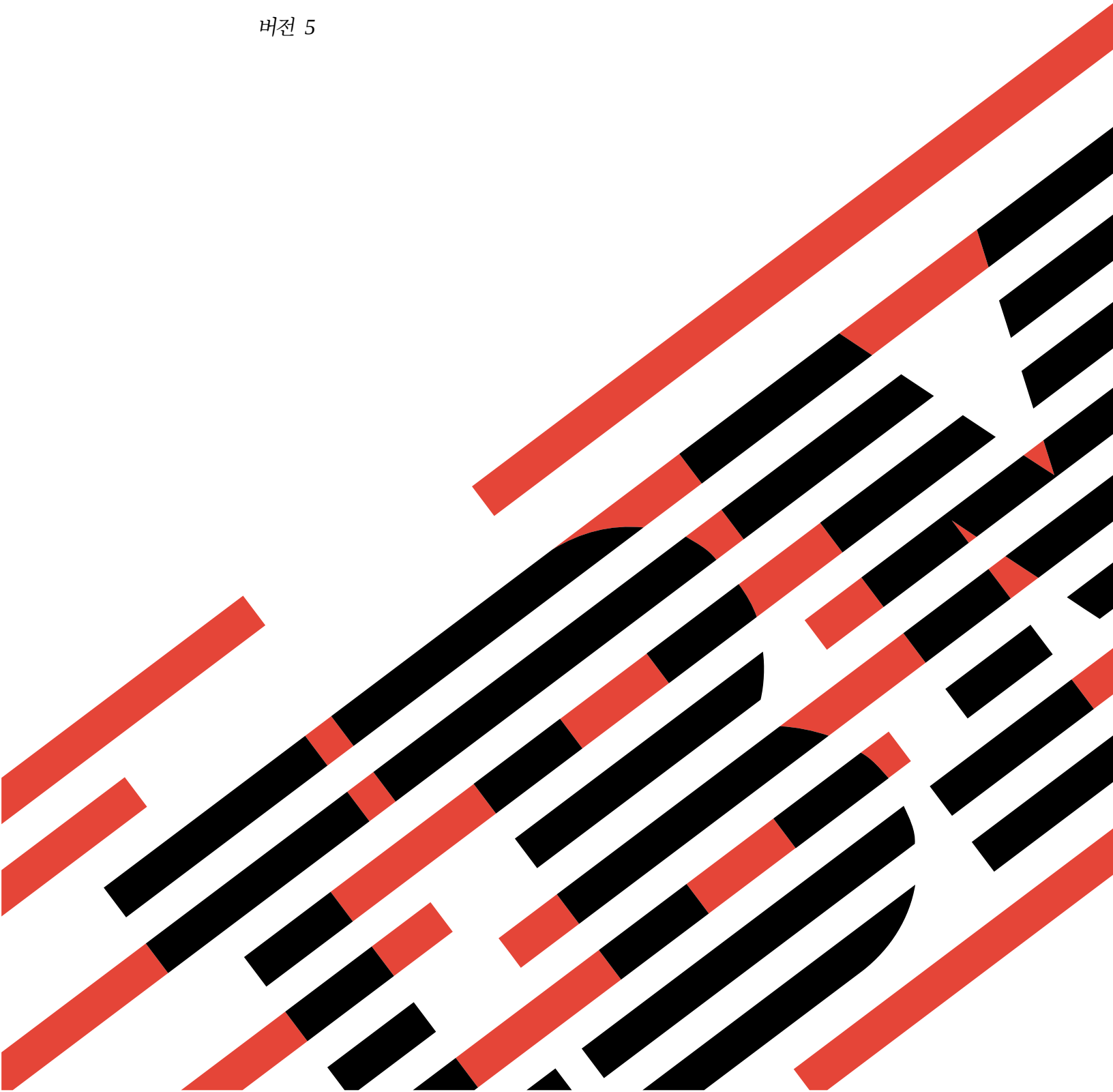
IBM

eserver

iSeries

iSeries용 DB2 Universal Database SQL 호출 레벨 인터페이스(ODBC)

버전 5



IBM

@server

iSeries

iSeries용 DB2 Universal Database SQL 호출 레벨 인터페이스(ODBC)

버전 5

목차

iSeries용 DB2 Universal Database SQL 호출 레벨 인터페이스 (ODBC) 정보

이 부분은 일반 DB2 UDB CLI 어플리케이션에 대한 개요로서 이 책에는 다음과 같은 정보가 들어 있습니다.

- DB2 UDB CLI를 소개하고 인터페이스의 배경과 내장 SQL과의 관계에 대해 설명합니다.
- DB2 UDB CLI 어플리케이션 내의 타스크 또는 단계에 대해 설명하며, 개념, 함수 및 이들 사이의 상호작용에 대해 소개합니다.
- DB2 UDB CLI를 구성하는 함수에 대한 참조 정보.
- 다음과 같은 부록이 들어 있습니다.
  - 275 페이지의 부록 A『DB2 UDB CLI 일반 진단 정보』에는 이 책 전반에서 참조되는 표가 있습니다.
  - 277 페이지의 부록 B『DB2 UDB CLI 포함 파일』에는 모든 DB2 UDB CLI 어플리케이션에 있는 헤더 파일을 나열합니다.
  - 297 페이지의 부록 C『DB2 UDB CLI 어플리케이션 코드 리스팅 예』에는 이 책 전반에서 사용되는 코드 세그먼트 예에 대한 전체 소스를 나열합니다.
  - 309 페이지의 부록 D『서버 모드로 DB2 UDB CLI 실행』에는 복수 사용자를 위해 CLI 어플리케이션 사용 방법에 대한 정보가 있습니다.

이 책에 관한 자세한 정보는 다음 주제를 참조하십시오.

- 『iSeries용 DB2 UDB SQL 호출 레벨 인터페이스(ODBC) 책의 사용자』
- 『iSeries용 DB2 UDB SQL 호출 레벨 인터페이스(ODBC) V5R2의 새로운 사항』
- viii 페이지의『코드 면책사항 관련 정보』

시작하려면 1 페이지의 제 1 장『CLI 소개』를 참조하십시오.

iSeries용 DB2 UDB SQL 호출 레벨 인터페이스(ODBC) 책의 사용자

이 책은 SQL과 C 프로그래밍 언어에 대한 지식을 갖고 있으면서 DB2 UDB CLI 함수를 사용하여 동적 SQL 문을 호출하고자 하는 어플리케이션 프로그래머를 대상으로 합니다.

iSeries용 DB2 UDB SQL 호출 레벨 인터페이스(ODBC) V5R2의 새로운 사항

이 릴리스에 다음 API가 추가되었습니다:

- SQLColumnPrivileges - 표의 열과 연관된 권한 확보
- SQLNextResult - 다음 결과 세트 처리
- SQLTablePrivileges - 표와 연관된 권한 확보

본 릴리스에서는 다음의 API가 삭제되었습니다.

- SQLBindParam - 버퍼를 매개변수 마커와 바인드
- SQLColAttributes - 열 속성
- SQLEndTran - 트랜잭션 확약 또는 롤백
- SQLGetConnectOption - 연결 옵션의 현재 설정 리턴
- SQLGetInfo - 일반 정보 얻기
- SQLGetLength - 스트링 값 길이 검색
- SQLGetStmtOption - 명령문 옵션의 현재 설정 리턴
- SQLProcedureColumns - 프로시듀어에 대한 입/출력(I/O) 매개변수 정보 얻기
- SQLProcedures - 프로시듀어명 리스트 얻기
- SQLSetConnectAttr - 연결 속성 설정
- SQLSetDescRec - 설명자 레코드 설정
- SQLSetEnvAttr - 환경 속성 설정
- SQLSetStmtAttr - 명령문 속성 설정
- SQLTables - 표 정보 얻기

CLI 소개 및 DB2® CLI 어플리케이션 작성의 일부 정보가 갱신되었습니다.

코드 면책사항 관련 정보

본 자료에는 프로그래밍 예들이 수록되어 있습니다.

IBM®은 귀하가 자신의 특정 요구에 맞게 유사한 기능을 생성한 모든 프로그래밍 코드 예를 사용함에 있어서 비독점적 저작권을 귀하에게 부여합니다.

IBM이 제공하는 모든 샘플 코드는 예시용입니다. 이 예들은 모든 환경에서 완벽하게 테스트가 이루어진 것이 아닙니다. 따라서 IBM은 이 프로그램의 신뢰성, 서비스 가능성 또는 기능을 보증 또는 암시하지 않습니다.

여기에 수록된 모든 프로그램들은 어떠한 종류의 보증없이 "현상태대로" 제공합니다. 비침해, 상품성, 특정 목적에의 적합성에 대한 암묵적 보증을 명시적으로 부인합니다.

제 1 장 CLI 소개

DB2 UDB 호출 레벨 인터페이스(CLI)는 zOS 및 OS/390®용 DB2 UDB 그리고 VSE 및 VM용 DB2 Server를 제외하고 모든 DB2 환경에서 지원되는 호출 가능 SQL(구조화 조회 언어) 프로그래밍 인터페이스입니다. 호출 가능한 SQL 인터페이스는 데이터베이스 액세스용 WinSock 어플리케이션 프로그램 인터페이스(API)로서, 함수 호출을 사용하여 동적 SQL문을 시작합니다.

DB2 UDB CLI는 내장 동적 SQL에 대한 하나의 대안입니다. 내장 동적 SQL과 DB2 UDB CLI 사이의 중요한 차이점은 SQL문이 시작되는 방법입니다. iSeries 시스템에서 이 인터페이스는 모든 ILE 언어에 사용할 수 있습니다.

DB2 UDB CLI는 또한 Microsoft® ODBC(Open Database Connectivity)는 물론 모든 레벨 2 함수를 지원합니다. 대부분의 경우 ODBC는 ANS와 ISO SQL CLI 표준의 수퍼세트입니다.

자세한 정보는 다음을 참조하십시오.

- 『DB2 UDB CLI의 기초 정보』
- 『호출 레벨 인터페이스』
- 4 페이지의『DB2 UDB CLI와 내장 SQL의 차이점』

DB2 UDB CLI의 기초 정보

DB2 UDB CLI 또는 호출 가능한 SQL 인터페이스의 기반을 이해하고 기존의 인터페이스와 비교하는 것이 중요합니다.

ISO 표준 9075:1999 - 데이터베이스 언어 SQL 파트 3: 호출 레벨 인터페이스는 CLI 표준 정의를 제공합니다. 이 인터페이스의 목적은 어플리케이션이 데이터베이스 서버와 독립적으로 작동하여 어플리케이션의 이식성을 높이기 위한 것입니다.

ODBC는 Windows®용 드라이버 관리자를 제공하는데, 이 기능은 각 ODBC 드라이버(ODBC 함수 호출을 구현하고 특정 DBMS와 상호작용하는 DLL 파일)에 대한 중앙 제어점 역할을 합니다.

호출 레벨 인터페이스

iSeries에서는 다음과 같은 호출 레벨 인터페이스 API를 사용하여 데이터베이스 액세스를 할 수 있습니다.

- 연결
  - 76 페이지의『SQLConnect - 자료 소스 연결』
  - 80 페이지의『SQLDataSources - 자료 소스 리스트 얻기』
  - 91 페이지의『SQLDisconnect - 자료 소스 단절』
  - 93 페이지의『SQLDriverConnect - 자료 소스에 (확장) 연결』

- 진단
  - 99 페이지의『SQLError - 오류 정보 검색』
  - 153 페이지의『SQLGetDiagField - 진단 정보(확장가능) 리턴』
  - 156 페이지의『SQLGetDiagRec - (간략한) 진단 정보 리턴』
- **MetaData**
  - 73 페이지의『SQLColumns - 표에 대한 열 정보 얻기』
  - 70 페이지의『SQLColumnPrivileges - 표의 열과 연관된 권한 확보』
  - 118 페이지의『SQLForeignKeys - 외부 키 열 리스트 얻기』
  - 164 페이지의『SQLGetInfo - 일반 정보 얻기』
  - 190 페이지의『SQLGetTypeInfo - 자료 유형 정보 얻기』
  - 195 페이지의『SQLLanguages - SQL 다이얼렉트 또는 적합성 정보 얻기』
  - 217 페이지의『SQLPrimaryKeys - 표의 1차 키 열 얻기』
  - 220 페이지의『SQLProcedureColumns - 프로시듀어에 대한 입/출력(I/O) 매개변수 정보 얻기』
  - 226 페이지의『SQLProcedures - 프로시듀어명 리스트 얻기』
  - 260 페이지의『SQLSpecialColumns - 특별한(행 ID) 열 얻기』
  - 264 페이지의『SQLStatistics - 기본 표에 대한 색인과 통계 정보 얻기』
  - 267 페이지의『SQLTablePrivileges - 표와 연관된 권한 확보』
  - 270 페이지의『SQLTables - 표 정보 얻기』
- **SQL문 처리**
  - 63 페이지의『SQLCancel - 명령문 취소』
  - 64 페이지의『SQLCloseCursor - 커서 명령문 닫기』
  - 65 페이지의『SQLColAttributes - 열 속성』
  - 84 페이지의『SQLDescribeCol - 열 속성 설명』
  - 88 페이지의『SQLDescribeParam - 매개변수 마커의 설명 리턴』
  - 97 페이지의『SQLEndTran - 트랜잭션 확약 또는 롤백』
  - 102 페이지의『SQLExecDirect - 명령문 직접 실행』
  - 104 페이지의『SQLExecute - 명령문 실행』
  - 106 페이지의『SQLExtendedFetch - 행 배열 페치』
  - 109 페이지의『SQLFetch - 다음 행 페치』
  - 116 페이지의『SQLFetchScroll - 스크롤할 수 있는 커서에서 페치』
  - 142 페이지의『SQLGetCursorName - 커서명 얻기』
  - 147 페이지의『SQLGetData - 열에서 자료 얻기』
  - 148 페이지의『SQLGetDescField - 설명자 필드 얻기』
  - 151 페이지의『SQLGetDescRec - 설명자 레코드 얻기』

- 249 페이지의『SQLSetEnvAttr - 환경 속성 설정』
- 254 페이지의『SQLSetParam - 매개변수 설정』
- 255 페이지의『SQLSetStmtAttr - 명령문 속성 설정』
- 258 페이지의『SQLSetStmtOption - 명령문 옵션 설정』

- **핸들에 대한 작업**
  - 29 페이지의『SQLAllocConnect - 연결 핸들 할당』
  - 32 페이지의『SQLAllocEnv - 환경 핸들 할당』
  - 35 페이지의『SQLAllocHandle - 핸들 할당』
  - 37 페이지의『SQLAllocStmt - 명령문 핸들 할당』
  - 79 페이지의『SQLCopyDesc - 설명문 복사』
  - 123 페이지의『SQLFreeConnect - 연결 핸들 해제』
  - 125 페이지의『SQLFreeEnv - 환경 핸들 해제』
  - 127 페이지의『SQLFreeHandle - 핸들 해제』
  - 129 페이지의『SQLFreeStmt - 명령문 핸들을 해제(또는 재설정)』
  - 232 페이지의『SQLReleaseEnv - 모든 환경 자원 해제』

DB2 UDB CLI와 내장 SQL의 차이점

내장 SQL 인터페이스를 사용하는 어플리케이션은 SQL문을 코드로 변환하기 위한 사전컴파일러가 필요합니다. 코드는 컴파일되어 데이터베이스로 바인드된 후 실행됩니다. 이와는 반대로, DB2 UDB CLI 어플리케이션은 사전컴파일이나 바인딩이 필요없는 대신 실행시 SQL문 및 관련 서비스를 실행하는 표준 함수 집합을 사용합니다.

이러한 차이가 중요한 이유는 사전컴파일러는 보통 어플리케이션과 데이터베이스 제품을 효과적으로 연계시켜 주는 데이터베이스 제품에 대해서만 사용 가능하기 때문입니다. DB2 UDB CLI를 사용하면, 특정 데이터베이스 제품에 독립적인 이식 가능한 어플리케이션에 기록할 수 있습니다. 이러한 독립성 때문에 DB2 UDB CLI 어플리케이션은 서로 다른 데이터베이스 제품에 액세스하기 위해 다시 컴파일되거나 다시 바인드될 필요가 없습니다. 어플리케이션은 실행시 적절한 데이터베이스 제품을 선택합니다.

DB2 UDB CLI와 내장 SQL의 차이점은 다음과 같습니다.

- DB2 UDB CLI는 커서에 대한 명시적 선언이 필요없습니다. DB2 UDB CLI는 필요한 만큼 커서를 생성합니다. 어플리케이션은 생성된 커서를 여러 행의 `SELECT` 명령문과 위치 지정된 `UPDATE` 및 `DELETE` 명령문에서 정상적 커서 페치 모델로 사용할 수 있습니다.
- `OPEN` 명령문은 DB2 UDB CLI에서 필요없습니다. 대신 `SELECT`를 실행시키면 자동으로 커서가 열립니다.
- 내장 SQL과는 달리, DB2 UDB CLI를 사용하면 `EXECUTE IMMEDIATE` 명령문(`SQLExecDirect()` 함수)에 해당하는 매개변수 마커를 사용할 수 있습니다.
- DB2 UDB CLI에서 `COMMIT` 또는 `ROLLBACK`은 SQL문으로서 전달되는 것이 아니라, `SQLTransact()`나 `SQLEndTran()` 함수 호출을 통해 제공됩니다.

- DB2 UDB CLI는 어플리케이션을 대신하여 명령문 관련 정보를 관리하며, 이 정보를 추상적 오브젝트로서 참조하기 위해 ***명령문 핸들***을 제공합니다. 이 핸들을 사용하면 제품 고유의 자료 구조를 사용하는 어플리케이션이 필요하지 않습니다.
- 명령문 핸들과 비슷하게, ***환경 핸들*** 및 ***연결 핸들***에는 모든 글로벌 변수와 연결에 따른 정보를 참조할 수 있는 방법이 있습니다.
- DB2 UDB CLI는 X/Open SQL CAE 스펙이 정의하는 SQLSTATE 값을 사용합니다. 그 형식과 여러 값이 IBM 관계형 데이터베이스 제품에서 사용되는 값과 일치하더라도 차이점은 분명히 있습니다.

이러한 차이에도 불구하고 내장 SQL과 DB2 UDB CLI 사이에는 중요한 공통 개념이 있습니다.

DB2 UDB CLI는 내장 SQL에서 동적으로 준비될 수 있는 모든 SQL문을 실행할 수 있습니다. 이렇게 되는 이유는 DB2 UDB CLI가 실제로 SQL문 자체를 *실행*하는 것이 아니라 동적 실행을 위해 DBMS로 SQL문을 전달하기 때문입니다.

표 1은 각 SQL문과 이 SQL문이 DB2 UDB CLI를 사용하여 실행될 수 있는지를 나타냅니다.

표 1. SQL문

SQL문	**Dyn**[a]	**CLI**[c]
ALTER TABLE	X	X
BEGIN DECLARE SECTION[b]		
CALL	X	X
CLOSE		SQLFreeStmt()
COMMENT ON	X	X
COMMIT	X	SQLTransact(), SQLEndTran()
CONNECT(Type 1)		SQLConnect()
CONNECT(Type 2)		SQLConnect()
CREATE INDEX	X	X
CREATE TABLE	X	X
CREATE VIEW	X	X
DECLARE CURSOR[b]		SQLAllocStmt()
DELETE	X	X
DESCRIBE		SQLDescribeCol(), SQLColAttributes()
DISCONNECT		SQLDisconnect()
DROP	X	X
END DECLARE SECTION[b]		
EXECUTE		SQLExecute()
EXECUTE IMMEDIATE		SQLExecDirect()
FETCH		SQLFetch()
GRANT	X	X
INCLUDE[b]		
INSERT	X	X
LOCK TABLE	X	X
OPEN		SQLExecute(), SQLExecDirect()

표 1. SQL문 (계속)

SQL문	Dyn[a]	CLI[c]
PREPARE		SQLPrepare()
RELEASE		SQLDisconnect()
REVOKE	X	X
ROLLBACK	X	SQLTransact(), SQLEndTran()
SELECT	X	X
SET CONNECTION		
UPDATE	X	X
WHENEVER[b]		

주:

[a] **Dyn**은 동적임을 의미합니다. 이 리스트에 나와 있는 **X**로 표시되는 명령문만 동적 SQL로 코딩되고, 그외의 명령문은 정적 SQL로 코딩될 수 있습니다.

[b] 실행할 수 없는 명령문을 나타냅니다.

[c] X는 이 명령문이 SQLExecDirect()를 사용하거나 SQLPrepare() 및 SQLExecute()를 사용하여 실행될 수 있음을 나타냅니다. 이에 해당하는 DB2 UDB CLI 함수가 있는 경우에는 함수 이름이 나열됩니다.

각 DBMS에는 동적으로 준비 가능한 추가 명령문이 있을 수 있으며 이 경우, DB2 UDB CLI는 해당 명령문을 DBMS로 전달합니다. 이 경우, COMMIT와 ROLLBACK은 일부 DBMS에 의해 동적으로는 준비 가능하지만, 전달되지 않는 것이 한 가지 예외입니다. 대신 SQLTransact()나 SQLEndTran()은 COMMIT나 ROLLBACK을 지정하는 데 사용해야 합니다.

자세한 정보는 다음을 참조하십시오.

- 『내장 SQL을 대신한 DB2 UDB CLI 사용의 장점』
- 7 페이지의 『DB2 UDB CLI, 동적 SQL 및 정적 SQL 사이의 결정』

내장 SQL을 대신한 DB2 UDB CLI 사용의 장점

DB2 UDB CLI 인터페이스는 내장 SQL에 비해 여러 가지 장점이 있습니다.

- DB2 CLI 인터페이스는 어플리케이션이 구축될 때 목표 데이터베이스를 알 수 없는 클라이언트/서버 환경에 이상적이며, 어플리케이션이 어떠한 데이터베이스 서버에 연결되는지 상관없이 SQL문 실행을 위한 일관적인 인터페이스를 제공합니다.
- 또한 사전컴파일러에 의존하지 않으므로 어플리케이션의 이식성을 향상시킵니다. 어플리케이션은 컴파일된 어플리케이션이나 실행시 라이브러리로 분배되는 것이 아니라 각 데이터베이스 제품에 대해 사전처리되는 소스 코드로 분배됩니다.
- DB2 UDB CLI 어플리케이션은 연결할 각 데이터베이스에 바인드될 필요가 없습니다.
- DB2 UDB CLI 어플리케이션은 여러 데이터베이스에 동시에 연결될 수 있습니다.
- DB2 UDB CLI 어플리케이션은 내장 SQL 어플리케이션과 함께 있을 때에는 SQLCA 및 SQLDA와 같은 글로벌 자료 영역을 제어하지 않아도 됩니다. 대신, DB2 UDB CLI는 필요한 자료 구조를 할당하고 제어하며, 자료 구조를 참조하기 위한 *핸*들을 어플리케이션에 제공합니다.

DB2 UDB CLI, 동적 SQL 및 정적 SQL 사이의 결정

어플리케이션에 따라 인터페이스를 선택해야 합니다.

DB2 UDB CLI는 특정 DBMS(예: 카탈로그 데이터베이스, 백업, 복원)가 제공하는 유틸리티나 API는 필요 없지만, 이식성이 필요한 조회 기반 어플리케이션에 이상적입니다. 그러나 DB2 UDB CLI를 사용한다고 해서 어플리케이션으로부터 DBMS 특정 API를 호출하는 것은 아닙니다. 즉, 어플리케이션이 더 이상 이식가능하지 않음을 의미합니다.

동적 SQL과 정적 SQL 사이의 성능 비교도 중요하게 고려해야 합니다. 동적 SQL은 실행시 준비되는 반면 정적 SQL은 사전컴파일 단계에서 준비됩니다. 명령문을 준비하는 데에는 처리 시간이 추가로 필요하므로, 정적 SQL이 보다 효율적일 수 있습니다. 동적 SQL 대신 정적 SQL을 선택하는 경우, DB2 UDB CLI는 옵션이 아닙니다.

대부분의 경우, 개인 기호에 따라 두 인터페이스 중 하나를 선택합니다. 이전 경험에 비추어 한 인터페이스가 다른 것 보다 더 직관적으로 보일 수 있습니다.

제 2 장 DB2 UDB CLI 어플리케이션 작성

DB2 UDB CLI 어플리케이션은 타스크 세트로 구성되고 타스크 세트는 일련의 불연속적인 단계로 구성됩니다. 어플리케이션 실행 과정에서, 어플리케이션 전반에 걸쳐 다른 타스크가 발생할 수 있습니다. 어플리케이션은 각 타스크를 수행하기 위해 하나 이상의 DB2 UDB CLI 함수를 호출합니다.

모든 DB2 UDB CLI 어플리케이션에는 그림 1에서와 같은 세 개의 기본 타스크가 있습니다. 그림과 같은 순서대로 함수가 호출되지 않으면 오류가 발생됩니다.

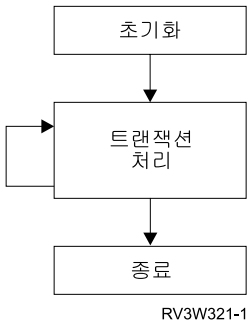


그림 1. DB2 UDB CLI 어플리케이션의 개념적 뷰

초기화 이 타스크는 *트랜잭션 처리* 기본 타스크를 준비할 때 일부 자원을 할당하고 초기화합니다. 세부사항은 10 페이지의 『DB2 UDB CLI 어플리케이션에서의 타스크 초기화 및 종료』를 참조하십시오.

트랜잭션 처리

이것은 어플리케이션의 기본 타스크입니다. SQL문은 그 조회 및 수정을 위해 DB2 UDB CLI로 전달됩니다. 세부사항은 13 페이지의 『DB2 UDB CLI 어플리케이션에서의 트랜잭션 처리 타스크』를 참조하십시오.

종료 이 타스크는 할당된 자원을 해제합니다. 자원은 일반적으로 고유 핸들에 의해 식별되는 자료 영역으로 구성됩니다. 자원을 해제한 후에는 다른 타스크가 이 핸들을 사용합니다. 세부사항은 10 페이지의 『DB2 UDB CLI 어플리케이션에서의 타스크 초기화 및 종료』를 참조하십시오.

위의 세 가지 타스크 외에도 진단 메세지 처리와 같은 일반 타스크가 있는데, 이런 타스크는 어플리케이션 전반에 걸쳐 발생합니다.

이 주제에는 이러한 함수가 DB2 UDB CLI 어플리케이션에서 어떻게 사용되는지에 관한 예가 나옵니다.

자세한 정보는 다음을 참조하십시오.

- 10 페이지의 『DB2 UDB CLI 어플리케이션에서의 타스크 초기화 및 종료』
- 13 페이지의 『DB2 UDB CLI 어플리케이션에서의 트랜잭션 처리 타스크』
- 18 페이지의 『DB2 UDB CLI 어플리케이션에서의 진단』
- 20 페이지의 『DB2 UDB CLI 함수에서의 자료 유형 및 자료 변환』

• 22 페이지의『DB2 UDB CLI 함수에서의 스트링 인수에 대한 작업』

각 함수에 대한 설명과 사용법은 25 페이지의 제 3 장『DB2 UDB CLI 함수』를 참조하십시오.

DB2 UDB CLI 어플리케이션에서의 타스크 초기화 및 종료

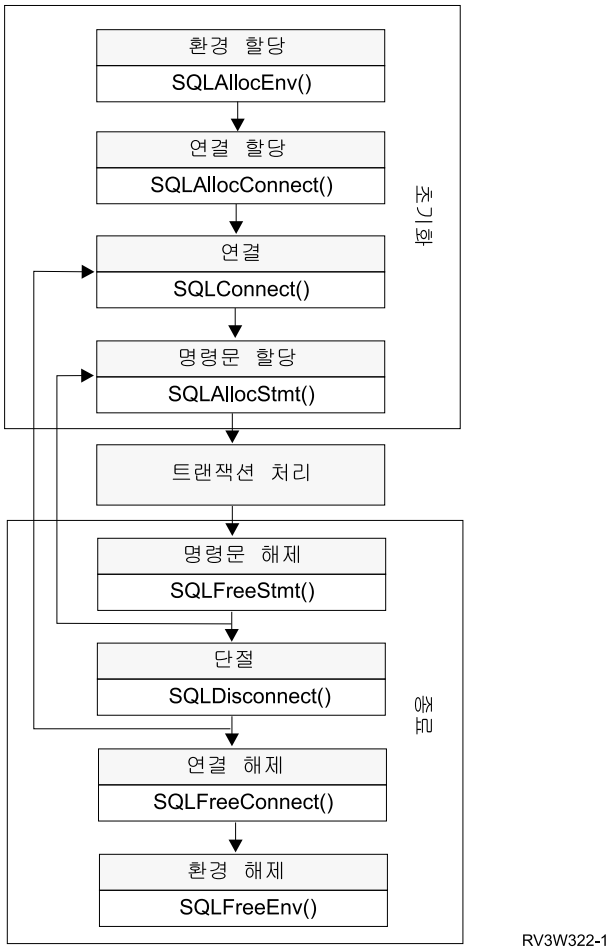


그림 2. 초기화 및 종료 타스크의 개념적 뷰

그림 2는 초기화 및 종료 타스크에 대한 함수 호출 순서입니다. 다이어그램 가운데 있는 트랜잭션 처리 타스크는 13 페이지의 그림 3에서 볼 수 있습니다.

초기화 및 종료는 환경 핸들 및 연결 핸들을 할당하고 초기화합니다. 종료 타스크는 이 핸들을 해제합니다. 핸들은 DB2 UDB CLI가 제어하는 자료 오브젝트를 참조하는 변수입니다. 핸들을 사용하면, 어플리케이션은 IBM DBMS용 내장 SQL 인터페이스에서 사용되는 SQLCA나 SQLDA와 같은 글로벌 변수나 자료 구조를 할당하고 관리할 필요가 없습니다. 어플리케이션은 다른 DB2 UDB CLI 함수를 호출할 때 적절한 핸들을 전달합니다. 핸들의 종류는 다음 세 가지입니다.

환경 핸들

환경 핸들은 어플리케이션의 상태에 대한 전체적인 정보를 담고 있는 자료 오브젝트를 참조합니다. 이

핸들은 SQLAllocEnv() 호출을 통해 할당되며, SQLFreeEnv() 호출을 통해 해제됩니다. 환경 핸들은 연결 핸들이 할당되기 전에 할당해야 합니다. 어플리케이션마다 단 하나의 환경 핸들만 할당될 수 있습니다.

연결 핸들

연결 핸들은 DB2 UDB CLI가 관리하는 연결과 연관된 정보를 담고 있는 자료 오브젝트를 참조합니다. 이 정보에는 일반적인 상태 정보, 트랜잭션 상태, 진단 정보가 있습니다. 각 연결 핸들은 SQLAllocConnect() 호출을 통해 할당되고, SQLFreeConnect() 호출을 통해 해제됩니다. 어플리케이션은 각 연결에 대한 연결 핸들을 데이터베이스 서버에 할당해야 합니다.

명령문 핸들

명령문 핸들은 다음 타스크에서 설명됩니다.

『예: DB2 UDB CLI 어플리케이션에서의 초기화 및 연결』을 참조하십시오.

