3113	80	.50	1982-03-01	1982-04-15
000260	AD3113	180	.50	1982-03-01	1982-04-15
000260	AD3113	180	1.00	1982-04-15	1982-06-01
000260	AD3113	180	1.00	1982-06-01	1982-07-01
000270	AD3113	60	.50	1982-03-01	1982-04-01
000270	AD3113	60	1.00	1982-04-01	1982-09-01
000270	AD3113	60	.25	1982-09-01	1982-10-15
000270	AD3113	70	.75	1982-09-01	1982-10-15
000270	AD3113	70	1.00	1982-10-15	1983-02-01
000270	AD3113	80	1.00	1982-01-01	1982-03-01
000270	AD3113	80	.50	1982-03-01	1982-04-01
000030	IF1000	10	.50	1982-06-01	1983-01-01
000130	IF1000	90	1.00	1982-10-01	1983-01-01
000130	IF1000	100	.50	1982-10-01	1983-01-01
000140	IF1000	90	.50	1982-10-01	1983-01-01
000030	IF2000	10	.50	1982-01-01	1983-01-01
000140	IF2000	100	1.00	1982-01-01	1982-03-01
000140	IF2000	100	.50	1982-03-01	1982-07-01
000140	IF2000	110	.50	1982-03-01	1982-07-01
000140	IF2000	110	.50	1982-10-01	1983-01-01
000010	MA2100	10	.50	1982-01-01	1982-11-01
000110	MA2100	20	1.00	1982-01-01	1983-03-01
000010	MA2110	10	1.00	1982-01-01	1983-02-01
000200	MA2111	50	1.00	1982-01-01	1982-06-15
000200	MA2111	60	1.00	1982-06-15	1983-02-01
000220	MA2111	40	1.00	1982-01-01	1983-02-01
000150	MA2112	60	1.00	1982-01-01	1982-07-15
000150	MA2112	180	1.00	1982-07-15	1983-02-01
000170	MA2112	60	1.00	1982-01-01	1983-06-01
000170	MA2112	70	1.00	1982-06-01	1983-02-01
000190	MA2112	70	1.00	1982-01-01	1982-10-01
000190	MA2112	80	1.00	1982-10-01	1983-10-01
000160	MA2113	60	1.00	1982-07-15	1983-02-01
000170	MA2113	80	1.00	1982-01-01	1983-02-01
000180	MA2113	70	1.00	1982-04-01	1982-06-15
000210	MA2113	80	.50	1982-10-01	1983-02-01
000210	MA2113	180	.50	1982-10-01	1983-02-01
000050	OP1000	10	.25	1982-01-01	1983-02-01

EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
000090	OP1010	10	1.00	1982-01-01	1983-02-01
000280	OP1010	130	1.00	1982-01-01	1983-02-01
000290	OP1010	130	1.00	1982-01-01	1983-02-01
000300	OP1010	130	1.00	1982-01-01	1983-02-01
000310	OP1010	130	1.00	1982-01-01	1983-02-01
000050	OP2010	10	.75	1982-01-01	1983-02-01
000100	OP2010	10	1.00	1982-01-01	1983-02-01
000320	OP2011	140	.75	1982-01-01	1983-02-01
000320	OP2011	150	.25	1982-01-01	1983-02-01
000330	OP2012	140	.25	1982-01-01	1983-02-01
000330	OP2012	160	.75	1982-01-01	1983-02-01
000340	OP2013	140	.50	1982-01-01	1983-02-01
000340	OP2013	170	.50	1982-01-01	1983-02-01
000020	PL2100	30	1.00	1982-01-01	1982-09-15

프로젝트 표(PROJECT)

프로젝트 표는 업무가 현재 진행되고 있는 각 프로젝트를 설명합니다. 각 행에 들어 있는 자료에는 프로젝트 번호, 이름, 책임자 및 일정 등이 포함됩니다. 프로젝트 표는 다음의 CREATE TABLE문과 ALTER TABLE문을 사용하여 작성됩니다.

```
CREATE TABLE PROJECT
      (PROJNO    CHAR(6)        NOT NULL,
       PROJNAME  VARCHAR(24)    NOT NULL DEFAULT,
       DEPTNO    CHAR(3)        NOT NULL,
       RESPEMP   CHAR(6)        NOT NULL,
       PRSTAFF   DECIMAL(5,2)           ,
       PRSTDATE  DATE                   ,
       PRENDATE  DATE                   ,
       MAJPROJ   CHAR(6)                ,
       PRIMARY KEY (PROJNO))

ALTER TABLE PROJECT
             ADD FOREIGN KEY (DEPTNO)
             REFERENCES DEPARTMENT
             ON DELETE RESTRICT

ALTER TABLE PROJECT
             ADD FOREIGN KEY (RESPEMP)
             REFERENCES EMPLOYEE
             ON DELETE RESTRICT

ALTER TABLE PROJECT
             ADD FOREIGN KEY  RPP (MAJPROJ)
             REFERENCES PROJECT
             ON DELETE CASCADE
```

다음의 색인이 작성됩니다.

```
CREATE UNIQUE INDEX XPROJ1
            ON PROJECT (PROJNO)

CREATE INDEX XPROJ2
            ON PROJECT (RESPEMP)
```

다음의 별명이 표에 대해 작성됩니다.

```
CREATE ALIAS PROJ FOR PROJECT
```

아래의 표는 열의 내용을 보여줍니다.

열 이름	설명
PROJNO	프로젝트 번호
PROJNAME	프로젝트명
DEPTNO	프로젝트를 담당한 부서의 부서 번호
RESPEMP	프로젝트를 담당한 사원의 사원 번호
PRSTAFF	예상 평균 사원 수
PRSTDATE	프로젝트의 예상 시작일
PRENDATE	프로젝트의 예상 종료일
MAJPROJ	서브 프로젝트에 대한 프로젝트 관리 번호

PROJECT의 전체 리스트는 『PROJECT』를 참조하십시오.

PROJECT

PROJNO	PROJNAME	DEPTNO	RESPEMP	PRSTAFF	PRSTDATE	PRENDATE	MAJPROJ
AD3100	ADMIN SERVICES	D01	000010	6.5	1982-01-01	1983-02-01	?
AD3110	GENERAL ADMIN SYSTEMS	D21	000070	6	1982-01-01	1983-02-01	AD3100
AD3111	PAYROLL PROGRAMMING	D21	000230	2	1982-01-01	1983-02-01	AD3110
AD3112	PERSONNEL PROGRAMMING	D21	000250	1	1982-01-01	1983-02-01	AD3110
AD3113	ACCOUNT PROGRAMMING	D21	000270	2	1982-01-01	1983-02-01	AD3110
IF1000	QUERY SERVICES	C01	000030	2	1982-01-01	1983-02-01	?
IF2000	USER EDUCATION	C01	000030	1	1982-01-01	1983-02-01	?
MA2100	WELD LINE AUTOMATION	D01	000010	12	1982-01-01	1983-02-01	?
MA2110	W L PROGRAMMING	D11	000060	9	1982-01-01	1983-02-01	MA2100
MA2111	W L PROGRAM DESIGN	D11	000220	2	1982-01-01	1982-12-01	MA2110
MA2112	W L ROBOT DESIGN	D11	000150	3	1982-01-01	1982-12-01	MA2110

PROJNO	PROJNAME	DEPTNO	RESPEMP	PRSTAFF	PRSTDATE	PRENDATE	MAJPROJ
MA2113	W L PROD CONT PROGS	D11	000160	3	1982-02-15	1982-12-01	MA2110
OP1000	OPERATION SUPPORT	E01	000050	6	1982-01-01	1983-02-01	?
OP1010	OPERATION	E11	000090	5	1982-01-01	1983-02-01	OP1000
OP2000	GEN SYSTEMS SERVICES	E01	000050	5	1982-01-01	1983-02-01	?
OP2010	SYSTEMS SUPPORT	E21	000100	4	1982-01-01	1983-02-01	OP2000
OP2011	SCP SYSTEMS SUPPORT	E21	000320	1	1982-01-01	1983-02-01	OP2010
OP2012	APPLICATIONS SUPPORT	E21	000330	1	1982-01-01	1983-02-01	OP2010
OP2013	DB/DC SUPPORT	E21	000340	1	1982-01-01	1983-02-01	OP2010
PL2100	WELD LINE PLANNING	B01	000020	1	1982-01-01	1982-09-15	MA2100

프로젝트 활동 표(PROJACT)

프로젝트 활동 표는 업무에서 현재 진행되고 있는 각 프로젝트를 설명합니다. 각 행에 들어 있는 자료에는 프로젝트 번호, 활동 번호 및 일정 등이 포함됩니다. 프로젝트 활동 표는 다음의 CREATE TABLE문과 ALTER TABLE문을 사용하여 작성됩니다.

```
CREATE TABLE PROJACT
             (PROJNO CHAR(6) NOT NULL,
              ACTNO SMALLINT NOT NULL,
              ACSTAFF DECIMAL(5,2),
              ACSTDATE DATE NOT NULL,
              ACENDATE DATE ,
              PRIMARY KEY (PROJNO, ACTNO, ACSTDATE))

ALTER TABLE PROJACT
              ADD FOREIGN KEY RPAP (PROJNO)
              REFERENCES PROJECT
              ON DELETE RESTRICT
```

다음의 외부 키가 나중에 추가됩니다.

```
ALTER TABLE PROJACT
            ADD FOREIGN KEY RPAA (ACTNO)
                REFERENCES ACT
              ON DELETE RESTRICT
```

다음의 색인이 작성됩니다.

```
CREATE UNIQUE INDEX XPROJAC1
              ON PROJACT (PROJNO, ACTNO, ACSTDATE)
```

아래의 표는 열의 내용을 보여줍니다.

열 이름	설명
PROJNO	프로젝트 번호
ACTNO	활동 번호
ACSTAFF	예상 평균 사원 수
ACSTDATE	활동 시작 날짜
ACENDATE	활동 종료 날짜

PROJACT의 전체 리스트는 『PROJACT』를 참조하십시오.

PROJACT

PROJNO	ACTNO	ACSTAFF	ACSTDATE	ACENDATE
AD3100	10	?	1982-01-01	?
AD3110	10	?	1982-01-01	?
AD3111	60	?	1982-01-01	?
AD3111	60	?	1982-03-15	?
AD3111	70	?	1982-03-15	?
AD3111	80	?	1982-04-15	?
AD3111	180	?	1982-10-15	?
AD3111	70	?	1982-02-15	?
AD3111	80	?	1982-09-15	?
AD3112	60	?	1982-01-01	?
AD3112	60	?	1982-02-01	?
AD3112	60	?	1983-01-01	?
AD3112	70	?	1982-02-01	?
AD3112	70	?	1982-03-15	?
AD3112	70	?	1982-08-15	?
AD3112	80	?	1982-08-15	?
AD3112	80	?	1982-10-15	?
AD3112	180	?	1982-08-15	?
AD3113	70	?	1982-06-15	?
AD3113	70	?	1982-07-01	?
AD3113	80	?	1982-01-01	?
AD3113	80	?	1982-03-01	?
AD3113	180	?	1982-03-01	?
AD3113	180	?	1982-04-15	?
AD3113	180	?	1982-06-01	?
AD3113	60	?	1982-03-01	?
AD3113	60	?	1982-04-01	?
AD3113	60	?	1982-09-01	?

PROJNO	ACTNO	ACSTAFF	ACSTDATE	ACENDATE
AD3113	70	?	1982-09-01	?
AD3113	70	?	1982-10-15	?
IF1000	10	?	1982-06-01	?
IF1000	90	?	1982-10-01	?
IF1000	100	?	1982-10-01	?
IF2000	10	?	1982-01-01	?
IF2000	100	?	1982-01-01	?
IF2000	100	?	1982-03-01	?
IF2000	110	?	1982-03-01	?
IF2000	110	?	1982-10-01	?
MA2100	10	?	1982-01-01	?
MA2100	20	?	1982-01-01	?
MA2110	10	?	1982-01-01	?
MA2111	50	?	1982-01-01	?
MA2111	60	?	1982-06-15	?
MA2111	40	?	1982-01-01	?
MA2112	60	?	1982-01-01	?
MA2112	180	?	1982-07-15	?
MA2112	70	?	1982-06-01	?
MA2112	70	?	1982-01-01	?
MA2112	80	?	1982-10-01	?
MA2113	60	?	1982-07-15	?
MA2113	80	?	1982-01-01	?
MA2113	70	?	1982-04-01	?
MA2113	80	?	1982-10-01	?
MA2113	180	?	1982-10-01	?
OP1000	10	?	1982-01-01	?
OP1010	10	?	1982-01-01	?
OP1010	130	?	1982-01-01	?
OP2010	10	?	1982-01-01	?
OP2011	140	?	1982-01-01	?
OP2011	150	?	1982-01-01	?
OP2012	140	?	1982-01-01	?
OP2012	160	?	1982-01-01	?
OP2013	140	?	1982-01-01	?
OP2013	170	?	1982-01-01	?
PL2100	30	?	1982-01-01	?

활동 표(ACT)

활동 표는 각 활동을 설명합니다. 활동 표는 다음의 CREATE TABLE문을 사용하여 작성됩니다.

```
CREATE TABLE ACT
             (ACTNO SMALLINT NOT NULL,
              ACTKWD CHAR(6) NOT NULL,
              ACTDESC VARCHAR(20) NOT NULL,
              PRIMARY KEY (ACTNO))
```

다음의 색인이 작성됩니다.

```
CREATE UNIQUE INDEX XACT1
             ON ACT (ACTNO)

CREATE UNIQUE INDEX XACT2
             ON ACT (ACTKWD)
```

아래의 표는 열의 내용을 보여줍니다.