예: DB2 UDB CLI 어플리케이션에서의 초기화 및 연결

코드 예와 관련된 정보는 viii 페이지의 『코드 면책사항 관련 정보』를 참조하십시오.

```
/*****************************************************
** file = basiccon.c
**    - demonstrate basic connection to two datasources.
**    - error handling  ignored for simplicity
**
** Functions used:
**
**    SQLAllocConnect  SQLDisconnect
**    SQLAllocEnv      SQLFreeConnect
**    SQLConnect       SQLFreeEnv
**
**
*****************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "sqlcli.h"

int
connect(SQLHENV henv,
        SQLHDBC * hdbc);

#define MAX_DSN_LENGTH    18
#define MAX_UID_LENGTH    10
#define MAX_PWD_LENGTH    10
#define MAX_CONNECTIONS   5

int
main()
{
    SQLHENV     henv;
    SQLHDBC         hdbc[MAX_CONNECTIONS];

    /* allocate an environment handle   */
    SQLAllocEnv(&henv);
```

```
    /* Connect to first data source */
    connect(henv, &hdbc[0];);

    /* Connect to second data source */
    connect(henv, &hdbc[1];);

    /*********   Start Processing Step  ************************/
    /* allocate statement handle, execute statement, and so forth       */
    /*********   End Processing Step  *************************/

    printf("\nDisconnecting .....\n");
    SQLDisconnect(hdbc[0]);     /* disconnect first connection   */
    SQLDisconnect(hdbc[1]);     /* disconnect second connection  */
    SQLFreeConnect(hdbc[0]);    /* free first connection handle  */
    SQLFreeConnect(hdbc[1]);    /* free second connection handle */
    SQLFreeEnv(henv);           /* free environment handle       */

    return (SQL_SUCCESS);
}

/*********************************************************************
**   connect - Prompt for connect options and connect              **
*********************************************************************/

int
connect(SQLHENV henv,
        SQLHDBC * hdbc)
{
    SQLRETURN   rc;
    SQLCHAR         server[MAX_DSN_LENGTH + 1], uid[MAX_UID_LENGTH + 1],
pwd[MAX_PWD_LENGTH
+ 1];
    SQLCHAR         buffer[255];
    SQLSMALLINT     outlen;

    printf("Enter Server Name:\n");
    gets((char *) server);
    printf("Enter User Name:\n");
    gets((char *) uid);
    printf("Enter Password Name:\n");
    gets((char *) pwd);

    SQLAllocConnect(henv, hdbc);/* allocate a connection handle     */

    rc = SQLConnect(*hdbc, server, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
    if (rc != SQL_SUCCESS) {
        printf("Error while connecting to database\n");
    return ( SQL_ERROR);
    } else {
        printf("Successful Connect\n");
    return (SQL_SUCCESS);
    }
}
```

DB2 UDB CLI 어플리케이션에서의 트랜잭션 처리 타스크

다음 그림은 DB2 UDB CLI 어플리케이션에서 생기는 일반적인 함수 호출 순서입니다. 모든 함수 또는 가능한 경로가 표시되지 않습니다.

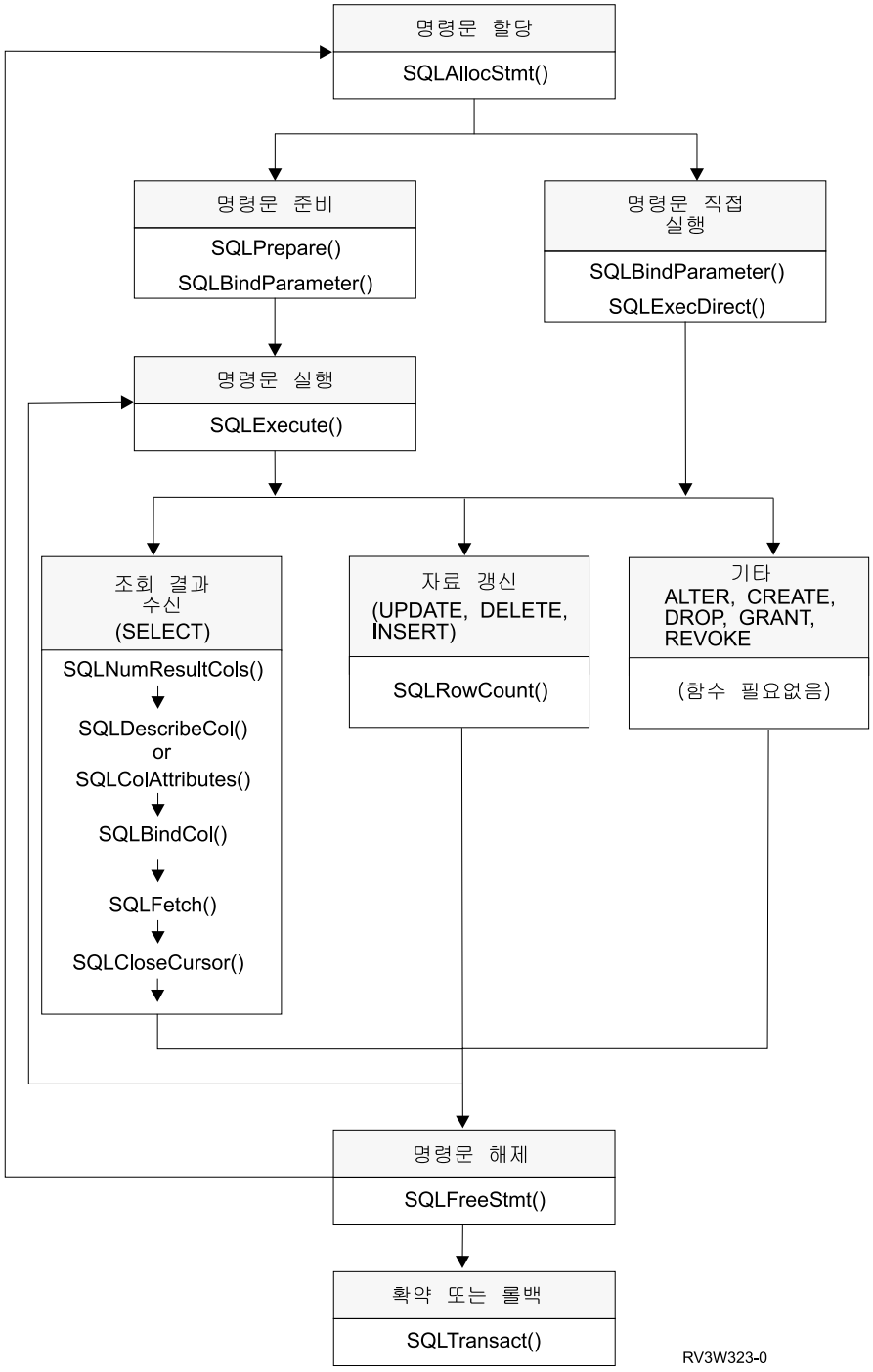


그림 3. 트랜잭션 처리

그림 3은 트랜잭션 처리 타스크의 단계와 DB2 UDB CLI 함수입니다. 이 타스크에는 다음과 같이 5단계가 있습니다.

- 『DB2 UDB CLI 어플리케이션에서의 명령문 핸들 할당』
- 『DB2 UDB CLI 어플리케이션에서의 준비 및 실행 타스크』
- 15 페이지의 『DB2 UDB CLI 어플리케이션에서의 결과 처리』
- 17 페이지의 『DB2 UDB CLI 어플리케이션에서의 명령문 핸들 해제』
- 17 페이지의 『DB2 UDB CLI 어플리케이션에서의 확약 또는 롤백』

SQLAllocStmt 함수는 SQL문을 처리할 때 사용할 명령문 핸들을 확보하기 위해 필요합니다. 명령문 실행에 사용할 수 있는 방법은 두 가지입니다. SQLPrepare와 SQLExecute를 사용하여 프로그램을 두 단계로 나눌 수 있습니다. SQLBindParameter 함수는 준비된 SQL문에 사용되는 호스트 변수의 프로그램 주소를 바인드하기 위해 필요합니다. 두 번째 방법은 SQLPrepare와 SQLExecute를 SQLExecDirect에 대한 단일 호출로 대체시키는 직접 실행 방법입니다.

일단 명령문이 실행되면 나머지 처리는 SQL문의 유형에 따라 결정됩니다. SELECT문의 경우 프로그램이 결과 세트 처리에 SQLNumResultCols, SQLDescribeCol, SQLBindCol, SQLFetch, SQLCloseCursor 등의 함수를 사용합니다. 자료를 갱신하는 명령문의 경우에는 영향을 받는 행 수를 알아보기 위해 SQLRowCount를 사용할 수 있습니다. 기타 유형의 SQL문에서는 명령문이 실행된 후 처리가 완료됩니다. 그런 후 SQLFreeStmt를 사용하여 모든 경우에서 더 이상 핸들이 필요 없음을 나타냅니다.

DB2 UDB CLI 어플리케이션에서의 명령문 핸들 할당

SQLAllocStmt()는 명령문 핸들을 할당합니다. 명령문 핸들은 DB2 UDB CLI가 관리하는 SQL문에 대한 정보를 담고 있는 자료 오브젝트를 참조합니다. 여기에는 동적 인수, 커서 정보, 동적 인수 및 열에 대한 바인딩, 결과 값, 상태 정보와 같은 정보들이 있습니다. (이러한 정보들은 나중에 설명됩니다.) 각 명령문 핸들은 연결 핸들과 연관이 있습니다.

명령문을 실행하기 위해 명령문 핸들을 할당합니다. 동시 할당 핸들 수의 최대 수는 80,000으로 제한됩니다. 이 제한은 모든 종류의 핸들에 적용되며, 구현 코드에 의해 암시적으로 할당되는 설명자 핸들에도 적용됩니다. 또한 리모트 연결의 경우 명령문 핸들이 400개로 제한됩니다.

DB2 UDB CLI 어플리케이션에서의 준비 및 실행 타스크

명령문 핸들이 할당되고 난 후에, SQL문을 지정하고 실행하는 방법에는 두 가지가 있습니다.

1. 준비 및 실행
   a. SQL문을 인수로 하여 SQLPrepare()를 호출하십시오.
   b. SQL문에 *매개변수 마커*가 있는 경우에는 SQLSetParam()을 호출하십시오.
   c. SQLExecute()를 호출하십시오.
2. 직접 실행
   a. SQL문에 *매개변수 마커*가 있는 경우에는 SQLSetParam()을 호출하십시오.
   b. SQL문을 인수로 하여 SQLExecDirect()를 호출하십시오.

첫 번째 방법은 명령문의 준비와 실행을 구분합니다. 이 방법이 사용되는 경우는 다음과 같습니다.

- 명령문이 반복적으로 실행됩니다(보통 매개변수 값은 다름). 이 경우에는 같은 명령문을 한 번만 준비하면 됩니다.
- 어플리케이션이 명령문 실행 전에, 결과 집합의 열에 대한 정보를 요구하는 경우.

두 번째 방법은 준비 단계와 실행 단계를 하나로 결합합니다. 이 방법이 사용되는 경우는 다음과 같습니다.

- 명령문이 한 번 실행됩니다. 이 경우에는 명령문을 실행하기 위해 두 개의 함수를 호출할 필요가 없습니다.
- 어플리케이션이 명령문 실행 전에, 결과 집합의 열에 대한 정보를 요구하지 않는 경우.

DB2 UDB CLI 어플리케이션에서의 SQL문 매개변수 바인딩

두 방법을 실행하면, SQL문의 *표현식*(또는 내장 SQL의 호스트 변수) 대신에 매개변수 마커를 사용할 수 있습니다.

매개변수 마커는 '?' 문자로 표시되며, 명령문이 실행될 때 어플리케이션 변수의 내용이 대체되는 SQL문 내의 위치를 표시합니다. 마커는 1부터 시작하여 왼쪽에서 오른쪽으로 순서대로 참조됩니다.

어플리케이션 변수가 매개변수 마커와 연관있는 경우, 어플리케이션 변수는 매개변수 마커에 *바인드*됩니다. 바인딩은 `SQLSetParam()` 함수를 다음과 함께 호출하면 실행됩니다.

- 매개변수 마커의 번호
- 어플리케이션 변수에 대한 포인터
- 매개변수의 SQL 유형
- 자료 유형 및 변수의 길이

어플리케이션 변수는 *연기된* 인수라고 하는데, 그 이유는 `SQLSetParam()`가 호출될 때 포인터만 전달되기 때문입니다. 명령문이 실행되어야만 변수로부터 자료를 읽어들일 수 있습니다. 이것은 버퍼 인수 및 버퍼의 자료 길이를 나타내는 인수에도 적용됩니다. 연기된 인수 때문에, 어플리케이션은 바인드된 매개변수의 내용을 수정하고 새 값으로 명령문을 반복 실행할 수 있습니다.

`SQLSetParam()`을 호출할 때, SQL문이 요구하는 것과 다른 유형의 변수를 바인드할 수 있습니다. 이 경우에는 DB2 UDB CLI가 바인드 변수의 내용을 올바른 유형으로 변환합니다. 예를 들면, SQL문은 정수 값을 요구하지만 어플리케이션은 정수를 스트링으로 표시할 수도 있습니다. 스트링은 매개변수에 바인드될 수 있으며, DB2 UDB CLI는 명령문이 실행될 때 스트링을 정수로 변환합니다. 자료 변환에 대한 자세한 정보는 20 페이지의 『DB2 UDB CLI 함수에서의 자료 유형 및 자료 변환』을 참조하십시오.

자세한 내용과 예에 대해서는 다음을 참조하십시오.

- 212 페이지의 『SQLPrepare - 명령문 준비』
- 254 페이지의 『SQLSetParam - 매개변수 설정』
- 104 페이지의 『SQLExecute - 명령문 실행』
- 102 페이지의 『SQLExecDirect - 명령문 직접 실행』

DB2 UDB CLI 어플리케이션에서의 결과 처리

명령이 실행된 후의 단계는 SQL문의 유형에 따라 달라집니다.

DB2 UDB CLI 어플리케이션에서의 SELECT문 처리

명령문이 SELECT인 경우, 결과 집합의 각 행을 검색하려면 보통 다음과 같은 단계가 필요합니다.

1. 결과 집합의 구조, 열의 수, 열의 유형과 길이를 설정합니다.
2. (선택사항) 자료를 받기 위해 어플리케이션 변수를 선택적으로 열에 바인드합니다.
3. 반복적으로 다음 자료 행을 인출하여 이를 바인드 어플리케이션 변수로 받아들입니다.
4. (선택사항) 미리 바인드되지 않은 열은 각 인출이 성공할 때마다 SQLGetData()를 호출하여 검색할 수 있습니다.

주: 위의 각 단계마다 진단 검사가 필요합니다.

첫 번째 단계에서는 실행되거나 준비된 명령문을 분석해야 합니다. SQL문이 어플리케이션에 의해 생성된 경우에는 이 단계가 필요없습니다. 왜냐하면 어플리케이션이 결과 집합의 구조와 각 열의 자료 유형을 알고 있기 때문입니다. SQL문이 실행시에 생성된 경우에는 (예를 들면, 사용자가 입력한 경우) 어플리케이션은 다음을 조회해야 합니다.

- 열의 수
- 각 열의 유형
- 결과 집합에서 각 열의 이름

이 정보는 명령문을 준비하거나 실행한 후, SQLNumResultCols()과 SQLDescribeCol() (또는 SQLColAttributes())를 호출하여 얻을 수 있습니다.

두 번째 단계에서 어플리케이션은 SQLFetch()의 다음 호출에서 열 자료를 어플리케이션 변수로 직접 변경할 수 있습니다. 변경할 각 열에 대해, 어플리케이션은 SQLBindCol()를 호출하여 어플리케이션 변수와 결과 집합의 열을 바인드합니다. SQLSetParam()를 사용하여 변수를 매개변수 마커에 바인드하는 경우와 비슷하게, 열은 연기된 인수를 사용하여 바인드됩니다. 이번에는 변수가 출력 인수이고, 자료는 SQLFetch()가 호출될 때 변수에 기록됩니다. SQLGetData()도 자료를 검색하는 데 사용될 수 있으므로, SQLBindCol() 호출은 선택적입니다.

세 번째 단계는 결과 집합의 처음 또는 다음 행을 인출하기 위해 SQLFetch()를 호출하는 것입니다. 임의의 열이 바인드되어 있는 경우에는, 어플리케이션 변수가 갱신됩니다. 자료 변환이 SQLBindCol 호출에 지정된 자료 유형에 의해 표시되었다면, 변환은 SQLFetch()이 호출될 때 발생됩니다. 자료 변환에 대한 설명은 20 페이지의 『DB2 UDB CLI 함수에서의 자료 유형 및 자료 변환』을 참조하십시오.

마지막 단계(선택사항)는 미리 바인드되지 않은 열을 검색하기 위해 SQLGetData()를 호출하는 것입니다. 모든 열이 바인드되어 있지 않다면 이 방법으로 검색할 수 있으며, 또한 두 가지 방법을 조합해서 사용할 수 있습니다. SQLGetData()는 더 작은 단위로 변수 길이 열을 검색하는 데 사용될 수 있지만, 바인드 열에는 사용할 수 없습니다. 자료 변환은 SQLBindCol()에서와 같이, 여기에서도 표시될 수 있습니다. 자세한 정보는 20 페이지의 『DB2 UDB CLI 함수에서의 자료 유형 및 자료 변환』을 참조하십시오.

자세한 내용과 예에 대해서는 다음을 참조하십시오.

- 39 페이지의 『SQLBindCol - 어플리케이션 변수에 열 바인드』

- 65 페이지의『SQLColAttributes - 열 속성』
- 84 페이지의『SQLDescribeCol - 열 속성 설명』
- 109 페이지의『SQLFetch - 다음 행 페치』
- 147 페이지의『SQLGetData - 열에서 자료 얻기』
- 206 페이지의『SQLNumResultCols - 결과 열의 수 얻기』

DB2 UDB CLI 어플리케이션에서의 UPDATE, DELETE 및 INSERT문 처리

명령문이 자료(UPDATE, DELETE 또는 INSERT)를 수정하는 경우, 진단 메세지에 대한 일반적인 검사를 제외한 다른 조치는 필요하지 않습니다. 이 경우에, SQL문의 영향을 받는 행의 수를 알기 위해 SQLRowCount()를 사용할 수 있습니다. 자세한 정보는 206 페이지의『SQLNumResultCols - 결과 열의 수 얻기』를 참조하십시오.

SQL문이 위치지정된 UPDATE 또는 DELETE인 경우, *커서*를 사용할 필요가 있습니다. 커서는 SELECT 명령문의 결과표에서 행에 대하여 이동 가능한 포인터입니다. 내장 SQL의 경우, 커서는 행을 검색하고, 갱신하거나 삭제하는 데 사용됩니다. DB2 UDB CLI를 사용하는 경우에는, 커서가 자동으로 생성되므로 커서를 정의할 필요가 없습니다.

위치지정된 UPDATE나 DELETE 명령문의 경우, SQL문 안에 커서의 이름을 지정해야 합니다. SQLSetCursorName()을 사용하여 커서명을 정의할 수 있고, SQLGetCursorName()을 사용하여, 생성된 커서의 이름을 조회할 수 있습니다. 모든 오류 메세지는 SQLSetCursorName()이 정의한 이름이 아니라, 생성된 커서의 이름을 참조하므로, 생성된 이름을 사용하는 것이 가장 좋습니다.

DB2 UDB CLI 어플리케이션에서의 기타 SQL문 처리

명령문이 자료를 조회하지도 않고 수정하지도 않는 경우, 진단 메세지에 일반적인 검사외의 다른 조치는 필요없습니다.

DB2 UDB CLI 어플리케이션에서의 명령문 핸들 해제

특정 명령문 핸들에 대한 처리를 종료하려면 SQLFreeStmt()를 호출하십시오. 이 함수는 다음과 같은 작업을 한 가지 이상 수행하는 데 사용됩니다.

- 모든 열의 바인드 해제
- 모든 매개변수 바인드 해제
- 커서 닫기 및 결과 삭제
- 명령문 핸들 제거 및 모든 연관 자원 해제

명령문 핸들은 제거되지만 않으면 재사용할 수 있습니다.

DB2 UDB CLI 어플리케이션에서의 확약 또는 롤백

마지막 단계는 SQLTransact()를 사용하여 *트랜잭션*을 확약하거나 롤백하는 것입니다.

트랜잭션이란 복구 가능한 작업 단위 또는 하나의 아토믹 조작으로 취급할 수 있는 SQL문의 그룹을 말합니다. 즉, 그룹 내의 모든 동작은 마치 그들이 하나의 동작인 것처럼 완료(확약)되거나 실행취소(롤백)됩니다.

DB2 UDB CLI를 사용하는 경우, SQLPrepare(), SQLExecDirect() 또는 SQLGetTypeInfo()를 사용하면 트랜잭션은 데이터베이스에 처음 액세스할 때 암시적으로 시작됩니다. 트랜잭션은 트랜잭션을 롤백하거나 확약하기 위해 SQLTransact()를 사용할 때 끝납니다. 즉, 이 둘 사이에서 실행된 SQL문들은 하나의 작업 단위로 처리됩니다.

DB2 UDB CLI 어플리케이션에서의 SQLTransact() 호출 시기

트랜잭션을 종료할 시점을 결정할 때는 다음을 고려하십시오.

- 현재의 트랜잭션은 확약하거나 롤백만 할 수 있으므로, 같은 트랜잭션 안에 독립적인 명령문을 두십시오.
- 처리못한 트랜잭션이 있는 동안에는 다양한 잠금 처리를 할 수 있습니다. 트랜잭션이 종료되면 잠금이 해제되며, 다른 사용자가 자료에 액세스할 수 있습니다. 이것은 SELECT 명령문을 비롯한 모든 SQL문에 해당됩니다.
- 트랜잭션이 성공적으로 확약되거나 롤백되었으면, 시스템 기록부로부터 완벽하게 복구시킬 수 있습니다(DBMS에 따라 달라짐). 열려있는 트랜잭션은 복구되지 않습니다.

DB2 UDB CLI 어플리케이션에서 SQLTransact() 호출이 미치는 영향

트랜잭션이 종료되는 경우에는 다음 사항을 확인하십시오.

- 모든 명령문은 다시 사용되기 전에 준비되어 있어야 합니다.
- 커서명, 바인드 매개변수, 열 바인딩은 트랜잭션 사이에서 유지보수됩니다.
- 모든 열린 커서가 닫힙니다.

자세한 정보와 예에 대해서는 273 페이지의 『SQLTransact - 트랜잭션 관리』를 참조하십시오.

DB2 UDB CLI 어플리케이션에서의 진단

진단이란 어플리케이션 내에서 생성된 경고나 오류 상태를 처리하는 것을 말합니다. DB2 UDB CLI 함수를 호출하는 경우, 진단에는 두 가지 레벨이 있습니다.

- 『DB2 UDB CLI 어플리케이션의 리턴 코드』
- 19 페이지의 『DB2 UDB CLI SQLSTATE』(진단 메세지)

오류 처리에 대한 예는 99 페이지의 『SQLError - 오류 정보 검색』을 참조하십시오.

DB2 UDB CLI 어플리케이션의 리턴 코드

다음 표는 DB2 UDB CLI 함수에 가능한 모든 리턴 코드입니다. 25 페이지의 제 3 장 『DB2 UDB CLI 함수』의 함수 설명에 각 함수에서 리턴되는 모든 코드가 나옵니다.

표 2. DB2 UDB CLI 함수 리턴 코드

리턴 코드	설명
SQL_SUCCESS	함수가 성공적으로 완료되었고, SQLSTATE 정보는 추가로 제공되지 않습니다.
SQL_SUCCESS_WITH_INFO	경고나 기타 정보와 함께 함수 실행이 성공적으로 완료되었습니다. SQLSTATE 및 기타 오류 정보를 수신하려면 SQLError() 를 호출하십시오. SQLSTATE의 클래스는 01입니다.
SQL_NO_DATA_FOUND	함수가 성공적으로 리턴되었지만 관련 자료를 찾을 수 없습니다.
SQL_ERROR	함수가 실패하였습니다. SQLSTATE 및 기타 오류 정보를 수신하려면 SQLError()를 호출하십시오.
SQL_INVALID_HANDLE	입력 핸들(환경, 접속 또는 명령문 핸들)이 유효하지 않아 함수가 실패하였습니다.

DB2 UDB CLI SQLSTATE

데이터베이스 서버마다 사용하는 진단 메세지 코드가 다르므로, DB2 UDB CLI는 X/Open SQL CAE 스펙이 정의하는 표준 집합, *SQLSTATE*를 제공합니다. 그래야만 서로 다른 데이터베이스 서버 간에도 일관성 있게 메세지를 처리할 수 있습니다.

SQLSTATE는 ccsss형식의 5자(바이트)로 된 영숫자 스트링입니다. 여기에서 cc는 클래스, sss는 하위 클래스를 나타냅니다. SQLSTATE의 클래스는 다음과 같습니다.

- '01': 경고
- 'HY': CLI(명령행 인터페이스) 드라이버(DB2 UDB CLI 또는 ODBC)에 의해 생성됩니다.

서버가 오류 코드를 생성한 경우, SQLError() 함수는 *원래* 오류 코드로 리턴합니다. IBM 데이터베이스 서버에 연결된 경우에는 원래 오류 코드가 SQLCODE이고, 서버가 아닌 DB2 UDB CLI가 코드를 생성한 경우에는 원래 오류 코드가 -99999로 설정됩니다.

DB2 UDB CLI SQLSTATE에는 데이터베이스 서버가 리턴하는 추가적인 IBM 정의 SQLSTATE와 X/Open 스펙에 정의되지 않은 조건에 대한 DB2 UDB CLI 정의 SQLSTATE가 있습니다. 그렇기 때문에, 가장 많은 양의 진단 정보가 리턴되게 됩니다. ODBC를 사용하여 Windows에서 어플리케이션을 실행할 경우, ODBC 정의 SQLSTATE를 받을 수도 있습니다.

어플리케이션에서 SQLSTATE를 사용하려면 다음 지침을 따르십시오.

- 진단 정보가 제공되는지 판별하려면, SQLError()를 호출하기 전에 함수 리턴 코드를 항상 점검하십시오.
- 원래 오류 코드보다는 SQLSTATE를 사용하십시오.
- 어플리케이션의 이식성을 높이려면, X/Open 스펙이 정의하는 DB2 UDB CLI SQLSTATE의 부분 집합에 대해서만 의존하도록 하고, 추가적인 것은 단지 정보로서 리턴하십시오. 의존이란 어플리케이션이 특정 SQLSTATE에 근거해서 논리 흐름을 결정하는 것을 말합니다.
- 최대한의 진단 정보를 얻으려면, SQLSTATE와 함께 텍스트 메세지를 리턴하십시오. (적용가능한 경우, 텍스트 메세지에는 IBM 정의 SQLSTATE가 있습니다.) 오류를 리턴한 함수의 이름을 출력하는 것도 어플리케이션에 도움이 됩니다.

DB2 UDB CLI 함수에서의 자료 유형 및 자료 변환

표 3은 지원되는 모든 SQL 유형 및 해당 기호명입니다. 기호명은 인수의 자료 유형을 표시하기 위해, SQLBindParam(), SQLBindParameter(), SQLSetParam(), SQLBindCol(), SQLGetData()에 사용됩니다.

각 열은 아래에서 설명하고 있습니다.

SQL 유형

이 열에는 SQL문에서처럼 SQL 자료 유형이 있습니다. SQL 자료 유형은 DBMS에 따라 달라집니다.

SQL-Symbolic

이 열에는 정수 값으로 정의된(`sqlcli.h`에서) SQL 기호명이 있습니다. 이 값은 첫 번째 열의 SQL 자료 유형을 식별하기 위해, 여러 함수가 사용합니다.

표 3. SQL 자료 유형 및 디폴트 C 자료 유형

SQL 유형	**SQL 기호**
CHAR	SQL_CHAR, SQL_WCHAR[2]
VARCHAR	SQL_VARCHAR, SQL_WVARCHAR[2]
GRAPHIC	SQL_GRAPHIC
VARGRAPHIC	SQL_VARGRAPHIC
SMALLINT	SQL_SMALLINT
BIGINT	SQL_BIGINT
INTEGER	SQL_INTEGER
DECIMAL	SQL_DECIMAL
NUMERIC	SQL_NUMERIC
DOUBLE	SQL_DOUBLE
FLOAT	SQL_FLOAT
REAL	SQL_REAL
DATE[1]	SQL_CHAR
TIME[1]	SQL_CHAR
TIMESTAMP[1]	SQL_CHAR
BLOB	SQL_BLOB
CLOB	SQL_CLOB
DBCLOB	SQL_DBCLOB
주:	
1 DATE, TIME, TIMESTAMP 값은 문자 형식으로 리턴됩니다.	
2 SQL_WCHAR 및 SQL_WVARCHAR은 유니코드 자료를 표시하는 데 사용할 수 있습니다.	

자세한 정보는 다음을 참조하십시오.

- 21 페이지의『DB2 UDB CLI 함수에서의 기타 C 자료 유형』
- 21 페이지의『DB2 UDB CLI 함수에서의 자료 변환』

DB2 UDB CLI 함수에서의 기타 C 자료 유형

SQL 자료 유형에 맵핑되는 자료 유형은 물론, 포인터, 핸들과 같은 다른 함수 인수에 사용되는 C 기호 유형도 있습니다.

표 4. 일반 자료 유형 및 실제 C 자료 유형

기호 유형	실제 C 유형	일반 사용법
SQLPOINTER	void *	자료 및 매개변수 기억장치에 대한 포인터
SQLHENV	long int	환경 정보를 참조하는 핸들
SQLHDBC	long int	데이터베이스 연결 정보를 참조하는 핸들
SQLHSTMT	long int	명령문 정보를 참조하는 핸들
SQLRETURN	long int	DB2 UDB CLI 함수에서 생긴 리턴 코드

DB2 UDB CLI 함수에서의 자료 변환

앞에서도 설명했듯이, DB2 UDB CLI는 어플리케이션과 DBMS 사이의 자료 전송 및 요구되는 자료 전환을 관리합니다. 자료 전송이 실제로 일어나기 전, `SQLBindParam()`, `SQLBindParameter()`, `SQLSetParam()`, `SQLBindCol()` 또는 `SQLGetData()`를 호출하면 소스, 목표 또는 이 두 자료 유형이 모두 표시됩니다. 이 함수들은 관련된 자료 유형을 식별하기 위해 20 페이지의 표 3에 있는 기호 유형 이름을 사용합니다. 기호 자료 유형을 사용하는 함수의 예로는 `SQLFetch()` 110 페이지의 『예』 또는 `SQLGetCol()`136 페이지의 『예』를 참조하십시오.

표 5는 DB2 UDB CLI가 지원하는 변환으로서, 디폴트 변환만 표시되어 있습니다. 다른 변환은 실행되는 명령문의 SQL 구문에서 SQL 스칼라 함수나 SQL CAST 함수를 사용하면 됩니다.

위에서 언급된 함수들은 자료를 다른 유형으로 변환하는데 사용할 수 있습니다. 모든 자료 변환이 지원되는 것은 아니며, 모든 자료 변환이 의미있는 것도 아닙니다. 표 5에서는 DB2 UDB CLI에 제공되는 변환을 보여줍니다.

표 5의 첫 번째 열은 소스 자료 유형이며, 나머지 열은 목표 자료 유형입니다. X는 DB2 UDB CLI가 변환을 지원한다는 것을 나타냅니다.

표 5. 지원되는 자료 변환

소스 자료 유형	VARGRAPHIC	GRAPHIC	TIMESTAMP	TIME	DATE	VARCHAR	DOUBLE	REAL	FLOAT	SMALLINT	BIGINT	INTEGER	DECIMAL	NUMERIC	CHAR	BLOB	CLOB	DBCLOB
CHAR VARCHAR			X	X	X	X				X					X		X	

표 5. 지원되는 자료 변환 (계속)

소스 자료 유형	VARGRAPHIC	GRAPHIC	TIMESTAMP	TIME	DATE	VARCHAR	DOUBLE	REAL	FLOAT	SMALLINT	BIGINT	INTEGER	DECIMAL	NUMERIC	CHAR	BLOB	CLOB	DBCLOB
GRAPHIC VARGRAPHIC	X	X																X
BLOB																X		
CLOB						X									X		X	
DBCLOB	X	X																X
INTEGER SMALLINT BIGINT DECIMAL NUMERIC DOUBLE FLOAT						X	X	1	X	X	X	X	X	2	X			
DATE			X		X	X									X			
TIME			X	X		X									X			
TIMESTAMP			X	X	X	X									X			

주:

1. OS/2®용 DB2 UDB 또는 AIX/6000용 DB2 UDB는 REAL을 지원하지 **않습니다**.
2. **오직** iSeries용 DB2 UDB만 NUMERIC을 지원합니다(다른 DBMS는 DECIMAL로 처리).