열 이름	설명
ACTNO	활동 번호
ACTKWD	활동에 대한 키워드
ACTDESC	활동의 설명

ACT의 전체 리스트는 『ACT』를 참조하십시오.

ACT

ACTNO	ACTKWD	ACTDESC
10	MANAGE	MANAGE/ADVISE
20	ECOST	ESTIMATE COST
30	DEFINE	DEFINE SPECS
40	LEADPR	LEAD PROGRAM/DESIGN
50	SPECS	WRITE SPECS
60	LOGIC	DESCRIBE LOGIC
70	CODE	CODE PROGRAMS
80	TEST	TEST PROGRAMS
90	ADMQS	ADM QUERY SYSTEM
100	TEACH	TEACH CLASSES
110	COURSE	DEVELOP COURSES
120	STAFF	PERS AND STAFFING
130	OPERAT	OPER COMPUTER SYS
140	MAINT	MAINT SOFTWARE SYS
150	ADMSYS	ADM OPERATING SYS
160	ADMDB	ADM DATA BASES

170	ADMDC	ADM DATA COMM
180	DOC	DOCUMENT

클래스 스케줄 표(CL_SCHED)

클래스 스케줄 표는 각 클래스들, 해당 클래스의 시작 시간과 종료 시간 및 해당 클래스 코드를 설명합니다. 클래스 스케줄 표는 다음의 CREATE TABLE문으로 작성됩니다.

```
CREATE TABLE CL_SCHED
      (CLASS_CODE          CHAR(7),
       "DAY"               SMALLINT,
       STARTING            TIME,
       ENDING              TIME)
```

다음 표에 열의 내용이 나와 있습니다.

열 이름	설명
CLASS_CODE	클래스 코드(room:teacher)
DAY	4일 일정 중에서 일 수
STARTING	클래스 시작 시간
ENDING	클래스 종료 시간

CL_SCHED의 전체 리스트는 『CL_SCHED』를 참조하십시오.

CL_SCHED

CLASS_CODE	DAY	STARTING	ENDING
042:BF	4	12:10:00	14:00:00
553:MJA	1	10:30:00	11:00:00
543:CWM	3	09:10:00	10:30:00
778:RES	2	12:10:00	14:00:00
044:HD	3	17:12:30	18:00:00

인 트레이 표(IN_TRAY)

인 트레이 표는 전자 수신함의 내용(메세지를 수신한 시간소인, 메세지를 송신한 사람의 사용자 ID, 메세지 내용)을 설명합니다. 인 트레이 표는 다음의 CREATE TABLE 문으로 작성됩니다.

```
CREATE TABLE IN_TRAY
    (RECEIVED            TIMESTAMP,
     SOURCE              CHAR(8),
     SUBJECT             CHAR(64),
     NOTE_TEXT           VARCHAR(3000))
```

다음 표에 열의 내용이 나와 있습니다.

열 이름	설명
RECEIVED	수신된 날짜와 시간
SOURCE	메모를 보낸 사람의 사용자 ID
SUBJECT	메모에 대한 간단한 설명
NOTE_TEXT	주석

IN_TRAY의 전체 리스트는 『IN_TRAY』를 참조하십시오.

IN_TRAY

RECEIVED	SOURCE	SUBJECT	NOTE_TEXT
1988-12-25-17.12.30.000000	BADAMSON	FWD: Fantastic year! 4th Quarter Bonus.	To: JWALKER Cc: QUINTANA, NICHOLLS Jim, Looks like our hard work has paid off. I have some good beer in the fridge if you want to come over to celebrate a bit. Delores and Heather, are you interested as well? Bruce <Forwarding from ISTERN> Subject: FWD: Fantastic year! 4th Quarter Bonus. To: Dept_D11 Congratulations on a job well done. Enjoy this year's bonus. Irv <Forwarding from CHAAS> Subject: Fantastic year! 4th Quarter Bonus. To: All_Managers Our 4th quarter results are in. We pulled together as a team and exceeded our plan! I am pleased to announce a bonus this year of 18%. Enjoy the holidays. Christine Haas

RECEIVED	SOURCE	SUBJECT	NOTE_TEXT
1988-12-23-08.53.58.000000	ISTERN	FWD: Fantastic year! 4th Quarter Bonus.	To: Dept_D11 Congratulations on a job well done. Enjoy this year's bonus. Irv <Forwarding from CHAAS> Subject: Fantastic year! 4th Quarter Bonus. To: All_Managers Our 4th quarter results are in. We pulled together as a team and exceeded our plan! I am pleased to announce a bonus this year of 18%. Enjoy the holidays. Christine Haas
1988-12-22-14.07.21.136421	CHAAS	Fantastic year! 4th Quarter Bonus.	To: All_Managers Our 4th quarter results are in. We pulled together as a team and exceeded our plan! I am pleased to announce a bonus this year of 18%. Enjoy the holidays. Christine Haas

구성 표(ORG)

조직 표는 회사의 조직을 설명합니다. 구성 표는 다음의 CREATE TABLE문을 사용하여 작성됩니다.

```
CREATE TABLE ORG
             (DEPTNUMB SMALLINT NOT NULL,
              DEPTNAME VARCHAR(14),
              MANAGER SMALLINT,
              DIVISION VARCHAR(10),
              LOCATION VARCHAR(13))
```

다음 표에 열의 내용이 나와 있습니다.

열 이름	설명
DEPTNUMB	부서 번호
DEPTNAME	부서명
MANAGER	부서의 관리자 수
DIVISION	부서의 과
LOCATION	부서의 위치

ORG의 전체 리스트는 413 페이지의 『ORG』를 참조하십시오.

ORG

DEPTNUMB	DEPTNAME	MANAGER	DIVISION	LOCATION
10	본점	160	공동	뉴욕
15	뉴잉글랜드	50	동부	보스턴
20	중대서양	10	동부	워싱턴
38	남대서양	30	동부	애틀랜타
42	5대호	100	중서부	시카고
51	평야	140	중서부	달라스
66	태평양	270	서부	샌프란시스코
84	산간	290	서부	덴버

직원 표(STAFF)

직원 표는 사원들을 설명합니다. 직원 표는 다음의 CREATE TABLE문을 사용하여 작성됩니다.

```
CREATE TABLE STAFF
            (ID  SMALLINT NOT NULL,
             NAME VARCHAR(9),
             DEPT SMALLINT,
             JOB CHAR(5),
             YEARS SMALLINT,
             SALARY DECIMAL(7,2),
             COMM DECIMAL(7,2))
```

아래의 표는 열의 내용을 보여줍니다.

열 이름	설명
ID	사원 번호
NAME	사원명
DEPT	부서 번호
JOB	직책
YEARS	회사에서 일한 연도수
SALARY	사원의 연봉
COMM	사원의 커미션

STAFF의 전체 리스트는 『STAFF』를 참조하십시오.

STAFF

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
10	Sanders	20	Mgr	7	18357.50	?
20	Pernal	20	Sales	8	18171.25	612.45
30	Marenghi	38	Mgr	5	17506.75	?

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
40	O’Brien	38	Sales	6	18006.00	846.55
50	Hanes	15	Mgr	10	20659.80	?
60	Quigley	38	Sales	7	16508.30	650.25
70	Rothman	15	Sales	7	16502.83	1152.00
80	James	20	Clerk	?	13504.60	128.20
90	Koonitz	42	Sales	6	18001.75	1386.70
100	Plotz	42	Mgr	7	18352.80	?
110	Ngan	15	Clerk	5	12508.20	206.60
120	Naughton	38	Clerk	?	12954.75	180.00
130	Yamaguchi	42	Clerk	6	10505.90	75.60
140	Fraye	51	Mgr	6	21150.00	?
150	Williams	51	Sales	6	19456.50	637.65
160	Molinare	10	Mgr	7	22959.20	?
170	Kermisch	15	Clerk	4	12258.50	110.10
180	Abrahams	38	Clerk	3	12009.75	236.50
190	Sneider	20	Clerk	8	14252.75	126.50
200	Scoutten	42	Clerk	?	11508.60	84.20
210	Lu	10	Mgr	10	20010.00	?
220	Smith	51	Sales	7	17654.50	992.80
230	Lundquist	51	Clerk	3	13369.80	189.65
240	Daniels	10	Mgr	5	19260.25	?
250	Wheeler	51	Clerk	6	14460.00	513.30
260	Jones	10	Mgr	12	21234.00	?
270	Lea	66	Mgr	9	18555.50	?
280	Wilson	66	Sales	9	18674.50	811.50
290	Quill	84	Mgr	10	19818.00	?
300	Davis	84	Sales	5	15454.50	806.10
310	Graham	66	Sales	13	21000.00	200.30
320	Gonzales	66	Sales	4	16858.20	844.00
330	Burke	66	Clerk	1	10988.00	55.50
340	Edwards	84	Sales	7	17844.00	1285.00
3650	Gafney	84	Clerk	5	13030.50	188.00

판매 표(SALES)

판매 표는 각 판매 담당자에 대한 각 판매량을 설명합니다. 판매 표는 다음의 CREATE TABLE문을 사용하여 작성됩니다.

```
CREATE TABLE SALES
             (SALES_DATE DATE,
              SALES_PERSON VARCHAR(15),
              REGION VARCHAR(15),
              SALES INTEGER)
```

다음 표에 열의 내용이 나와 있습니다.

열 이름	설명
SALES_DATE	판매된 날짜
SALES_PERSON	판매한 담당자
REGION	판매한 지역
SALES	판매 수

SALES의 전체 리스트는 『SALES』를 참조하십시오.

SALES

SALES_DATE	SALES_PERSON	REGION	SALES
12/31/1995	LUCCHESSI	Ontario-South	1
12/31/1995	LEE	Ontario-South	3
12/31/1995	LEE	Quebec	1
12/31/1995	LEE	Manitoba	2
12/31/1995	GOUNOT	Quebec	1
03/29/1996	LUCCHESSI	Ontario-South	3
03/29/1996	LUCCHESSI	Quebec	1
03/29/1996	LEE	Ontario-South	2
03/29/1996	LEE	Ontario-North	2
03/29/1996	LEE	Quebec	3
03/29/1996	LEE	Manitoba	5
03/29/1996	GOUNOT	Ontario-South	3
03/29/1996	GOUNOT	Quebec	1
03/29/1996	GOUNOT	Manitoba	7
03/30/1996	LUCCHESSI	Ontario-South	1
03/30/1996	LUCCHESSI	Quebec	2
03/30/1996	LUCCHESSI	Manitoba	1
03/30/1996	LEE	Ontario-South	7
03/30/1996	LEE	Ontario-North	3
03/30/1996	LEE	Quebec	7
03/30/1996	LEE	Manitoba	4
03/30/1996	GOUNOT	Ontario-South	2
03/30/1996	GOUNOT	Quebec	18
03/30/1996	GOUNOT	Manitoba	1
03/31/1996	LUCCHESSI	Manitoba	1

SALES_DATE	SALES_PERSON	REGION	SALES
03/31/1996	LEE	Ontario-South	14
03/31/1996	LEE	Ontario-North	3
03/31/1996	LEE	Quebec	7
03/31/1996	LEE	Manitoba	3
03/31/1996	GOUNOT	Ontario-South	2
03/31/1996	GOUNOT	Quebec	1
04/01/1996	LUCCHESSI	Ontario-South	3
04/01/1996	LUCCHESSI	Manitoba	1
04/01/1996	LEE	Ontario-South	8
04/01/1996	LEE	Ontario-North	?
04/01/1996	LEE	Quebec	8
04/01/1996	LEE	Manitoba	9
04/01/1996	GOUNOT	Ontario-South	3
04/01/1996	GOUNOT	Ontario-North	1
04/01/1996	GOUNOT	Quebec	3
04/01/1996	GOUNOT	Manitoba	7

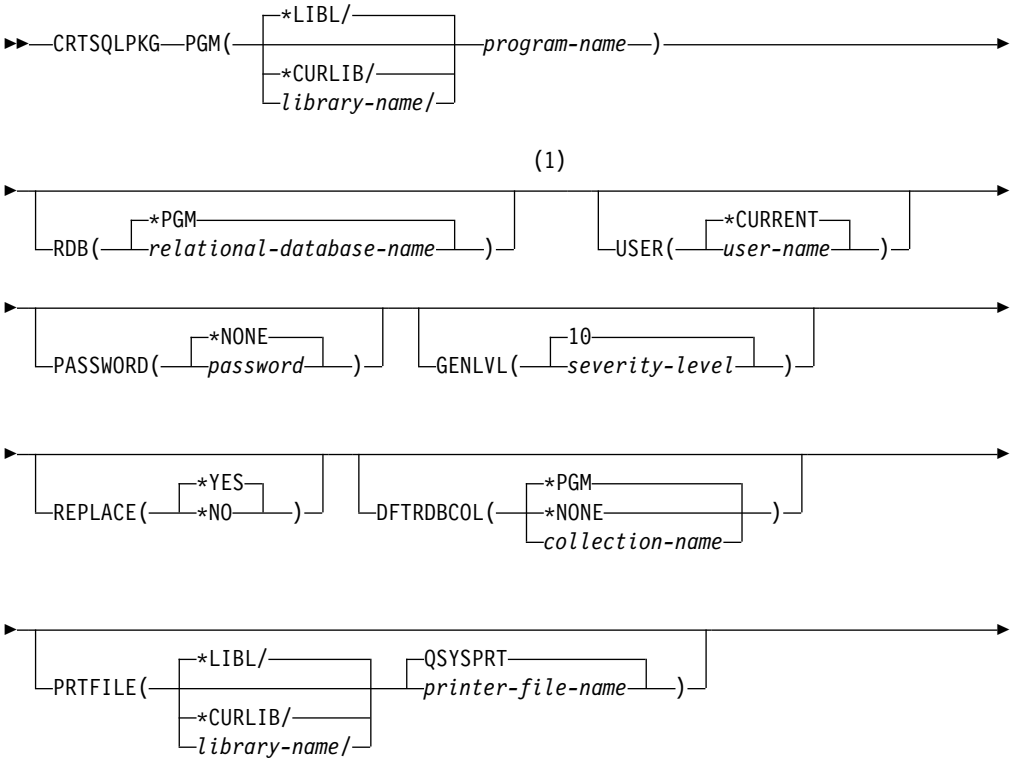
부록 B. iSeries용 DB2 UDB CL 명령 설명

이 부록에는 이 안내서와 SQL 참조서에서 참조되고 사용된 구문 다이어그램이 들어 있습니다.