값의 반올림 삭제 또는 자료 유형의 비호환 문제가 함수 호출에서 발생할 때마다 SQL_ERROR 또는 SQL_SUCCESS_WITH_INFO가 리턴됩니다. 자세한 정보는 SQLError()가 리턴하는 다른 정보와 SQLSTATE 값에 의해 표시됩니다.

DB2 UDB CLI 함수에서의 스트링 인수에 대한 작업

다음은 DB2 UDB CLI 함수에서 스트링 인수 작업에 대한 다양한 규약입니다.

- 23 페이지의『DB2 UDB CLI 함수에서 스트링 인수의 길이』
- 23 페이지의『DB2 UDB CLI 함수에서의 스트링 절단』
- 23 페이지의『DB2 UDB CLI 함수에서의 스트링 해석』

DB2 UDB CLI 함수에서 스트링 인수의 길이

입력 스트링 인수에는 연관된 길이 인수가 있습니다. 이 인수는 할당된 버퍼의 길이(널 바이트 종료자는 제외) 또는 특수 값 SQL_NTS를 DB2 UDB CLI에 표시합니다. SQL_NTS가 전달되면, DB2 UDB CLI는 널 종료 문자를 찾아 스트링의 길이를 결정합니다.

출력 스트링 인수에는 연관된 길이 인수가 두 개 있는데, 하나는 할당된 버퍼의 길이를 지정하는 것이고, 다른 하나는 DB2 UDB CLI가 리턴하는 스트링의 길이를 리턴하는 것입니다. 리턴된 길이 값은 그 길이가 버퍼에 맞는지에는 상관없이, 리턴에 사용 가능한 스트링의 총 길이입니다.

SQL 열 자료의 경우, 출력이 빈 스트링이면 길이 인수에 SQL_NULL_DATA가 리턴됩니다.

출력 길이 인수에 대한 널(null) 포인터로 함수가 호출되는 경우에는 DB2 UDB CLI는 길이를 리턴하지 않습니다. 이것은 버퍼가 가능한 모든 결과를 담을 수 있을 정도로 충분히 큰 경우 유용하게 사용될 수 있습니다. DB2 UDB CLI가 SQL_NULL_DATA 값을 리턴하여 열에 널 자료가 있다는 것을 나타내려고 하면, 함수 호출은 실패합니다.

DB2 UDB CLI가 리턴하는 모든 문자 스트링은 널 종료 문자(hex 00)로 끝나지만, 그래픽 자료 유형으로부터 리턴되는 스트링은 예외입니다. 모든 버퍼는 널 종료 문자마다, 하나의 간격을 더함으로써 최대 예상 문자 수에 대한 충분한 간격을 할당해야 합니다.

DB2 UDB CLI 함수에서의 스트링 절단

출력 스트링이 버퍼 길이보다 더 길면, DB2 UDB CLI는 스트링을 버퍼 크기보다 하나 작은 길이로 절단한 후, 널 종료자를 기록합니다. 절단이 발생하는 경우, 함수는 절단을 표시함으로써 SQLSTATE 및 SQL_SUCCESS_WITH_INFO를 리턴합니다. 그러면 어플리케이션은 버퍼 길이를 출력 길이와 비교하여 절단된 스트링을 알아냅니다.

예를 들어, SQLFetch()가 SQL_SUCCESS_WITH_INFO와 01004 값의 SQLSTATE를 리턴하는 경우, 열에 바인드된 버퍼 중 적어도 하나는 자료를 담기에 너무 작습니다. 열에 바인드된 각 버퍼에 대해, 어플리케이션은 버퍼 길이를 출력 길이와 비교하고 절단된 열을 알아냅니다.

DB2 UDB CLI 함수에서의 스트링 해석

DB2 UDB CLI는 대소문자를 구분하지 않으며, 열 이름이나 커서명과 같은 모든 스트링 입력 인수의 앞뒤 공백을 제거하지만, 다음의 경우에는 예외입니다.

- 데이터베이스 자료
- 분리 ID(큰 따옴표 안에 있음)
- 암호 인수

제 3 장 DB2 UDB CLI 함수

이 주제에서는 각각의 함수에 대해 설명합니다. 각각의 DB2 UDB CLI 함수에는 다음 정보가 있습니다.

- **목적**

  여기에서는 함수의 기능을 간단히 설명합니다. 설명되는 함수를 호출하기 전이나 후에 다른 함수를 호출해야 하는지도 알려줍니다.

- **구문**

  여기에는 OS/400® 환경에 대한 'C' 프로토타입도 있습니다.

- **인수**

  여기에서는 각 함수의 인수를 인수의 자료 유형, 설명, 그리고 인수가 입력 인수인지 아니면 출력 인수인지를 나타냅니다.

  각각의 DB2 UDB CLI 인수가 입력 인수이거나 출력 인수입니다. `SQLGetInfo()`는 제외하고, DB2 UDB CLI는 출력 인수로 지정된 인수만을 수정합니다.

  일부 함수에는 *연기된*, 또는 *바인드된* 인수라고 알려진 입력 또는 출력 인수가 있습니다. 이 인수는 어플리케이션에 의해 할당된 버퍼에 대한 포인터입니다. 이 인수는 SQL문의 매개변수 또는 결과 집합의 열과 연관됩니다. (또는 바인드됩니다.) 함수에 의해 지정된 자료 영역은 나중에 DB2 UDB CLI에 의해 액세스됩니다. 이 연기된 자료 영역은 DB2 UDB CLI가 이 자료 영역을 액세스할 때도 여전히 유효해야 합니다.

- **사용**

  여기에서는 함수를 사용하는 방법과 특별한 고려사항을 설명합니다. 가능한 오류 상태는 여기서 설명되지 않고 진단 설명 섹션에 나열됩니다.

- **리턴 코드**

  여기에서는 모든 가능한 함수의 리턴 코드를 나열합니다. SQL_ERROR 또는 SQL_SUCCESS_WITH_INFO가 리턴되면 `SQLError()`을 호출하여 오류 정보를 얻을 수 있습니다.

  리턴 코드에 대한 자세한 정보는 18 페이지의 『DB2 UDB CLI 어플리케이션에서의 진단』을 참조하십시오.

- **진단**

  여기에서는 DB2 UDB CLI에 의해 명시적으로 리턴되는 SQLSTATE(데이터베이스 관리 시스템(DBMS)에 의해 생성된 SQLSTATE도 리턴될 수 있음)를 나열하는 표가 있으며 오류의 원인을 표시합니다. 함수가 SQL_ERROR 또는 SQL_SUCCESS_WITH_INFO를 리턴한 후에 `SQLError()`를 호출하여 이 값을 얻게 됩니다.

  첫 번째 열의 『*****』는 SQLSTATE가 DB2 UDB CLI에 의해서만 리턴되고 다른 ODBC 드라이버에 의해서는 리턴되지 않음을 표시합니다.

  진단에 대한 자세한 정보는 18 페이지의 『DB2 UDB CLI 어플리케이션에서의 진단』을 참조하십시오.

- **제한사항**

  여기에서는 어플리케이션에 영향을 줄 수 있는 ODBC와 DB2 UDB CLI 사이의 차이점과 제한을 설명합니다.

- **예**

  여기에서는 함수 사용을 보여주는 코드가 나타납니다. 모든 코드의 완전한 소스는 297 페이지의 부록 C『DB2 UDB CLI 어플리케이션 코드 리스팅 예』에 들어 있습니다.

- **참조**

  여기에서는 관련된 DB2 UDB CLI 함수를 나열합니다.

함수들은 다음과 같습니다.

SQLAllocConnect - 연결 핸들 할당

목적

SQLAllocConnect()는 입력 환경 핸들에 의해 식별된 환경 내에서 자원과 연결 핸들을 할당합니다. fInfoType을 SQL_ACTIVE_CONNECTIONS으로 설정하여 SQLGetInfo()를 호출하여 한번에 할당될 수 있는 연결 수를 조회합니다.

이 함수를 호출하기 전에 SQLAllocEnv()를 호출해야 합니다.

구문

```
SQLRETURN SQLAllocConnect (SQLHENV    henv,
                           SQLHDBC    *phdbc);
```

함수 인수

표 6. SQLAllocConnect 인수

자료 유형	인수	사용	설명
SQLHENV	*henv*	입력	환경 핸들
SQLHDBC *	*phdbc*	출력	연결 핸들을 가리키는 포인터

사용법

일반 상태 정보, 트랜잭션 상태, 오류 정보를 포함하여, 연결과 관련되는 모든 정보를 참조하기 위해 출력 연결 핸들이 DB2 UDB CLI에 의해 사용됩니다.

연결 핸들을 가리키는 포인터(*phdbc*)가 SQLAllocConnect()에 의해 할당된 유효한 연결 핸들을 가리키면, 이 호출의 결과에 의해 원래의 값이 바뀝니다. 이것은 어플리케이션 프로그래밍 오류로, DB2 UDB CLI에 의해 감지되지 않습니다.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

SQL_ERROR가 리턴되면 *phdbc* 인수가 SQL_NULL_HDBC로 설정됩니다. 어플리케이션은 환경 핸들(*henv*)과 함께 *hdbc*와 *hstmt* 인수를 각각 SQL_NULL_HDBC와 SQL_NULL_HSTMT로 설정하여 SQLError()를 호출해야 합니다.

진단

표 7. SQLAllocConnect SQLSTATEs

CLI SQLSTATE	설명	설명
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*phdbc*가 널(null) 포인터입니다.

예

다음 예는 연결과 환경에 대한 진단 정보를 얻는 방법을 보여줍니다. SQLError()를 사용하는 추가 예에 대해서는 typical.c의 전체 리스트에 대한 300 페이지의 『예: 대화식 SQL 및 그에 해당되는 DB2 UDB CLI 함수 호출』을 참조하십시오.

코드 예와 관련된 정보는 viii 페이지의 『코드 면책사항 관련 정보』를 참조하십시오.

```
/*********************************************************************
** initialize
**  - allocate environment handle
**  - allocate connection handle
**  - prompt for server, user id, & password
**  - connect to server
*********************************************************************/

int initialize(SQLHENV *henv,
               SQLHDBC *hdbc)
{
SQLCHAR     server[SQL_MAX_DSN_LENGTH],
            uid[30],
            pwd[30];
SQLRETURN   rc;

    SQLAllocEnv (henv);         /* allocate an environment handle   */
            if (rc != SQL_SUCCESS )
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);

    SQLAllocConnect (*henv, hdbc);  /* allocate a connection handle     */
            if (rc != SQL_SUCCESS )
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);

    printf("Enter Server Name:\n");
    gets(server);
    printf("Enter User Name:\n");
    gets(uid);
    printf("Enter Password Name:\n");
    gets(pwd);

    if (uid[0] == '\0')
    {   rc = SQLConnect (*hdbc, server, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS);
            if (rc != SQL_SUCCESS )
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
    }
    else
    {   rc = SQLConnect (*hdbc, server, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
            if (rc != SQL_SUCCESS )
```

```
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
    }
}/* end initialize */


/*****************************************************************/
int check_error (SQLHENV    henv,
                       SQLHDBC        hdbc,
                 SQLHSTMT   hstmt,
                 SQLRETURN  frc)
{
SQLRETURN   rc;

    print_error(henv, hdbc, hstmt);

    switch (frc){
    case SQL_SUCCESS : break;
    case SQL_ERROR :
    case SQL_INVALID_HANDLE:
        printf("\n ** FATAL ERROR, Attempting to rollback transaction **\n");
    rc = SQLTransact(henv, hdbc, SQL_ROLLBACK);
        if (rc != SQL_SUCCESS)
            printf("Rollback Failed, Exiting application\n");
    else
            printf("Rollback Successful, Exiting application\n");
    terminate(henv, hdbc);
        exit(frc);
        break;
    case SQL_SUCCESS_WITH_INFO :
        printf("\n ** Warning Message, application continuing\n");
        break;
    case SQL_NO_DATA_FOUND :
        printf("\n ** No Data Found ** \n");
        break;
    default :
        printf("\n ** Invalid Return Code ** \n");
        printf(" ** Attempting to rollback transaction **\n");
        SQLTransact(henv, hdbc, SQL_ROLLBACK);
    terminate(henv, hdbc);
        exit(frc);
        break;
    }
    return(SQL_SUCCESS);

}
```

참조

- 32 페이지의『SQLAllocEnv - 환경 핸들 할당』
- 76 페이지의『SQLConnect - 자료 소스 연결』
- 91 페이지의『SQLDisconnect - 자료 소스 단절』
- 123 페이지의『SQLFreeConnect - 연결 핸들 해제』
- 138 페이지의『SQLGetConnectAttr - 연결 속성 값 얻기』
- 241 페이지의『SQLSetConnectOption - 연결 옵션 설정』

SQLAllocEnv - 환경 핸들 할당

목적

SQLAllocEnv()는 환경 핸들과 연관된 자원을 할당합니다.

어플리케이션은 SQLAllocConnect() 또는 다른 DB2 UDB CLI 함수를 호출하기 전에 이 함수를 먼저 호출해야 합니다이후에 입력으로 환경 핸들을 필요로 하는 모든 함수에 *henv* 값이 전달됩니다.

구문

```
SQLRETURN SQLAllocEnv (SQLHENV    *phenv);
```

함수 인수

표 8. SQLAllocEnv 인수

자료 유형	인수	사용	설명
SQLHENV *	*phenv*	출력	환경 핸들을 가리키는 포인터

사용법

어플리케이션마다 한 번에 단 하나의 환경만 활동 상태일 수 있습니다. 나중에 SQLAllocEnv()를 호출하면 기존 환경 핸들을 리턴합니다.

디폴트로 SQLFreeEnv()에 대하여 최초로 성공한 호출은 핸들과 연관된 자원을 해제합니다. 이것은 SQLAllocEnv()가 몇 번이나 성공적으로 호출되었는지에 관계없이 발생합니다.
SQL_ATTR_ENVHNDL_COUNTER 환경 속성이 SQL_TRUE로 설정된 경우에는 핸들과 연관된 자원이 해제되기 전에 SQLFreeEnv()가 각각의 성공한 SQLAllocEnv() 호출에 대해 호출되어야 합니다.

모든 DB2 UDB CLI 자원을 계속 활동 상태로 유지하려면, SQLAllocEnv()를 호출한 프로그램이 종료되거나 스택을 벗어나서는 안됩니다. 그렇지 않으면, 어플리케이션은 열린 커서, 명령문 핸들 그리고 할당한 다른 자원을 잃을 수 있습니다.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR

SQL_ERROR가 리턴되고 *phenv*가 SQL_NULL_HENV이면, 추가 진단 정보와 연관된 핸들이 없으므로 SQLError()는 호출될 수 없습니다.

리턴 코드가 SQL_ERROR이고 환경 핸들을 가리키는 포인터가 SQL_NULL_HENV가 아니면, 핸들은 **제한된 핸들**입니다. 이는 핸들이 오류 정보를 더 얻기 위해 SQLError() 호출에만 사용되거나 SQLFreeEnv() 호출에만 사용될 수 있음을 의미합니다.

진단

표 9. SQLAllocEnv SQLSTATEs

SQLSTATE	설명	설명
58004	시스템 오류	복구할 수 없는 시스템 오류

예

코드 예와 관련된 정보는 viii 페이지의 『코드 면책사항 관련 정보』를 참조하십시오.

```
/*************************************************
** file = basiccon.c
**    - demonstrate basic connection to two datasources.
**    - error handling  ignored for simplicity
**
** Functions used:
**
**    SQLAllocConnect  SQLDisconnect
**    SQLAllocEnv      SQLFreeConnect
**    SQLConnect       SQLFreeEnv
**
**
*************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "sqlcli.h"

int
connect(SQLHENV henv,
        SQLHDBC * hdbc);

#define MAX_DSN_LENGTH    18
#define MAX_UID_LENGTH    10
#define MAX_PWD_LENGTH    10
#define MAX_CONNECTIONS   5

int
main()
{
    SQLHENV     henv;
    SQLHDBC         hdbc[MAX_CONNECTIONS];

    /* allocate an environment handle   */
    SQLAllocEnv(&henv);

    /* Connect to first data source */
    connect(henv, &hdbc[0];);

    /* Connect to second data source */
    connect(henv, &hdbc[1];);

    /*********   Start Processing Step  ************************/
    /* allocate statement handle, execute statement, etc.       */
    /*********   End Processing Step  *************************/
```

```
    printf("\nDisconnecting .....\n");
    SQLFreeConnect(hdbc[0]);    /* free first connection handle  */
    SQLFreeConnect(hdbc[1]);    /* free second connection handle */
    SQLFreeEnv(henv);           /* free environment handle       */

    return (SQL_SUCCESS);
}

/*********************************************************************
**   connect - Prompt for connect options and connect              **
*********************************************************************/

int
connect(SQLHENV henv,
        SQLHDBC * hdbc)
{
    SQLRETURN   rc;
    SQLCHAR         server[MAX_DSN_LENGTH + 1], uid[MAX_UID_LENGTH + 1],
pwd[MAX_PWD_LENGTH
+ 1];
    SQLCHAR         buffer[255];
    SQLSMALLINT     outlen;

    printf("Enter Server Name:\n");
    gets((char *) server);
    printf("Enter User Name:\n");
    gets((char *) uid);
    printf("Enter Password Name:\n");
    gets((char *) pwd);

    SQLAllocConnect(henv, hdbc);/* allocate a connection handle     */

    rc = SQLConnect(*hdbc, server, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
    if (rc != SQL_SUCCESS) {
        printf("Error while connecting to database\n");
        return (SQL_ERROR);
    } else {
        printf("Successful Connect\n");
    return (SQL_SUCCESS);
    }
}
```

참조

- 29 페이지의『SQLAllocConnect - 연결 핸들 할당』
- 125 페이지의『SQLFreeEnv - 환경 핸들 해제』
- 37 페이지의『SQLAllocStmt - 명령문 핸들 할당』

SQLAllocHandle - 핸들 할당

목적

SQLAllocHandle()은 모든 유형의 핸들을 할당합니다.

구문

```
SQLRETURN SQLAllocHandle (SQLSMALLINT htype,
                          SQLINTEGER ihandle,
                          SQLINTEGER *handle);
```

함수 인수

표 10. SQLAllocHandle 인수

자료 유형	인수	사용	설명
SQLSMALLINT	*htype*	입력	할당할 핸들의 유형. SQL_HANDLE_ENV, SQL_HANDLE_DBC, SQL_HANDLE_DESC 또는 SQL_HANDLE_STMT여야 합니다.
SQLINTEGER	*ihandle*	입력	새로운 핸들이 할당되는 문맥을 설명하는 핸들. 그러나 *htype*이 SQL_HANDLE_ENV이면 이 값은 SQL_NULL_HANDLE입니다.
SQLINTEGER *	*handle*	출력	핸들을 가리키는 포인터

사용법

이 함수가 SQLAllocEnv(), SQLAllocConnect() 및 SQLAllocStmt() 함수를 결합합니다.

*htype*이 SQL_HANDLE_ENV이면, *ihandle*은 SQL_NULL_HANDLE이어야 합니다. *htype*이 SQL_HANDLE_DBC이면, *ihandle*은 유효한 환경 변수여야 합니다. *htype*이 SQL_HANDLE_DESC이거나 SQL_HANDLE_STMT이면, *ihandle*은 유효한 연결 핸들이어야 합니다.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

인수 핸들이 널(null) 포인터인 경우에는 SQL_ERROR이 리턴됩니다.

표 11. SQLAllocHandle SQLSTATEs

SQLSTATE	설명	설명
58004	시스템 오류	복구할 수 없는 시스템 오류
HY014	핸들이 너무 많음	최대 핸들 수가 할당되었습니다.

SQLAllocHandle

참조

- 29 페이지의『SQLAllocConnect - 연결 핸들 할당』
- 32 페이지의『SQLAllocEnv - 환경 핸들 할당』
- 37 페이지의『SQLAllocStmt - 명령문 핸들 할당』

SQLAllocStmt - 명령문 핸들 할당

목적

SQLAllocStmt()는 새 명령문 핸들을 할당하여 이를 연결 핸들에 의해 지정된 연결과 연관시킵니다. 한 번에 할당될 수 있는 명령문 핸들의 수에 대한 제한이 정의되지 않았습니다.

이 함수를 호출하기 전에 SQLConnect()를 호출해야 합니다.

SQLBindParam(), SQLPrepare(), SQLExecute(), SQLExecDirect() 또는 명령문 핸들을 입력 인수로 갖는 다른 함수 이전에 이 함수를 호출해야 합니다.

구문

```
SQLRETURN SQLAllocStmt (SQLHDBC    hdbc,
                        SQLHSTMT   *phstmt);
```

함수 인수

표 12. SQLAllocStmt 인수

자료 유형	인수	사용	설명
SQLHDBC	*hdbc*	입력	연결 핸들
SQLHSTMT *	*phstmt*	출력	명령문 핸들을 가리키는 포인터

사용법

DB2 UDB CLI는 각각의 명령문 핸들을 사용하여 모든 설명자, 결과 값, 커서 정보, 상태 정보를 처리된 SQL 문과 연관시킵니다. 각각의 SQL문에 명령문 핸들이 있어야 하지만 다른 명령문에서 핸들을 다시 사용할 수 있습니다.

이 함수에 대한 호출에서는 *hdbc*가 사용 중인 데이터베이스 연결을 참조해야 합니다.

위치지정된 갱신이나 삭제를 실행하기 위해서는 어플리케이션이 SELECT문과 UPDATE문 또는 DELETE문에 대해 다른 명령문 핸들을 사용해야 합니다.

명령문 핸들을 가리키는 입력 포인터(*phstmt*)가 이전의 SQLAllocStmt() 호출에 의해 할당된 유효한 명령문 핸들을 가리키면 원래의 값이 이 호출의 결과에 따라 바뀝니다. 이것은 어플리케이션 프로그래밍 오류로 DB2 UDB CLI에 의해 감지되지 않습니다.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

SQLAllocStmt

SQL_ERROR가 리턴되면 *phstmt* 인수가 SQL_NULL_HSTMT로 설정됩니다. 어플리케이션은 *hdbc*와 *hstmt* 인수를 SQL_NULL_HSTMT로 설정하여 SQLError()를 호출해야 합니다.

진단

표 13. SQLAllocStmt SQLSTATEs

SQLSTATE	설명	설명
08003	연결이 열려 있지 않음	*hdbc* 인수에 의해 지정된 연결이 열리지 않았습니다. *hstmt*를 할당하기 위해 이 연결이 드라이버에 대해 성공적으로 설정되어야 합니다(연결도 열려야 함).
40003 *	명령문 완료 상태를 알 수 없음	함수가 처리를 완료하기 전에 CLI와 자료 소스와의 통신 링크가 끊어졌습니다.
58004	시스템 오류	복구할 수 없는 시스템 오류
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*phstmt*가 널(null) 포인터입니다.
HY013 *	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

예

SQLFetch()110 페이지의 『예』를 참조하십시오.

참조

- 76 페이지의 『SQLConnect - 자료 소스 연결』
- 129 페이지의 『SQLFreeStmt - 명령문 핸들을 해제(또는 재설정)』
- 185 페이지의 『SQLGetStmtOption - 명령문 옵션의 현재 설정 리턴』
- 258 페이지의 『SQLSetStmtOption - 명령문 옵션 설정』

SQLBindCol - 어플리케이션 변수에 열 바인드

목적

SQLBindCol()은 모든 자료 유형에 대해 결과 집합의 열을 어플리케이션 변수(기억장치 버퍼)에 연관(바인드)시킵니다. SQLFetch()가 호출되면 자료가 데이터베이스 관리 시스템(DBMS)에서 어플리케이션으로 전송됩니다.

이 함수는 필요한 자료 변환을 지정하기 위해서도 사용됩니다. 이 함수는 어플리케이션이 검색할 필요가 있는 결과 집합의 각 열에 대해 한 번 호출됩니다.

이 함수 전에 일반적으로 SQLPrepare() 또는 SQLExecDirect()가 호출됩니다. SQLDescribeCol() 또는 SQLColAttributes()를 호출할 수도 있습니다.

이 호출에 의해 지정된 기억장치 버퍼에 자료를 전송하기 위해, SQLFetch() 전에 SQLBindCol()을 호출해야 합니다.

구문

```
SQLRETURN SQLBindCol (SQLHSTMT       hstmt,
                      SQLSMALLINT    icol,
                      SQLSMALLINT    fCType,
                      SQLPOINTER     rgbValue,
                      SQLINTEGER     cbValueMax,
                      SQLINTEGER     *pcbValue);
```

함수 인수

표 14. SQLBindCol 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLSMALLINT	*icol*	입력	열을 식별하는 번호. 열은 왼쪽에서 오른쪽으로 1부터 순차적으로 번호가 지정됩니다.

SQLBindCol

표 14. SQLBindCol 인수 (계속)

자료 유형	인수	사용	설명
SQLSMALLINT	*fCType*	입력	결과 집합의 열 번호 *icol*에 대한 어플리케이션 자료 유형. 다음 유형들이 지원됩니다. • SQL_CHAR • SQL_VARCHAR • SQL_NUMERIC • SQL_DECIMAL • SQL_INTEGER • SQL_SMALLINT • SQL_BIGINT • SQL_FLOAT • SQL_REAL • SQL_DOUBLE • SQL_GRAPHIC • SQL_VARGRAPHIC • SQL_DATETIME • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_BLOB • SQL_CLOB • SQL_DBCLOB • SQL_BLOB_LOCATOR • SQL_CLOB_LOCATOR • SQL_DBCLOB_LOCATOR SQL_DEFAULT를 지정하면 자료가 기본 자료 유형으로 전송됩니다. 자세한 정보는 20 페이지의 표 3을 참조하십시오.
SQLPOINTER	*rgbValue*	출력(지연됨)	페치가 발생할 때 DB2 UDB CLI가 열 자료를 저장할 버퍼를 가리키는 포인터 *rgbValue*가 널이면 열이 바인드되지 않습니다.

표 14. SQLBindCol 인수 (계속)

자료 유형	인수	사용	설명
SQLINTEGER	*cbValueMax*	입력	열 자료를 저장하기 위해 사용할 수 있는 *rgbValue* 버퍼 크기(바이트) *fCType*이 SQL_CHAR 또는 SQL_DEFAULT이면, *cbValueMax*가 0보다 커야 하며 그렇지 않으면 오류가 리턴됩니다. *fcType*이 SQL_DECIMAL 또는 SQL_NUMERIC이면, *cbValueMax*는 실제로 정밀도와 소수 자릿수(scale)여야 합니다. 두 값을 지정하는 방법은 *(정밀도 * 256) + 소수 자릿수*를 사용하는 것입니다. 이 값은 또한 SQLColAttributes()를 사용할 때 이 자료 유형의 LENGTH로 리턴된 값입니다. *fcType*이 2바이트 문자 자료 형식을 지정한 경우, *cbValueMax*는 바이트 수가 아닌 2바이트 문자의 수가 되어야 합니다.
SQLINTEGER *	*pcbValue*	출력(지연됨)	*rgbValue* 버퍼에서 DB2 UDB CLI가 리턴하기 위해 사용할 수 있는 바이트 수를 나타내는 값을 가리키는 포인터. 열의 자료 값이 널이면 SQLFetch()는 이 인수에서 SQL_NULL_DATA를 리턴합니다. 열의 자료 값이 널 종료 스트링으로 리턴되면 이 인수에서 SQL_NTS가 리턴됩니다.

주:

이 함수에 대해 *rgbValue*와 *pcbValue*는 연기된 출력으로, SQLFetch()가 호출될 때까지 이 포인터가 가리키는 기억장치 위치가 갱신되지 않음을 의미합니다. 이 포인터에 의해 참조된 위치는 SQLFetch()가 호출될 때까지 유효한 상태로 남아 있어야 합니다.

사용법

어플리케이션은 검색하려는 결과 집합의 각 열에 대해 SQLBindCol()을 한 번 호출합니다. SQLFetch()가 호출되면 이 *바인드된* 각 열의 자료가 할당된 위치(포인터 *rgbValue*와 *pcbValue*에 의해 주어짐)에 놓입니다.

어플리케이션은 먼저 SQLDescribeCol() 또는 SQLColAttributes()를 호출하여 열의 속성(자료 유형과 길이 같은)을 조회할 수 있습니다. 기억장치 위치의 정확한 자료 유형을 지정하거나 다른 자료 유형으로의 자료 변환을 표시하기 위해 이 정보를 사용할 수 있습니다. 자세한 정보는 20 페이지의 『DB2 UDB CLI 함수에서의 자료 유형 및 자료 변환』을 참조하십시오.

나중에 페치할 때 어플리케이션은 SQLBindCol()을 호출하여 바인드되지 않은 열을 바인드하거나 이 열의 바인딩을 변경할 수 있습니다. 새로운 바인딩은 페치된 자료에는 적용되지 않으며 다음 번 SQLFetch()를 호출

할 때 사용됩니다. 단일 열을 바인드하지 않으려면, *rgbValue*를 널(null)로 설정해서 SQLBindCol()을 호출하십시오.모든 열을 바인드하지 않으려면 어플리케이션은 *fOption* 입력을 SQL_UNBIND로 설정하여 SQLFreeStmt()를 호출해야 합니다.

왼쪽에서 오른쪽으로 1부터 시작되는 번호에 의해 열이 식별됩니다. *fdescType* 인수를 SQL_DESC_COUNT로 설정하여 SQLNumResultCols() 또는 SQLColAttributes()를 호출하여 결과 집합의 열 번호를 판별할 수 있습니다.

어플리케이션은 모든 열을 바인드하지 않도록 선택하거나 모든 열을 바인드하지 않도록 선택할 수도 있습니다. 바인드되지 않은 열(그리고 바인드되지 않은 열만)의 자료는 SQLFetch()를 호출한 후에 SQLGetData()를 사용하여 검색할 수 있습니다. SQLBindCol()은 SQLGetData()보다 더 효율적이며, 가능할 때마다 사용되어야 합니다.

어플리케이션은 검색할 자료에 대해 충분한 기억장치를 할당해야 합니다. 버퍼에 가변 길이 자료가 있어야 한다면, 어플리케이션은 바인드된 열이 요구하는 최대 길이의 기억장치를 할당해야 하며, 그렇지 않으면 자료가 절단될 수 있습니다.

스트링이 절단되면 SQL_SUCCESS_WITH_INFO가 리턴되고 *pcbValue*가 어플리케이션에 리턴하기 위해 사용할 수 있는 *rgbValue*의 실제 *크기*로 설정됩니다.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 15. SQLBindCol SQLSTATEs

SQLSTATE	설명	설명
40003 *****	명령문 완료 상태를 알 수 없음	함수가 처리를 완료하기 전에 CLI와 자료 소스와의 통신 링크가 끊어졌습니다.
58004	시스템 오류	복구할 수 없는 시스템 오류
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY002	유효하지 않은 열 번호	*icol* 인수에 대해 지정된 값이 0입니다. *icol* 인수에 대해 지정된 값이 자료 소스에 의해 지원되는 열의 최대 수를 초과했습니다.
HY003	프로그램 유형이 범위 밖의 값임	*fCType*이 유효한 자료 유형이 아닙니다.
HY009	유효하지 않은 인수 값	*rgbValue*가 널(null) 포인터입니다. *cbValueMax* 인수에 대해 지정된 값이 1 미만이며 *fCType* 인수가 SQL_CHAR 또는 SQL_DEFAULT입니다.

표 15. SQLBindCol SQLSTATEs (계속)

SQLSTATE	설명	설명
HY013 *	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.
HY014	핸들이 너무 많음	최대 핸들 수가 할당되었으며, 이 함수를 사용하려면 추가 설명자 핸들이 필요합니다.
HYC00	드라이버가 지원되지 않음	드라이버가 인식은 하지만 *fCType* 인수에 지정된 자료 유형을 지원하지 않습니다(**HY**003도 참조).

예

SQLFetch()110 페이지의『예』를 참조하십시오.

참조

- 102 페이지의『SQLExecDirect - 명령문 직접 실행』
- 104 페이지의『SQLExecute - 명령문 실행』
- 109 페이지의『SQLFetch - 다음 행 페치』
- 212 페이지의『SQLPrepare - 명령문 준비』

SQLBindFileToCol - LOB 열에 LOB 파일 참조 바인드

목적

SQLBindFileToCol()은 결과 집합의 LOB 열을 파일 참조 또는 파일 참조의 배열에 연관(바인드)시킵니다. 이렇게 하면 명령문 핸들에 대해 각 행이 페치될 때 해당 열의 자료를 파일로 직접 전송할 수 있습니다.