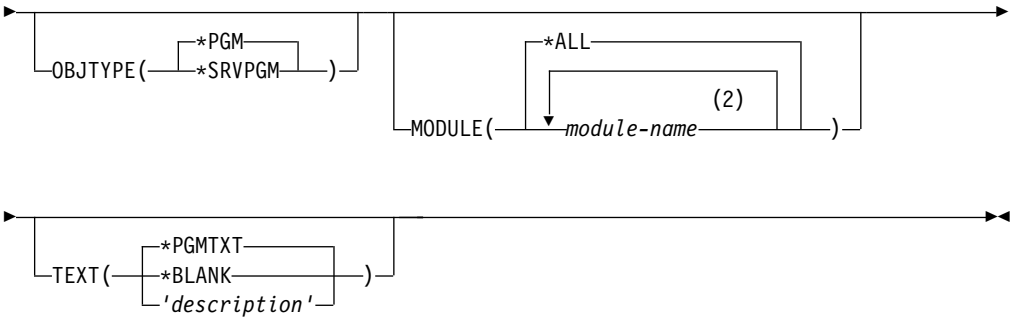
명령 설명은 다음의 주제를 참조하십시오.

CRTSQLPKG(구조화 조회 언어 패키지 작성) 명령

Job: B,I Pgm: B,I REXX: B,I Exec



CRTSQLPKG



주:

1 이 앞에 오는 모든 매개변수는 앞뒤 관계에 따라 지정할 수 있습니다.

2 최대 256 모듈이 지정될 수 있습니다.

목적:

구조화 조회 언어 패키지 작성(CRTSQLPKG) 명령은 기존의 분산 SQL 프로그램으로부터 관계형 데이터베이스의 SQL 패키지를 작성(또는 재작성)하기 위해 사용될 수 있습니다. 분산 SQL 프로그램은 CRTSQLxxx(여기서 xxx=C, CI, CBL, CBLI, FTN, PLI 또는 RPG 또는 RPGI) 명령에 RDB 매개변수를 지정하여 작성된 프로그램입니다.

매개변수:

PGM

SQL 패키지가 작성되는 프로그램의 규정화명을 지정합니다. 프로그램은 분산 SQL 프로그램이어야 합니다.

프로그램의 이름은 다음 값 중 하나로 규정될 수 있습니다.

***LIBL:** 일치되는 첫 번째 항목이 발견될 때까지 작업의 라이브러리 리스트에 있는 모든 라이브러리가 탐색됩니다.

***CURLIB:** 작업에 대한 현재 라이브러리가 탐색됩니다. 작업에 대한 현재 라이브러리로 지정된 라이브러리가 없으면 QGPL 라이브러리가 사용됩니다.

library-name: 탐색될 라이브러리의 이름을 지정하십시오.

program-name: 패키지가 작성된 프로그램의 이름을 지정하십시오.

RDB

SQL 패키지가 작성될 관계형 데이터베이스명을 지정합니다.

***PGM:** SQL 프로그램에 지정된 관계형 데이터베이스명이 사용됩니다. 관계형 데이터베이스명이 분산 SQL 프로그램의 RDB 매개변수에 지정됩니다.

relational-database-name: SQL 패키지가 작성될 관계형 데이터베이스명을 지정하십시오. 이 매개변수에 유효한 관계형 데이터베이스명을 보려면 WRKRDBDIRE(관계형 데이터베이스 디렉토리 항목에 대한 작업) 명령을 사용하십시오.

USER

대화를 시작할 때 리모트 시스템으로 송신된 사용자명을 지정합니다.

***CURRENT:** 현재 작업과 연관된 사용자명이 사용됩니다.

user-name: 어플리케이션 서버 작업에 사용되는 사용자명을 지정하십시오.

PASSWORD

리모트 시스템에서 사용될 암호를 지정합니다.

***NONE:** 암호가 송신되지 않습니다. 이 값이 지정되면 USER(*CURRENT) 또한 지정되어야 합니다.

password: USER 매개변수에 지정된 사용자명의 암호를 지정하십시오.

GENLVL

SQL 패키지 작성중 발견된 오류에 대해 허용된 최대 심각도 레벨을 지정합니다. 오류가 지정 레벨을 이상에서 발생하면, SQL 패키지가 작성되지 않습니다.

10: 디폴트 심각도 레벨은 10입니다.

severity-level: 최대 심각도 레벨을 지정하십시오. 유효한 값의 범위는 0-40 사이입니다.

REPLACE

기존의 패키지가 새로운 패키지로 대체되는지의 여부를 지정합니다. 이 매개변수에 대한 자세한 내용은 *CL 참조서*에서 부록 A, ″확장 매개변수 설명″에 있습니다.

***YES:** 이름이 동일한 기존의 SQL 패키지가 새로운 SQL 패키지로 대체됩니다.

***NO:** 동일한 이름을 가진 기존의 SQL 패키지가 새로운 SQL 패키지로 대체되지 않으며, 패키지가 지정 라이브러리에 이미 있는 경우에는 신규 SQL 패키지가 작성되지 않습니다.

DFTRDBCOL

규정되지 않은 표, 보기, 색인 및 SQL 패키지명에 대해 사용될 콜렉션명을 지정합니다. 이 매개변수는 패키지에 있는 정적 SQL문에만 적용됩니다.

***PGM:** SQL 프로그램에 지정된 콜렉션명이 사용됩니다. 디폴트 관계형 데이터베이스 콜렉션명은 분산 SQL 프로그램의 DFTRDBCOL 매개변수에 지정됩니다.

***NONE:** 표, 보기, 색인 및 SQL 패키지에 대해 규정되지 않은 이름은 프로그램 작성에 사용되는 CRTSQLxxx 명령의 OPTION 매개변수에 지정된 탐색 규칙을 사용합니다.

collection-name: 규정되지 않은 표, 보기, 색인 및 SQL 패키지에 사용되는 콜렉션명을 지정하십시오.

PRTFILE

SQL 패키지 오류 작성 리스트가 전달되는 프린터 파일의 규정화명을 지정합니다. SQL 패키지 작성중 오류가 검출되지 않는 경우에는 리스트가 작성되지 않습니다.

프린터 파일명은 다음의 라이브러리 값 중 하나로 규정될 수 있습니다.

***LIBL:** 일치되는 첫 번째 항목이 발견될 때까지 작업의 라이브러리 리스트에 있는 모든 라이브러리가 탐색됩니다.

***CURLIB:** 작업에 대한 현재 라이브러리가 탐색됩니다. 작업에 대한 현재 라이브러리로 지정된 라이브러리가 없으면 QGPL 라이브러리가 사용됩니다.

library-name: 탐색될 라이브러리의 이름을 지정하십시오.

QSYSPRT: 파일명이 지정되지 않으면, SQL 패키지 작성 오류 리스트가 IBM 제공 프린터 파일 QSYSPRT로 전달됩니다.

printer-file-name: SQL 패키지 작성 오류 리스트가 전달되는 프린터 파일의 이름을 지정하십시오.

OBJTYPE

SQL 패키지가 작성될 프로그램의 유형을 지정합니다.

***PGM:** PGM 매개변수에 지정된 프로그램에서 SQL 패키지를 작성합니다

***SRVPGM:** PGM 매개변수에 지정된 서비스 프로그램에서 SQL 패키지를 작성합니다.

MODULE

모듈의 리스트를 바운드 프로그램에 지정합니다.

***ALL:** 프로그램 내 각 모듈에 대한 SQL 패키지가 작성됩니다. 프로그램의 모듈에 SQL문이 없거나 모듈이 분산 모듈이 아닌 경우에는 오류 메세지가 전달됩니다.

주: CRTSQLPKG는 1024 이하의 모듈이 있는 프로그램을 처리할 수 있습니다.

module-name: SQL 패키지가 작성될 프로그램 내의 최대 256개 모듈명을 지정하십시오. SQL 패키지 작성에 필요한 모듈이 256 모듈 이상일 경우, 복수 CRTSQLPKG 명령을 사용해야 합니다.

동일한 프로그램에 복제 모듈명이 허용됩니다. 이 명령에 의해 프로그램의 각 모듈이 탐색되며 *ALL 또는 모듈명이 MODULE 매개변수에 지정된 경우, SQL 패키지를 작성할 것인지의 여부를 결정하기 위해 처리가 계속됩니다. 모듈이 SQL을 사용하여 작성되고 RDB 매개변수가 사전컴파일 명령에 지정된 경우, SQL 패키지가 모듈에 대해 작성됩니다. SQL 패키지는 바운드 프로그램의 모듈과 연관됩니다.

TEXT

SQL 패키지와 그 기능을 간단히 설명하는 텍스트를 지정합니다.

***PGMTXT:** SQL 패키지가 작성중인 프로그램의 텍스트가 사용됩니다.

***BLANK:** 텍스트가 지정되지 않습니다.

'description': 50자 이하의 텍스트를 어포스트로피로 묶어서 지정하십시오.

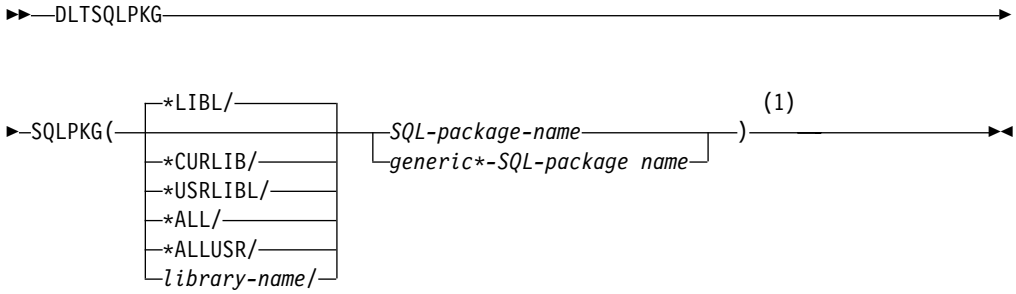
예:

```
CRTSQLPKG   PAYROLL  RDB(SYSTEMA)
  TEXT('Payroll Program')
```

이 명령은 분산 SQL 프로그램의 SQL 패키지를 관계형 데이터베이스 SYSTEMA에 작성합니다.

DLTSQLPKG(구조화 조회 언어 패키지 삭제) 명령

Job: B,I Pgm: B,I REXX: B,I Exec



주:

1 이 앞에 오는 모든 매개변수는 앞뒤 관계에 따라 지정할 수 있습니다.

목적:

구조화 조회 언어 패키지 삭제(DLTSQLPKG) 명령은 두 개 이상의 SQL 패키지를 삭제하기 위해 사용됩니다.

DLTSQLPKG는 로컬 명령이며 삭제될 SQL 패키지가 위치한 iSeries 시스템에서 사용되어야 합니다.

또한 iSeries 시스템인 리모트 시스템의 SQL 패키지를 삭제하려면, 리모트 시스템의 DLTSQLPKG 명령을 실행할 리모트 명령 제출(SBMRMTCMD) 명령을 사용하십시오

iSeries 시스템이 아닌 리모트 시스템에서는 SQL 패키지를 삭제하기 위하여 다음을 수행할 수 있습니다.

- CONNECT 및 DROP PACKAGE 작업을 실행하기 위한 대화식 SQL을 사용합니다.
- 리모트 시스템을 사인 온하여 그 시스템에 로컬 명령을 사용합니다.
- DROP PACKAGE SQL문이 있는 SQL 프로그램을 작성하고 실행합니다.

DLTSQLPKG

매개변수:

SQLPKG

삭제될 SQL 패키지의 규정화명을 지정합니다. 특정 또는 총칭 SQL 패키지명이 지정될 수 있습니다.

SQL 패키지명은 다음의 라이브러리 값 중 하나로 규정될 수 있습니다.