LOB 파일 참조 인수(파일명, 파일명 길이, 파일 참조 옵션)는 (클라이언트의) 어플리케이션 환경 내의 파일을 참조합니다. 각 행을 페치하기 전에 어플리케이션은 이 변수에 파일명, 파일명 길이, 파일 옵션(신규/겹쳐 쓰기/추가)이 있는지 확인해야 합니다. 이 값은 각 페치 사이에서 변경될 수 있습니다.

구문

```
SQLRETURN   SQLBindFileToCol (SQLHSTMT          StatementHandle,
                              SQLSMALLINT       ColumnNumber,
                              SQLCHAR           *FileName,
                              SQLSMALLINT       *FileNameLength,
                              SQLINTEGER        *FileOptions,
                              SQLSMALLINT       MaxFileNameLength,
                              SQLINTEGER        *StringLength,
                              SQLINTEGER        *IndicatorValue);
```

함수 인수

표 16. SQLBindFileToCol 인수

자료 유형	인수	사용	설명
SQLHSTMT	*StatementHandle*	입력	명령문 핸들.
SQLSMALLINT	*ColumnNumber*	입력	열을 식별하는 번호. 열은 왼쪽에서 오른쪽으로 1부터 순차적으로 번호가 지정됩니다.
SQLCHAR *	*FileName*	입력(지연됨)	*StatementHandle*을 사용하여 다음에 페치할 때의 파일명 배열이나 파일명이 있는 위치를 가리키는 포인터. 이 값은 파일의 전체 경로명이거나 상대 파일명입니다. 상대 파일명일 경우 실행 중인 어플리케이션의 현재 경로에 추가됩니다. 이 포인터는 널(null)이 될 수 없습니다.
SQLSMALLINT *	*FileNameLength*	입력(지연됨)	*StatementHandle*을 사용하여 다음에 페치할 때의 파일명 길이가 있는 위치를 가리키는 포인터. 이 포인터가 널(null)이면 SQL_NTS의 길이가 가정됩니다. 파일명 길이의 최대 값은 255입니다.

표 16. SQLBindFileToCol 인수 (계속)

자료 유형	인수	사용	설명
SQLINTEGER *	*FileOptions*	입력(지연됨)	*StatementHandle*을 사용하여 다음에 페치할 때의 파일을 쓰기 위해 사용할 파일 옵션이 있는 위치를 가리키는 포인터. 다음 *FileOptions*이 지원됩니다. **SQL_FILE_CREATE** 새 파일을 작성합니다. 이 이름의 파일이 이미 존재하면 SQL_ERROR가 리턴됩니다. **SQL_FILE_OVERWRITE** 파일이 이미 존재하면 겹쳐씁니다. 그렇지 않으면 새로운 파일을 작성합니다. **SQL_FILE_APPEND** 파일이 이미 존재하면 이 파일에 자료를 추가합니다. 그렇지 않으면 새로운 파일을 작성합니다. 파일당 하나의 옵션만 선택할 수 있으며 디폴트 옵션은 없습니다.
SQLSMALLINT	*MaxFileNameLength*	입력	*FileName* 버퍼 길이를 지정합니다.
SQLINTEGER *	*StringLength*	출력(지연됨)	리턴된 LOB 자료의 바이트 단위 길이가 있는 위치를 가리키는 포인터. 이 포인터가 널(null)이면 리턴되는 값이 없습니다.
SQLINTEGER *	*IndicatorValue*	출력(지연됨)	인디케이터 값이 있는 위치를 가리키는 포인터.

사용법

행이 페치될 때 직접 파일에 전송되어야 하는 각 열에 대해 어플리케이션은 SQLBindFileToCol()을 한 번 호출합니다. 널 종료자를 추가하지 않고, 자료 변환 없이 LOB 자료가 직접 파일로 기록됩니다.

페치하기 전마다 *FileName*, *FileNameLength*, *FileOptions*을 설정해야 합니다. SQLFetch() 또는 SQLFetchScroll()이 호출되면 LOB 파일 참조에 바인드된 열에 대한 자료가 이 파일 참조가 가리키는 파일(들)에 기록됩니다. SQLBindFileToCol()의 연기된 입력 인수 값과 연관된 오류가 페치시에 보고됩니다. LOB 파일 참조와 연기된 *StringLength*와 *IndicatorValue* 출력 인수가 페치 작업과 작업 사이에 갱신됩니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

오류 조건

표 17. SQLBindFileToCol SQLSTATEs

SQLSTATE	설명	설명
58004	예기치 않은 시스템 실패	복구할 수 없는 시스템 오류

표 17. SQLBindFileToCol SQLSTATEs (계속)

SQLSTATE	설명	설명
HY002	유효하지 않은 열 번호	*icol* 인수에 대해 지정된 값이 1 미만입니다. *icol* 인수에 대해 지정된 값이 자료 소스에 의해 지원되는 열의 최대 수를 초과했습니다.
HY009	유효하지 않은 인수 값	*FileName*, *StringLength* 또는 *FileOptions*는 널(null) 포인터입니다.
HY010	함수 순서 오류	함수가 실행시 자료(SQLParamData(), SQLPutData()) 조작중에 호출됩니다. 함수가 BEGIN COMPOUND와 END COMPOUND SQL 작업 내에서 호출됩니다.
HY090	유효하지 않은 스트링 또는 버퍼 길이	*MaxFileNameLength* 인수에 대해 지정된 값이 0 미만입니다.
HYC00	드라이버가 지원되지 않음	어플리케이션이 큰 오브젝트를 지원하지 않는 자료 소스에 현재 연결되었습니다.

제한사항

큰 오브젝트 자료 유형을 지원하지 않는 DB2 서버에 연결될 때는 이 함수를 사용할 수 없습니다.

참조

- 39 페이지의『SQLBindCol - 어플리케이션 변수에 열 바인드』
- 109 페이지의『SQLFetch - 다음 행 페치』
- 47 페이지의『SQLBindFileToParam - LOB 매개변수에 LOB 파일 참조 바인드』

SQLBindFileToParam - LOB 매개변수에 LOB 파일 참조 바인드

목적

SQLBindFileToParam()이 SQL문의 매개변수 마커를 파일 참조나 파일 참조 배열에 연관(바인드)시킵니다. 이렇게 하면 이 명령문이 연이어 실행될 때 해당 파일의 자료가 직접 LOB 열로 전송됩니다.

LOB 파일 참조 인수(파일명, 파일명 길이, 파일 참조 옵션)는 (클라이언트의) 어플리케이션 환경 내의 파일을 참조합니다. SQLExecute() 또는 SQLExecDirect()를 호출하기 전에 어플리케이션은 이 정보를 연기된 입력 버퍼에서 사용할 수 있는지를 확인해야 합니다. 이 값은 SQLExecute() 호출 사이에서 변경될 수 있습니다.

구문

```
SQLRETURN SQLBindFileToParam (SQLHSTMT          StatementHandle,
                              SQLSMALLINT       ParameterNumber,
                              SQLSMALLINT       DataType,
                              SQLCHAR           *FileName,
                              SQLSMALLINT       *FileNameLength,
                              SQLINTEGER        *FileOptions,
                              SQLSMALLINT       MaxFileNameLength,
                              SQLINTEGER        *IndicatorValue);
```

함수 인수

표 18. SQLBindFileToParam 인수

자료 유형	인수	사용	설명
SQLHSTMT	*StatementHandle*	입력	명령문 핸들.
SQLSMALLINT	*ParameterNumber*	입력	매개변수 마커 번호. 매개변수는 왼쪽에서 오른쪽으로 1부터 순차적으로 번호가 지정됩니다.
SQLSMALLINT	*DataType*	입력	열의 SQL 자료 유형. 자료 유형은 다음 중 하나여야 합니다. • SQL_BLOB • SQL_CLOB • SQL_DBCLOB
SQLCHAR *	*FileName*	입력(지연됨)	명령문(*StatementHandle*)이 실행될 때 파일명이나 파일명 배열이 있는 위치를 가리킵니다. 이 값은 파일의 전체 경로명이거나 상대 파일명입니다. 상대 파일명인 경우 클라이언트 프로세스의 현재 경로에 추가됩니다. 이 인수는 널(null)이 될 수 없습니다.
SQLSMALLINT *	*FileNameLength*	입력(지연됨)	다음에 SQLExecute() 또는 SQLExecDirect()가 *StatementHandle*을 사용할 때의 파일명 길이(또는 길이 배열)가 있는 위치를 가리키는 포인터 이 포인터가 널(null)이면 SQL_NTS의 길이가 가정됩니다. 파일명 길이의 최대 값은 255입니다.

표 18. SQLBindFileToParam 인수 (계속)

자료 유형	인수	사용	설명
SQLINTEGER *	*FileOptions*	입력(지연됨)	파일을 읽을 때 사용할 파일 옵션(또는 파일 옵션 배열)이 있는 위치를 가리키는 포인터. 명령문(*StatementHandle*)이 실행될 때 이 위치에 액세스합니다. 하나의 옵션만 지원됩니다. (그 옵션만 지정해야 합니다.) **SQL_FILE_READ** 열고, 읽고, 닫을 수 있는 정규 파일. (파일을 열 때 길이가 계산됩니다.) 이 포인터는 널(null)이 될 수 없습니다.
SQLSMALLINT	*MaxFileNameLength*	입력	*FileName* 버퍼 길이를 지정합니다. 어플리케이션이 SQLParamOptions()을 호출하여 각 매개변수에 대해 복수 값을 지정하면, 이 값은 *FileName* 배열의 각 요소 길이입니다.
SQLINTEGER *	*IndicatorValue*	출력(지연됨)	인디케이터 값(또는 값 배열)이 있는 위치를 가리키는 포인터로, 매개변수의 자료 값이 널이어야 하면 SQL_NULL_DATA로 설정됩니다. 자료 값이 널이 아니면 이 값은 0으로 설정되어야 합니다. (또는 포인터가 널로 설정될 수 있습니다.)

사용법

명령문이 실행될 때 파일에서 직접 얻어야 하는 값을 가진 각 매개변수 마커에 대해 어플리케이션은 한 번씩 SQLBindFileToParam()을 호출합니다. 명령문을 실행하기 전에 *FileName*, *FileNameLength*, *FileOptions* 값을 설정해야 합니다. 명령문이 실행될 때, SQLBindFIleToParam()을 사용하여 바인드되는 매개변수에 대한 자료가 참조 파일에서 읽혀져서 서버로 전달됩니다.

LOB 매개변수 마커는 SQLBindFileToParam()을 사용하여 입력 파일, 또는 SQLBindParameter()를 사용하여 저장된 버퍼와 연관(바인드)될 수 있습니다. 가장 최근의 바인드 매개변수 함수 호출에 의해 유효한 바인딩 유형이 결정됩니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

오류 조건

표 19. SQLBindFileToParam SQLSTATEs

SQLSTATE	설명	설명
58004	예기치 않은 시스템 실패	복구할 수 없는 시스템 오류

표 19. SQLBindFileToParam SQLSTATEs (계속)

SQLSTATE	설명	설명
HY004	SQL 자료 유형이 범위 밖의 값임	*DataType*에 대해 지정된 값이 이 함수 호출에 대해 유효한 SQL 유형이 아닙니다.
HY009	유효하지 않은 인수 값	*FileName*, *FileOptions* *FileNameLength*가 널(null) 포인터입니다.
HY010	함수 순서 오류	함수가 실행시 자료(SQLParamData(), SQLPutData()) 조작중에 호출됩니다. 함수가 BEGIN COMPOUND와 END COMPOUND SQL 작업 내에서 호출됩니다.
HY090	유효하지 않은 스트링 또는 버퍼 길이	*MaxFileNameLength* 입력 인수에 대해 지정된 값이 0 미만입니다.
HY093	유효하지 않은 매개변수 번호	*ParameterNumber*에 대해 지정된 값이 1 미만이거나 지원되는 매개변수의 최대수보다 큽니다.
HYC00	드라이버가 지원되지 않음	서버는 큰 오브젝트 자료 유형을 지원하지 않습니다.

제한사항

큰 오브젝트 자료 유형을 지원하지 않는 DB2 서버에 연결될 때는 이 함수를 사용할 수 없습니다.

참조

- 50 페이지의『SQLBindParam - 매개변수 마커에 버퍼 바인드』
- 104 페이지의『SQLExecute - 명령문 실행』
- 210 페이지의『SQLParamOptions - 매개변수에 대한 입력 배열 지정』

SQLBindParam - 매개변수 마커에 버퍼 바인드

목적

SQLBindParam()은 어플리케이션 변수를 SQL문의 매개변수 마커에 바인드합니다. 이 함수는 매개변수가 입력 또는 출력되는 저장된 프로시듀어 CALL문의 매개변수에 어플리케이션 변수를 바인드하기 위해서도 사용될 수 있습니다. 이 함수는 SQLSetParam()과 같습니다.

구문

```
SQLRETURN SQLBindParam (SQLHSTMT    hstmt,
                        SQLSMALLINT   ipar,
                        SQLSMALLINT   fCType,
                        SQLSMALLINT   fSqlType,
                        SQLINTEGER    cbParamDef,
                        SQLSMALLINT   ibScale,
                        SQLPOINTER    rgbValue,
                        SQLINTEGER    *pcbValue);
```

함수 인수

표 20. SQLBindParam 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLSMALLINT	*ipar*	입력	매개변수 마커 번호는 왼쪽에서 오른쪽으로 1부터 순차적으로 번호가 지정됩니다.

표 20. SQLBindParam 인수 (계속)

자료 유형	인수	사용	설명
SQLSMALLINT	*fCType*	입력	매개변수의 어플리케이션 자료 유형. 다음 유형들이 지원됩니다. • SQL_CHAR • SQL_VARCHAR • SQL_NUMERIC • SQL_DECIMAL • SQL_INTEGER • SQL_SMALLINT • SQL_BIGINT • SQL_FLOAT • SQL_REAL • SQL_DOUBLE • SQL_GRAPHIC • SQL_VARGRAPHIC • SQL_DATETIME • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_BLOB • SQL_CLOB • SQL_DBCLOB • SQL_BLOB_LOCATOR • SQL_CLOB_LOCATOR • SQL_DBCLOB_LOCATOR SQL_DEFAULT를 지정하면 자료가 기본 어플리케이션 자료 유형에서 *fSqlType*에 표시된 유형으로 전송됩니다.

SQLBindParam

표 20. SQLBindParam 인수 (계속)

자료 유형	인수	사용	설명
SQLSMALLINT	*fSqlType*	입력	매개변수의 SQL 자료 유형 지원되는 유형은 다음과 같습니다. • SQL_CHAR • SQL_VARCHAR • SQL_NUMERIC • SQL_DECIMAL • SQL_INTEGER • SQL_SMALLINT • SQL_BIGINT • SQL_FLOAT • SQL_REAL • SQL_DOUBLE • SQL_GRAPHIC • SQL_VARGRAPHIC • SQL_DATETIME • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_BLOB • SQL_CLOB • SQL_DBCLOB • SQL_BLOB_LOCATOR • SQL_CLOB_LOCATOR • SQL_DBCLOB_LOCATOR
SQLINTEGER	*cbParamDef*	입력	대응하는 매개변수 마커의 정밀도. 다음 *fSqlType*에 따라 의미가 달라집니다. • 1바이트 문자 스트링(예를 들면 SQL_CHAR)의 경우, 이 매개변수에 대하여 송신된 바이트 단위의 최대 길이입니다. 이 길이에는 널 종료 문자가 있습니다. • 2바이트 문자 스트링(예를 들면 SQL_GRAPHIC)의 경우, 이 매개변수에 대한 2바이트 문자 단위의 최대 길이입니다. • SQL_DECIMAL 또는 SQL_NUMERIC인 경우, 최대 십진 정밀도입니다. • 그 외의 경우 이 인수는 사용되지 않습니다.

표 20. SQLBindParam 인수 (계속)

자료 유형	인수	사용	설명
SQLSMALLINT	*ibScale*	입력	*fSqlType*이 SQL_DECIMAL 또는 SQL_NUMERIC인 경우 대응하는 매개변수의 소수 자릿수(scale). *fSqlType*이 SQL_TIMESTAMP인 경우, 이는 시간소인의 문자 표시에서 소수점 오른쪽의 자릿수입니다. (예를 들면, yyyy-mm-dd hh:mm:ss.fff의 경우는 3입니다.) 여기서 설명된 *fSqlType* 값 이외의 경우는 *ibScale*이 사용되지 않습니다.
SQLPOINTER	*rgbValue*	입력(지연됨) 또는 출력(지연됨)	실행시에 *pcbValue*에 SQL_NULL_DATA나 SQL_DATA_AT_EXEC이 없으면, *rgbValue*가 매개변수에 대한 실제 자료를 포함하는 버퍼를 가리킵니다. *pcbValue*에 SQL_DATA_AT_EXEC이 있으면, *rgbValue*가 이 매개변수와 연관이 있는 어플리케이션이 정의한 32비트 값입니다. 이 32비트 값은 나중에 SQLParamData() 호출에 의해 어플리케이션에 리턴됩니다.
SQLINTEGER *	*pcbValue*	입력(지연됨) 또는 출력(지원됨) 또는 둘다	명령문이 실행될 때 값이 분석되는 변수 • 널값(null value)이 매개변수로 사용되면 *pcbValue*에는 SQL_NULL_DATA 값이 반드시 있어야 합니다. • ParamData()와 PutData()를 호출하여 실행시 동적 인수가 제공되면, *pcbValue*에는 SQL_DATA_AT_EXEC 값이 반드시 있어야 합니다. • *fcType*이 SQL_CHAR이고 *rgbValue*의 자료에 널 종료 스트링이 있으면, *pcbValue*에 *rgbValue*의 자료 길이나 SQL_NTS 값이 들어갑니다. • *fcType*이 SQL_CHAR이고 *rgbValue*에 널 종료 스트링이 없으면, *pcbValue*에 *rgbValue*의 자료 길이가 있습니다. • *fcType*이 LOB 유형이면, *pcbValue*에 *rgbValue*의 자료 길이가 있습니다. • 그 외의 경우 *pcbValue*는 0이어야 합니다.

사용법

어플리케이션 변수를 저장된 프로시듀어에 대한 출력 매개변수와 바인드하기 위해 SQLBindParam()을 사용할 때, *rgbValue* 버퍼가 메모리에서 *pcbValue* 버퍼 바로 뒤에 있으면 DB2 UDB CLI에 의해 성능이 다소 향상됩니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 21. SQLBindParam SQLSTATEs

SQLSTATE	설명	설명
07006	제한된 자료 유형 속성 위반	SQLSetParam()과 같습니다.
40003 *	명령문 완료 상태를 알 수 없음	함수가 처리를 완료하기 전에 CLI와 자료 소스와의 통신 링크가 끊어졌습니다.
58004	시스템 오류	복구할 수 없는 시스템 오류
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY003	프로그램 유형이 범위 밖의 값임	SQLSetParam()과 같습니다.
HY004	SQL 자료 유형이 범위 밖의 값임	SQLSetParam()과 같습니다.
HY009	유효하지 않은 인수 값	*rgbValue*, *pcbValue* 둘 다 널(null) 포인터이거나 *ipar*가 1 미만입니다.
HY010	함수 순서 오류	SQLExecute() 또는 SQLExecDirect()가 SQL_NEED_DATA를 리턴한 후에 함수가 호출되었으나 자료가 모든 *실행시 자료* 매개변수에 대해 송신되지 않았습니다.
HY013 *	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.
HY014	핸들이 너무 많음	최대 핸들 수가 할당되었습니다.

SQLBindParameter - 버퍼에 매개변수 마커 바인드

목적

SQL문의 매개변수 마커를 어플리케이션 변수에 연관(바인드)시키기 위해 SQLBindParameter()가 사용됩니다. SQLExecute() 또는 SQLExecDirect()가 호출될 때 자료가 어플리케이션에서 데이터베이스 관리 시스템(DBMS)으로 전송됩니다. 자료가 전송될 때 자료가 변환될 수도 있습니다.

이 함수는 매개변수가 입력, 출력, 또는 두 경우 모두가 될 수 있는 저장된 프로시듀어 CALL문의 매개변수에 어플리케이션 기억장치를 바인드하기 위해서도 사용할 수 있습니다. 이 함수는 SQLSetParam()의 확장이어야 합니다.

구문

```
SQLRETURN SQLBindParameter(SQLHSTMT          StatementHandle,
                           SQLSMALLINT       ParameterNumber,
                           SQLSMALLINT       InputOutputType,
                           SQLSMALLINT       ValueType,
                           SQLSMALLINT       ParameterType,
                           SQLINTEGER        ColumnSize,
                           SQLSMALLINT       DecimalDigits,
                           SQLPOINTER        ParameterValuePtr,
                           SQLINTEGER        BufferLength,
                           SQLINTEGER       *StrLen_or_IndPtr);
```

함수 인수

표 22. SQLBindParameter 인수

자료 유형	인수	사용	설명
SQLHSTMT	StatementHandle	입력	명령문 핸들
SQLSMALLINT	ParameterNumber	입력	매개변수 마커 번호는 왼쪽에서 오른쪽으로 1부터 순차적으로 번호가 지정됩니다.

SQLBindParameter

표 22. SQLBindParameter 인수 (계속)

자료 유형	인수	사용	설명
SQLSMALLINT	InputOutputType	입력	매개변수 유형. IPD의 SQL_DESC_PARAMETER_TYPE 필드 값이 이 인수에 설정됩니다. 지원되는 유형은 다음과 같습니다. • SQL_PARAM_INPUT: 매개변수 마커가 저장된 프로시듀어 CALL이 아닌 SQL문과 연관됩니다. 또는 호출된 저장된 프로시듀어의 입력 변수로 표시됩니다. 명령문이 실행되면 매개변수에 대한 실제 자료 값이 서버로 송신됩니다. *ParameterValuePtr* 버퍼에 유효한 입력 자료 값이 있어야 합니다. *StrLen_or_IndPtr* 버퍼에 대응하는 길이 값이나 SQL_NTS, SQL_NULL_DATA 또는 (값이 SQLParamData()와 SQLPutData()를 통해 송신되어야 하는 경우) SQL_DATA_AT_EXEC이 있어야 합니다. • SQL_PARAM_INPUT_OUTPUT: 매개변수 마커가 호출된 저장된 프로시듀어의 입/출력 매개변수와 연관됩니다. 명령문이 실행되면 매개변수에 대한 실제 자료 값이 서버로 송신됩니다. *ParameterValuePtr* 버퍼에 유효한 입력 자료 값이 있어야 합니다. *StrLen_or_IndPtr* 버퍼에 대응하는 길이 값이나 SQL_NTS, SQL_NULL_DATA 또는 (값이 SQLParamData()와 SQLPutData()를 통해 송신되어야 하는 경우) SQL_DATA_AT_EXEC이 있어야 합니다. • SQL_PARAM_OUTPUT: 매개변수 마커가 호출된 저장 프로시듀어의 출력 매개변수 또는 저장 프로시듀어의 리턴 값과 연관됩니다. 명령문이 실행되면, 출력 매개변수에 대한 자료가 *ParameterValuePtr* 및 *StrLen_or_IndPtr*에 지정된 어플리케이션 버퍼에 리턴됩니다(둘 다 널(null) 포인터가 아닌 경우). 이 경우 출력 자료가 삭제됩니다. 출력 매개변수에 리턴 값이 없으면 *StrLen_or_IndPtr*이 SQL_NULL_DATA로 설정됩니다.

표 22. SQLBindParameter 인수 (계속)

자료 유형	인수	사용	설명
SQLSMALLINT	ValueType	입력	매개변수의 C 자료 유형. 다음 유형들이 지원됩니다. • SQL_CHAR • SQL_VARCHAR • SQL_NUMERIC • SQL_DECIMAL • SQL_INTEGER • SQL_SMALLINT • SQL_BIGINT • SQL_FLOAT • SQL_REAL • SQL_DOUBLE • SQL_GRAPHIC • SQL_VARGRAPHIC • SQL_DATETIME • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_BLOB • SQL_CLOB • SQL_DBCLOB • SQL_BLOB_LOCATOR • SQL_CLOB_LOCATOR • SQL_DBCLOB_LOCATOR SQL_C_DEFAULT를 지정하면 자료가 디폴트 C 자료 유형에서 *ParameterType*에 표시된 유형으로 전송됩니다.
SQLSMALLINT	ParameterType	입력	매개변수의 SQL 자료 유형
SQLINTEGER	ColumnSize	입력	대응하는 매개변수 마커의 정밀도. 다음 *ParameterType*에 따라 의미가 달라집니다. • 2진 또는 1바이트 문자 스트링(예를 들면 SQL_CHAR)의 경우, 이 매개변수 마커에 대한 바이트 단위의 최대 길이입니다. • 2바이트 문자 스트링(예를 들면 SQL_GRAPHIC)의 경우, 이 매개변수에 대한 2바이트 문자 단위의 최대 길이입니다. • SQL_DECIMAL 또는 SQL_NUMERIC의 경우, 최대 십진 정밀도입니다. • 그 외의 경우 이 인수는 무시됩니다.

SQLBindParameter

표 22. SQLBindParameter 인수 (계속)

자료 유형	인수	사용	설명
SQLSMALLINT	DecimalDigits	입력	*ParameterType*이 SQL_DECIMAL 또는 SQL_NUMERIC인 경우 대응하는 매개변수의 소수 자릿수 (scale). *ParameterType*이 SQL_TYPE_TIMESTAMP인 경우, 이는 시간소인의 문자 표시에서 소수점 오른쪽의 자릿수입니다. (예를 들면, yyyy-mm-dd hh:mm:ss.fff의 경우는 3입니다.) 여기서 설명된 *ParameterType* 값 이외의 경우는 *DecimalDigits*가 사용되지 않습니다.
SQLPOINTER	ParameterValuePtr	입력(지연됨) 및/또는 출력 (지연됨)	• 입력시(*InputOutputType*이 SQL_PARAM_INPUT 또는 SQL_PARAM_INPUT_OUTPUT으로 설정됨): 실행 시 *StrLen_or_IndPtr*에 SQL_NULL_DATA나 SQL_DATA_AT_EXEC이 없으면, *ParameterValuePtr*이 매개변수에 대한 실제 자료가 있는 버퍼를 가리킵니다. *StrLen_or_IndPtr*에 SQL_DATA_AT_EXEC이 있으면, *ParameterValuePtr*이 이 매개변수와 연관이 있는 어플리케이션이 정의한 32비트 값입니다. 이 32비트 값은 후속 `SQLParamData()` 호출을 통해 리턴됩니다. `SQLParamOptions()`이 호출되어 매개변수에 대해 여러 값을 지정하면, *ParameterValuePtr*은 *BufferLength*바이트의 입력 버퍼 배열을 가리키는 포인터입니다. • 출력시(*InputOutputType*이 SQL_PARAM_OUTPUT 또는 SQL_PARAM_INPUT_OUTPUT으로 설정됨): *ParameterValuePtr*는 저장된 프로시듀어의 출력 매개변수 값이 저장된 버퍼를 가리킵니다. *InputOutputType*가 SQL_PARAM_OUTPUT으로 설정되고, *ParameterValuePtr* 및 *StrLen_or_IndPtr*가 널(null) 포인터이면, 저장된 프로시듀어 호출의 리턴 값이나 출력 매개변수 값이 삭제됩니다.
SQLINTEGER	BufferLength	입력	사용되지 않음

표 22. SQLBindParameter 인수 (계속)

자료 유형	인수	사용	설명
SQLINTEGER *	StrLen_or_IndPtr	입력(지연됨) 및/또는 출력 (지연됨)	이 값이 입력 또는 입/출력 매개변수인 경우 다음과 같습니다. (명령문이 실행될 때) *ParameterValuePtr*에 저장된 매개변수 마커 값의 길이가 있는 위치를 가리키는 포인터입니다. 매개변수 마커에 대해 널값을 지정하려면 이 기억장치 위치에 SQL_NULL_DATA가 있어야 합니다. *ValueType*이 SQL_C_CHAR이면, 이 기억장치 위치에는 *ParameterValuePtr*에 저장된 자료의 정확한 길이 또는 SQL_NTS(*ParameterValuePtr*의 내용이 널로 끝나는 경우)가 있습니다. *ValueType*이 LOB 자료를 가리키는 경우, 이 기억장치 위치에는 *ParameterValuePtr*에 저장된 자료 길이가 들어 있어야 합니다. *ValueType*이 문자 자료를 표시하고(명시적 또는 내포적으로 SQL_C_DEFAULT를 사용하여) 이 포인터가 널(null)로 설정된 경우, 어플리케이션이 *ParameterValuePtr*에 널 종료 스트링을 제공한다고 가정됩니다. 또한 이는 이 매개변수 마커는 결코 널값을 갖지 않음을 의미합니다. SQLExecute() 또는 SQLExecDirect()가 호출되고, *StrLen_or_IndPtr*가 SQL_DATA_AT_EXEC 값을 가리킬 때, 매개변수에 대한 자료가 SQLPutData()와 함께 송신됩니다. 이 매개변수는 **실행시 자료** 매개변수로 참조됩니다.

사용법

매개변수 마커는 "?" 문자로 표시되며 표시되며 이 명령문이 실행될 때 어플리케이션이 제공한 값이 대체되는 명령문에서의 위치를 표시하는데 사용됩니다. 이 값은 어플리케이션 변수에서 얻을 수 있습니다.

어플리케이션은 SQL문을 실행하기 전에 SQL문의 각 매개변수 마커에 변수를 바인드합니다. 이 함수의 경우, *ParameterValuePtr* 및 *StrLen_or_IndPtr*이 지연되는 인수입니다. 명령문이 실행될 때 기억장치 위치가 유효해야 하며 입력 자료 값이 있어야 합니다. 이는 SQLExecDirect() 또는 SQLExecute() 호출을 같은 프로시듀어 범위에서 SQLBindParameter() 호출로 보존하거나 이 기억장치 위치가 동적으로 할당되거나 정적 또는 전역 변수로 선언되어야 함을 의미합니다.

매개변수 마커는 숫자(*ParameterNumber*)에 의해 참조되며 왼쪽에서 오른쪽으로, 1부터 순차적으로 번호가 지정됩니다.

이 함수에 의해 바인드된 모든 매개변수는 SQLFreeStmt()가 SQL_DROP 또는 SQL_RESET_PARAMS 옵션과 함께 호출되거나 SQLBindParameter()가 같은 *ParameterNumber* 숫자에 의해 다시 호출될 때까지 유효합니다.

SQLBindParameter

SQL문이 실행되고 결과가 처리되고 나면, 어플리케이션은 다른 SQL문을 실행하기 위해 명령문 핸들을 다시 사용할 수도 있습니다. 이 매개변수 마커 스펙이 다르면(매개변수 수, 길이 또는 유형) SQL_RESET_PARAMS와 함께 SQLFreeStmt()가 호출되어 매개변수 바인딩을 지우거나 재설정해야 합니다.

*ValueType*에 의해 주어진 C 버퍼 자료 유형은 *ParameterType*에 의해 표시된 SQL 자료 유형과 호환되어야 하며 그렇지 않으면 오류가 발생합니다.

어플리케이션은 매개변수에 대한 값을 *ParameterValuePtr* 버퍼로 전달하거나 한 번 이상의 SQLPutData() 호출로 전달할 수 있습니다. 후자의 경우 이 매개변수는 실행시 자료 매개변수입니다. 어플리케이션은 SQL_DATA_AT_EXEC 값을 *StrLen_or_IndPtr* 버퍼로 가져 가서 data-at-execution 매개변수를 DB2 UDB CLI에 알려줍니다. 어플리케이션은 후속 SQLParamData() 호출에서 리턴되어 매개변수 위치를 식별하기 위해 사용될 수 있는 32비트 값으로 *ParameterValuePtr* 입력 인수를 설정합니다.