***LIBL:** 일치되는 첫 번째 항목이 발견될 때까지 작업의 라이브러리 리스트에 있는 모든 라이브러리가 탐색됩니다.

***CURLIB:** 작업에 대한 현재 라이브러리가 탐색됩니다. 작업에 대한 현재 라이브러리로 지정된 라이브러리가 없으면 QGPL 라이브러리가 사용됩니다.

***USRLIBL:** 작업 라이브러리 리스트의 사용자 부분에 있는 라이브러리만이 탐색됩니다.

***ALL:** QSYS를 비롯한 시스템의 모든 라이브러리가 탐색됩니다.

***ALLUSR:** 모든 사용자 라이브러리가 탐색됩니다. 다음을 제외한 문자 Q로 시작되지 않는 이름의 모든 라이브러리가 탐색됩니다.

```
#CGULIB     #DFULIB     #RPGLIB     #SEULIB
#COBLIB     #DSULIB     #SDALIB
```

IBM에서 다음의 Qxxx 라이브러리를 제공했더라도 여기에는 일반적으로 자주 변경되는 사용자 자료가 들어 갑니다. 따라서, 이러한 라이브러리들은 사용자 라이브러리로 간주되어 탐색됩니다.

```
QDSNX          QRCL          QUSRBRM          QUSRSYS
QGPL           QS36F         QUSRIJS          QUSRVxRxMx
QGPL38         QUSER38       QUSRINFSKR
QPFRDATA       QUSRADSM      QUSRRDARS
```

주: QUSRVxRxMx 양식의 다른 라이브러리명이 IBM에서 지원하는 각 릴리스에 대해 사용자에 의해 작성됩니다. VxRxMx는 라이브러리의 버전, 릴리스 및 수정판 레벨입니다.

library-name: 탐색될 라이브러리의 이름을 지정하십시오.

SQL-package-name: 삭제될 SQL 패키지의 이름을 지정하십시오.

generic-SQL-package-name:* 삭제될 SQL 패키지의 총칭명을 지정하십시오. 총칭명은 별표(*)가 뒤에 오는 두 개 이상의 문자 스트링입니다. 예를 들면, ABC*와 같이 사용됩니다. 총칭명이 지정된 경우, 총칭명으로 시작되는 모든 SQL 패키지명과 사용자의 권한은 없어집니다. 별표(*)에 총칭명(접두부)이 없다면, 시스템은 별표를 완전한 SQL 패키지명으로 간주합니다.

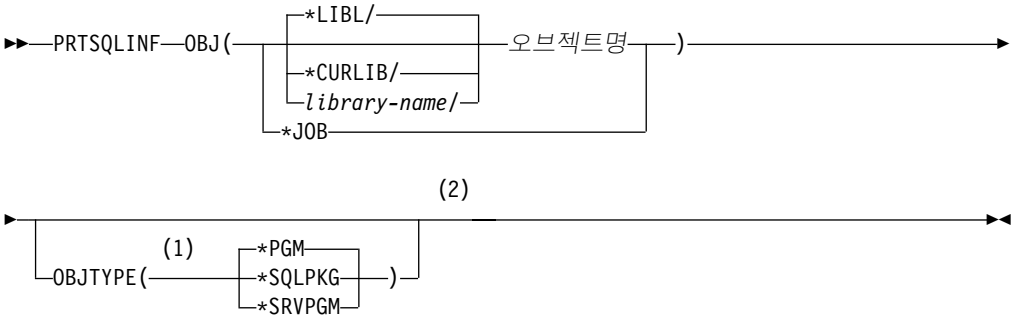
예:

```
DLTSQLPKG  SQLPKG(JONES)
```

이 명령은 SQL 패키지 JONES를 삭제합니다.

PRTSQLINF(SQL(구조화 조회 언어) 정보 인쇄) 명령

Job: B,I Pgm: B,I REXX: B,I Exec



주:

1 OBJTYPE 매개변수는 OBJ(*JOB)가 지정되면 허용되지 않습니다.

2 이 지점 전의 모든 매개변수는 위치 형식으로 지정할 수 있습니다.

목적:

PRTSQLINF(SQL(구조화 조회 언어) 정보 인쇄) 명령은 프로그램, SQL 패키지, 서비스 프로그램 또는 작업에 있는 SQL문에 대한 정보를 인쇄합니다. 이 정보에는 SQL문, 명령문 실행 중 사용된 액세스 계획 및 오브젝트에 대해 소스 멤버 사전컴파일 중이나 SQL문 실행 시에 정의된 명령 매개변수의 리스트가 들어 있습니다.

매개변수:

OBJ

SQL 정보를 인쇄하려는 오브젝트의 이름이나 작업의 SQL 정보가 인쇄될 것임을 나타내는 '*JOB'을 지정하십시오. 명명된 오브젝트는 프로그램, SQL 패키지 또는 서비스 프로그램이 될 수 있습니다.

오브젝트의 이름은 다음의 라이브러리 값 중 하나에 의해 규정할 수 있습니다.

***LIBL:** 일치되는 첫 번째 항목이 발견될 때까지 작업의 라이브러리 리스트에 있는 모든 라이브러리가 탐색됩니다.

***CURLIB:** 작업에 대한 현재 라이브러리가 탐색됩니다. 작업에 대한 현재 라이브러리로 지정된 라이브러리가 없으면 QGPL 라이브러리가 사용됩니다.

library-name: 탐색될 라이브러리의 이름을 지정하십시오.

object-name: 정보를 인쇄하려는 프로그램, SQL 패키지 또는 서비스 프로그램의 이름을 지정하십시오.

- ***JOB:** 현재 작업에 대한 SQL 정보가 인쇄될 것임을 나타냅니다.

PRTSQLINF

선택적 매개변수

OBJTYPE

오브젝트의 유형을 지정합니다.

- ***PGM:** 오브젝트는 프로그램입니다.
- ***SQLPKG:** 오브젝트는 SQL 패키지입니다.
- ***SRVPGM:** 오브젝트는 서비스 프로그램입니다.

예:

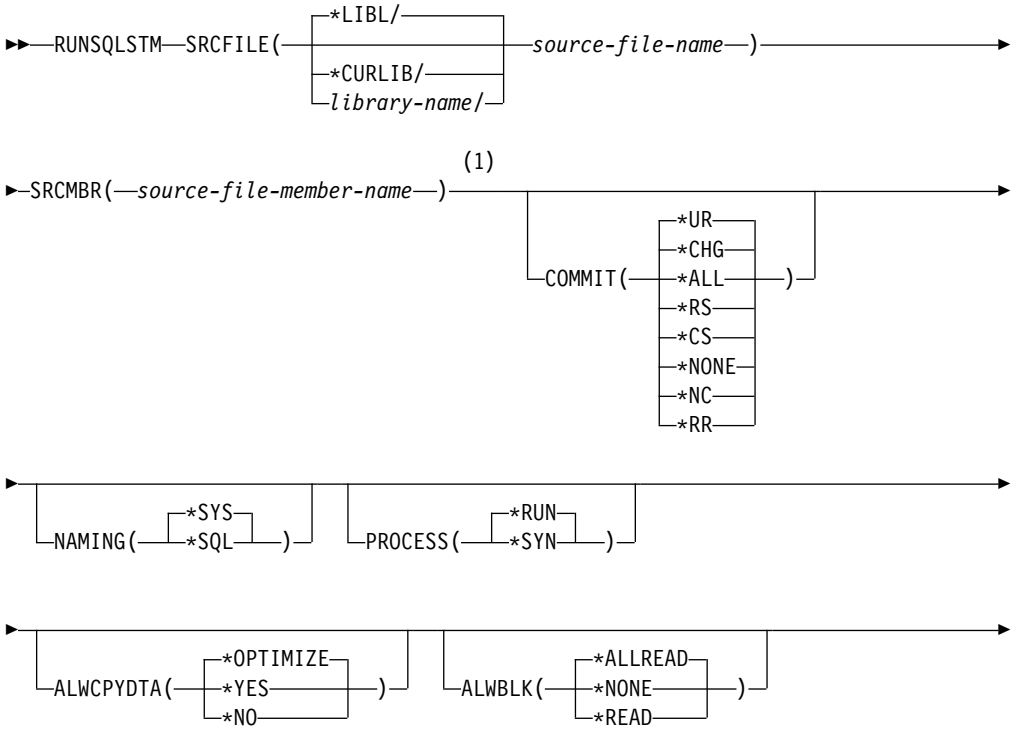
```
PRTSQLINF PAYROLL
```

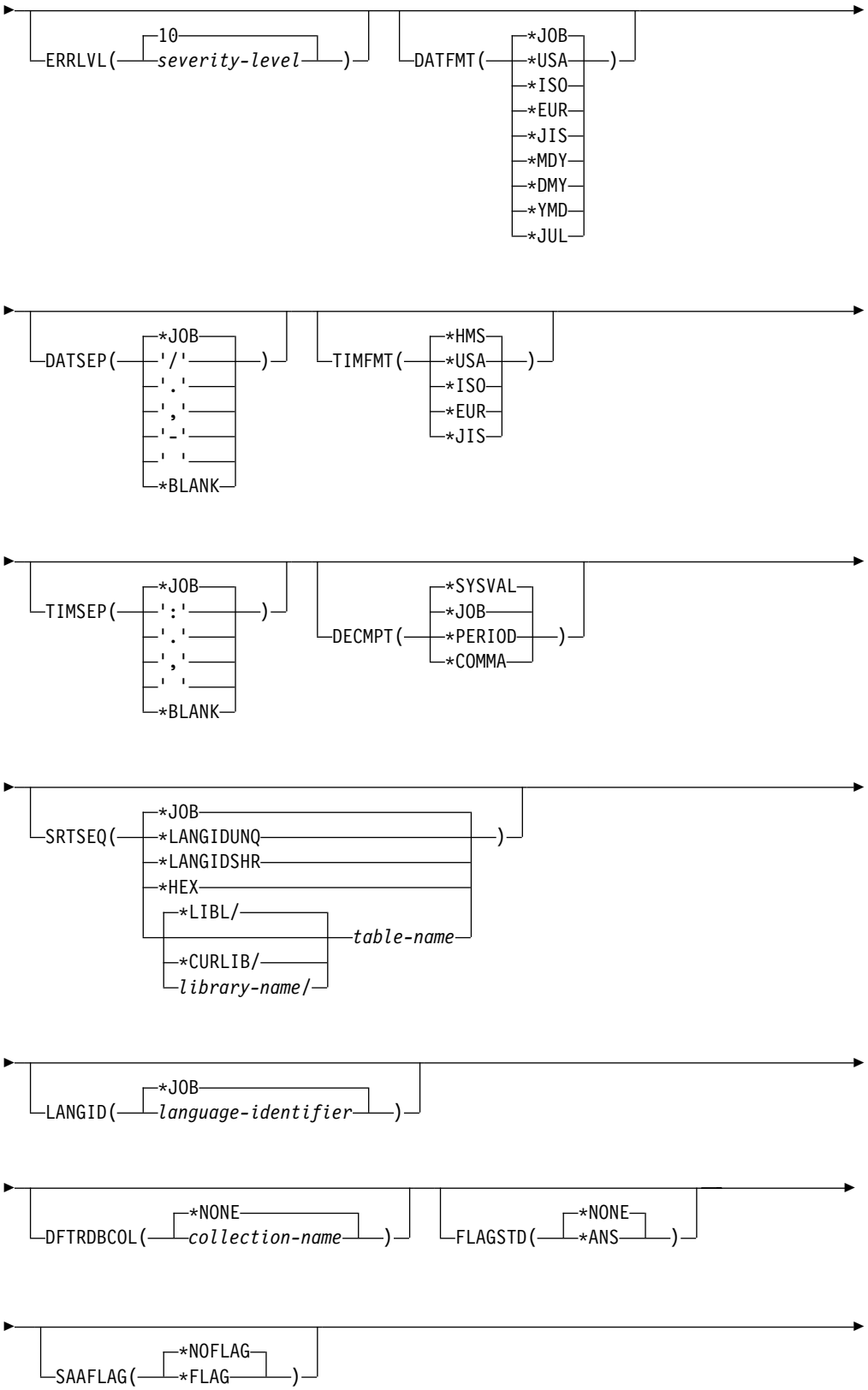
이 명령은 프로그램 PAYROLL에 들어 있는 SQL문에 대한 정보를 인쇄합니다.

OBJTYPE은 현재 종속 키워드이며 OBJ 매개변수가 *JOB인 경우에만 프롬트가 나타난다는 점을 유의하십시오.

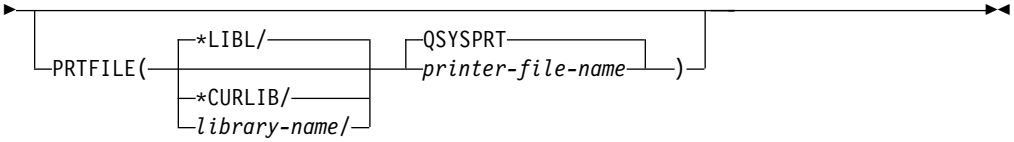
RUNSQLSTM(구조화 조회 언어문 실행) 명령

Job: B,I Pgm: B,I REXX: B,I Exec

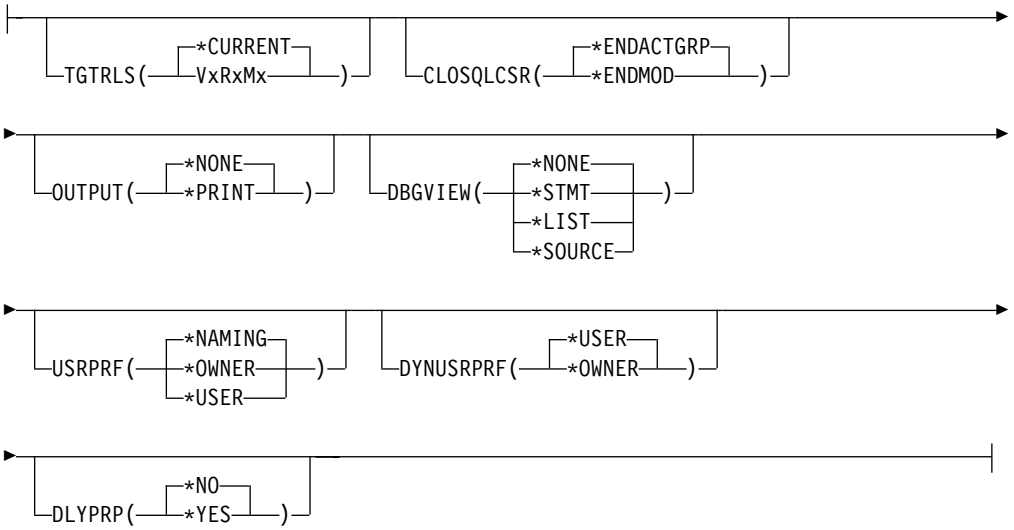



ERRLVL(
10
severity-level
)
DATFMT(
*JOB
*USA
*ISO
*EUR
*JIS
*MDY
*DMY
*YMD
*JUL
)
DATSEP(
*JOB
'/'
'.'
','
'-'
' '
*BLANK
)
TIMFMT(
*HMS
*USA
*ISO
*EUR
*JIS
)
TIMSEP(
*JOB
':'
'.'
','
' '
*BLANK
)
DECMPT(
*SYSVAL
*JOB
*PERIOD
*COMMA
)
SRTSEQ(
*JOB
*LANGIDUNQ
*LANGIDSHR
*HEX
*LIBL/
*CURLIB/
library-name/
table-name
)
LANGID(
*JOB
language-identifier
)
DFTRDBCOL(
*NONE
collection-name
)
FLAGSTD(
*NONE
*ANS
)
SAAFLAG(
*NOFLAG
*FLAG
)

RUNSQLSTM



SQL 루틴 매개변수:



주:

1 이 앞에 오는 모든 매개변수는 앞뒤 관계에 따라 지정할 수 있습니다.

목적:

RUNSQLSTM(구조화 조회 언어문 수행) 명령은 SQL문에 대한 소스 파일을 처리합니다.

매개변수:

SRCFILE

실행될 SQL문이 있는 소스 파일의 규정화명을 지정합니다.

소스 파일명은 다음의 라이브러리 값 중 하나로 규정될 수 있습니다.

***LIBL:** 일치되는 첫 번째 항목이 발견될 때까지 작업의 라이브러리 리스트에 있는 모든 라이브러리가 탐색됩니다.

***CURLIB:** 작업에 대한 현재 라이브러리가 탐색됩니다. 작업에 대한 현재 라이브러리로 지정된 라이브러리가 없으면 QGPL 라이브러리가 사용됩니다.

library-name: 탐색될 라이브러리명을 지정하십시오.

source-file-name: 실행될 SQL문이 들어 있는 소스 파일의 이름을 지정하십시오. 소스 파일은 데이터베이스 파일 또는 인라인 자료 파일이 될 수 있습니다.

SRCMBR

실행될 SQL문이 있는 소스 파일 멤버명을 지정합니다.

COMMIT

소스 파일의 SQL문이 확약 제어하에 실행되는지 여부를 지정합니다.

***CHG 또는 *UR:** SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON 및 REVOKE문에서 참조된 오브젝트들을 지정하며, 갱신, 삭제 및 삽입된 행들은 작업 단위(트랜잭션)의 끝까지 잠겨 있습니다. 다른 작업의 확약되지 않은 변경 내용을 볼 수 있습니다.

***ALL 또는 *RS:** SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME 및 REVOKE문에서 참조되는 오브젝트들을 지정하며, 선택, 갱신, 삭제 및 삽입된 행은 작업 단위(트랜잭션)의 끝까지 잠깁니다. 다른 작업의 확약되지 않은 변경 내용을 볼 수 없습니다.

***CS:** SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME 및 REVOKE문에서 참조되는 오브젝트와 갱신, 삭제 및 삽입된 행은 작업 단위(트랜잭션)의 끝까지 잠깁니다. 선택은 되었지만 갱신되지 않은 행은 다음 행이 선택될 때까지 잠깁니다. 다른 작업의 확약되지 않은 변경 내용을 볼 수 없습니다.

***NONE 또는 *NC:** 확약 제어가 사용되지 않음을 지정합니다. 다른 작업의 확약되지 않은 변경 내용을 볼 수 있습니다. SQL DROP COLLECTION문이 프로그램에 포함되는 경우에는 *NONE 또는 *NC가 반드시 사용되어야 합니다. RDB 매개변수에 관계형 데이터베이스가 지정되어 있고 관계형 데이터베이스가 AS/400이 아닌 시스템에 있으면, *NONE 또는 *NC를 지정할 수 없습니다.

***RR:** SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME 및 REVOKE문에서 참조되는 오브젝트들을 지정하며, 선택, 갱신, 삭제 및 삽입되는 행은 작업 단위(트랜잭션)의 끝까지 잠깁니다. 다른 작업의 확약되지 않은 변경 내용을 볼 수 없습니다. SELECT, UPDATE, DELETE 및 INSERT문에 참조된 모든 표는 작업 단위(트랜잭션)의 끝까지 배타적으로 잠겨 있습니다.

NAMING

SQL문에서 오브젝트를 명명하는데 사용되는 명명 규칙을 지정합니다.

***SYS:** 시스템 명명 규칙(library-name/file-name)이 사용됩니다.

***SQL:** SQL 명명 규칙(collection-name.table-name)이 사용됩니다.

PROCESS

소스 파일 멤버의 SQL문 실행 여부 또는 구문 검사 여부를 지정합니다.

***RUN:** 명령문이 구문 검사 후 실행됩니다.

***SYN:** 명령문의 구문 검사만 이루어집니다.

RUNSQLSTM

ALWCPYDTA

SELECT문에 자료의 사본이 사용될 수 있는지를 지정합니다.

***OPTIMIZE:** 시스템이 데이터베이스에서 직접 검색된 자료를 사용할지 또는 자료의 사본을 사용할지를 판별합니다. 결정은 최상의 성능을 제공할 수 있는 방법에 근거하여 이루어집니다. COMMIT가 *CHG 또는 *CS이면서 ALWBLK가 *ALLREAD가 아니거나, COMMIT가 *ALL 또는 *RR이면, 조회 실행에 필요할 경우에만 자료의 사본이 사용됩니다.

***YES:** 필요할 때만 자료의 사본이 사용됩니다.

***NO:** 자료의 사본이 사용되지 않습니다. 조회를 수행하는데 자료의 임시 사본이 필요하면 오류 메세지가 표시됩니다.

ALWBLK

데이터베이스 관리자가 레코드 블록화를 사용할 수 있는지 여부와 블록화가 읽기 전용 커서에 사용되는 범위를 지정합니다.

***ALLREAD:** COMMIT 매개변수에 대해 *NONE 또는 *CHG가 지정된 경우 읽기 전용 커서에 대해 행이 블록화됩니다. 명시적으로 갱신될 수 없는 프로그램 내의 모든 커서들은 EXECUTE 또는 EXECUTE IMMEDIATE문이 프로그램 내에 있어도 읽기 전용으로 열립니다.

*ALLREAD 지정:

- *READ에 대해 허용된 블록화 이외의 확약 제어 레벨 *CHG하에서 레코드의 블록화를 허용합니다.
- 프로그램에서 거의 모든 읽기 전용 커서의 성능이 향상될 수 있으나 다음과 같은 방법으로 조회를 제한합니다.
  - 롤백(ROLLBACK) 명령, 호스트 언어의 ROLLBACK문 또는 ROLLBACK HOLD SQL문은 *ALLREAD가 지정된 경우 읽기 전용 커서의 위치를 재지정하지 않습니다.
  - 커서에 대한 DECLARE문에 FOR UPDATE절이 없는 한, 지정된 UPDATE 또는 DELETE문의 동적 실행(예를 들면, EXECUTE IMMEDIATE 사용)은 커서의 행을 갱신하는데 사용될 수 없습니다.

***NONE:** 커서에 대한 자료의 검색을 위해 행이 블록화되지 않습니다.

*NONE을 지정하는 경우:

- 검색된 자료가 현재의 자료임을 보장합니다.
- 조회시 자료의 첫 번째 행을 검색하는데 필요한 시간을 줄일 수 있습니다.
- 조회가 닫히기 전에 조회의 처음 몇 행만이 검색될 때, 프로그램에 의해 사용되지 않은 자료 행 블록의 검색으로부터 데이터베이스 관리자를 중지시킵니다.
- 많은 행을 검색하는 조회의 전반적인 성능이 저하될 수 있습니다.

***READ:** 다음과 같은 경우에 커서에 대한 자료를 읽기 전용으로 검색하기 위해 레코드가 블록화됩니다.

- *NONE이 COMMIT 매개변수에 지정되어 확약 제어가 사용되지 않음을 표시하는 경우
- FOR FETCH ONLY절로 커서가 선언되었거나 커서에 대해 지정된 UPDATE 또는 DELETE문을 실행할 수 있는 동적 명령문이 없는 경우

*READ를 지정하면 위의 조건을 만족하는 조회의 전반적인 성능이 향상되고 많은 수의 레코드를 검색할 수 있습니다.

ERRLVL

SQL문 처리에 의해 생성된 메시지의 심각도에 근거하여 처리가 제대로 되었는지를 지정합니다. 처리중 이 매개변수에 지정된 값보다 큰 오류가 발생하면, 명령문은 더 이상 처리될 수 없으며 명령문이 확약 제어하에서 실행된다면 명령문은 원위치로 롤백합니다.

10: 명령문 처리는 10보다 큰 심각도 레벨의 오류 메세지가 수신될 때 중단됩니다.

severity-level: 사용될 심각도 레벨을 지정하십시오.

DATFMT

날짜 결과 열에 액세스할 때 사용된 형식을 지정합니다. 입력 시간 스트링의 경우 지정된 값이 날짜가 유효한 형식으로 지정되는지를 결정하는데 사용됩니다.

주: 형식 *USA, *ISO, *EUR 또는 *JIS를 사용하는 입력 날짜 스트링은 항상 유효합니다.

***JOB:** 작업에 지정된 형식이 사용됩니다. 작업 표시(DSPJOB) 명령을 사용하여 작업에 대한 현재 날짜 형식을 판별하십시오.

***USA:** 미국식 날짜 형식(mm/dd/yyyy)이 사용됩니다.

***ISO:** 국제 표준화 기구(ISO) 날짜 형식(yyyy-mm-dd)이 사용됩니다.

***EUR:** 유럽식 날짜 형식(dd.mm.yyyy)이 사용됩니다.

***JIS:** 일본 산업 표준 날짜 형식(yyyy-mm-dd)이 사용됩니다.

***MDY:** 날짜 형식(mm/dd/yy)이 사용됩니다.

***DMY:** 날짜 형식(dd/mm/yy)이 사용됩니다.

***YMD:** 날짜 형식(yy/mm/dd)이 사용됩니다.

***JUL:** 줄리안 날짜 형식(yy/ddd)이 사용됩니다.

DATSEP

날짜 결과 열에 액세스하기 위해 사용되는 분리자를 지정합니다.

주: 이 매개변수는 *JOB, *MDY, *DMY, *YMD 또는 *JUL이 DATFMT 매개변수에 지정되어 있을 때만 적용됩니다.

***JOB:** 작업에 지정된 날짜 분리자가 사용됩니다. 작업에 대한 현재 값을 결정하려면 DSPJOB(작업 표시) 명령을 사용하십시오.

'/': 슬래시(/)가 사용됩니다.

'.': 마침표(.)가 사용됩니다.

',': 쉼표(,)가 사용됩니다.

'-': 대시(-)가 사용됩니다.

' ': 공백( )이 사용됩니다.

***BLANK:** 공백( )이 사용됩니다.

TIMFMT

시간 결과 열에 액세스하기 위해 사용되는 형식을 지정합니다. 입력 시간 스트링의 경우, 지정된 값은 시간이 유효한 형식으로 지정되었는지 판별하는 데 사용됩니다.

주: 형식 *USA, *ISO, *EUR 또는 *JIS를 사용하는 입력 날짜 스트링은 항상 유효합니다.

***HMS:** hh:mm:ss 형식이 사용됩니다.

***USA:** 미국식 시간 형식 hh:mm xx가 사용됩니다. 여기서, xx는 AM 또는 PM입니다.

***ISO:** 국제 표준화 기구(ISO) 시간 형식 hh.mm.ss가 사용됩니다.

***EUR:** 유럽식 시간 형식 hh.mm.ss가 사용됩니다.

***JIS:** 일본 산업 표준 시간 형식 hh:mm:ss가 사용됩니다.

TIMSEP

시간 결과 열에 액세스하기 위해 사용되는 분리자를 지정합니다.

주: 이 매개변수는 *HMS가 TIMFMT 매개변수에 지정되어 있을 때에만 적용됩니다.

***JOB:** 작업에 지정된 시간 분리 문자가 사용됩니다. 작업에 대한 현재 값을 결정하려면 DSPJOB(작업 표시) 명령을 사용하십시오.

':': 콜론(:)이 사용됩니다.

'.': 마침표(.)가 사용됩니다.

',': 쉼표(,)가 사용됩니다.

' ': 공백( )이 사용됩니다.

***BLANK:** 공백( )이 사용됩니다.

DECMPT

SQL문의 숫자 상수에 대해 사용되는 소수점 값을 지정합니다.

***JOB:** SQL문에 있는 숫자 상수에 대한 소수점으로 사용되는 값이 명령문을 실행중인 작업에 대해 지정된 소수점 표현입니다.

***SYSVAL:** QDECFMT 시스템 값이 10진 소수점으로 사용됩니다.

***PERIOD:** 마침표가 소수점으로 사용됩니다.

***COMMA:** 쉼표가 소수점으로 사용됩니다.

SRTSEQ

SQL문의 스트링 비교에 사용될 정렬 순서표를 지정합니다.