ParameterValuePtr 및 *StrLen_or_IndPtr*에 의해 참조된 변수의 자료가 명령문이 실행될 때까지 확인되지 않으므로 자료 내용이나 형식 오류는 SQLExecute() 또는 SQLExecDirect()가 호출될 때까지 보고되거나 감지되지 않습니다.

SQLBindParameter()는 매개변수가 입력, 입/출력, 또는 출력인지를 지정하는 방법을 제공하여 SQLSetParam() 함수의 기능을 확장합니다. 이 정보는 저장된 프로시듀어에 대한 매개변수의 적절한 처리를 위해 필요합니다.

InputOutputType 인수는 매개변수 유형을 지정합니다. 프로시듀어를 호출하지 않는 SQL문의 모든 매개변수는 입력 매개변수입니다. 저장된 프로시듀어 호출의 매개변수는 입력, 입/출력 또는 출력 매개변수일 수 있습니다. DB2의 저장 프로시듀어 인수 규약은 일반적으로 모든 프로시듀어 인수가 입/출력(I/O) 인수임을 가정하지만, 어플리케이션 프로그래머는 좀 더 정확한 코딩 방식을 따르기 위해 SQLBindParameter()에 입력 또는 출력 속성을 정확하게 지정할 수 있습니다. 또한 이 유형은 저장된 프로시듀어가 SQL CREATE PROCEDURE 문과 함께 등록되었을 때 지정된 매개변수 유형과 일치해야 합니다.

- 어플리케이션에서 프로시듀어 호출의 매개변수 유형을 판별할 수 없으면 *InputOutputType*을 SQL_PARAM_INPUT으로 설정합니다. 자료 소스가 매개변수에 대한 값을 리턴하면 DB2 UDB CLI는 이 값을 삭제합니다.
- 어플리케이션이 매개변수를 SQL_PARAM_INPUT_OUTPUT 또는 SQL_PARAM_OUTPUT으로 표시하고 자료 소스가 값을 리턴하지 않으면, DB2 UDB CLI는 *StrLen_or_IndPtr* 버퍼를 SQL_NULL_DATA로 설정합니다.
- 어플리케이션이 매개변수를 SQL_PARAM_OUTPUT으로 표시하면 CALL문이 처리된 후 매개변수에 대한 자료가 어플리케이션으로 리턴됩니다. *ParameterValuePtr* 및 *StrLen_or_IndPtr* 인수가 둘 다 널(null) 포인터이면, DB2 UDB CLI는 출력 값을 삭제합니다. 자료 소스가 출력 매개변수에 대한 값을 리턴하지 않으면 DB2 UDB CLI는 *StrLen_or_IndPtr* 버퍼를 SQL_NULL_DATA로 설정합니다.
- 이 함수의 경우 *ParameterValuePtr* 및 *StrLen_or_IndPtr*는 연기된 인수입니다. *InputOutputType*이 SQL_PARAM_INPUT 또는 SQL_PARAM_INPUT_OUTPUT으로 설정되는 경우, 명령문이 실행될 때 기억장치 위치가 유효해야 하며 입력 자료 값이 있어야 합니다. 이는 SQLExecDirect() 또는 SQLExecute() 호출을 SQLBindParameter() 호출과 같은 프로시듀어 범위 내에서 보존하거나 이 기억장치 위치가 동적으로 할당되거나 정적/전역 변수로 선언되어야 함을 의미합니다.

마찬가지로, *InputOutputType*이 SQL_PARAM_OUTPUT 또는 SQL_PARAM_INPUT_OUTPUT으로 설정되면, ParameterValuePtr와 StrLen_or_IndPtr 버퍼 위치는 CALL문이 실행될 때까지 유효해야 합니다.

어플리케이션은 매개변수에 대한 값을 *ParameterValuePtr* 버퍼로 전달하거나 한 번 이상의 SQLPutData() 호출로 전달할 수 있습니다. 후자의 경우 이 매개변수는 실행시 자료 매개변수입니다. 어플리케이션은 SQL_DATA_AT_EXEC 값을 StrLen_or_IndPtr 버퍼로 가져 가서 실행시 자료 매개변수를 DB2 UDB CLI에 알려줍니다. 어플리케이션은 후속 SQLParamData() 호출에서 리턴되어 매개변수 위치를 식별하기 위해 사용될 수 있는 32비트 값으로 *ParameterValuePtr* 입력 인수를 설정합니다.

어플리케이션 변수를 저장된 프로시듀어에 대한 출력 매개변수와 바인드하기 위해 SQLBindParameter()를 사용할 때, *ParameterValuePtr* 버퍼가 메모리에서 *StrLen_or_IndPtr* 버퍼 바로 뒤에 있으면 DB2 UDB CLI에 의해 성능이 다소 향상됩니다. 예를 들면 다음과 같습니다.

```
struct {  SQLINTEGER  StrLen_or_IndPtr;
          SQLCHAR     ParameterValuePtr[MAX_BUFFER];
       } column;
```

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

오류 조건

표 23. SQLBindParameter SQLSTATEs

SQLSTATE	설명	설명
07006	변환이 유효하지 않음	*ValueType* 인수에 의해 식별된 자료 값에서 *ParameterType* 인수에 의해 식별된 자료 유형으로 변환하는 것(예를 들면, SQL_C_DATE를 SQL_DOUBLE로 변환하는 것)은 의미가 있는 변환이 아닙니다.
40003 **08**S01	통신 링크 실패	함수가 완료되기 전에 어플리케이션과 자료 소스와의 통신 링크가 끊어졌습니다.
58004	예기치 않은 시스템 실패	복구할 수 없는 시스템 오류
HY001	메모리 할당 실패	DB2 UDB CLI가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY003	프로그램 유형이 범위 밖의 값임	*ParameterNumber* 인수에 의해 지정된 값이 유효한 자료 유형이나 SQL_C_DEFAULT가 아닙니다.
HY004	SQL 자료 유형이 범위 밖의 값임	*ParameterType* 인수에 대해 지정된 값이 유효한 SQL 자료 유형이 아닙니다.
HY009	인수 값이 유효하지 않음	*ParameterValuePtr* 인수가 널(null) 포인터이고 *StrLen_or_IndPtr* 인수도 널(null) 포인터이며, *InputOutputType*은 SQL_PARAM_OUTPUT이 아닙니다.
HY010	함수 순서 오류	SQLExecute() 또는 SQLExecDirect()가 SQL_NEED_DATA를 리턴한 후에 함수가 호출되었으나 자료가 모든 *실행시 자료* 매개변수에 대해 송신되지 않았습니다.

SQLBindParameter

표 23. SQLBindParameter SQLSTATEs (계속)

SQLSTATE	설명	설명
HY013	예기치 않은 메모리 처리 오류	DB2 UDB CLI가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.
HY014	핸들이 너무 많음	최대 핸들 수가 할당되었습니다.
HY021	일치하지 않는 설명자 정보	일관성 체크시에 체크된 설명자 정보가 일관되지 않습니다.
HY090	유효하지 않은 스트링 또는 버퍼 길이	*BufferLength* 인수에 대해 지정된 값이 0 미만입니다.
HY093	유효하지 않은 매개변수 번호	*ValueType* 인수에 대해 지정된 값이 1 미만이거나 서버에 의해 지원되는 매개변수의 최대 수를 초과했습니다.
HY094	소수 자릿수(scale) 값이 유효하지 않음	*ParameterType*에 대해 지정된 값이 SQL_DECIMAL 또는 SQL_NUMERIC이며, *DecimalDigits*에 대해 지정된 값이 0 미만이거나 *ParamDef*(정밀도) 인수에 대한 값보다 큽니다. *ParameterType*에 대해 지정된 값이 SQL_C_TIMESTAMP이고 *ParameterType*에 대한 값이 SQL_CHAR 또는 SQL_VARCHAR이며, *DecimalDigits*에 대한 값이 0 미만이고 6보다 큽니다.
HY104	정밀도 값이 유효하지 않음	*ParameterType*에 대해 지정된 값이 SQL_DECIMAL 또는 SQL_NUMBER이고 *ParamDef*에 대해 지정된 값이 1 미만입니다.
HY105	매개변수 유형이 유효하지 않음	*InputOutputType*이 SQL_PARAM_INPUT, SQL_PARAM_OUTPUT, SQL_PARAM_INPUT_OUTPUT 중 하나가 아닙니다.
HYC00	드라이버가 지원되지 않음	DB2 UDB CLI 또는 자료 소스는 *ValueType* 인수에 대해 지정된 값과 *ParameterType* 인수에 대해 지정된 값의 결합에 의해 지정된 변환을 지원하지 않습니다. *ParameterType* 인수에 대해 지정된 값은 DB2 UDB CLI 또는 자료 소스에 의해 지원되지 않습니다.

참조

- 102 페이지의『SQLExecDirect - 명령문 직접 실행』
- 104 페이지의『SQLExecute - 명령문 실행』
- 208 페이지의『SQLParamData - 자료 값이 필요한 다음 매개변수 얻기』
- 230 페이지의『SQLPutData - 매개변수에 대한 자료 값 전달』

SQLCancel - 명령문 취소

목적

SQLCancel()은 비동기식으로 실행되고 있는 진행 중인 SQL문 작업 처리를 종료시키려고 시도합니다.

SQLCancel()은 호환성 목적만을 위한 것이며 SQL문 실행에는 영향을 주지 않습니다.

구문

```
SQLRETURN SQLCancel (SQLHSTMT     hstmt);
```

함수 인수

표 24. SQLCancel 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.

사용법

성공적인 리턴 코드는 구현 프로그램에서 취소 요구를 받아들였음을 표시하며, 처리가 취소되었음은 확인할 수 없습니다.

리턴 코드

- SQL_SUCCESS
- SQL_INVALID_HANDLE
- SQL_ERROR

진단

표 25. SQLCancel SQLSTATEs

SQLSTATE	설명	설명
HY009 *****	유효하지 않은 인수 값	*hstmt*는 명령문 핸들이 아닙니다.

제한사항

DB2 UDB CLI는 비동기식 명령문 실행을 지원하지 않습니다.

SQLCloseCursor - 커서 명령문 닫기

목적

SQLCloseCursor()는 명령문 핸들에서 열린 커서를 닫습니다.

구문

```
SQLRETURN SQLCloseCursor (SQLHSTMT     hstmt);
```

함수 인수

표 26. SQLCancel 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.

사용법

SQLCloseCursor()를 호출하면 명령문 핸들과 연관된 커서를 닫고 지연되고 있는 결과를 삭제합니다. 명령문 핸들과 연관된 열린 커서가 없으면 함수는 어떤 영향도 주지 않습니다.

명령문 핸들이 복수의 결과 세트를 갖고 있는 저장된 프로시듀어를 참조하는 경우, SQLCloseCursor()는 현재의 결과 세트만 닫습니다. 다른 결과 세트들은 열려 있고 사용 가능한 상태로 남아 있습니다.

리턴 코드

- SQL_SUCCESS
- SQL_INVALID_HANDLE
- SQL_ERROR

진단

표 27. SQLCancel SQLSTATEs

SQLSTATE	설명	설명
08003 *	연결이 열려 있지 않음	hstmt에 대한 연결이 설정되지 않았습니다.
HY009 *	유효하지 않은 인수 값	*hstmt*는 명령문 핸들이 아닙니다.

SQLColAttributes - 열 속성

목적

SQLColAttributes()는 결과 세트 열에 대한 속성을 가져오며 열의 수를 판별하는데 사용됩니다. SQLColAttributes()는 SQLDescribeCol()함수를 더 확장할 수 있게 하는 대체 함수입니다.

이 함수를 호출하기 전에 SQLPrepare() 또는 SQLExecDirect()를 호출해야 합니다.

어플리케이션이 다양한 속성(자료 유형과 길이 등)을 모를 경우 SQLBindCol()을 호출하기 전에 이 함수(또는 SQLDescribeCol())를 호출해야 합니다.

구문

```
SQLRETURN SQLColAttributes (SQLHSTMT       hstmt,
                            SQLSMALLINT    icol,
                            SQLSMALLINT    fDescType,
                            SQLCHAR        *rgbDesc,
                            SQLINTEGER     cbDescMax,
                            SQLINTEGER     *pcbDesc,
                            SQLINTEGER     *pfDesc);
```

함수 인수

표 28. SQLColAttributes 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLSMALLINT	*icol*	입력	결과 집합의 열 번호(1부터 결과 집합의 열 수까지). SQL_DESC_COUNT가 지정되면 이 인수는 무시됩니다.
SQLSMALLINT	*fDescType*	입력	지원되는 값이 표 29에 나옵니다.
SQLCHAR *	*rgbDesc*	출력	스트링 열 속성에 대한 버퍼를 가리키는 포인터
SQLINTEGER	*cbDescMax*	입력	설명자 버퍼 길이(*rgbDesc*)
SQLINTEGER *	*pcbDesc*	출력	리턴될 설명자의 실제 바이트 수. 이 인수에 *rgbDesc* 버퍼의 길이보다 크거나 같은 값이 있으면 값이 절단됩니다. 이 설명자는 *cbDescMax* - 1 바이트로 절단됩니다.
SQLINTEGER *	*pfDesc*	출력	숫자 열 속성에 대한 정보를 나타내는 정수를 가리키는 포인터

표 29. fDescType 설명자 유형

설명자	유형	설명
SQL_DESC_COUNT	SMALLINT	결과 집합의 열 번호는 *pfDesc*에 리턴됩니다.
SQL_DESC_NAME	CHAR(128)	*icol* 열의 이름이 *rgbDesc*에 리턴됩니다. 열이 표현식이면 리턴된 결과는 제품에 따라 달라집니다.

표 29. fDescType 설명자 유형 (계속)

설명자	유형	설명
SQL_DESC_TYPE	SMALLINT	*icol*에 식별된 열의 SQL 자료 유형이 *pfDesc*에 리턴됩니다. *pfSqlType*에 대해 사용할 수 있는 값이 21 페이지의 표 5에 나옵니다.
SQL_DESC_LENGTH	INTEGER	열과 연관된 자료의 *바이트* 수는 *pfDesc*에 리턴됩니다. *icol*에 식별된 열이 예를 들어, SQL_CHAR, SQL_VARCHAR, 또는 SQL_LONG_VARCHAR를 기초로 한 문자이면 실제 길이 또는 최대 길이가 리턴됩니다. 열 유형이 SQL_DECIMAL 또는 SQL_NUMERIC이면, SQL_DESC_LENGTH는 *(정밀도 * 256) + 소수 자릿수*입니다. 같은 값이 SQLBindCol()의 입력으로 전달되도록 이 값이 리턴됩니다. SQL_DESC_PRECISION과 SQL_DESC_SCALE을 사용하여 이 자료 유형에 대한 분리된 값으로 정밀도와 소수 자릿수(scale)를 구할 수도 있습니다.
SQL_DESC_PRECISION	SMALLINT	열의 정밀도 속성이 리턴됩니다.
SQL_DESC_SCALE	SMALLINT	열의 소수 자릿수(scale) 속성이 리턴됩니다.
SQL_DESC_NULLABLE	SMALLINT	*icol*에 의해 식별된 열에 널이 있으면 SQL_NULLABLE이 *pfDesc*에 리턴됩니다. 열이 널을 받아들이지 않도록 제한되면 SQL_NO_NULLS이 *pfDesc*에 리턴됩니다.
SQL_DESC_UNNAMED	SMALLINT	NAME 필드가 실제 이름이면 SQL_NAMED이고 NAME 필드가 구현 프로그램에서 생성한 이름이면 SQL_UNNAMED입니다.
SQL_DESC_AUTO_INCREMENT	INTEGER	새로운 행이 표에 삽입됨에 따라 열이 자동으로 증가하면 SQL_TRUE입니다. 열이 자동으로 증가하지 않으면 SQL_FALSE입니다.

표 29. fDescType 설명자 유형 (계속)

설명자	유형	설명
SQL_DESC_SEARCHABLE	INTEGER	열에서 **WHERE**절이 사용되지 않으면 SQL_UNSEARCHABLE입니다. 열에서 **WHERE** 절이 **LIKE** 술부와 함께만 사용되는 경우에만 SQL_LIKE_ONLY입니다. 열에서 **WHERE**절이 **LIKE**를 제외한 모든 비교 연산자와 함께 사용되면 SQL_ALL_EXCEPT_LIKE입니다. 열에서 **WHERE** 절이 어떤 비교 연산자든지 함께 사용되면 SQL_SEARCHABLE입니다. 이 속성이 검색되도록 하려면 명령문 핸들이나 연결 핸들에 SQL_ATTR_EXTENDED_COL_INFO 속성을 반드시 SQL_TRUE로 설정해야 합니다.
SQL_DESC_UPDATABLE	INTEGER	정의된 상수에 대한 값으로 열을 서술합니다. SQL_ATTR_READONLY SQL_ATTR_WRITE SQL_ATTR_READWRITE_UNKNOWN SQL_COLUMN_UPDATABLE는 결과 세트에서 열을 갱신할 수 있는지 서술합니다. 열의 갱신 가능 여부는 자료 유형, 사용자 특권, 결과 세트 자체의 정의 등에 따라 결정됩니다. 열을 갱신할 수 있는지 확실하지 않으면 SQL_ATTR_READWRITE_UNKNOWN이 리턴됩니다. 이 속성이 검색되도록 하려면 명령문 핸들이나 연결 핸들에 SQL_ATTR_EXTENDED_COL_INFO 속성을 반드시 SQL_TRUE로 설정해야 합니다.
SQL_DESC_BASE_TABLE	CHAR(128)	작성된 열의 기초가 되는 표의 이름. 이 속성이 검색되도록 하려면 명령문 핸들이나 연결 핸들에 SQL_ATTR_EXTENDED_COL_INFO 속성을 반드시 SQL_TRUE로 설정해야 합니다.

표 29. fDescType 설명자 유형 (계속)

설명자	유형	설명
SQL_DESC_BASE_COLUMN	CHAR(128)	작성된 열의 기초가 되는 표에서 실제 열의 이름. 이 속성이 검색되도록 하려면 명령문 핸들이나 연결 핸들에 SQL_ATTR_EXTENDED_COL_INFO 속성을 반드시 SQL_TRUE로 설정해야 합니다.
SQL_DESC_BASE_SCHEMA	CHAR(128)	작성된 열의 기초가 되는 표의 스키마명. 이 속성이 검색되도록 하려면 명령문 핸들이나 연결 핸들에 SQL_ATTR_EXTENDED_COL_INFO 속성을 반드시 SQL_TRUE로 설정해야 합니다.
SQL_DESC_LABEL	CHAR(128)	이 열의 레이블(있을 경우). 아니면 0 길이 스트링. 이 속성이 검색되도록 하려면 명령문 핸들이나 연결 핸들에 SQL_ATTR_EXTENDED_COL_INFO 속성을 반드시 SQL_TRUE로 설정해야 합니다.

사용법

SQLDescribeCol()과 같은 특정 인수 집합을 리턴하는 대신, SQLColAttributes()를 사용하여 특정 열에 대해 수신하려는 속성을 지정할 수 있습니다. 원하는 정보가 스트링이면 *rgbDesc*에 리턴됩니다. 원하는 정보가 숫자이면 *pfDesc*에 리턴됩니다.

SQLColAttributes()는 나중 확장에 대해 허용하지만, 각 열에 대해 SQLDescribeCol()보다 동일 자료를 수신하기 위한 더 많은 호출을 필요로 합니다.

fDescType 설명자 유형이 데이터베이스 서버에 적용되지 않으면 설명자의 예상되는 결과에 따라 0이 *pfDesc*에 리턴되거나 빈 스트링이 *rgbDesc*에 리턴됩니다.

열은 번호(왼쪽에서 오른쪽으로, 1부터 번호가 지정됨)에 의해 식별되며 다른 순서로 서술할 수도 있습니다.

리턴된 열이 있는 지를 판별하기 위해 SQLNumResultCols()를 호출하는 대신, *fDescType*을 SQL_DESC_COUNT로 설정하여 SQLColAttributes()를 호출할 수도 있습니다.

결과 집합의 존재 여부를 판별하기 위해 SQLColAttributes()를 호출하기 위해 SQLNumResultCols()를 호출합니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

진단

표 30. SQLColAttributes SQLSTATEs

SQLSTATE	설명	설명
07009	유효하지 않은 열 번호	*icol* 인수에 대해 지정된 값이 1 미만입니다.
HY009	유효하지 않은 인수 값	*fDescType* 인수에 대해 지정된 값이 65 페이지의 표 29에 지정된 값과 같지 않습니다. *rgbDesc*, *pcbDesc* 또는 *pfDesc* 인수가 널(null) 포인터입니다.
HY010	함수 순서 오류	*hstmt*에 대해 SQLPrepare() 또는 SQLExecDirect()를 호출하기 전에 이 함수를 호출합니다.
HYC00	드라이버가 지원되지 않음	*icol* 열에 대한 데이터베이스에 의해 리턴된 SQL 자료 유형이 DB2 UDB CLI에 의해 인식되지 않습니다.

참조

- 39 페이지의『SQLBindCol - 어플리케이션 변수에 열 바인드』
- 84 페이지의『SQLDescribeCol - 열 속성 설명』
- 102 페이지의『SQLExecDirect - 명령문 직접 실행』
- 104 페이지의『SQLExecute - 명령문 실행』
- 212 페이지의『SQLPrepare - 명령문 준비』

SQLColumnPrivileges - 표의 열과 연관된 권한 확보

목적

SQLColumnPrivileges()은 지정된 표에 대한 열 리스트 및 연관된 권한을 리턴합니다. 조회에서 생성된 결과 집합을 프로세스하는 데 사용된 동일한 함수를 사용하여 검색할 수 있는 SQL 결과 집합에 이 정보가 리턴됩니다.

구문

```
SQLRETURN SQLColumnPrivileges  (
                SQLHSTMT            StatementHandle,
                SQLCHAR             *CatalogName,
                SQLSMALLINT         NameLength1,
                SQLCHAR             *SchemaName,
                SQLSMALLINT         NameLength2,
                SQLCHAR             *TableName
                SQLSMALLINT         NameLength3,
                SQLCHAR             *ColumnName,
                SQLSMALLINT         NameLength4);
```

함수 인수

표 31. SQLColumnPrivileges 인수

자료 유형	인수	사용	설명
SQLHSTMT	*명령문 핸들*	입력	명령문 핸들
SQLCHAR *	*CatalogName*	입력	세 부분으로 된 표 이름의 카탈로그 규정자. 이 포인터는 널(null) 포인터이거나 길이가 0인 스트링이어야 합니다.
SQLSMALLINT	*NameLength1*	입력	*CatalogName*의 길이0으로 설정되어야 합니다.
SQLCHAR *	*SchemaName*	입력	표 이름의 스키마 규정자.
SQLSMALLINT	*NameLength2*	입력	*SchemaName*의 길이
SQLCHAR *	*TableName*	입력	표 이름.
SQLSMALLINT	*NameLength3*	입력	*TableName*의 길이.
SQLCHAR *	*ColumnName*	입력	열 이름으로 결과 집합을 규정하기 위한 *패턴 값*이 있는 버퍼.
SQLSMALLINT	*NameLength4*	입력	*ColumnName*의 길이.

사용법

71 페이지의 표 32에 나열된 열이 있는 표준 결과 집합으로서 결과가 리턴됩니다. 결과 집합이 TABLE_CAT, TABLE_SCHEM, TABLE_NAME, COLUMN_NAME 및 PRIVILEGE에 의해 주문됩니다.다중 권한이 지정된 열과 연관된 경우, 각 권한은 분리 행으로서 리턴됩니다. 일반적인 어플리케이션에서 열 권한 정보를 판별하기 위해 SQLColumns()으로 호출한 다음 이 함수를 호출하고자 할 수 있습니다. 어플리케이션은 이 함수에 대한 입력 인수로서 SQLColumns() 결과 집합의 TABLE_SCHEM, TABLE_NAME, COLUMN_NAME 열에 리턴된 문자 스트링을 사용하여야 합니다.

많은 경우에 ince calls to SQLColumnPrivileges() 호출은 시스템 카탈로그에 반해 복잡하고 비용이 많이 드는 조회에맵핑되므로 자주 사용하지 않는 것이 좋으며 호출을 반복하기 보다는 결과를 저장해야 합니다.

카탈로그 함수 결과 집합의 VARCHAR 열이 SQL 92 제한 길이와 일치하도록 최대 길이 인수인 128로 선언됩니다. DB2 이름이 128 보다 적기 때문에 어플리케이션이 출력 버퍼에 128문자(및 널 터미네이터)를 항상 별개로 설정하거나, SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_TABLE_NAME_LEN 및 SQL_MAX_COLUMN_NAME_LEN을 사용해서 SQLGetInfo()를 호출하여 연결된 DBMS에 의해 지원되는 TABLE_CAT, TABLE_SCHEM, TABLE_NAME 및 COLUMN_NAME 열의 실제 길이를 각각 판별하도록 선택할 수 있습니다.

ColumnName 인수가 탐색 패턴을 수용하는 것에 유의하십시오.

새로운 열이 추가되고 기존 열의 이름이 이후의 릴리스에서 변경될 수 있지만 현재 열의 위치는 변경되지 않습니다.

표 32. SQLColumnPrivileges에 의해 리턴되는 열

열 번호/이름	자료 유형	설명
TABLE_CAT	VARCHAR(128)	이 열은 항상 널(null)입니다.
TABLE_SCHEM	VARCHAR(128)	TABLE_NAME이 있는 스키마명.
TABLE_NAME	VARCHAR(128)는 널(null)이 아님	표나 뷰의 이름.
COLUMN_NAME	VARCHAR(128)는 널(null)이 아님	지정된 표나 뷰의 열에 대한 이름.
권한 부여자	VARCHAR(128)	권한을 부여한 사용자의 권한 부여 ID.
권한 수여자	VARCHAR(128)	권한을 부여받은 사용자의 권한 부여 ID.
권한	VARCHAR(128)	열 권한. 다음 중 하나일 수 있습니다. • INSERT • REFERENCES • SELECT • UPDATE
IS_GRANTABLE	VARCHAR(3)	권한 수여자가 권한을 다른 사용자에게 부여하는 지를 표시합니다. 예 또는 아니오.

주: DB2 CLI에 의해 사용된 열 이름은 X/Open CLI CAE 스펙 방식을 따릅니다. 열 유형, 내용 및 순서는 ODBC의 SQLColumnPrivileges() 결과 집합에 정의된 것과 동일합니다.

열과 연관된 권한이 하나 이상일 경우, 각 권한은 결과 집합에 별도의 행으로 리턴됩니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

SQLColumnPrivileges

- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 33. SQLColumnPrivileges SQLSTATEs

SQLSTATE	설명	설명
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 스트링 또는 버퍼 길이	이름 길이 인수 중 하나의 값이 0 미만이지만 SQL_NTS와 같지 않습니다.
HY010	함수 순서 오류	명령문 핸들에 대한 커서가 열려 있습니다. 이 명령문 핸들에 대한 연결이 없습니다.

제한사항

없음

예

```
/* From the CLI sample TBINFO.C */
/* ... */

    /* call SQLColumnPrivileges */
    printf("\n Call SQLColumnPrivileges for:\n");
    printf(" tbSchema = %s\n", tbSchema);
    printf(" tbName = %s\n", tbName);
    sqlrc = SQLColumnPrivileges( hstmt, NULL, 0,
                                 tbSchema, SQL_NTS,
                                 tbName, SQL_NTS,
                                 colNamePattern, SQL_NTS);
```

참조

- 73 페이지의『SQLColumns - 표에 대한 열 정보 얻기』
- 270 페이지의『SQLTables - 표 정보 얻기』

SQLColumns - 표에 대한 열 정보 얻기

목적

SQLColumns()은 지정된 표에 열 리스트를 리턴합니다. SELECT문에 의해 생성된 결과 집합을 페치하기 위해 사용된 함수와 같은 함수를 사용하여 검색될 수 있는 SQL 결과 집합에 이 정보가 리턴됩니다.

구문

```
SQLRETURN SQLColumns  (SQLHSTMT       hstmt,
                       SQLCHAR        *szCatalogName,
                       SQLSMALLINT    cbCatalogName,
                       SQLCHAR        *szSchemaName,
                       SQLSMALLINT    cbSchemaName,
                       SQLCHAR        *szTableName,
                       SQLSMALLINT    cbTableName,
                       SQLCHAR        *szColumnName,
                       SQLSMALLINT    cbColumnName);
```

함수 인수

표 34. SQLColumns 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들
SQLCHAR *	*szCatalogName*	입력	결과 집합을 규정하기 위한 *패턴* 값이 있는 버퍼. *카탈로그*는 세 부분 표 이름의 첫 번째 부분입니다. 이 포인터는 널(null) 포인터이거나 길이가 0인 스트링이어야 합니다.
SQLSMALLINT	*cbCatalogName*	입력	*szCatalogName*의 길이. 0으로 설정되어야 합니다.
SQLCHAR *	*szSchemaName*	입력	*스키마*명에 의해 결과 집합을 규정하기 위한 *패턴* 값이 있는 버퍼.
SQLSMALLINT	*cbSchemaName*	입력	*szSchemaName*의 길이.
SQLCHAR *	*szTableName*	입력	표 이름에 의해 결과 집합을 규정하기 위한 *패턴* 값이 있는 버퍼.
SQLSMALLINT	*cbTableName*	입력	*szTableName*의 길이.
SQLCHAR *	*szColumnName*	입력	열명(column name)에 의해 결과 집합을 규정하기 위한 *패턴* 값이 있는 버퍼.
SQLSMALLINT	*cbColumnName*	입력	*szColumnName*의 길이.

사용법

이 함수는 표 열이나 표 리스트에 대한 정보를 검색합니다.

SQLColumns()은 표준 결과 집합을 리턴합니다. 74 페이지의 표 35는 결과 집합의 열을 나열합니다. 어플리케이션은 REMARKS 열 이후의 추가 열이 나중에 해제될 때 추가될 수 있다고 예상해야 합니다.

SQLColumns

szCatalogName, szSchemaName, szTableName, szColumnName 인수는 탐색 패턴을 허용합니다. 이탈 문자는 실제 문자가 탐색 패턴에서 사용되도록 와일드카드 문자와 연계하여 지정될 수 있습니다. 이탈 문자는 SQL_ATTR_ESCAPE_CHAR 환경 속성에서 지정됩니다.

이 함수는 SQLDescribeCol() 또는 SQLColAttributes()에 의해 검색되는 결과 집합에 열에 대한 정보를 리턴하지 않습니다. 어플리케이션이 결과 집합에 대한 열 정보를 얻으려면, 효율성을 위해 SQLDescribeCol() 또는 SQLColAttributes()를 항상 호출해야 합니다. SQLColumns()는 복잡한 조회를 시스템 카탈로그에 대해 맵핑하며, 대량의 시스템 자원을 요구할 수 있습니다.