***JOB:** 작업에 대한 LANGID 값이 검색됩니다.

***LANGIDSHR:** 정렬 순서표는 여러 문자에 대해 동일한 가중치를 사용하며, LANGID 매개변수에 지정된 언어와 연관된 공유 가중치 정렬 순서표입니다.

***LANGIDUNQ:** LANGID 매개변수에 지정된 언어에 대한 고유 가중치 정렬표가 사용됩니다.

***HEX:** 정렬 순서표가 사용되지 않습니다. 문자의 16진수 값이 정렬 순서를 결정하기 위하여 사용됩니다.

표의 이름은 다음 값 중 하나로 규정될 수 있습니다.

***LIBL:** 일치되는 첫 번째 항목이 발견될 때까지 작업의 라이브러리 리스트에 있는 모든 라이브러리가 탐색됩니다.

***CURLIB:** 작업에 대한 현재 라이브러리가 탐색됩니다. 작업에 대한 현재 라이브러리로 지정된 라이브러리가 없으면 QGPL 라이브러리가 사용됩니다.

library-name: 탐색될 라이브러리명을 지정하십시오.

table-name: 대화식 SQL 세션과 함께 사용될 정렬 순서표의 이름을 지정하십시오.

LANGID

SRTSEQ(*LANGIDUNQ) 또는 SRTSEQ(*LANGIDSHR) 지정시 사용될 언어 ID를 지정합니다.

***JOB:** 작업에 대한 LANGID 값이 사전컴파일중에 검색됩니다.

language-identifier: 언어 ID를 지정하십시오.

DFTRDBCOL

표, 보기, 색인 및 SQL 패키지의 규정되지 않은 이름에 사용되는 콜렉션명을 지정합니다.

***NONE:** OPTION 매개변수에 정의된 명명 규칙이 사용됩니다.

collection-name: 콜렉션 ID의 이름을 지정하십시오. 이 값은 OPTION 매개변수에 지정된 명명 규칙 대신 사용됩니다.

FLAGSTD

미국 표준 협회(ANSI) 플래그 기능을 지정합니다. 이 매개변수는 SQL문이 다음과 같은 표준을 따르고 있는지 확인하기 위해 SQL문을 플래그합니다.

```
ANSI X3.135-1992 항목
ISO 9075-1992 항목
FIPS 127.2 항목
```

***NONE:** SQL문이 ANSI 표준을 따르는지 알아보기 위해 SQL문을 검사하지 않습니다.

***ANS:** SQL문이 ANSI 표준을 따르는지 알아보기 위해 SQL문을 검사합니다.

SAAFLAG

IBM SQL 표시 기능을 지정합니다. 이 매개변수는 SQL문이 IBM SQL 구문을 따르는지 확인하기 위해 SQL문에 플래그를 붙입니다. IBM 데이터베이스 제품 IBM SQL 구문에 대한 자세한 정보는 *DRDA IBM SQL 참조서*, SC26-3255-00을 참조하십시오.

***NOFLAG:** SQL문이 IBM SQL 구문을 따르는지 알아보기 위해 SQL문을 검사하지 않습니다.

***FLAG:** SQL문이 IBM SQL 구문을 따르는지 알아보기 위해 SQL문을 검사합니다.

PRTFILE

RUNSQLSTM 인쇄 출력이 전달될 프린터 파일의 규정화명을 지정합니다. 파일의 길이는 최소 132바이트여야 합니다. 132바이트보다 적은 레코드 길이로 파일이 지정되는 경우, 정보가 유실됩니다.

프린터 파일명은 다음의 라이브러리 값 중 하나로 규정될 수 있습니다.

***LIBL:** 일치되는 첫 번째 항목이 발견될 때까지 작업의 라이브러리 리스트에 있는 모든 라이브러리가 탐색됩니다.

***CURLIB:** 작업에 대한 현재 라이브러리가 탐색됩니다. 작업에 대한 현재 라이브러리로 지정된 라이브러리가 없으면 QGPL 라이브러리가 사용됩니다.

library-name: 탐색될 라이브러리명을 지정하십시오.

QSYSPRT: 파일명이 지정되지 않으면, RUNSQLSTM 인쇄 출력이 IBM 제공 프린터 파일 QSYSPRT로 전달됩니다.

printer-file-name: RUNSQLSTM 인쇄 출력이 전달되는 프린터 파일의 이름을 지정하십시오.

SQL 루틴에 대한 매개변수:

아래 나열된 매개변수는 SQL 프로시듀어, SQL 기능 및 SQL 트리거를 작성하는 소스 파일 내의 명령문에만 적용됩니다. 매개변수는 SQL 프로시듀어, SQL 기능 및 SQL 트리거와 연관된 프로그램 오브젝트를 작성하는 동안 사용됩니다.

TGTRLS

작성될 오브젝트를 사용하려고 하는 오퍼레이팅 시스템의 릴리스 레벨을 지정합니다.

*CURRENT 값이 주어질 때와 *릴리스 레벨* 값을 지정할 때에는 VxRxMx의 형식으로 릴리스를 지정하게 되며, 여기에서 Vx는 버전, Rx는 릴리스, Mx는 수정 레벨을 나타냅니다. 예를 들어, V2R3M0은 버전 2, 릴리스 3, 수정 레벨 0을 나타냅니다.

***CURRENT:** 오브젝트는 사용자 시스템에서 현재 실행중인 오퍼레이팅 시스템의 릴리스에 사용됩니다. 예를 들어, 시스템에서 V2R3M5가 실행되고 있으면, *CURRENT는 V2R3M5가 설치된 시스템에서 오브젝트를 사용할 것임을 의미합니다. 또한 사용자는 설치되어 있는 오퍼레이팅 시스템의 어느 후속 릴리스에서도 오브젝트를 사용할 수 있습니다.

주: 시스템에서 V2R3M5가 실행되고 있으며, V2R3M0가 설치된 시스템에서 오브젝트가 사용될 경우에는 TGTRLS(*CURRENT)가 아니고 TGTRLS(V2R3M0)를 지정해야 합니다.

release-level: VxRxMx 형식의 릴리스를 지정하십시오. 지정된 릴리스의 시스템이나 설치되어 있는 오퍼레이팅 시스템의 어느 후속 릴리스의 시스템에서도 오브젝트를 사용할 수 있습니다.

유효한 값들은 현재 버전, 릴리스, 수정 레벨에 기초하고 있으며, 새로운 릴리스로 변경될 수 있습니다. 이 명령이 지원하는 가장 이전의 릴리스 레벨보다 더 앞의 릴리스 레벨을 지정하면, 지원되는 릴리스 레벨 가운데 가장 이전의 릴리스가 표시된 오류 메세지를 받게 됩니다.

CLOSQLCSR

SQL 커서가 내재적으로 닫힐 때 SQL 준비 명령문이 내재적으로 취소되고 LOCK TABLE 잠금이 해제됨을 지정합니다. SQL 커서는 CLOSE, COMMIT 또는 ROLLBACK(HOLD 없이) SQL문을 발행할 때 명시적으로 닫힙니다.

***ENDACTGRP:** SQL 커서는 닫히고 SQL 준비 명령문은 즉시 취소됩니다.

ENDMOD: SQL 커서는 닫히고 SQL 준비 명령문은 내재적으로 모듈이 종료할 때 취소됩니다. 호출 스택의 첫 번째 SQL 프로그램이 종료할 때 LOCK TABLE 잠금이 해제됩니다.

OUTPUT

사전컴파일러 리스트가 생성되었는지를 지정합니다.

***NONE:** 사전컴파일러 리스트가 생성되지 않습니다

***PRINT:** 사전컴파일러 리스트가 생성됩니다.

DBGVIEW

처리되는 테이프 자료 파일의 레코드 유형과 블록화 속성을 지정합니다.

***NONE:** 소스 보기가 생성되지 않습니다.

***STMT:** 컴파일된 모듈이 프로그램 명령문 번호와 기호 ID를 사용하여 디버그되도록 합니다.

***LIST:** 컴파일된 모듈 오브젝트를 디버깅하기 위한 리스트 보기를 생성합니다.

***SOURCE:** SQL 프로시듀어, 기능 및 트리거에 대한 소스 보기가 생성됩니다.

USRPRF

정적 SQL문에서 각 오브젝트에 대해 프로그램 오브젝트가 가지는 권한을 포함하여 컴파일된 프로그램 오브젝트를 실행할 때 사용되는 사용자 프로파일을 지정합니다. 프로그램 소유자 또는 프로그램 사용자의 프로파일이 프로그램 오브젝트가 사용할 수 있는 오브젝트를 제어하는 데 사용됩니다.

***NAMING:** 사용자 프로파일은 명명 규칙에 의해 판별됩니다. 명명 규칙이 *SQL이면, USRPRF(*OWNER)가 사용됩니다. 명명 규칙이 *SYS이면, USRPRF(*USER)가 사용됩니다.

***USER:** 프로그램 오브젝트를 실행중인 사용자의 프로파일이 사용됩니다.

***OWNER:** 프로그램 소유자와 프로그램 사용자 모두의 사용자 프로파일이 프로그램 실행시 사용됩니다.

DYNUSRPRF

동적 SQL문에 대해 사용될 사용자 프로파일을 지정합니다.

***USER:** 로컬의 경우, 동적 SQL문은 프로그램 사용자의 프로파일하에서 실행됩니다. 분산 동적 SQL문은 SQL 패키지의 사용자 프로파일하에서 실행됩니다.

***OWNER:** 로컬의 경우, 동적 SQL문은 프로그램 소유자의 프로파일하에서 실행됩니다. 분산 동적 SQL문은 SQL 패키지 소유자의 사용자 프로파일하에서 실행됩니다.

DLYPRP

PREPARE문에 대한 동적 명령문 유효성 검사가 OPEN, EXECUTE 또는

DESCRIBE문 실행시까지 지연되는지 여부를 지정합니다. 유효성 검사가 지연되면 불필요한 확인이 제거되므로 성능이 향상됩니다.

***NO:** 동적 명령문 유효성 검사가 지연되지 않습니다. 동적 명령문이 준비될 때 액세스 계획의 유효성이 확인됩니다. 동적 명령문이 OPEN 또는 EXECUTE문에 사용될 때 액세스 계획의 유효성이 재확인됩니다. 동적 명령문에 의해 참조되는 오브젝트의 존재 또는 권한으로 인해, 동적 명령문이 아직 유효한지 확인하기 위해 OPEN 또는 EXECUTE문을 발행하기 전에 SQLCODE 또는 SQLSTATE를 체크해야 합니다.

***YES:** 동적 명령문 유효성 검사는 동적 명령문이 OPEN, EXECUTE 또는 DESCRIBE SQL문에 사용될 때까지 지연됩니다. 동적 명령문이 사용될 때 유효성 검사가 완료되고 액세스 계획이 수립됩니다. 이 매개변수에 *YES를 지정하면 OPEN, EXECUTE 또는 DESCRIBE문을 수행한 후에 SQLCODE와 SQLSTATE를 검사하여 동적 명령문이 유효한지 확인해야 합니다.

주: *YES를 지정하면 PREPARE에 INTO절이 사용되거나, 동적 명령문에 대한 OPEN문이 발행되기 전에 DESCRIBE문에서 그 동적 명령문을 사용하는 경우 성능은 향상되지 않습니다.

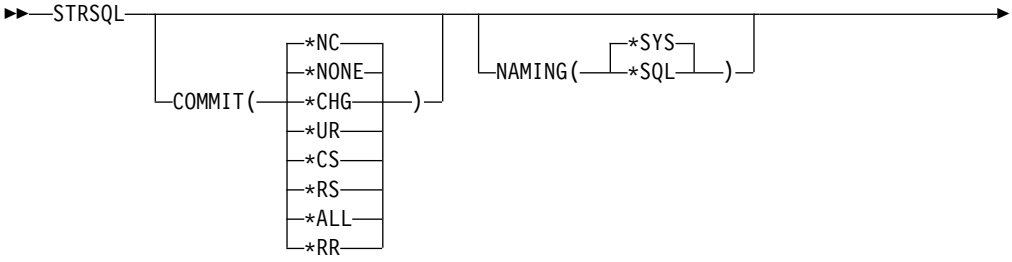
예:

```
RUNSQLSTM  SRCFILE(MYLIB/MYFILE)  SRCMBR(MYMBR)
```