표 35. SQLColumns에 의해 리턴되는 열

열명(column name)	자료 유형	설명
TABLE_CAT	VARCHAR(128)	현재 서버.
TABLE_SCHEM	VARCHAR(128)	TABLE_NAME이 있는 스키마명.
TABLE_NAME	VARCHAR(128)	표나 뷰의 이름.
COLUMN_NAME	VARCHAR(128)	열 ID. 지정된 표나 뷰의 열에 대한 이름.
DATA_TYPE	SMALLINT는 널(null)이 아님	열의 SQL 자료 유형을 식별합니다.
TYPE_NAME	VARCHAR(128)는 널(null)이 아님	DATA_TYPE에 대응하는 자료 유형명을 나타내는 문자 스트링.
LENGTH_PRECISION	INTEGER	DATA TYPE이 대략적인 숫자 자료 유형이면 이 열에는 열의 가수(mantissa) 정밀도의 비트 수가 있습니다. 정확한 숫자 자료 유형인 경우 이 열에는 열에 허용되는 십진 숫자의 총 자릿수가 있습니다. 시간, 시간소인 자료 유형인 경우 이 열에는 소수 초 부분의 정밀도 자릿수가 있습니다. 그 외의 경우 이 열은 널(null)입니다. **주:** 정밀도의 ODBC 정의는 일반적으로 자료 유형을 저장하는 자릿수입니다.
BUFFER_LENGTH	INTEGER	SQL_DEFAULT가 SQLBindCol(), SQLGetData()와 SQLBindParam() 호출에서 지정된 경우, 이 열에서부터 자료를 저장하는 최대 바이트 수.
NUM_SCALE	SMALLINT	열의 소수 자릿수(scale). 소수 자릿수를 적용할 수 없는 자료 유형에 대해서는 널(null)이 리턴됩니다.
NUM_PREC_RADIX	SMALLINT	10, 2 또는 널(null)입니다. DATA_TYPE이 대략적인 숫자 자료 유형인 경우 이 열에는 값 2가 있으며, LENGTH_PRECISION 열에는 이 열에 허용되는 비트 수가 있습니다. DATA_TYPE이 정확한 숫자 자료 유형인 경우 이 열에는 값 10이 있으며 LENGTH_PRECISION과 NUM_SCALE 열에는 이 열에 허용된 십진 자릿수가 있습니다. 숫자 자료 유형의 경우 데이터베이스 관리 시스템(DBMS)은 10이나 2의 NUM_PREC_RADIX를 리턴할 수 있습니다. 기수(radix)를 적용할 수 없는 자료 유형의 경우 널(null)이 리턴됩니다.
NULLABLE	SMALLINT는 널(null)이 아님	열이 널(null)값을 허용하지 않을 경우 SQL_NO_NULLS 열이 널(null)값을 허용할 경우 SQL_NULLABLE

표 35. SQLColumns에 의해 리턴되는 열 (계속)

열명(column name)	자료 유형	설명
REMARKS	VARCHAR(254)	열에 대한 설명 정보가 있을 수 있습니다.
COLUMN_DEF	VARCHAR(254)	열의 디폴트 값. 디폴트 값이 숫자 리터럴이면, 열에는 닫는 작은 따옴표가 없는 숫자 리터럴의 문자 표시가 있습니다. 디폴트 값이 문자 스트링이면 이 열은 작은 따옴표로 묶인 스트링입니다. 디폴트 값이 DATE, TIME, TIMESTAMP 열에 대한 값과 같이 *의사 리터럴*일 경우 이 열에는 닫는 작은 따옴표가 없는 의사 리터럴(예: CURRENT DATE) 키워드가 있습니다. 널(null)이 디폴트 값으로 지정되었으면 이 열은 작은 따옴표로 묶인 단어가 아니라 널(null)을 리턴합니다. 디폴트 값을 절단하지 않고는 나타낼 수 없으면 이 열에는 닫는 작은 따옴표가 없는 TRUNCATED가 있습니다. 디폴트 값이 지정되지 않으면 이 열은 NULL입니다.
DATETIME_CODE	INTEGER	이 열은 현재 널(null)입니다.
CHAR_OCTET_LENGTH	INTEGER	문자 자료 유형 열에 대한 옥텟(octet) 단위의 최대 길이가 있습니다. 1바이트 문자 세트의 경우 이 값은 LENGTH_PRECISION과 같습니다. 기타 자료 유형의 경우 이 값은 널(null)입니다.
ORDINAL_POSITION	INTEGER NOT NULL	표의 열의 원래 위치. 표의 처음 열은 번호가 1입니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 36. SQLColumns SQLSTATEs

SQLSTATE	설명	설명
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 스트링 또는 버퍼 길이	이름 길이 인수 중 하나의 값이 0 미만이지만 SQL_NTS와 같지 않습니다.
HY010	함수 순서 오류	명령문 핸들에 대한 커서가 열려 있습니다. 이 명령문 핸들에 대한 연결이 없습니다.

SQLConnect - 자료 소스 연결

목적

SQLConnect()는 목표 데이터베이스와의 연결을 설정합니다. 어플리케션은 목표 SQL 데이터베이스와 옵션으로 권한 부여 이름, 인증 스트링을 제공해야 합니다.

이 함수를 호출하기 전에 SQLAllocConnect()를 호출해야 합니다.

SQLAllocStmt()를 호출하기 전에 이 함수를 호출해야 합니다.

구문

```
SQLRETURN SQLConnect (SQLHDBC         hdbc,
                      SQLCHAR         *szDSN,
                      SQLSMALLINT     cbDSN,
                      SQLCHAR         *szUID,
                      SQLSMALLINT     cbUID,
                      SQLCHAR         *szAuthStr,
                      SQLSMALLINT     cbAuthStr);
```

함수 인수

표 37. SQLConnect 인수

자료 유형	인수	사용	설명
SQLHDBC	*hdbc*	입력	연결 핸들
SQLCHAR *	*szDSN*	입력	자료 소스: 데이터베이스명 또는 별명
SQLSMALLINT	*cbDSN*	입력	*szDSN* 인수의 내용 길이
SQLCHAR *	*szUID*	입력	권한 부여 이름(사용자 ID)
SQLSMALLINT	*cbUID*	입력	*szUID* 인수의 내용 길이
SQLCHAR *	*szAuthStr*	입력	인증 스트링(암호)
SQLSMALLINT	*cbAuthStr*	입력	*szAuthStr* 인수의 내용 길이

사용법

SQLSetConnectOption()을 사용하여 어플리케이션에서 다양한 연결 특성(옵션)을 정의할 수 있습니다.

SQLConnect()의 입력 길이 인수(*cbDSN, cbUID, cbAuthStr*)는 연관된 자료의 실제 길이로 설정될 수 있습니다. 여기에는 연관된 자료가 널로 종료된다는 것을 표시하기 위한 SQL_NTS나 널 종료 문자가 없습니다.

*szDSN*과 *szUID* 인수 값의 선행 공백이나 후미 공백은 작은 따옴표로 묶여 있지 않으면 처리하기 전에 삭제됩니다.

자료 소스는 연결이 작동되기 위해 시스템에서 이미 정의되어 있어야 합니다. iSeries에서 WRKRDBDIRE(관계형 데이터베이스 디렉토리 항목에 대한 작업) 명령을 사용하여, 어떤 자료 소스가 이미 정의되어 있는지 판별하고 자료 소스를 추가로 정의할 수 있는 옵션을 사용할 수 있습니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 38. SQLConnect SQLSTATEs

SQLSTATE	설명	설명
08001	자료 소스에 연결할 수 없음	드라이버는 자료 소스(서버)와의 연결을 설정할 수 없습니다.
08002	연결 사용 중	지정된 *hdbc*가 자료 소스와의 연결을 설정하기 위해 사용되었으며 연결이 아직 열려 있습니다.
08004	자료 소스의 연결 설정이 거부되었음	자료 소스(서버)의 연결 설정이 거부되었습니다.
28000	유효하지 않은 권한 부여 스펙	*szUID* 인수에 대해 지정된 값 또는 *szAuthStr* 인수에 대해 지정된 값이 자료 소스에 의해 정의된 제한사항을 위반했습니다.
58004	시스템 오류	복구할 수 없는 시스템 오류
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*cbDSN* 인수에 대해 지정된 값은 0 미만이었으나 SQL_NTS와 같지 않았고, *szDSN* 인수가 널(null) 포인터가 아닙니다. *cbUID* 인수에 대해 지정된 값은 0 미만이었으나 SQL_NTS와 같지 않았고, *szUID* 인수가 널(null) 포인터가 아닙니다. *cbAuthStr* 인수에 대해 지정된 값은 0 미만이었으나 SQL_NTS와 같지 않았고, *szAuthStr* 인수가 널(null) 포인터가 아닙니다. *szDSN*, *szUID*, 또는 *szAuthStr* 인수에서 짝이 맞지 않는 큰 따옴표(")가 있습니다.
HY013 *	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.
HY501 *	유효하지 않은 자료 소스명	*szDSN* 인수에 유효하지 않은 자료 소스명이 지정되었습니다.

제한사항

IBM 데이터베이스 관리 시스템(DBMS)에 대한 내포적인 연결(또는 디폴트 데이터베이스) 옵션이 지원되지 않습니다. SQL문이 실행되기 전에 SQLConnect()가 호출되어야 합니다. iSeries는 하나의 작업에서 같은 자료 소스로의 여러 개의 동시 연결을 지원하지 않습니다.

신규 릴리스에서 DB2 UDB CLI를 사용하면 SQLConnect()가 SQL0144 메세지를 받을 수 있습니다. 이는 자료 소스(서버)에 삭제되어야 하는 불필요한 SQL 패키지가 있음을 표시합니다. 이 패키지를 삭제하려면 서버 시스템에서 다음 명령을 실행합니다.

```
DLTSQLPKG SQLPKG(QGPL/QSQCLI*)
```

SQLConnect

다음 SQLConnect()가 새로운 SQL 패키지를 작성할 것입니다.

예

SQLAllocEnv()33 페이지의 『예』를 참조하십시오.

참조

- 29 페이지의 『SQLAllocConnect - 연결 핸들 할당』
- 37 페이지의 『SQLAllocStmt - 명령문 핸들 할당』

SQLCopyDesc - 설명문 복사

목적

SQLCopyDesc()는 소스 핸들과 연관된 자료 구조 필드를 목표 핸들과 연관된 자료 구조로 복사합니다.

ALLOC_TYPE 필드를 제외하고, 목표 핸들과 연관된 자료 구조의 기존 자료가 바뀝니다.

구문

```
SQLRETURN SQLCopyDesc (SQLHDESC     sDesc)
                      (SQLHDESC     tDesc);
```

함수 인수

표 39. SQLCancel 인수

자료 유형	인수	사용	설명
SQLHDESC	*sDesc*	입력	소스 설명자 핸들
SQLHDESC	*tDesc*	입력	목표 설명자 핸들

사용법

GetStmtAttr()를 호출하여 명령문의 매개변수 설명자와 자동으로 생성된 행에 대한 핸들을 얻을 수 있습니다.

리턴 코드

- SQL_SUCCESS
- SQL_INVALID_HANDLE
- SQL_ERROR

SQLDataSources - 자료 소스 리스트 얻기

목적

SQLDataSources()는 사용할 수 있는 목표 데이터베이스 리스트를 한 번에 하나씩 리턴합니다. 데이터베이스는 사용할 수 있도록 카탈로그화되어야 합니다. 카탈로그화에 대한 자세한 정보는 SQLConnect()에 대한 사용 주를 참조하거나 WRKRDBDIRE(관계형 데이터베이스(RDB) 디렉토리 항목에 대한 작업) 명령에 대한 온라인 도움말을 참조하십시오.

연결할 수 있는 데이터베이스를 판별하기 위해 SQLDataSources()는 일반적으로 연결하기 전에 호출되어야 합니다.

구문

```
SQLRETURN   SQLDataSources   (SQLHENV           EnvironmentHandle,
                              SQLSMALLINT       Direction,
                              SQLCHAR           *ServerName,
                              SQLSMALLINT       BufferLength1,
                              SQLSMALLINT       *NameLength1Ptr,
                              SQLCHAR           *Description,
                              SQLSMALLINT       BufferLength2,
                              SQLSMALLINT       *NameLength2Ptr);
```

함수 인수

표 40. SQLDataSources 인수

자료 유형	인수	사용	설명
SQLHENV	*EnvironmentHandle*	입력	환경 핸들
SQLSMALLINT	*Direction*	입력	리스트의 첫 번째 자료 소스명이나 리스트의 다음 이름을 요구하기 위해 어플리케이션에 의해 사용됩니다. *Direction*은 다음 값만을 가질 수 있습니다. • SQL_FETCH_FIRST • SQL_FETCH_NEXT
SQLCHAR *	*ServerName*	출력	검색된 자료 소스명을 갖는 버퍼를 가리키는 포인터.
SQLSMALLINT	*BufferLength1*	입력	*ServerName*이 가리키는 버퍼의 최대 길이. 이 값은 SQL_MAX_DSN_LENGTH + 1 이하여야 합니다.
SQLSMALLINT *	*NameLength1Ptr*	출력	리턴하기 위해 사용할 수 있는 *ServerName*의 최대 바이트 수가 저장될 위치를 가리키는 포인터.
SQLCHAR *	*Description*	출력	자료 소스의 설명이 리턴되는 버퍼를 가리키는 포인터. DB2 UDB CLI는 데이터베이스 관리 시스템(DBMS)에 카탈로그화된 데이터베이스와 연관된 ***Comment*** 필드를 리턴합니다.
SQLSMALLINT	*BufferLength2*	입력	*Description* 버퍼의 최대 길이.
SQLSMALLINT *	*NameLength2Ptr*	출력	자료 소스의 설명에 대해 리턴하기 위해 사용할 수 있는 실제 바이트 수를 이 함수가 리턴하는 위치를 가리키는 포인터.

사용법

*Direction*을 SQL_FETCH_FIRST 또는 SQL_FETCH_NEXT로 설정하여 어플리케이션은 이 함수를 언제나 호출할 수 있습니다.

SQL_FETCH_FIRST가 지정되면 리스트의 첫 번째 데이터베이스가 항상 리턴됩니다.

SQL_FETCH_NEXT는 다음과 같이 지정됩니다.

- SQL_FETCH_FIRST 호출 바로 다음에 지정되면, 리스트의 두 번째 데이터베이스가 리턴됩니다.
- `SQLDataSources()` 호출 전에 지정되면 리스트의 첫 번째 데이터베이스가 리턴됩니다.
- 리스트에 더 이상의 데이터베이스가 없으면 SQL_NO_DATA_FOUND가 리턴됩니다. 함수가 다시 호출되면 첫 번째 데이터베이스가 리턴됩니다.
- 그 외의 경우에는 리스트의 다음 데이터베이스가 리턴됩니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

오류 조건

표 41. SQLDataSources SQLSTATEs

SQLSTATE	설명	설명
01004	자료가 절단됨	*ServerName* 인수에 리턴된 자료 소스명이 *BufferLength1* 인수에 지정된 값보다 깁니다. *NameLength1Ptr* 인수에는 전체 자료 소스명의 길이가 있습니다. (함수가 SQL_SUCCESS_WITH_INFO를 리턴합니다.) *Description* 인수에 리턴된 자료 소스명이 *BufferLength2* 인수에 지정된 값보다 깁니다. *NameLength2Ptr* 인수에는 전체 자료 소스 설명의 길이가 있습니다. (함수가 SQL_SUCCESS_WITH_INFO를 리턴합니다.)
58004	예기치 않은 시스템 실패	복구할 수 없는 시스템 오류
HY000	일반 오류	특정한 SQLSTATE가 없거나 특정한 SQLSTATE가 정의되지 않은 오류가 발생했습니다. *ErrorMsg* 인수에서 `SQLError()`에 의해 리턴된 오류 메세지가 오류와 그 원인을 설명합니다.
HY001	메모리 할당 실패	DB2 UDB CLI가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*ServerName*, *NameLength1Ptr*, *Description* 또는 *NameLength2Ptr* 인수가 널(null) 포인터입니다. 방향에 대해 유효하지 않은 값
HY013	예기치 않은 메모리 처리 오류	DB2 UDB CLI가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

표 41. SQLDataSources SQLSTATEs (계속)

SQLSTATE	설명	설명
HY103	방향 옵션이 범위 밖의 값임	*Direction* 인수에 대해 지정된 값이 SQL_FETCH_FIRST 또는 SQL_FETCH_NEXT와 같지 않습니다.

권한

없음

예

```
/* From CLI sample datasour.c */
/* ... */

#include <stdio.h>
#include <stdlib.h>
#include <sqlcli1.h>
#include "samputil.h"          /* Header file for CLI sample code */

/* ... */

/******************************************************************
** main
** - initialize
** - terminate
******************************************************************/
int main() {

    SQLHANDLE henv ;
    SQLRETURN rc ;
    SQLCHAR source[SQL_MAX_DSN_LENGTH + 1], description[255] ;
    SQLSMALLINT buffl, desl ;

/* ... */

    /* allocate an environment handle */
    rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv ) ;
    if ( rc != SQL_SUCCESS ) return( terminate( henv, rc ) ) ;

    /* list the available data sources (servers) */
    printf( "The following data sources are available:\n" ) ;
    printf( "ALIAS NAME                      Comment(Description)\n" ) ;
    printf( "------------------------------------------------\n" ) ;

    while ( ( rc = SQLDataSources( henv,
                                   SQL_FETCH_NEXT,
                                   source,
                                   SQL_MAX_DSN_LENGTH + 1,
                                   &buffl,
                                   description,
                                   255,
                                   &desl
                                 )
            ) != SQL_NO_DATA_FOUND
          ) printf( "%-30s  %s\n", source, description ) ;
```

```
    rc = SQLFreeHandle( SQL_HANDLE_ENV,  henv ) ;
    if ( rc != SQL_SUCCESS ) return( terminate( henv, rc ) ) ;

    return( SQL_SUCCESS ) ;


}
```

참조

없음

SQLDescribeCol - 열 속성 설명

목적

SQLDescribeCol()은 SELECT문에 의해 생성된 결과 집합의 표시된 열에 대한 결과 설명자 정보(열명(column name), 유형, 정밀도)를 리턴합니다.

어플리케이션이 설명자 정보의 속성 하나만을 필요로 하는 경우, SQLColAttributes() 함수가 SQLDescribeCol() 대신 사용될 수 있습니다. 자세한 정보는 65 페이지의 『SQLColAttributes - 열 속성』을 참조하십시오.

이 함수를 호출하기 전에 SQLPrepare() 또는 SQLExecDirect()를 호출해야 합니다.

SQLBindCol()을 호출하기 전에 일반적으로 이 함수(또는 SQLColAttributes())를 호출해야 합니다.

구문

```
SQLRETURN SQLDescribeCol (SQLHSTMT       hstmt,
                          SQLSMALLINT    icol,
                          SQLCHAR        *szColName,
                          SQLSMALLINT    cbColNameMax,
                          SQLSMALLINT    *pcbColName,
                          SQLSMALLINT    *pfSqlType,
                          SQLINTEGER     *pcbColDef,
                          SQLSMALLINT    *pibScale,
                          SQLSMALLINT    *pfNullable);
```

함수 인수

표 42. SQLDescribeCol 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLSMALLINT	*icol*	입력	설명될 열 번호
SQLCHAR *	*szColName*	출력	열명(column name) 버퍼를 가리키는 포인터
SQLSMALLINT	*cbColNameMax*	입력	*szColName* 버퍼 크기
SQLSMALLINT *	*pcbColName*	출력	*szColName* 인수에 대해 리턴하기 위해 사용할 수 있는 바이트. *pcbColName*이 *cbColNameMax* 이상이면 열명(column name)(*szColName*)이 *cbColNameMax - 1*바이트로 절단됩니다.
SQLSMALLINT *	*pfSqlType*	출력	열의 SQL 자료 유형
SQLINTEGER *	*pcbColDef*	출력	데이터베이스에 정의된 열의 정밀도 *fSqlType*이 그래픽 SQL 자료 유형을 나타낼 경우에는 이 변수가 열이 가질 수 있는 최대 2바이트 *문자* 수를 나타냅니다.
SQLSMALLINT *	*pibScale*	출력	데이터베이스에 정의된 열의 소수 자릿수(scale)(SQL_DECIMAL, SQL_NUMERIC, SQL_TIMESTAMP에만 적용됨)

표 42. SQLDescribeCol 인수 (계속)

자료 유형	인수	사용	설명
SQLSMALLINT *	*pfNullable*	출력	이 열에 대해 널(null)이 허용되는 지를 나타냅니다. • SQL_NO_NULLS • SQL_NULLABLE

사용법

열은 번호에 의해 식별되며 왼쪽에서 오른쪽으로, 1부터 번호가 지정되고, 다른 순서로도 서술할 수 있습니다.

유효한 포인터와 버퍼 영역을 *szColName* 인수에 대해 사용할 수 있어야 합니다. 널(null) 포인터가 남아있는 포인터 인수에 대해 지정된 경우, DB2 UDB CLI는 어플리케이션에 정보가 필요없는 것으로 가정하고 아무것도 리턴시키지 않습니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

`SQLDescribeCol()`이 SQL_ERROR이나 SQL_SUCCESS_WITH_INFO 중 하나를 리턴하는 경우, `SQLError()` 함수를 호출하여 다음의 SQLSTATE 중 하나를 얻을 수 있습니다.

표 43. SQLDescribeCol SQLSTATEs

SQLSTATE	설명	설명
01004	자료가 절단됨	*szColName* 인수에 리턴된 열명(column name)이 *cbColNameMax* 인수에 지정된 값보다 깁니다. *pcbColName* 인수에는 전체 열명(column name) 길이가 있습니다. (함수가 SQL_SUCCESS_WITH_INFO를 리턴합니다.)
07005 *****	SELECT문이 아님	*hstmt*와 연관된 명령문이 결과 집합을 리턴하지 않았습니다. 설명하는 열이 없습니다. (결과 집합에 행이 있는지 판별하려면 먼저 `SQLNumResultCols()`를 호출하십시오.)
07009	유효하지 않은 열 번호	*icol* 인수에 대해 지정된 값이 1 미만입니다. *icol* 인수에 대해 지정된 값이 결과 집합의 열 번호보다 큽니다.
40003 *****	명령문 완료 상태를 알 수 없음	함수가 처리를 완료하기 전에 CLI와 자료 소스와의 통신 링크가 끊어졌습니다.
58004	시스템 오류	복구할 수 없는 시스템 오류
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.

표 43. SQLDescribeCol SQLSTATEs (계속)

SQLSTATE	설명	설명
HY009	유효하지 않은 인수 값	*cbColNameMax* 인수에 지정된 길이가 1 미만입니다. *szColName* 또는 *pcbColName* 인수가 널(null) 포인터입니다.
HY010	함수 순서 오류	*hstmt*에 대해 SQLPrepare() 또는 SQLExecDirect()를 호출하기 전에 이 함수를 호출합니다.
HY013 *	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.
HYC00	드라이버가 지원되지 않음	*icol* 열의 SQL 자료 유형이 DB2 UDB CLI에 의해 인식되지 않습니다.

예

다음 예의 전체 리스트를 보려면 300 페이지의 『예: 대화식 SQL 및 그에 해당되는 DB2 UDB CLI 함수 호출』을 참조하십시오.

```
/*********************************************************************
** file = typical.c
...
/*********************************************************************
** display_results
**
**  - for each column
**      - get column name
**      - bind column
**  - display column headings
**  - fetch each row
**      - if value truncated, build error message
**      - if column null, set value to "NULL"
**      - display row
**      - print truncation message
**  - free local storage
*********************************************************************/
display_results(SQLHSTMT hstmt,
                SQLSMALLINT nresultcols)
{
SQLCHAR         colname[32];
SQLSMALLINT     coltype;
SQLSMALLINT     colnamelen;
SQLSMALLINT     nullable;
SQLINTEGER      collen[MAXCOLS];
SQLSMALLINT     scale;
SQLINTEGER      outlen[MAXCOLS];
SQLCHAR *       data[MAXCOLS];
SQLCHAR         errmsg[256];
SQLRETURN   rc;
SQLINTEGER      i;
SQLINTEGER      displaysize;

        for (i = 0; i < nresultcols; i++)
    {
        SQLDescribeCol (hstmt, i+1, colname, sizeof (colname),
        &colnamelen, &coltype, &collen[i], &scale, &nullable);
```

```
        /* get display lenght for column */
        SQLColAttributes (hstmt, i+1, SQL_COLUMN_DISPLAY_SIZE, NULL, 0,
            NULL, &displaysize);

        /* set column length to max of display length, and column name
           length.  Plus one byte for null terminator       */
        collen[i] = max(displaysize, strlen((char *) colname) ) + 1;


        /* allocate memory to bind column                              */
        data[i] = (SQLCHAR *) malloc (collen[i]);

        /* bind columns to program vars, converting all types to CHAR */
        SQLBindCol (hstmt, i+1, SQL_CHAR, data[i], collen[i],
&outlen[i]);
    }
    printf("\n");

    /* display result rows                                             */
    while ((rc = SQLFetch (hstmt)) != SQL_NO_DATA_FOUND)
    {
        errmsg[0] = '\0';
        for (i = 0; i < nresultcols; i++)
        {
            /* Build a truncation message for any columns truncated */
            if (outlen[i] >= collen[i])
            {    sprintf ((char *) errmsg + strlen ((char *) errmsg),
                          "%d chars truncated, col %d\n",
                           outlen[i]-collen[i]+1, i+1);
            }
            if (outlen[i] == SQL_NULL_DATA)
    else
        } /* for all columns in this row  */

        printf ("\n%s", errmsg);  /* print any truncation messages    */
    } /* while rows to fetch */

    /* free data buffers                                               */
        for (i = 0; i < nresultcols; i++)
    {
        free (data[i]);
    }

}/* end display_results
```

참조

- 65 페이지의『SQLColAttributes - 열 속성』
- 102 페이지의『SQLExecDirect - 명령문 직접 실행』
- 206 페이지의『SQLNumResultCols - 결과 열의 수 얻기』
- 212 페이지의『SQLPrepare - 명령문 준비』

SQLDescribeParam - 매개변수 마커의 설명 리턴

목적

SQLDescribeParam()은 준비된 SQL문과 연관된 매개변수 마커의 설명을 리턴합니다. 이 정보는 IPD(implementation parameter descriptor)의 필드에도 있습니다.

구문

```
SQLRETURN   SQLDescribeParam (SQLHSTMT          StatementHandle,
                              SQLSMALLINT       ParameterNumber,
                              SQLSMALLINT       *DataTypePtr,
                              SQLINTEGER        *ParameterSizePtr,
                              SQLSMALLINT       *DecimalDigitsPtr,
                              SQLSMALLINT       *NullablePtr);
```

함수 인수

표 44. SQLDescribeParam 인수

자료 유형	인수	사용	설명
SQLHSTMT	StatementHandle	입력	명령문 핸들.
SQLSMALLINT	ParameterNumber	입력	1부터 시작하여 순서가 지정된 매개변수 마커 번호.
SQLSMALLINT *	DataTypePtr	출력	매개변수의 SQL 자료 유형을 리턴할 버퍼를 가리키는 포인터.
SQLINTEGER *	ParameterSizePtr	출력	자료 소스에 의해 정의된대로 대응하는 매개변수 마커의 표현식 또는 열 크기를 리턴할 버퍼를 가리키는 포인터.
SQLSMALLINT *	DecimalDigitsPtr	출력	자료 소스에 의해 정의된대로 대응하는 매개변수의 표현식 또는 열의 십진 자릿수를 리턴할 버퍼를 가리키는 포인터.
SQLSMALLINT *	NullablePtr	출력	매개변수가 널(null) 값을 허용하는 지를 표시하는 값을 리턴할 버퍼를 가리키는 포인터. 이 값은 IPD의 SQL_DESC_NULLABLE 필드로부터 읽습니다. 다음 중 하나입니다. • SQL_NO_NULLS - 매개변수가 널(null) 값을 허용하지 않습니다 (디폴트 값입니다). • SQL_NULLABLE - 매개변수가 널(null) 값을 허용합니다. • SQL_NULLABLE_UNKNOWN - 매개변수의 널(null) 값 허용을 판별할 수 없습니다.

사용법

매개변수 마커는 SQL문에 나타나는 순서대로 1부터 증가하는 매개변수 순서로 번호가 지정됩니다.

SQLDescribeParam()은 SQL문의 매개변수 유형(입력, 출력, 또는 입력과 출력 모두)을 리턴하지 않습니다. 프로시듀어 호출의 경우를 제외하고 SQL문의 모든 매개변수는 입력 매개변수입니다. 프로시듀어 호출에서의 매개변수 유형을 판별하기 위해 어플리케이션은 SQLProcedureColumns()을 호출합니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

오류 조건

표 45. SQLDescribeParam SQLSTATEs

SQLSTATE	설명	설명
01000	경고	정보용 메세지(함수가 SQL_SUCCESS_WITH_INFO를 리턴합니다.)
07009	유효하지 않은 설명자 색인	*ParameterNumber* 인수에 대해 지정된 값이 1 미만입니다. *ParameterNumber* 인수에 대해 지정된 값이 연관된 SQL문의 매개변수 수보다 많습니다. 매개변수 마커가 비DML문의 일부입니다. 매개변수 마커가 SELECT 리스트의 일부입니다.
08S01	통신 링크 실패	함수가 처리를 완료하기 전에 연결되었던 자료 소스와 DB2 UDB CLI 사이의 통신 링크가 끊어졌습니다.
21S01	삽입 값 리스트가 열 리스트와 일치하지 않습니다.	INSERT문의 매개변수 수가 명령문에 이름이 지정된 표의 열 수와 일치하지 않습니다.
HY000	일반 오류	
HY001	메모리 할당 실패	DB2 UDB CLI가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY008	조작 취소됨	
HY009	유효하지 않은 인수 값	*DataTypePtr*, *ParameterSizePtr*, *DecimalDigitsPtr* 또는 *NullablePtr* 인수가 널(null) 포인터입니다.
HY010	함수 순서 오류	*StatementHandle*에 대해 `SQLPrepare()` 또는 `SQLExecDirect()`를 호출하기 전에 함수가 호출됩니다.
HY013	예기치 않은 메모리 처리 오류	기초가 되는 메모리 오브젝트에 액세스하지 못하여 함수 호출을 처리할 수 없습니다. 사용할 수 있는 메모리가 부족할 가능성이 있습니다.

제한사항

없음

SQLDescribeParam

참조

- 50 페이지의『SQLBindParam - 매개변수 마커에 버퍼 바인드』
- 63 페이지의『SQLCancel - 명령문 취소』
- 104 페이지의『SQLExecute - 명령문 실행』
- 212 페이지의『SQLPrepare - 명령문 준비』

SQLDisconnect - 자료 소스 단절

목적

SQLDisconnect()는 데이터베이스 연결 핸들과 연관된 연결을 닫습니다.

이 함수를 호출한 후에 다른 데이터베이스에 연결하기 위해 SQLConnect()를 호출하거나 SQLFreeConnect()를 호출합니다.

구문

```
SQLRETURN SQLDisconnect (SQLHDBC    hdbc);
```

함수 인수

표 46. SQLDisconnect 인수

자료 유형	인수	사용	설명
SQLHDBC	*hdbc*	입력	연결 핸들

사용법

연결과 연관된 명령문 핸들을 모두 해제하기 전에 어플리케이션이 SQLDisconnect를 호출하면, DB2 UDB CLI는 데이터베이스를 성공적으로 단절한 후에 핸들을 해제합니다.

SQL_SUCCESS_WITH_INFO가 리턴되면, 이는 데이터베이스 단절이 성공적이지만 추가 오류 또는 특정 구현 프로그램 정보가 있음을 의미합니다. 예를 들면 다음과 같습니다.

- 단절 후에 정리하는 과정에서 문제가 발생한 경우
- 어플리케이션과는 독립적으로 발생한 이벤트(통신 실패와 같은)로 인해 현재는 연결되지 않은 경우

성공적인 SQLDisconnect() 호출 후에 또 다른 SQLConnect() 요구를 하기 위해 어플리케이션은 *hdbc*를 다시 사용할 수 있습니다.

*hdbc*가 DUOW 2단계 확약(commit) 연결의 일부라면 즉시 단절되지 않을 수 있습니다. 실제 단절은 분배된 트랜잭션에 대해 발행된 다음 번 확약시에 일어납니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 47. SQLDisconnect SQLSTATEs

SQLSTATE	설명	설명
01002	단절 오류	단절하는 동안 오류가 발생했습니다. 그러나 성공적으로 단절되었습니다. (함수가 SQL_SUCCESS_WITH_INFO를 리턴합니다.)
08003	연결이 열려 있지 않음	*hdbc* 인수에 지정된 연결이 열리지 않았습니다.
25000	유효하지 않은 트랜잭션 상태	*hdbc* 인수에 의해 지정된 연결에 처리 중인 트랜잭션이 있습니다. 트랜잭션을 사용 중이며 연결을 단절할 수 없습니다.
58004	시스템 오류	복구할 수 없는 시스템 오류
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY013 *****	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

예

SQLAllocEnv()33 페이지의 『예』를 참조하십시오.

참조

- 29 페이지의 『SQLAllocConnect - 연결 핸들 할당』
- 76 페이지의 『SQLConnect - 자료 소스 연결』
- 273 페이지의 『SQLTransact - 트랜잭션 관리』

SQLDriverConnect - 자료 소스에 (확장) 연결

목적

SQLDriverConnect()는 SQLConnect() 대신 사용할 수 있습니다. 두 함수 모두 목표 데이터베이스와의 연결을 설정하지만 SQLDriverConnect()는 자료 소스명, 사용자 ID, 암호를 판별하기 위해 연결 스트링을 사용합니다. 두 함수는 같으며 호환성을 위해 둘 다 지원됩니다.

구문

```
SQLRETURN   SQLDriverConnect (SQLHDBC           ConnectionHandle,
                              SQLHWND           WindowHandle,
                              SQLCHAR           *InConnectionString,
                              SQLSMALLINT       StringLength1,
                              SQLCHAR           *OutConnectionString,
                              SQLSMALLINT       BufferLength,
                              SQLSMALLINT       *StringLength2Ptr,
                              SQLSMALLINT       DriverCompletion);
```

함수 인수

표 48. SQLDriverConnect 인수

자료 유형	인수	사용	설명
SQLHDBC	*ConnectionHandle*	입력	연결 핸들
SQLHWND	*hwindow*	입력	창 핸들(플랫폼 종속적): Windows에서 이 핸들은 상위 Windows 핸들입니다. OS/2에서는 상위 PM 창 핸들입니다. AIX®에서는 상위 MOTIF 위젯 창 핸들입니다. iSeries에서는 무시됩니다.
SQLCHAR *	*InConnectionString*	입력	전체, 일부 또는 빈(널(null) 포인터) 연결 스트링(아래 설명과 구문 참조).
SQLSMALLINT	*StringLength1*	입력	*InConnectionString* 길이
SQLCHAR *	*OutConnectionString*	출력	전체 연결 스트링에 대한 버퍼를 가리키는 포인터. 연결이 성공적으로 설정되면 이 버퍼에는 전체 연결 스트링이 있습니다.
SQLSMALLINT	*BufferLength*	입력	*OutConnectionString*가 가리키는 버퍼의 최대 크기.
SQLSMALLINT *	*StringLength2Ptr*	출력	*OutConnectionString* 버퍼에 리턴되는 사용할 수 있는 바이트 수를 가리키는 포인터. *StringLength2Ptr* 값이 *BufferLength* 이상이면, *OutConnectionString*의 전체 연결 스트링이 *BufferLength* - 1바이트로 절단됩니다.
SQLSMALLINT	*DriverCompletion*	입력	DB2 UDB CLI가 사용자로부터 추가 정보를 받기 위한 메세지를 표시하는 시기를 지정합니다. 가능한 값은 다음과 같습니다. • SQL_DRIVER_COMPLETE • SQL_DRIVER_COMPLETE_REQUIRED • SQL_DRIVER_NOPROMPT

사용법

이 연결 스트링은 연결을 완료하기 위해 필요한 하나 이상의 값을 전달하기 위해 사용됩니다. 연결 스트링의 내용과 *DriverCompletion*의 값은 연결 방법을 결정합니다.

Connection String Syntax

위의 각 키워드는 다음과 같은 속성을 갖습니다.

DSN 자료 소스명. 데이터베이스명 또는 별명. *DriverCompletion*이 SQL_DRIVER_NOPROMPT이면 자료 소스명이 필요합니다.

UID 권한 부여 이름(사용자 ID)

PWD

권한 부여 이름에 대응하는 암호. 사용자 ID에 대한 암호가 없으면 아무 것도 지정되지 않습니다 (PWD=;).

iSeries에는 현재 DB2 UDB CLI 정의 키워드가 없습니다.

DriverCompletion 값은 유효하지만 모두 같이 작동합니다. 연결 스트링에 있는 정보를 사용하여 연결이 시도됩니다. 충분한 정보가 없으면, SQL_ERROR가 리턴됩니다.

일단 연결되면 전체 연결 스트링이 리턴됩니다. 주어진 사용자 ID에 대해 같은 데이터베이스와 복수 연결을 설정할 필요가 있는 어플리케이션은 이 출력 연결 스트링을 저장해야 합니다. 이 스트링은 이후의 `SQLDriverConnect()` 호출에서 입력 연결 스트링으로 사용될 수 있습니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NO_DATA_FOUND
- SQL_INVALID_HANDLE
- SQL_ERROR

오류 조건

76 페이지의 『SQLConnect - 자료 소스 연결』에 의해 생성된 모든 진단도 역시 여기로 리턴됩니다. 다음 표는 리턴될 수 있는 추가 진단을 표시합니다.

표 49. SQLDriverConnect SQLSTATEs

SQLSTATE	설명	설명
01004	자료가 절단됨	*szConnstrOut* 버퍼가 전체 연결 스트링을 저장할 정도로 크지 않습니다. *StringLength2Ptr* 인수에는 리턴하기 위해 사용할 수 있는 연결 스트링의 실제 길이가 있습니다. (함수가 SQL_SUCCESS_WITH_INFO를 리턴합니다.)
01S00	유효하지 않은 연결 스트링 속성	유효하지 않은 키워드 또는 속성 값이 입력 연결 스트링에 지정되었지만, 다음 처리를 통해 자료 소스 연결이 이루어졌습니다. • 인식할 수 없는 키워드를 무시합니다. • 유효하지 않은 속성 값을 무시하고 디폴트 값을 대신 사용합니다. (함수가 SQL_SUCCESS_WITH_INFO를 리턴합니다.)
HY009	유효하지 않은 인수 값	*InConnectionString*, *OutConnectionString* 또는 *StringLength2PTR* 인수가 널(null) 포인터입니다. DriverCompletion 인수가 1이 아닙니다.
HY090	유효하지 않은 스트링 또는 버퍼 길이	*StringLength1*에 대해 지정된 값은 0 미만이었으나 SQL_NTS와 같지 않습니다. *BufferLength*에 대해 지정된 값은 0 미만입니다.
HY110	유효하지 않은 드라이버 종료	*fCompletion* 인수에 대해 지정된 값은 유효한 값 중 하나와 같지 않습니다.

제한사항

없음

예

```
/* From CLI sample drivrcon.c */
/* ... */
/*********************************************************************
**   drv_connect - Prompt for connect options and connect             **
*********************************************************************/

int
drv_connect(SQLHENV henv,
            SQLHDBC * hdbc,
            SQLCHAR con_type)
{
    SQLRETURN       rc;
    SQLCHAR         server[SQL_MAX_DSN_LENGTH + 1];
    SQLCHAR         uid[MAX_UID_LENGTH + 1];
    SQLCHAR         pwd[MAX_PWD_LENGTH + 1];
    SQLCHAR         con_str[255];
    SQLCHAR         buffer[255];
    SQLSMALLINT     outlen;
```

```
|
|     printf("Enter Server Name:\n");
|     gets((char *) server);
|     printf("Enter User Name:\n");
|     gets((char *) uid);
|     printf("Enter Password Name:\n");
|     gets((char *) pwd);
|
|     /* Allocate a connection handle */
|     SQLAllocHandle( SQL_HANDLE_DBC,
|                         henv,
|                         hdbc
|                       );
|     CHECK_HANDLE( SQL_HANDLE_DBC, *hdbc, rc);
|
|     sprintf((char *)con_str, "DSN=%s;UID=%s;PWD=%s;",
|             server, uid, pwd);
|
|     rc = SQLDriverConnect(*hdbc,
|             (SQLHWND) NULL,
|             con_str,
|             SQL_NTS,
|             buffer, 255, &outlen,
|             SQL_DRIVER_NOPROMPT);
|     if (rc != SQL_SUCCESS) {
|         printf("Error while connecting to database, RC= %ld\n", rc);
|         CHECK_HANDLE( SQL_NULL_HENV, *hdbc, rc);
|         return (SQL_ERROR);
|     } else {
|         printf("Successful Connect\n");
|         return (SQL_SUCCESS);
|     }
| }
```

참조

- 76 페이지의『SQLConnect - 자료 소스 연결』

SQLEndTran - 트랜잭션 확약 또는 롤백

목적

SQLEndTran()은 연결의 현재 트랜잭션을 확약하거나 롤백합니다.

연결 시간이나 이전 SQLEndTran() 호출(둘 중 가장 최근의 것) 이후의 연결에서 수행된 데이터베이스의 모든 변경이 확약되거나 롤백됩니다.

트랜잭션이 연결에서 사용중이면 어플리케이션은 데이터베이스를 단절하기 전에 SQLEndTran()을 호출해야 합니다.

구문

```
SQLRETURN SQLEndTran (SQLSMALLINT    hType,
                      SQLINTEGER     handle,
                      SQLSMALLINT    fType);
```

함수 인수

표 50. SQLEndTran 인수

자료 유형	인수	사용	설명
SQLSMALLINT	*hType*	입력	핸들 유형, 반드시 SQL_HANDLE_ENV나 SQL_HANDLE_DBC가 있어야 합니다.
SQLINTEGER	*handle*	입력	COMMIT 또는 ROLLBACK을 수행할 때 사용할 핸들
SQLSMALLINT	*fType*	입력	트랜잭션에 대해 원하는 조치. 이 인수에 대한 값은 다음 중 하나여야 합니다. • SQL_COMMIT • SQL_ROLLBACK • SQL_COMMIT_HOLD • SQL_ROLLBACK_HOLD • SQL_SAVEPOINT_NAME_ROLLBACK • SQL_SAVEPOINT_NAME_RELEASE

사용법

트랜잭션을 SQL_COMMIT 또는 SQL_ROLLBACK으로 완료하면 다음과 같은 효과가 있습니다.

- SQLEndTran()을 호출한 후에도 명령문 핸들이 유효합니다.
- 커서명, 바인드 매개변수, 열 바인딩이 트랜잭션보다 오래 살아 있습니다.
- 열린 커서가 닫히고 검색을 지연 중인 결과 집합이 삭제됩니다.

트랜잭션을 SQL_COMMIT_HOLD 또는 SQL_ROLLBACK_HOLD로 종료하면 데이터베이스 변경을 확약하거나 롤백하지만 커서가 닫히지 않습니다.

SQLEndTran

연결에서 현재 사용 중인 트랜잭션이 없으면, SQLEndTran()를 호출해도 데이터베이스 서버에 아무런 영향을 주지 않으며 SQL_SUCCESS를 리턴합니다.

연결 손실로 인해 COMMIT 또는 ROLLBACK을 실행하는 동안 SQLEndTran()이 실패할 수 있습니다. 이 경우에는 어플리케이션이 COMMIT 또는 ROLLBACK이 처리되었는지 판별할 수 없으므로 데이터베이스 관리자에게 도움을 요청해야 합니다. 트랜잭션 기록부와 기타 트랜잭션 관리 타스크에 대한 자세한 정보는 데이터베이스 관리 시스템(DBMS) 제품 정보를 참조하십시오.

SQL_SAVEPOINT_NAME_ROLLBACK나 SQL_SAVEPOINT_NAME_RELEASE를 사용할 때는 SQLSetConnectAttr을 사용하여 savepoint 이름을 설정했어야 합니다.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 51. SQLEndTran SQLSTATEs

SQLSTATE	설명	설명
08003	연결이 열려 있지 않음	*hdbc*가 연결 상태가 아닙니다.
08007	트랜잭션이 실행되는 동안 연결 실패	*hdbc*와 연관된 연결이 함수가 실행되는 동안 실패하였으며 실패하기 전에 요구된 COMMIT 또는 ROLLBACK이 처리되었는지 판별할 수 없습니다.
58004	시스템 오류	복구할 수 없는 시스템 오류.
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY010	함수 순서 오류	SQL_SAVEPOINT_NAME_ROLLBACK나 SQL_SAVEPOINT_NAME_RELEASE가 사용되었지만 SQL_ATTR_SAVEPOINT_NAME 속성에 대하여 SQLSetConnectAttr()를 호출하여 savepoint 이름을 작성하지 않았습니다.
HY012	유효하지 않은 트랜잭션 조작 상태	*fType* 인수에 대해 지정된 값이 SQL_COMMIT나 SQL_ROLLBACK이 아닙니다.
HY013 *	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

SQLError - 오류 정보 검색

목적

SQLError()는 특정 명령문, 연결 또는 환경 핸들에 대해 가장 최근에 호출된 DB2 UDB CLI 함수와 연관된 진단 정보를 리턴합니다.

정보는 표준화된 SQLSTATE, 원래의 오류 코드, 텍스트 메세지로 구성됩니다. 자세한 정보는 18 페이지의 『DB2 UDB CLI 어플리케이션에서의 진단』을 참조하십시오.

다른 함수 호출에서 SQL_ERROR 또는 SQL_SUCCESS_WITH_INFO 리턴 코드를 수신한 후 SQLError()를 호출합니다.

구문

```
SQLRETURN SQLError (SQLHENV       henv,
                    SQLHDBC       hdbc,
                    SQLHSTMT      hstmt,
                    SQLCHAR       *szSqlState,
                    SQLINTEGER    *pfNativeError,
                    SQLCHAR       *szErrorMsg,
                    SQLSMALLINT   cbErrorMsgMax,
                    SQLSMALLINT   *pcbErrorMsg);
```

함수 인수

표 52. SQLError 인수

자료 유형	인수	사용	설명
SQLHENV	*henv*	입력	환경 핸들 환경과 연관된 진단 정보를 얻으려면 유효한 환경 핸들을 전달하십시오. *hdbc*와 *hstmt*를 각각 SQL_NULL_HDBC와 SQL_NULL_HSTMT로 설정하십시오.
SQLHDBC	*hdbc*	입력	데이터베이스 연결 핸들 연결과 연관된 진단 정보를 얻으려면, 유효한 데이터베이스 연결 핸들을 전달하고 *hstmt*를 SQL_NULL_HSTMT로 설정하십시오. *henv* 인수는 무시됩니다.
SQLHSTMT	*hstmt*	입력	명령문 핸들. 명령문과 연관된 진단 정보를 얻으려면 유효한 명령문 핸들을 전달하십시오. *henv*와 *hdbc* 인수는 무시됩니다.
SQLCHAR *	*szSqlState*	출력	널 문자로 종료된 5자 스트링의 SQLSTATE. 처음 2자는 오류 클래스를 표시하며, 다음 3자는 하위 클래스를 표시합니다. 이 값은 X/Open SQL CAE 스펙과 ODBC 스펙에 정의된 SQLSTATE 값에 직접 대응하며 앞에 IBM 고유의 제품별 SQLSTATE 값이 붙습니다.

SQLError

표 52. SQLError 인수 (계속)

자료 유형	인수	사용	설명
SQLINTEGER *	*pfNativeError*	출력	원래의 오류 코드. DB2 UDB CLI에서 *pfNativeError* 인수에는 데이터베이스 관리 시스템(DBMS)에 의해 리턴된 SQLCODE 값이 있습니다. 오류가 DBMS가 아니라 DB2 UDB CLI에 의해 생성된 경우 이 필드는 -99999로 설정됩니다.
SQLCHAR *	*szErrorMsg*	출력	구현 프로그램에서 정의된 메세지 텍스트를 포함하는 버퍼를 가리키는 포인터. DB2 UDB CLI에서는 데이터베이스 관리 시스템(DBMS)이 생성한 메세지만 리턴됩니다. DB2 UDB CLI는 문제를 설명하는 어떤 메세지 텍스트도 리턴하지 않습니다.
SQLSMALLINT	*cbErrorMsgMax*	입력	*szErrorMsg* 버퍼의 최대(즉, 할당된) 길이. 권장되는 할당 길이는 SQL_MAX_MESSAGE_LENGTH + 1입니다.
SQLSMALLINT *	*pcbErrorMsg*	출력	*szErrorMsg* 버퍼에 리턴되는 사용할 수 있는 전체 바이트 수를 가리키는 포인터.

사용법

SQLSTATE는 X/OPEN SQL CAE와 X/Open SQL CLI 스냅샷에 의해 정의된 것과 같으나 앞에 IBM 고유의 제품별 SQLSTATE 값이 붙습니다.

다음과 연관된 진단 정보를 얻을 수 있습니다.

- 환경과 연관된 진단 정보를 얻으려면 유효한 환경 핸들을 전달하십시오. *hdbc*와 *hstmt*를 각각 SQL_NULL_HDBC와 SQL_NULL_HSTMT로 설정하십시오.
- 연결과 연관된 진단 정보를 얻으려면, 유효한 데이터베이스 연결 핸들을 전달하고 *hstmt*를 SQL_NULL_HSTMT로 설정하십시오. *henv* 인수는 무시됩니다.
- 명령문과 연관된 진단 정보를 얻으려면 유효한 명령문 핸들을 전달하십시오. *henv*와 *hdbc* 인수는 무시됩니다.

한 DB2 UDB CLI 함수에 의해 생성된 진단 정보가 SQLError() 이외의 함수가 같은 핸들로 호출되기 전에 검색되지 않으면, 이전 함수 호출에 대한 정보가 손실됩니다. 두 번째 DB2 UDB CLI 함수 호출에 대해 생성된 진단 정보가 있는지에 관계없이 항상 해당됩니다.

오류 메세지가 절단되지 않게 하려면 버퍼 길이를 SQL_MAX_MESSAGE_LENGTH + 1로 선언합니다. 메세지 텍스트는 이보다 길어질 수 없습니다.

리턴 코드

- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND
- SQL_SUCCESS

진단

SQLError()가 자신에 대한 진단 정보를 생성하지 않으므로 SQLSTATE가 정의되지 않습니다. szSqlState, pfNativeError, szErrorMsg 또는 pcbErrorMsg 인수가 널(null) 포인터인 경우에는 SQL_ERROR이 리턴됩니다.

예

다음 예의 전체 리스트를 보려면 300 페이지의 『예: 대화식 SQL 및 그에 해당되는 DB2 UDB CLI 함수 호출』을 참조하십시오.

```
/******************************************************************************
** file = typical.c
******************************************************************************/
int print_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt)
{
SQLCHAR     buffer[SQL_MAX_MESSAGE_LENGTH + 1];
SQLCHAR     sqlstate[SQL_SQLSTATE_SIZE + 1];
SQLINTEGER  sqlcode;
SQLSMALLINT length;


    while ( SQLError(henv, hdbc, hstmt, sqlstate, &sqlcode, buffer,
                     SQL_MAX_MESSAGE_LENGTH + 1, &length) == SQL_SUCCESS )
    {
        printf("\n **** ERROR *****\n");
        printf("         SQLSTATE: %s\n", sqlstate);
        printf("Native Error Code: %ld\n", sqlcode);
        printf("%s \n", buffer);
    };
    return (0);

}
```

SQLExecDirect - 명령문 직접 실행

목적

SQLExecDirect는 지정된 SQL문을 직접 실행합니다. 명령문은 한 번만 실행될 수 있습니다. 또한 연결된 데이터베이스 서버는 명령문을 준비할 수 있어야 합니다.

구문

```
SQLRETURN SQLExecDirect (SQLHSTMT      hstmt,
                         SQLCHAR       *szSqlStr,
                         SQLINTEGER    cbSqlStr);
```

함수 인수

표 53. SQLExecDirect 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들. *hstmt*와 연관된 열린 커서가 없어야 합니다. 자세한 정보는 129 페이지의 『SQLFreeStmt - 명령문 핸들을 해제(또는 재설정)』를 참조하십시오.
SQLCHAR *	*szSqlStr*	입력	SQL문 스트링. 연결된 데이터베이스 서버는 명령문을 준비할 수 있어야 합니다.
SQLINTEGER	*cbSqlStr*	입력	*szSqlStr* 인수의 내용 길이. 길이는 명령문의 정확한 길이 또는 명령문이 널로 종료될 경우 SQL_NTS로 설정되어야 합니다.

사용법

SQL문은 COMMIT 또는 ROLLBACK문일 수 없습니다. 대신, COMMIT 또는 ROLLBACK을 발행하기 위해 SQLTransact()가 호출되어야 합니다. 지원되는 SQL문에 대한 자세한 정보는 5 페이지의 표 1을 참조하십시오.

SQL문 스트링에는 매개변수 마커가 있을 수 있습니다. 매개변수 마커는 "?" 문자로 표시되며 SQLExecDirect()가 호출될 때 어플리케이션 변수 값이 대체되어야 할 명령문의 위치를 지정합니다. SQLBindParam()은 어플리케이션 변수를 각 매개변수 마커에 바인드(연관)시켜서, 자료가 전송될 때 자료 변환이 수행되어야 하는지를 표시합니다. 모든 매개변수는 SQLExecDirect()를 호출하기 전에 바인드되어야 합니다.

SQL문이 SELECT문이면, SQLExecDirect()는 커서명을 생성하고 커서를 엽니다. 어플리케이션이 SQLSetCursorName()을 사용하여 커서명을 명령문 핸들과 연관시키면 DB2 UDB CLI는 어플리케이션이 생성한 커서명을 내부적으로 생성된 커서명과 연관시킵니다.

SELECT문에 의해 생성된 결과 집합에서 행을 검색하기 위해 SQLExecDirect()가 성공적으로 리턴된 후 SQLFetch()를 호출합니다.

SQL문이 위치지정된 DELETE 또는 위치지정된 UPDATE문일 경우 명령문이 참조한 커서는 행에 위치해야 합니다. 뿐만 아니라 SQL문은 같은 연결 핸들 아래의 분리된 명령문 핸들에 정의되어야 합니다.

명령문 핸들에 열린 커서가 있어야 합니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

SQL문이 탐색된 UPDATE 또는 탐색된 DELETE문이고 탐색 조건을 만족하는 행이 없으면 SQL_NO_DATA_FOUND가 리턴됩니다.

진단

표 54. SQLExecDirect SQLSTATEs

SQLSTATE	설명	설명
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*szSqlStr* 인수가 널(null) 포인터입니다. *cbSqlStr* 인수가 1 미만이었으나 SQL_NTS와 같지 않습니다.
HY010	함수 순서 오류	이 명령문 핸들에 대한 열린 커서가 있거나 연결되지 않았습니다.
HY013 *	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

주: 명령문 실행 시 데이터베이스 관리 시스템(DBMS)에 의해 생성되는 다른 SQLSTATE 값이 많이 있습니다.

예

SQLFetch()110 페이지의 『예』를 참조하십시오.

참조

- 104 페이지의 『SQLExecute - 명령문 실행』
- 109 페이지의 『SQLFetch - 다음 행 페치』
- 254 페이지의 『SQLSetParam - 매개변수 설정』

SQLExecute - 명령문 실행

목적

SQLExecute()는 SQLPrepare()를 사용하여 성공적으로 준비된 명령문을 한 번 또는 여러 번 실행합니다. 명령문은 SQLBindParam()에 의해 매개변수 마커에 바인드된 어플리케이션 변수의 현재 값을 사용하여 실행됩니다.

구문

```
SQLRETURN SQLExecute (SQLHSTMT     hstmt);
```

함수 인수

표 55. SQLExecute 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들. hstmt와 연관된 열린 커서가 없어야 합니다. 자세한 정보는 129 페이지의 『SQLFreeStmt - 명령문 핸들을 해제(또는 재설정)』를 참조하십시오.

사용법

SQL문 스트링에는 매개변수 마커가 있을 수 있습니다. 매개변수 마커는 "?" 문자로 표시되며 SQLExecute()가 호출될 때 어플리케이션 변수 값이 대체되어야 할 명령문의 위치를 지정합니다. SQLBindParam()은 어플리케이션 변수를 각 매개변수 마커에 바인드(연관)시켜서 자료가 전송될 때 자료 변환이 수행되어야 하는지를 표시합니다. 모든 매개변수는 SQLExecute()를 호출하기 전에 바인드되어야 합니다.

일단 어플리케이션이 SQLExecute() 호출의 결과를 처리하면, 어플리케이션 변수에서 새로운(또는 같은) 값을 사용하여 명령문을 다시 실행할 수 있습니다.

SQLExecDirect()에 의해 실행된 명령문은 SQLExecute()를 호출하여 다시 실행할 수 없습니다. SQLPrepare()를 먼저 호출해야 합니다.

준비된 SQL문이 SELECT문이면, SQLExecute()는 커서명을 생성하고 커서를 엽니다. 어플리케이션이 SQLSetCursorName()을 사용하여 커서명을 명령문 핸들과 연관시키면 DB2 UDB CLI는 어플리케이션이 생성한 커서명을 내부적으로 생성된 커서명과 연관시킵니다.

SELECT문을 한 번 이상 실행하려면 어플리케이션은 SQL_CLOSE 옵션과 함께 SQLFreeStmt()를 호출하여 커서를 닫아야 합니다. SQLExecute()를 호출할 때 명령문 핸들에 열린 커서가 없어야 합니다.

SELECT문에 의해 생성된 결과 집합에서 행을 검색하려면, SQLExecute()가 성공적으로 리턴된 후 SQLFetch()를 호출하십시오.

SQL문이 위치지정된 DELETE 또는 위치지정된 UPDATE문이면 SQLExecute()가 호출될 때 명령문에 의해 참조된 커서는 행에 위치해야 하며 같은 연결 핸들 아래의 분리된 명령문 핸들에 정의되어야 합니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

SQL문이 탐색된 UPDATE 또는 탐색된 DELETE문이고 탐색 조건을 만족하는 행이 없으면 SQL_NO_DATA_FOUND가 리턴됩니다.

진단

SQLExecute()에 대한 SQLSTATE에는 **HY**009를 제외하고 다음 표의 SQLSTATE를 추가한 SQLExecDirect()(103 페이지의 표 54 참조)에 대한 모든 내용이 있습니다.

표 56. SQLExecute SQLSTATEs

SQLSTATE	설명	설명
HY010	함수 순서 오류	지정된 *hstmt*는 준비된 상태가 아닙니다. SQLExecute() 가 먼저 SQLPrepare를 호출하지 않고 호출됩니다.

주: 명령문 실행 시 데이터베이스 관리 시스템(DBMS)에 의해 생성되는 다른 SQLSTATE 값이 많이 있습니다.

예

Refer to the SQLPrepare()213 페이지의 『예』를 참조하십시오.

참조

- 102 페이지의 『SQLExecDirect - 명령문 직접 실행』
- 39 페이지의 『SQLBindCol - 어플리케이션 변수에 열 바인드』
- 212 페이지의 『SQLPrepare - 명령문 준비』
- 109 페이지의 『SQLFetch - 다음 행 페치』
- 254 페이지의 『SQLSetParam - 매개변수 설정』

SQLExtendedFetch - 행 배열 페치

목적

SQLExtendedFetch()는 각각의 바운드시킨 열에 배열 형식의 여러 행(*행 집합*이라고 함)을 포함하는 자료 블록을 리턴하여 SQLFetch()를 확장합니다. 행 집합의 *크기*는 SQLSetStmtAttr() 호출의 SQL_ROWSET_SIZE 속성에 의해 결정됩니다.

한 번에 자료의 한 행씩 페치하려면 어플리케이션이 SQLFetch()를 호출해야 합니다.

구문

```
SQLRETURN   SQLExtendedFetch (SQLHSTMT          StatementHandle,
                              SQLSMALLINT       FetchOrientation,
                              SQLINTEGER        FetchOffset,
                              SQLINTEGER        *RowCountPtr,
                              SQLSMALLINT       *RowStatusArray);
```

함수 인수

표 57. SQLExtendedFetch 인수

자료 유형	인수	사용	설명
SQLHSTMT	StatementHandle	입력	명령문 핸들.
SQLSMALLINT	FetchOrientation	입력	페치 방향. 가능한 값은 116 페이지의 표 62를 참조하십시오.
SQLINTEGER	FetchOffset	입력	상대 위치에 대한 행 오프셋
SQLINTEGER *	RowCountPtr	출력	실제로 페치된 행 수. 처리하는 동안 오류가 발생하면, *RowCountPtr*은 오류가 발생한 행 이전 행의 (행 집합에서의) 순서적 위치를 가리킵니다. 첫 번째 행 검색시 오류가 발생하면 *RowCountPtr*는 값 0을 가리킵니다.
SQLSMALLINT *	RowStatusArray	출력	상태 값 배열. 요소의 수는 행 집합의 수 (SQL_ROWSET_SIZE 속성에 의해 정의됨)와 같아야 합니다. 페치된 각 행에 대한 상태 값이 리턴됩니다. • SQL_ROW_SUCCESS 페치된 행 수가 상태 배열의 요소 수 미만이면(즉, 행 집합 크기 미만), 나머지 상태 요소가 SQL_ROW_NOROW로 설정됩니다. 행이 페치가 시작된 이후에 갱신되거나 삭제되었는지를 DB2 UDB CLI는 감지할 수 없습니다. 그러므로 다음 ODBC 정의 상태 값이 보고되지 않습니다. • SQL_ROW_DELETED • SQL_ROW_UPDATED

사용법

SQLExtendedFetch()가 사용되어 행 집합의 배열 페치를 수행합니다. SQL_ROWSET_SIZE 속성과 함께 SQLSetStmtAttr()를 호출하여 어플리케이션은 배열 *크기*를 지정합니다.

SQLExtendedFetch()가 처음으로 호출되기 전에 커서는 첫 행 앞에 놓입니다. SQLExtendedFetch()를 호출한 후 커서는 지금 검색된 행 집합의 마지막 행 요소에 대응하는 결과 집합의 행에 위치합니다.

SQLBindCol() 함수를 통해 바인드된 결과 집합의 열에 대해 DB2 UDB CLI는 바인드된 열에 대한 자료를 필요한대로 변환하여 이 열에 바인드된 위치에 저장합니다. 결과 집합은 행 방식 형태로 바인드되어야 합니다. 이는 첫 번째 행의 모든 열에 대한 값이 인접해 있고 두 번째 행부터의 값이 바로 뒤에 나와야 합니다. 또한 인디케이터 변수가 사용되면, 모두 하나의 인접한 기억장치 위치에 리턴될 것입니다.

이 프로시듀어를 사용하여 여러 행을 검색할 경우, 모든 열이 바인드되어야 하며 기억장치는 인접해 있어야 합니다. 이 함수를 사용하여 SQL 프로시듀어 결과 집합에서 행을 검색할 경우, SQL_FETCH_NEXT 방향만이 지원됩니다. 사용자는 SQL_ROWSET_SIZE에 지정된 행 수에 대해 충분한 기억장치를 할당해야 합니다.

SQL_FETCH_NEXT 이외의 방향을 사용하려면 커서는 SQLExtendedFetch()에 대한 스크롤할 수 있는 커서여야합니다. SQL_ATTR_CURSOR_SCROLLABLE 속성을 설정하는데 대한 정보은 255 페이지의 『SQLSetStmtAttr - 명령문 속성 설정』을 참조하십시오.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

오류 조건

표 58. SQLExtendedFetch SQLSTATEs

SQLSTATE	설명	설명
HY009	유효하지 않은 인수 값	*RowCountPtr* 또는 *RowStatusArray* 인수 값은 널(null) 포인터입니다. FetchOrientation 인수에 지정된 값이 인식되지 않습니다.
HY010	함수 순서 오류	SQLFetch()가 호출된 후에 그리고 SQL_CLOSE 옵션과 함께 SQLFreeStmt()를 호출하기 전에 StatementHandle에 대해 SQLExtendedFetch()가 호출됩니다. *StatementHandle*에 대해 SQLPrepare() 또는 SQLExecDirect()를 호출하기 전에 함수가 호출됩니다. 함수가 실행시 자료(SQLParamData(), SQLPutData()) 조작중에 호출됩니다.

SQLExtendedFetch

제한사항

없음

참조

- 39 페이지의『SQLBindCol - 어플리케이션 변수에 열 바인드』
- 104 페이지의『SQLExecute - 명령문 실행』
- 102 페이지의『SQLExecDirect - 명령문 직접 실행』
- 109 페이지의『SQLFetch - 다음 행 페치』

SQLFetch - 다음 행 페치

목적

SQLFetch()는 결과 집합의 다음 행으로 커서를 진행시키고 바인드된 열을 검색합니다.

SQLFetch()를 사용하여 사용자가 SQLBindCol()으로 지정한 변수로 자료를 직접 수신하거나 SQLGetData()를 호출하여 페치 후에 열을 개별적으로 수신할 수 있습니다. 열이 바인드될 때 변환이 지정된 경우 SQLFetch()가 호출될 때 자료 변환도 발생합니다.

구문

```
SQLRETURN SQLFetch (SQLHSTMT    hstmt);
```

함수 인수

표 59. SQLFetch 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.