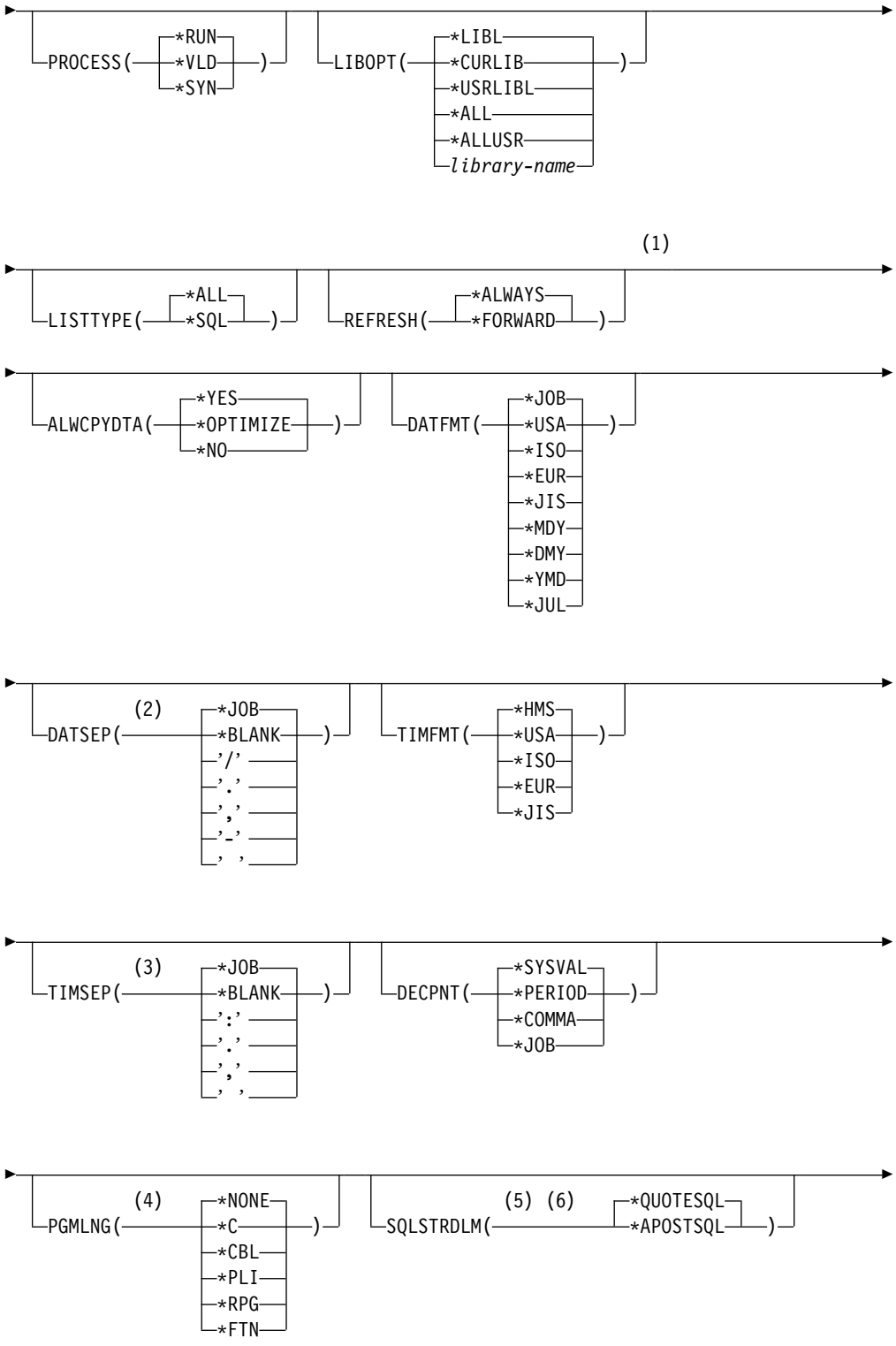
이 명령은 라이브러리 MYLIB의 파일 MYFILE에서 발견된 멤버 MYMBR의 SQL문을 처리합니다.

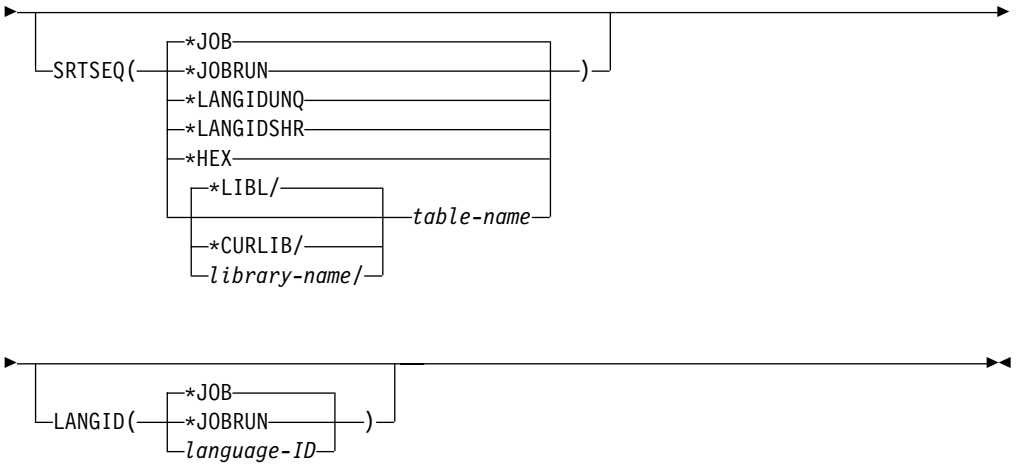
STRSQL(구조화 조회 언어 시작) 명령

Job: I Pgm: I REXX: I Exec



STRSQL


PROCESS(*RUN *VLD *SYN)
LIBOPT(*LIBL *CURLIB *USRLIBL *ALL *ALLUSR library-name)
(1)
LISTTYPE(*ALL *SQL)
REFRESH(*ALWAYS *FORWARD)
ALWCPYDTA(*YES *OPTIMIZE *NO)
DATFMT(*JOB *USA *ISO *EUR *JIS *MDY *DMY *YMD *JUL)
(2)
DATSEP(*JOB *BLANK '/' '.' ',' '-' ' ')
TIMFMT(*HMS *USA *ISO *EUR *JIS)
(3)
TIMSEP(*JOB *BLANK ':' '.' ',' ' ')
DECPNT(*SYSVAL *PERIOD *COMMA *JOB)
(4)
PGMLNG(*NONE *C *CBL *PLI *RPG *FTN)
(5) (6)
SQLSTRDLM(*QUOTESQL *APOSTSQL)



주:

1 이 앞에 오는 모든 매개변수는 앞뒤 관계에 따라 지정할 수 있습니다.

2 DATSEP은 *MDY, *DMY, *YMD 또는 *JUL이 DATFMT 매개변수에 지정될 때에만 유효합니다.

3 TIMSEP은 TIMFMT(*HMS)가 지정될 때에만 유효합니다.

4 PGMLNG와 SQLSTRDLM은 PROCESS(*SYN)가 지정될 때에만 유효합니다.

5 PGMLNG와 SQLSTRDLM은 PROCESS(*SYN)가 지정될 때에만 유효합니다.

6 SQLSTRDLM은 PGMLNG(*CBL)가 지정될 때에만 유효합니다.

목적:

STRSQL(구조화 조회 언어 시작) 명령은 대화식 구조화 조회 언어(SQL) 프로그램을 시작합니다. 프로그램은 SQL 입력 화면을 즉시 표시하는 대화식 SQL 프로그램의 명령문 입력을 시작합니다. 이 화면에서는 대화식 환경으로 SQL문을 빌드, 편집, 입력, 실행하는 것이 가능합니다. 프로그램 실행중 수신된 메세지가 이 화면에 표시됩니다.

매개변수:

COMMIT

SQL문이 확약 제어하에서 실행되는지 여부를 지정합니다.

***NONE 또는 *NC:** 확약 제어가 사용되지 않음을 지정합니다. 다른 작업의 확약되지 않은 변경 내용을 볼 수 있습니다. SQL DROP COLLECTION문이 프로그램에 포함되는 경우에는 *NONE 또는 *NC가 반드시 사용되어야 합니다. RDB 매개변수에 관계형 데이터베이스가 지정되어 있고 관계형 데이터베이스가 iSeries이 아닌 시스템에 있으면, *NONE 또는 *NC를 지정할 수 없습니다

***CHG or *UR:** SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME 및 REVOKE문에서 참조되는 오브젝트와 갱

신, 삭제 및 삽입된 행은 작업 단위(트랜잭션)의 끝까지 잠깁니다. 다른 작업의 확약되지 않은 변경 내용을 볼 수 있습니다.

***CS:** SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME 및 REVOKE문에서 참조되는 오브젝트와 갱신, 삭제 및 삽입된 행은 작업 단위(트랜잭션)의 끝까지 잠깁니다. 선택은 되었지만 갱신되지 않은 행은 다음 행이 선택될 때까지 잠깁니다. 다른 작업의 확약되지 않은 변경 내용을 볼 수 없습니다.

***ALL** 또는 ***RS:** SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME 및 REVOKE문에서 참조되는 오브젝트들을 지정하며, 선택, 갱신, 삭제 및 삽입된 행은 작업 단위(트랜잭션)의 끝까지 잠깁니다. 다른 작업의 확약되지 않은 변경 내용을 볼 수 없습니다.

***RR:** SQL ALTER, CALL, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, RENAME 및 REVOKE문에서 참조되는 오브젝트들을 지정하며, 선택, 갱신, 삭제 및 삽입되는 행은 작업 단위(트랜잭션)의 끝까지 잠깁니다. 다른 작업의 확약되지 않은 변경 내용을 볼 수 없습니다. SELECT, UPDATE, DELETE 및 INSERT문에 참조된 모든 표는 작업 단위(트랜잭션)의 끝까지 배타적으로 잠겨 있습니다.

주: CRTSQLXXX 명령(XXX=CI, CPPI, CBL, FTN, PLI, CBLI, RPG 또는 RPGI)의 경우 이 매개변수에 대한 디폴트는 *CHG입니다.

NAMING

SQL문에서 오브젝트를 명명하는데 사용되는 명명 규칙을 지정합니다.

***SYS:** 시스템 명명 규칙(library-name/file-name)이 사용됩니다.

***SQL:** SQL 명명 규칙(collection-name.table-name)이 사용됩니다.

PROCESS

SQL문의 처리에 사용되는 값을 지정합니다.

***RUN:** 명령문이 구문 검사, 자료 검사 후 실행됩니다.

***VLD:** 명령문이 구문 검사, 자료 검사 후 실행되지 않습니다.

***SYN:** 명령문에 구문 검사만 이루어집니다.

LIBOPT

F4, F16, F17 또는 F18 기능 키를 눌렀을 때 콜렉션 리스트 빌드 기준으로 사용되는 콜렉션과 라이브러리를 지정합니다.

콜렉션 리스트의 이름은 다음 값 중 하나로 규정될 수 있습니다.

***LIBL:** 일치되는 첫 번째 항목이 발견될 때까지 작업의 라이브러리 리스트에 있는 모든 라이브러리가 탐색됩니다.

***CURLIB:** 작업에 대한 현재 라이브러리가 탐색됩니다. 작업에 대한 현재 라이브러리로 지정된 라이브러리가 없으면 QGPL 라이브러리가 사용됩니다.

***USRLIBL:** 작업 라이브러리 리스트의 사용자 부분에 있는 라이브러리만이 탐색됩니다.

***ALL:** QSYS를 비롯한 시스템의 모든 라이브러리가 탐색됩니다.

***ALLUSR:** 모든 사용자 라이브러리가 탐색됩니다. 다음을 제외한 문자 Q로 시작되지 않는 이름의 모든 라이브러리가 탐색됩니다.

```
#CGULIB     #DFULIB     #RPGLIB     #SEULIB
#COBLIB     #DSULIB     #SDALIB
```

IBM에서 다음의 Qxxx 라이브러리를 제공했더라도 여기에는 일반적으로 자주 변경되는 사용자 자료가 들어 갑니다. 따라서, 이러한 라이브러리들은 사용자 라이브러리로 간주되어 탐색됩니다.

```
QDSNX          QRCL          QUSRBRM          QUSRSYS
QGPL           QS36F         QUSRIJS          QUSRVxRxMx
QGPL38         QUSER38       QUSRINFSKR
QPFRDATA       QUSRADSM      QUSRRDARS
```

주: QUSRVxRxMx 양식으로 이루어진 서로 다른 라이브러리명은 IBM이 지원하는 각 릴리스용으로 사용자에 의해 작성될 수 있습니다. VxRxMx는 라이브러리의 버전, 릴리스 및 수정판 레벨입니다.

library-name: 탐색될 라이브러리명을 지정하십시오.

LISTTYPE

F4, F16, F17 또는 F18 기능 키를 누를 때 리스트 지원에 대해 표시되는 오브젝트 유형을 지정합니다.

***ALL:** 모든 오브젝트가 표시됩니다.

***SQL:** SQL 작성 오브젝트만이 표시됩니다.

REFRESH

화면 선택 출력 자료가 화면정리되는 시기를 지정합니다.

***ALWAYS:** 자료가 앞뒤로 화면을 이동하는 중에 정상적으로 정리됩니다.

***FORWARD:** 맨 처음 자료의 끝을 향해 앞으로 화면을 이동할 때에만 자료가 정리됩니다. 뒤로 화면을 이동할 때에는 이미 열람했던 자료의 사본이 표시됩니다.

ALWCPYDTA

SELECT문에 자료의 사본이 사용될 수 있는지를 지정합니다. COMMIT(*ALL)가 지정되면 SQL 실행시 ALWCPYDTA 값이 무시되고 현재의 자료가 사용됩니다.

***YES:** 필요할 경우 자료의 사본이 사용됩니다.

***OPTIMIZE:** 시스템이 데이터베이스에서 검색된 자료를 사용할지 아니면 자료의 사본을 사용할지를 결정합니다. 이 결정은 최상의 성능을 제공하는 방법에 근거하여 이루어집니다.

***NO:** 자료의 사본이 허용되지 않습니다. 조회를 수행하는데 자료의 임시 사본이 필요하면 오류 메세지가 표시됩니다.

DATFMT

SQL문에 사용되는 날짜 형식을 지정합니다.

***JOB:** 작업 속성 DATFMT에 지정된 형식이 사용됩니다.

***USA:** 미국식 날짜 형식(mm/dd/yyyy)이 사용됩니다.

***ISO:** ISO 날짜 형식(yyyy-mm-dd)이 사용됩니다.

***EUR:** 유럽식 날짜 형식(dd.mm.yyyy)이 사용됩니다.

***JIS:** 일본 산업 표준 날짜 형식(yyyy-mm-dd)이 사용됩니다.

***MDY:** 월, 일, 년 날짜 형식(mm/dd/yy)이 사용됩니다.

***DMY:** 일, 월, 년 날짜 형식(dd/mm/yy)이 사용됩니다.

***YMD:** 년, 월, 일 날짜 형식(yy/mm/dd)이 사용됩니다.

***JUL:** 줄리안 날짜 형식(yy/ddd)이 사용됩니다.

DATSEP

SQL문에 사용된 날짜 분리 문자를 지정합니다.

***JOB:** 작업 속성에 지정된 날짜 분리 문자가 사용됩니다. 새로운 대화식 SQL 세션에 *JOB을 지정하면 현재 값이 저장되어 사용됩니다. 작업의 날짜 분리 문자에 대한 차후의 변경은 대화식 SQL에 의해 검출되지 않습니다.

***BLANK:** 공백( )이 사용됩니다.

'/': 슬래시(/)가 사용됩니다.

'.': 마침표(.)가 사용됩니다.

',': 쉼표(,)가 사용됩니다.

'-': 대시(-)가 사용됩니다.

' ': 공백( )이 사용됩니다.

TIMFMT

SQL문에 사용된 시간 형식을 지정합니다.

***HMS:** 시 분 초 시간 형식(hh:mm:ss)이 사용됩니다.

***USA:** 미국식 시간 형식(hh:mm xx, xx는 AM 또는 PM)이 사용됩니다.

***ISO:** ISO 시간 형식(hh.mm.ss)이 사용됩니다.

***EUR:** 유럽식 시간 형식(hh.mm.ss)이 사용됩니다.

***JIS:** 일본 산업 표준 서기 시간 형식(hh:mm:ss)이 사용됩니다.

TIMSEP

SQL문에 사용된 시간 분리 문자를 지정합니다.

***JOB:** 작업 속성에 지정된 시간 분리 문자가 사용됩니다. 새로운 대화식 SQL 세션에 *JOB을 지정하면 현재 값이 저장되어 사용됩니다. 작업의 시간 분리 문자에 대한 차후의 변경은 대화식 SQL에 의해 검출되지 않습니다.

***BLANK:** 공백( )이 사용됩니다.

':': 콜론(:)이 사용됩니다.

'.': 마침표(.)가 사용됩니다.

',': 쉼표(,)가 사용됩니다.

' ': 공백( )이 사용됩니다.

DECPNT

사용하는 소수점의 종류를 지정합니다.

***JOB:** SQL문에 있는 숫자 상수에 대한 소수점으로 사용되는 값이 명령문을 실행중인 작업에 대해 지정된 소수점 표현입니다.

***SYSVAL:** 소수점은 시스템 값으로부터 발췌됩니다. 새로운 대화식 SQL 세션에 *SYSVAL을 지정하면 현재 값이 저장되어 사용됩니다. 시스템의 시간 분리 문자에 대한 차후의 변경은 대화식 SQL에 의해 검출되지 않습니다.

***PERIOD:** 마침표가 소수점으로 사용됩니다.

***COMMA:** 쉼표가 소수점으로 사용됩니다.

PGMLNG

사용할 프로그램 언어 구문 규칙을 지정합니다. 이 매개변수를 사용하려면 PROCESS 매개변수에 *SYN이 선택되어야 합니다.

***NONE:** 특정 언어의 구문 검사가 사용되지 않습니다.

지원 언어는 다음과 같습니다.

***C:** C 언어 구문 규칙에 따라 구문 검사가 수행됩니다.

***CBL:** COBOL 언어 구문 규칙에 따라 구문 검사가 수행됩니다.

***PLI:** PL/I 언어 구문 규칙에 따라 구문 검사가 수행됩니다.

***RPG:** RPG 언어 구문 규칙에 따라 구문 검사가 수행됩니다.

***FTN:** FORTRAN 언어 구문 규칙에 따라 구문 검사가 수행됩니다.

SQLSTRDLM

SQL 스트링 분리문자를 지정합니다. 이 매개변수를 사용하려면 COBOL(*CBL) 문자 세트도 사용해야 합니다.

***QUOTESQL:** 따옴표가 SQL 스트링 분리문자로 사용됩니다.

***APOSTSQL:** 어포스트로피가 SQL 스트링 분리문자로 사용됩니다.

SRTSEQ

입력 SQL문 화면상의 SQL 명령문의 스트링 비교를 위해 사용된 정렬 순서표를 지정합니다.

***JOB:** 작업에 대한 SRTSEQ 값이 검색됩니다.

***JOBRUN:** 작업에 대한 SRTSEQ 값은 사용자가 대화식 SQL을 시작할 때마다 검색됩니다.

***LANGIDUNQ:** LANGID 매개변수에 지정된 언어에 대한 고유 가중치 정렬표가 사용됩니다.

***LANGIDSHR:** LANGID 매개변수에 지정된 언어에 대한 공유 가중치 정렬표가 사용됩니다.

***HEX:** 정렬 순서표가 사용되지 않습니다. 문자의 16진수 값이 정렬 순서를 결정하기 위하여 사용됩니다.

표의 이름은 다음 값 중 하나로 규정될 수 있습니다.

***LIBL:** 일치되는 첫 번째 항목이 발견될 때까지 작업의 라이브러리 리스트에 있는 모든 라이브러리가 탐색됩니다.

***CURLIB:** 작업에 대한 현재 라이브러리가 탐색됩니다. 작업에 대한 현재 라이브러리로 지정된 라이브러리가 없으면 QGPL 라이브러리가 사용됩니다.

library-name: 탐색될 라이브러리명을 지정하십시오.

table-name: 대화식 SQL 세션과 함께 사용될 정렬 순서표의 이름을 지정하십시오.

LANGID

SRTSEQ(*LANGIDUNQ) 또는 SRTSEQ(*LANGIDSHR) 지정시 사용될 언어 ID를 지정합니다.

***JOB:** 작업에 대한 LANGID 값이 검색됩니다.

***JOBRUN:** 대화식 SQL이 시작될 때마다 작업에 대한 LANGID 값이 검색됩니다.

language-ID: 사용될 언어 ID를 지정하십시오.

예:

```
STRSQL   PROCESS(*SYN)  NAMING(*SQL)
  DECPNT(*COMMA)  PGMLNG(*CBL)
  SQLSTRDLM(*APOSTSQL)
```

STRSQL

이 명령은 SQL문의 구문만을 검사하는 대화식 SQL 세션을 시작합니다. 구문 검사 프로그램에 의해 사용된 문자 세트에는 COBOL 언어 구문 규칙이 사용됩니다. 이 세션에서는 SQL 명명 규칙이 사용됩니다. 소수점이 쉼표로 사용되고 SQL 스트링 분리문자로는 어포스트로피가 사용됩니다.

참고 문헌

참고 문헌에는 본문에서 사용되거나 참조된 주제 항목에 대한 추가 정보가 들어 있는 참고 책들이 나와 있습니다. 여기에는 책들의 원제목과 주문 번호가 모두 나오지만 본문에서 참조할 때는 축약하여 사용됩니다.

- 백업 및 회복

  이 안내서에는 백업 및 회복 책에 있는 정보의 서브세트가 들어 있습니다. 설명서에는 백업 및 회복 전략 계획에 대한 정보, 저장 및 복원 프로시듀어에 사용할 수 있는 여러 가지 유형의 매체 및 디스크 회복 절차가 들어 있습니다. 또한 백업으로 시스템을 다시 설치하는 방법이 설명됩니다.

- 파일 관리

  이 책에는 어플리케이션 프로그램의 파일을 사용하는 방법에 대한 정보가 나와 있습니다.

- iSeries용 DB2 UDB 데이터베이스 프로그래밍

  이 책은 시스템에서 데이터베이스 파일을 작성, 설명 및 갱신하는 방법에 대한 정보를 포함하여 iSeries용 DB2 UDB 데이터베이스 구성의 상세한 설명을 제공합니다.

- CL 프로그래밍

  이 안내서는 오브젝트 및 라이브러리에 대한 일반적인 설명, CL 프로그래밍, 프로그램간 흐름 제어와 통신, CL 프로그램에서 오브젝트에 대한 작업 및 CL 프로그램 작성을 비롯한 iSeries용 DB2 UDB 프로그래밍 주제를 광범위하게 설명합니다. 이 외에도 디버그 모드, 중단점, 추적 및 화면 기능을 포함하여 사전 정의 및 즉시 메세지, 사용자 정의 명령과 메뉴 처리, 정의, 작성과 어플리케이션 테스트에 대한 사항도 들어 있습니다.

- Control Language (CL)

  이 안내서는 iSeries용 DB2 UDB 제어 언어(CL) 및 OS/400 명령의 설명을 제공합니다(비OS/400 명령은 각 사용권 프로그램 참고 책에서 설명됩니다). 또한, 서버에 대한 모든 CL 명령의 개요를 제공하며 CL 명령을 코드화하는 데 필요한 구문 규칙을 설명합니다.

- iSeries 보안 참조서

  이 책에는 시스템 보안 개념, 보안 계획 및 시스템에서 보안 설정에 대한 정보가 들어 있습니다. 또한 해당 권한이 없는 사용자가 사용할 수 없도록 시스템과 자원을 보호하고 자료를 고의적으로 또는 부주의한 손상이나 파기로부터 보호하며 시스템에 보안을 설정하고 그 보안을 최신의 것으로 유지하는 방법에 대한 정보도 제공됩니다.

- iSeries용 DB2 UDB SQL 참조서

  이 책은 iSeries용 DB2 UDB 명령문과 매개변수에 관한 정보를 제공합니다. SQL 통신 영역(SQLCA)과 설명 영역(SQLDA)을 설명하는 부록도 들어 있습니다.

- IDDU 사용

  이 책은 iSeries용 DB2 UDB 대화식 자료 정의 유틸리티(IDDU)를 사용하여 시스템에 대한 자료 사전, 파일 및 레코드를 기술하는 방법에 대해 설명합니다.

- WebSphere Development Studio: ILE COBOL 프로그래머 안내서

  이 안내서는 iSeries 시스템에서 iSeries용 COBOL 프로그램을 설계, 기록, 테스트 및 유지보수하는 데 필요한 정보를 제공합니다.

- WebSphere Development Studio: ILE RPG 프로그래머 안내서

  이 안내서는 iSeries 시스템에서 iSeries용 ILE RPG 프로그램을 설계, 기록, 테스트 및 유지보수하는 데 필요한 정보를 제공합니다.

- ILE C for AS/400 Language Reference 

  이 안내서는 iSeries 시스템에서 iSeries용 ILE C 프로그램을 설계, 기록, 테스트 및 유지보수하는 데 필요한 정보를 제공합니다.
- WebSphere Development Studio: ILE C/C++ 프로그래머 안내서 

  이 안내서는 iSeries 시스템에서 iSeries용 ILE C 프로그램을 설계, 기록, 테스트 및 유지보수하는 데 필요한 정보를 제공합니다.
- WebSphere Development Studio: ILE COBOL 참조서 

  이 안내서는 iSeriesCOBOL 시스템에서 iSeries용 COBOL 프로그램을 설계, 기록, 테스트 및 유지보수하는 데 필요한 정보를 제공합니다.
- REXX/400 Programmer's Guide 

  이 안내서는 iSeries 시스템에서 REXX/400 프로그램을 설계, 기록, 테스트 및 유지보수하는 데 필요한 정보를 제공합니다.
- DB2 Multisystem

  이 책은 분산 관계형 데이터베이스 파일, 노드 그룹 및 파티션의 개념에 대해 설명합니다. 이 책은 여러 iSeries 시스템에 걸쳐 파티션되어 있는 데이터베이스 파일을 작성하고 사용하는 데 필요한 정보를 제공합니다. 제공되는 정보로는 시스템의 구성법, 파일의 작성법, 어플리케이션에서 파일을 사용하는 방법 등이 있습니다.
- iSeries용 성능 분석 툴 

  이 책은 프로그래머에게 시스템, 작업 또는 프로그램 성능에 관한 자료를 수집하는데 필요한 정보를 제공합니다. 이 책에는 존재할 가능성이 있는 비효율성을 식별하고 정정하기 위해 성능을 인쇄하고 분석하는데 대한 요령도 나옵니다. 관리자 및 에이전트 피처에 대한 정보도 포함되어 있습니다.
- iSeries용 DB2 UDB SQL 호출 레벨 인터페이스 (ODBC)

  이 책은 DB2 호출 레벨 인터페이스를 사용하여 어플리케이션을 작성하기 위해 어플리케이션 프로그래머가 필요로 하는 정보를 제공합니다.
- IBM Developer Kit for Java

  이 안내서는 iSeries 시스템에서 Java 프로그램을 설계, 기록, 테스트 및 유지보수하는 데 필요한 정보를 제공합니다. 또한, iSeries Developer Kit for Java JDBC 드라이버에 대한 장도 있어서 JDBC 또는 SQLj를 사용하여 Java 프로그램에서 데이터베이스 파일에 액세스하는 것에 대한 정보를 제공합니다.

색인

[가]

[나]

[다]

[라]

[마]

[바]

[사]

[아]

[자]

[차]

[카]

[타]

[파]

[하]

[숫자]

A

B

C

D

E

F

G

H

I

J

L

N

O

P

Q

R

S

U

W

X

Printed in U.S.A