사용법

SQLFetch()는 *hstmt*에서 가장 최근에 실행된 명령문이 SELECT문인 경우에만 호출될 수 있습니다.

SQLBindCol() 과 바인드된 어플리케이션 변수 갯수는 결과 집합의 열 수를 초과해서는 안됩니다. 초과할 경우 SQLFetch()가 실패합니다.

SQLBindCol()이 호출되었지만 바인드된 열이 없으면, SQLFetch()는 어플리케이션에 자료를 리턴하지는 않지만 커서를 진행시킵니다. 이 경우 SQLGetData()가 호출되어 모든 열을 개별적으로 가져올 수 있습니다. SQLFetch()가 다음 행으로 커서를 진행시키면 바인드되지 않은 열의 자료가 삭제됩니다.

바인드된 변수가 SQLFetch()에 의해 리턴된 자료를 저장할만큼 크지 않으면, 자료가 절단됩니다. 문자 자료가 절단되면, SQL_SUCCESS_WITH_INFO가 리턴되고 절단을 표시하는 SQLSTATE가 생성됩니다. SQLBindCol()의 연기된 출력 인수 *pcbValue*에는 서버에서 검색한 열 자료의 실제 길이가 있습니다. 어플리케이션은 출력 길이와 입력 길이(SQLBindCol()의 *pcbValue*와 *cbValueMax* 인수)를 비교하여 절단된 문자 열을 판별해야 합니다.

절단이 소수점 오른쪽 자릿수와 관련되면 숫자 자료 유형 절단은 보고되지 않습니다. 소수점 왼쪽에서 절단이 발생하면 오류가 리턴됩니다(진단 관련 섹션 참조).

그래픽 자료 유형의 절단은 문자 자료 유형과 똑같이 취급됩니다. *rgbValue* 버퍼는 SQLBindCol()에 지정된 *cbValueMax*와 같거나 적은 2바이트의 배수에 가까운 곳으로 채워진다는 점만 다릅니다. DB2 UDB CLI와 어플리케이션 간에 전송되는 그래픽 자료는 널로 종료되지 않습니다.

SQLFetch

결과 집합의 모든 행이 검색되었거나 나머지 행이 필요하지 않으면, SQLFreeStmt()가 호출되어 커서를 닫고 나머지 자료와 연관된 자원을 삭제해야 합니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

결과 집합에 행이 없거나 이전의 SQLFetch() 호출에서 결과 집합의 모든 행을 페치한 경우 SQL_NO_DATA_FOUND가 리턴됩니다.

진단

표 60. SQLFetch SQLSTATEs

SQLSTATE	설명	설명
01004	자료가 절단됨	하나 이상의 열에 대해 리턴된 자료가 절단되었습니다. 스트링 값은 오른쪽으로 절단됩니다. (오류가 없으면 SQL_SUCCESS_WITH_INFO가 리턴됩니다.)
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY010	함수 순서 오류	지정된 *hstmt*는 실행된 상태가 아닙니다. 먼저 SQLExecute 또는 SQLExecDirect를 호출하지 않고 함수가 호출됩니다.
HY013 *	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

예

코드 예와 관련된 정보는 viii 페이지의 『코드 면책사항 관련 정보』를 참조하십시오.

```
/*****************************************************************************
** file = fetch.c
**
** Example of executing an SQL statement.
** SQLBindCol & SQLFetch is used to retrive data from the result set
** directly into application storage.
**
** Functions used:
**
**        SQLAllocConnect      SQLFreeConnect
**        SQLAllocEnv          SQLFreeEnv
**        SQLAllocStmt         SQLFreeStmt
**        SQLConnect           SQLDisconnect
**
**        SQLBindCol           SQLFetch
**        SQLTransact          SQLExecDirect
**        SQLError
```

```
**
******************************************************************************/

#include <stdio.h>
#include <string.h>
#include "sqlcli.h"

#define MAX_STMT_LEN 255

int initialize(SQLHENV *henv,
               SQLHDBC *hdbc);

int terminate(SQLHENV henv,
              SQLHDBC hdbc);

int print_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt);

int check_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt,
                 SQLRETURN  frc);

/******************************************************************
** main
** - initialize
** - terminate
******************************************************************/
int main()
{
    SQLHENV     henv;
    SQLHDBC     hdbc;
    SQLCHAR     sqlstmt[MAX_STMT_LEN + 1]="";
    SQLRETURN   rc;

    rc = initialize(&henv, &hdbc);
    if (rc == SQL_ERROR) return(terminate(henv, hdbc));


    {SQLHSTMT   hstmt;
     SQLCHAR    sqlstmt[]="SELECT deptname, location from org where division = 'Eastern'";
     SQLCHAR    deptname[15],
                location[14];
     SQLINTEGER rlength;

        rc = SQLAllocStmt(hdbc, &hstmt);
        if (rc != SQL_SUCCESS )
           check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

        rc = SQLExecDirect(hstmt, sqlstmt, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt, rc);

        rc = SQLBindCol(hstmt, 1, SQL_CHAR, (SQLPOINTER) deptname, 15,
                        &rlength);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt, rc);
```

```
        rc = SQLBindCol(hstmt, 2, SQL_CHAR, (SQLPOINTER) location, 14,
                        &rlength);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt, rc);

        printf("Departments in Eastern division:\n");
        printf("DEPTNAME       Location\n");
        printf("-------------- -------------\n");

        while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS)
        {
            printf("%-14.14s %-13.13s \n", deptname, location);
        }
        if (rc != SQL_NO_DATA_FOUND )
           check_error (henv, hdbc, hstmt, rc);

        rc = SQLFreeStmt(hstmt, SQL_DROP);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, SQL_NULL_HSTMT, rc);
    }


    rc = SQLTransact(henv, hdbc, SQL_COMMIT);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

    terminate(henv, hdbc);
    return (0);
}/* end main */

/*****************************************************************
** initialize
**  - allocate environment handle
**  - allocate connection handle
**  - prompt for server, user id, & password
**  - connect to server
*****************************************************************/

int initialize(SQLHENV *henv,
               SQLHDBC *hdbc)
{
SQLCHAR     server[SQL_MAX_DSN_LENGTH],
            uid[30],
            pwd[30];
SQLRETURN   rc;

    rc = SQLAllocEnv (henv);         /* allocate an environment handle    */
    if (rc != SQL_SUCCESS )
        check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);

    rc = SQLAllocConnect (*henv, hdbc);  /* allocate a connection handle   */
    if (rc != SQL_SUCCESS )
        check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);

    printf("Enter Server Name:\n");
    gets(server);
    printf("Enter User Name:\n");
    gets(uid);
```

```
    printf("Enter Password Name:\n");
    gets(pwd);

    if (uid[0] == '\0')
    {   rc = SQLConnect (*hdbc, server, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
    }
    else
    {   rc = SQLConnect (*hdbc, server, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
    }

    return(SQL_SUCCESS);
}/* end initialize */

/*****************************************************************
** terminate
**  - disconnect
**  - free connection handle
**  - free environment handle
*****************************************************************/
int terminate(SQLHENV henv,
              SQLHDBC hdbc)
{
SQLRETURN   rc;

    rc = SQLDisconnect (hdbc);              /* disconnect from database  */
    if (rc != SQL_SUCCESS )
        print_error (henv, hdbc, SQL_NULL_HSTMT);
    rc = SQLFreeConnect (hdbc);             /* free connection handle    */
    if (rc != SQL_SUCCESS )
        print_error (henv, hdbc, SQL_NULL_HSTMT);
    rc = SQLFreeEnv (henv);                 /* free environment handle   */
    if (rc != SQL_SUCCESS )
        print_error (henv, hdbc, SQL_NULL_HSTMT);

    return(rc);
}/* end terminate */

/*****************************************************************
**  - print_error   - call SQLError(), display SQLSTATE and message
*****************************************************************/

int print_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt)
{
SQLCHAR     buffer[SQL_MAX_MESSAGE_LENGTH + 1];
SQLCHAR     sqlstate[SQL_SQLSTATE_SIZE + 1];
SQLINTEGER  sqlcode;
SQLSMALLINT length;


    while ( SQLError(henv, hdbc, hstmt, sqlstate, &sqlcode, buffer,
                     SQL_MAX_MESSAGE_LENGTH + 1, &length) == SQL_SUCCESS )
    {
```

```
        printf("\n **** ERROR *****\n");
        printf("         SQLSTATE: %s\n", sqlstate);
        printf("Native Error Code: %ld\n", sqlcode);
        printf("%s \n", buffer);
    };

    return ( SQL_ERROR);
} /* end print_error */

/*********************************************************************
**  - check_error   - call print_error(), checks severity of return code
*********************************************************************/
int check_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt,
                 SQLRETURN  frc)
{
SQLRETURN   rc;

    print_error(henv, hdbc, hstmt);

    switch (frc){
    case SQL_SUCCESS : break;
    case SQL_ERROR :
    case SQL_INVALID_HANDLE:
        printf("\n ** FATAL ERROR, Attempting to rollback transaction **\n");
        rc = SQLTransact(henv, hdbc, SQL_ROLLBACK);
        if (rc != SQL_SUCCESS)
            printf("Rollback Failed, Exiting application\n");
    else
            printf("Rollback Successful, Exiting application\n");
        terminate(henv, hdbc);
        exit(frc);
        break;
    case SQL_SUCCESS_WITH_INFO :
        printf("\n ** Warning Message, application continuing\n");
        break;
    case SQL_NO_DATA_FOUND :
        printf("\n ** No Data Found ** \n");
        break;
    default :
        printf("\n ** Invalid Return Code ** \n");
        printf(" ** Attempting to rollback transaction **\n");
        SQLTransact(henv, hdbc, SQL_ROLLBACK);
        terminate(henv, hdbc);
        exit(frc);
        break;
    }
    return(SQL_SUCCESS);

} /* end check_error */
```

참조

- 39 페이지의『SQLBindCol - 어플리케이션 변수에 열 바인드』
- 104 페이지의『SQLExecute - 명령문 실행』

- 102 페이지의『SQLExecDirect - 명령문 직접 실행』
- 132 페이지의『SQLGetCol - 결과 집합 행의 한 열을 검색』
- 116 페이지의『SQLFetchScroll - 스크롤할 수 있는 커서에서 페치』

SQLFetchScroll - 스크롤할 수 있는 커서에서 페치

목적

SQLFetchScroll()은 요구된 방향을 기준으로 커서를 위치시키고 바인드된 열을 검색합니다.

SQLFetchScroll()을 사용하여 SQLBindCol()으로 지정한 변수로 직접 자료를 수신하거나 SQLGetData()를 호출하여 페치한 후에 열을 개별적으로 수신할 수 있습니다. 열이 바인드될 때 변환이 지정된 경우 SQLFetchScroll()이 호출될 때 자료 변환도 발생합니다.

구문

```
SQLRETURN SQLFetchScroll (SQLHSTMT    hstmt,
                          SQLSMALLINT fOrient,
                          SQLINTEGER  fOffset);
```

함수 인수

표 61. SQLFetchScroll 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLSMALLINT	*fOrient*	입력	페치 방향. 가능한 값은 표 62를 참조하십시오.
SQLINTEGER	*fOffset*	입력	상대 위치에 대한 행 오프셋

사용법

*hstmt*에서 가장 최근에 실행된 명령문이 SELECT문인 경우에만 SQLFetchScroll()이 호출됩니다.

자료가 검색되기 전에 *fOrient* 매개변수가 커서를 위치시키는 것을 제외하고는 SQLFetchScroll()은 SQLFetch()같이 작동됩니다. SQL_FETCH_NEXT 이외의 방향을 사용하려면 커서는 SQLFetchScroll()에 대한 스크롤할 수 있는 커서여야 합니다. SQL_ATTR_CURSOR_SCROLLABLE 속성을 설정하는데 대한 정보은 255 페이지의 『SQLSetStmtAttr - 명령문 속성 설정』을 참조하십시오.

이 함수를 사용하여 SQL 프로시듀어 결과 집합에서 행을 검색할 경우, SQL_FETCH_NEXT 방향만이 지원됩니다.

표 62. 명령문 속성

fOrient	설명
SQL_FETCH_NEXT	현재 커서 위치 다음 행으로 이동합니다.
SQL_FETCH_FIRST	결과 집합의 첫 행으로 이동합니다.
SQL_FETCH_LAST	결과 집합의 마지막 행으로 이동합니다.
SQL_FETCH_PRIOR	현재 커서 위치 이전 행으로 이동합니다.

표 62. 명령문 속성 (계속)

fOrient	설명
SQL_FETCH_RELATIVE	*fOffset*의 값에 따라 커서 이동이 달라집니다. • 양수인 경우, 해당 행 수만큼 커서를 진행시킵니다. • 음수인 경우, 해당 행 수만큼 커서를 뒤로 이동합니다. • 0인 경우, 커서를 움직이지 않습니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

진단

표 63. SQLFetchScroll SQLSTATEs

SQLSTATE	설명	설명
01004	자료가 절단됨	하나 이상의 열에 대해 리턴된 자료가 절단되었습니다. 스트링 값은 오른쪽으로 절단됩니다. (오류가 없으면 SQL_SUCCESS_WITH_INFO가 리턴됩니다.)
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	유효하지 않은 방향
HY010	함수 순서 오류	지정된 *hstmt*는 실행된 상태가 아닙니다. 먼저 SQLExecute 또는 SQLExecDirect를 호출하지 않고 함수가 호출됩니다.
HY013 *	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

참조

- 39 페이지의『SQLBindCol - 어플리케이션 변수에 열 바인드』
- 104 페이지의『SQLExecute - 명령문 실행』
- 102 페이지의『SQLExecDirect - 명령문 직접 실행』
- 132 페이지의『SQLGetCol - 결과 집합 행의 한 열을 검색』
- 109 페이지의『SQLFetch - 다음 행 페치』

SQLForeignKeys - 외부 키 열 리스트 얻기

목적

SQLForeignKeys()는 지정된 표에 대한 외부 키 정보를 리턴합니다. 조회에 의해 생성된 결과를 검색하기 위해 사용된 함수와 같은 함수를 사용하여 처리될 수 있는 SQL 결과 집합에 이 정보가 리턴됩니다.

구문

```
SQLRETURN   SQLForeignKeys   (SQLHSTMT          StatementHandle,
                              SQLCHAR           *PKCatalogName,
                              SQLSMALLINT       NameLength1,
                              SQLCHAR           *PKSchemaName,
                              SQLSMALLINT       NameLength2,
                              SQLCHAR           *PKTableName,
                              SQLSMALLINT       NameLength3,
                              SQLCHAR           *FKCatalogName,
                              SQLSMALLINT       NameLength4,
                              SQLCHAR           *FKSchemaName,
                              SQLSMALLINT       NameLength5,
                              SQLCHAR           *FKTableName,
                              SQLSMALLINT       NameLength6);
```

함수 인수

표 64. SQLForeignKeys 인수

자료 유형	인수	사용	설명
SQLHSTMT	StatementHandle	입력	명령문 핸들.
SQLCHAR *	PKCatalogName	입력	1차 키 표의 카탈로그 규정자. 이 포인터는 널(null) 포인터이거나 길이가 0인 스트링이어야 합니다.
SQLSMALLINT	NameLength1	입력	*PKCatalogName*의 길이. 0으로 설정되어야 합니다.
SQLCHAR *	PKSchemaName	입력	1차 키 표의 스키마 규정자
SQLSMALLINT	NameLength2	입력	*PKSchemaName*의 길이
SQLCHAR *	PKTableName	입력	1차 키가 들어 있는 표
SQLSMALLINT	NameLength3	입력	*PKTableName*의 길이
SQLCHAR *	FKCatalogName	입력	외부 키가 들어 있는 표의 카탈로그 규정자 이 포인터는 널(null) 포인터이거나 길이가 0인 스트링이어야 합니다.
SQLSMALLINT	NameLength4	입력	*FKCatalogName*의 길이. 0으로 설정되어야 합니다.
SQLCHAR *	FKSchemaName	입력	외부 키가 들어 있는 표의 스키마 규정자
SQLSMALLINT	NameLength5	입력	*FKSchemaName*의 길이
SQLCHAR *	FKTableName	입력	외부 키가 들어 있는 표 이름
SQLSMALLINT	NameLength6	입력	*FKTableName*의 길이

사용법

*PKTableName*에 표 이름이 들어 있고 *FKTableName*이 빈 스트링이면, SQLForeignKeys()는 지정된 표의 1차 키와 이를 참조하는(다른 표의) 모든 외부 키가 들어 있는 결과 집합을 리턴합니다.

*FKTableName*에 표 이름이 들어 있고, *PKTableName*이 빈 스트링이면, SQLForeignKeys()는 지정된 표의 모든 외부 키와 이 외부 키가 참조하는(다른 표의) 1차 키가 들어 있는 결과 집합을 리턴합니다.

*PKTableName*과 *FKTableName* 모두에 표 이름이 들어 있으면, SQLForeignKeys()는 *PKTableName*에 지정된 표의 1차 키를 참조하는 *FKTableName*에 지정된 표의 포린 키를 리턴합니다. 이 키는 하나여야 합니다.

표 이름과 연관된 스키마 규정자 인수가 지정되지 않으면, 스키마명의 디폴트 값은 현재 연결에 대해 유효한 이름입니다.

SQLForeignKeys() 호출에 의해 생성된 결과 집합의 열이 표 65에 나열됩니다. 1차 키와 연관된 외부 키가 요구되면, 결과 집합은 FKTABLE_CAT, FKTABLE_SCHEM, FKTABLE_NAME, ORDINAL_POSITION에 의해 순서가 정해집니다. 외부 키와 연관된 1차 키가 요구되면, 결과 집합은 PKTABLE_CAT, PKTABLE_SCHEM, PKTABLE_NAME, ORDINAL_POSITION에 의해 순서가 정해집니다.

새로운 열이 추가되고 기존 열의 이름이 이후의 릴리스에서 변경될 수 있지만 현재 열의 위치는 변경되지 않습니다.

표 65. SQLForeignKeys에 의해 리턴되는 열

열 번호/이름	자료 유형	설명
1 PKTABLE_CAT	VARCHAR(128)	현재 서버.
2 PKTABLE_SCHEM	VARCHAR(128)	PKTABLE_NAME이 들어 있는 스키마명.
3 PKTABLE_NAME	VARCHAR(128)는 널(null)이 아님	1차 키가 들어 있는 표 이름.
4 PKCOLUMN_NAME	VARCHAR(128)는 널(null)이 아님	1차 키 열명
5 FKTABLE_CAT	VARCHAR(128)	현재 서버.
6 FKTABLE_SCHEM	VARCHAR(128)	FKTABLE_NAME이 들어 있는 스키마명.
7 FKTABLE_NAME	VARCHAR(128)는 널(null)이 아님	외부 키가 들어 있는 표 이름.
8 FKCOLUMN_NAME	VARCHAR(128)는 널(null)이 아님	외부 키 열명
9 ORDINAL_POSITION	SMALLINT는 널(null)이 아님	1부터 시작하는 키에서 열의 순서적 위치.
10 UPDATE_RULE	SMALLINT	SQL 작업이 UPDATE일 때 외부 키에 적용되는 조치 • SQL_RESTRICT • SQL_NO_ACTION IBM DB2 DBMS에 대한 갱신 규칙은 RESTRICT 또는 SQL_NO_ACTION입니다. 그러나 ODBC 어플리케이션이 IBM RDBMS가 아닌 데이터베이스 관리 시스템(DBMS)에 연결될 때 다음과 같은 UPDATE_RULE이 적용될 수 있습니다. • SQL_CASCADE • SQL_SET_NULL

표 65. SQLForeignKeys에 의해 리턴되는 열 (계속)

열 번호/이름	자료 유형	설명
11 DELETE_RULE	SMALLINT	SQL 작업이 DELETE일 때 외부 키에 적용되는 조치 • SQL_CASCADE • SQL_NO_ACTION • SQL_RESTRICT • SQL_SET_DEFAULT • SQL_SET_NULL
12 FK_NAME	VARCHAR(128)	외부 키 ID. 자료 소스에 적용할 수 없는 경우 널(null).
13 PK_NAME	VARCHAR(128)	1차 키 ID. 자료 소스에 적용할 수 없는 경우 널(null).

주: DB2 UDB CLI에 의해 사용된 열명(column name)은 X/Open CLI CAE 스펙 방식을 준수합니다. 열 유형, 내용과 순서는 ODBC의 `SQLForeignKeys()` 결과 집합에 대해 정의된 값과 같습니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 66. SQLForeignKeys SQLSTATEs

SQLSTATE	설명	설명
24000	유효하지 않은 커서 상태	커서가 명령문 핸들에서 이미 열렸습니다.
40003 **08**S01	통신 링크 실패	함수가 완료되기 전에 어플리케이션과 자료 소스와의 통신 링크가 끊어졌습니다.
HY001	메모리 할당 실패	DB2 UDB CLI가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*PKTableName*과 *FKTableName* 인수가 모두 널(null) 포인터입니다.
HY010	함수 순서 오류	
HY014	더 이상의 핸들이 없음	DB2 UDB CLI는 내부 자원 때문에 핸들을 할당할 수 없습니다.
HY090	유효하지 않은 스트링 또는 버퍼 길이	이름 길이 인수 중 하나의 값이 0 미만이지만 SQL_NTS와 같지 않습니다. 표 길이나 소유자명이 서버에 의해 지원되는 최대 길이보다 큽니다. 164 페이지의 『SQLGetInfo - 일반 정보 얻기』를 참조하십시오.
HYC00	드라이버가 지원되지 않음	DB2 UDB CLI는 *catalog*를 표 이름에 대한 규정자로 지원하지 않습니다.
HYT00	시간 종료가 만료됨	

제한사항

없음

예

```
/* From CLI sample browser.c */
/* ... */
SQLRETURN list_foreign_keys( SQLHANDLE hstmt,
                             SQLCHAR * schema,
                             SQLCHAR * tablename
                           ) {

/* ... */
    rc = SQLForeignKeys(hstmt, NULL, 0,
                        schema, SQL_NTS, tablename, SQL_NTS,
                        NULL, 0,
                        NULL, SQL_NTS, NULL, SQL_NTS);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 2, SQL_C_CHAR, (SQLPOINTER) pktable_schem.s, 129,
                    &pktable_schem.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 3, SQL_C_CHAR, (SQLPOINTER) pktable_name.s, 129,
                    &pktable_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 4, SQL_C_CHAR, (SQLPOINTER) pkcolumn_name.s, 129,
                    &pkcolumn_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 6, SQL_C_CHAR, (SQLPOINTER) fktable_schem.s, 129,
                    &fktable_schem.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 7, SQL_C_CHAR, (SQLPOINTER) fktable_name.s, 129,
                    &fktable_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 8, SQL_C_CHAR, (SQLPOINTER) fkcolumn_name.s, 129,
                    &fkcolumn_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 10, SQL_C_SHORT, (SQLPOINTER) &update_rule,
                    0, &update_ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 11, SQL_C_SHORT, (SQLPOINTER) &delete_rule,
                    0, &delete_ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 12, SQL_C_CHAR, (SQLPOINTER) fkey_name.s, 129,
                    &fkey_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 13, SQL_C_CHAR, (SQLPOINTER) pkey_name.s, 129,
                    &pkey_name.ind);
```

SQLForeignKeys

```
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    printf("Primary Key and Foreign Keys for %s.%s\n", schema, tablename);
    /* Fetch each row, and display */
    while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS) {
        printf("  %s  %s.%s.%s\n      Update Rule ",
               pkcolumn_name.s, fktable_schem.s, fktable_name.s, fkcolumn_name.s);
        if (update_rule == SQL_RESTRICT) {
            printf("RESTRICT ");  /* always for IBM DBMSs */
        } else {
            if (update_rule == SQL_CASCADE) {
               printf("CASCADE ");  /* non-IBM only */
            } else {
               printf("SET NULL ");
            }
        }
        printf(", Delete Rule: ");
        if (delete_rule== SQL_RESTRICT) {
            printf("RESTRICT ");  /* always for IBM DBMSs */
        } else {
            if (delete_rule == SQL_CASCADE) {
               printf("CASCADE ");  /* non-IBM only */
            } else {
               if (delete_rule == SQL_NO_ACTION) {
                  printf("NO ACTION ");  /* non-IBM only */
               } else {
                    printf("SET NULL ");
                 }
            }
        }
        printf("\n");
        if (pkey_name.ind > 0 ) {
            printf("     Primary Key Name: %s\n", pkey_name.s);
        }
        if (fkey_name.ind > 0 ) {
            printf("     Foreign Key Name: %s\n", fkey_name.s);
        }
    }
```

참조

- 217 페이지의『SQLPrimaryKeys - 표의 1차 키 열 얻기』
- 264 페이지의『SQLStatistics - 기본 표에 대한 색인과 통계 정보 얻기』

SQLFreeConnect - 연결 핸들 해제

목적

SQLFreeConnect()는 연결 핸들을 무효화하고 해제합니다. 연결 핸들과 연관된 모든 DB2 UDB CLI 자원이 해제됩니다.

이 함수를 호출하기 전에 SQLDisconnect()를 호출해야 합니다.

어플리케이션 종료를 계속하기 위해 SQLFreeEnv()를 그 다음으로 호출하거나 새로운 연결 핸들을 할당하기 위해 SQLAllocHandle()을 호출해야 합니다.

구문

```
SQLRETURN SQLFreeConnect (SQLHDBC    hdbc);
```

함수 인수

표 67. SQLFreeConnect 인수

자료 유형	인수	사용	설명
SQLHDBC	*hdbc*	입력	연결 핸들

사용법

연결이 되어 있는 상태에서 이 함수를 호출하면, SQL_ERROR가 리턴되고 연결 핸들은 계속 유효합니다.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 68. SQLFreeConnect SQLSTATEs

SQLSTATE	설명	설명
58004	시스템 오류	복구할 수 없는 시스템 오류.
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY010	함수 순서 오류	*hdbc*에 대해 SQLDisconnect()를 호출하기 전에 이 함수를 호출합니다.
HY013 *	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

예

SQLAllocEnv()33 페이지의 『예』를 참조하십시오.

참조

- 91 페이지의 『SQLDisconnect - 자료 소스 단절』
- 125 페이지의 『SQLFreeEnv - 환경 핸들 해제』

SQLFreeEnv - 환경 핸들 해제

목적

SQLFreeEnv는 환경 핸들을 무효화하고 해제합니다. 환경 핸들과 연관된 모든 DB2 UDB CLI 자원이 해제됩니다.

이 함수를 호출하기 전에 SQLFreeConnect()를 호출해야 합니다.

이 함수는 종료하기 전에 어플리케이션에서 필요로 하는 마지막 DB2 UDB CLI단계입니다.

구문

```
SQLRETURN SQLFreeEnv (SQLHENV    henv);
```

함수 인수

표 69. SQLFreeEnv 인수

자료 유형	인수	사용	설명
SQLHENV	*henv*	입력	환경 핸들

사용법

유효한 연결 핸들이 아직 있는 상태에서 이 함수를 호출하면 SQL_ERROR가 리턴되고 환경 핸들은 계속 유효합니다.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 70. SQLFreeEnv SQLSTATEs

SQLSTATE	설명	설명
58004	시스템 오류	복구할 수 없는 시스템 오류.
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY010	함수 순서 오류	할당되었거나 연결된 상태의 *hdbc*가 있습니다. SQLFreeEnv를 호출하기 전에 *hdbc*에 대해 SQLDisconnect와 SQLFreeConnect를 호출합니다.
HY013 *****	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

예

SQLAllocEnv()33 페이지의 『예』를 참조하십시오.

참조

- 123 페이지의 『SQLFreeConnect - 연결 핸들 해제』

SQLFreeHandle - 핸들 해제

목적

SQLFreeHandle()은 핸들을 무효화하고 해제합니다.

구문

```
SQLRETURN SQLFreeHandle (SQLSMALLINT htype,
                         SQLINTEGER handle);
```

함수 인수

표 71. SQLFreeHandle 인수

자료 유형	인수	사용	설명
SQLSMALLINT	*hType*	입력	핸들 유형. SQL_HANDLE_ENV, SQL_HANDLE_DBC, SQL_HANDLE_STMT 또는 SQL_HANDLE_DESC여야 합니다.
SQLINTEGER	*handle*	입력	해제할 핸들

사용법

SQLFreeHandle()은 SQLFreeEnv(), SQLFreeConnect()와 SQLFreeStmt() 함수를 결합합니다.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 72. SQLFreeHandle SQLSTATEs

SQLSTATE	설명	설명
58004	시스템 오류	복구할 수 없는 시스템 오류.
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY010	함수 순서 오류	할당되었거나 연결된 상태의 *hdbc*가 있습니다. SQLFreeHandle을 호출하기 전에 *hdbc*에 대해 SQLDisconnect와 SQLFreeConnect를 호출합니다.
HY013 *	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

SQLFreeHandle

참조

- 123 페이지의『SQLFreeConnect - 연결 핸들 해제』
- 125 페이지의『SQLFreeEnv - 환경 핸들 해제』
- 129 페이지의『SQLFreeStmt - 명령문 핸들을 해제(또는 재설정)』

SQLFreeStmt - 명령문 핸들을 해제(또는 재설정)

목적

SQLFreeStmt()는 명령문 핸들에 의해 참조된 명령문의 처리를 종료합니다. 이 함수를 사용하여 다음 작업을 수행합니다.

- 커서 닫기
- 매개변수 재설정
- 변수에서 열의 바인드 해제
- 명령문 핸들을 제거하고 명령문 핸들과 연관된 DB2 UDB CLI 자원 해제

SQL문을 실행하고 결과를 처리한 후 SQLFreeStmt()를 호출합니다.

구문

```
SQLRETURN SQLFreeStmt (SQLHSTMT       hstmt,
                       SQLSMALLINT    fOption);
```

함수 인수

표 73. SQLFreeStmt 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLSMALLINT	*fOption*	입력	명령문 핸들을 해제하는 방식을 지정하는 옵션. 옵션은 다음 값 중 하나입니다. • SQL_CLOSE • SQL_DROP • SQL_UNBIND • SQL_RESET_PARAMS

사용법

다음 옵션과 함께 SQLFreeStmt()를 호출할 수 있습니다.

- SQL_CLOSE

  연관된 핸들(*hstmt*)과 연관된 커서(있는 경우)가 닫히고 지연 중인 모든 결과가 삭제됩니다. *hstmt*에 바인드된 어플리케이션 변수(있는 경우)에서 같거나 다른 값과 함께 SQLExecute()를 호출하여 어플리케이션은 커서를 다시 열 수 있습니다. 명령문 핸들이 제거되거나 다음 번에 SQLSetCursorName() 호출이 성공할 때까지 커서명이 유지됩니다. 명령문 핸들과 연관된 커서가 없으면 이 옵션은 아무런 영향도 주지 않습니다(경고나 오류가 표시되지 않음).

- SQL_DROP

  입력 명령문 핸들과 연관된 DB2 UDB CLI 자원이 해제되고 핸들이 무효화됩니다. 열린 커서가 있는 경우 닫히고 지연 중인 모든 결과가 삭제됩니다.

SQLFreeStmt

- SQL_UNBIND

  이 명령문 핸들에서 이전의 SQLBindCol() 호출에 의해 바인드된 모든 열이 해제됩니다. (어플리케이션 변수나 파일 참조와 결과 집합 열 사이의 연관이 없어집니다.)

- SQL_RESET_PARAMS

  이 명령문 핸들에서 이전의 SQLBindParam() 호출에 의해 설정된 모든 매개변수가 해제됩니다. 어플리케이션 변수 또는 파일 참조와 명령문 핸들의 SQL문 매개변수 마커 사이의 연관이 없어집니다.

명령문 핸들을 다시 사용하여 다른 명령문을 처리할 경우 이전 명령문이

- SELECT이면 커서를 닫아야 합니다.
- 다른 매개변수 유형이나 갯수를 사용했으면 매개변수를 재설정해야 합니다.
- 다른 열 바인딩 유형이나 갯수를 사용했으면 열을 바인드 해제해야 합니다.

또는 명령문 핸들을 제거하고 새로운 핸들을 할당할 수 있습니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

*fOption*이 SQL_DROP으로 설정되면, SQLError()가 호출될 때 사용할 명령문 핸들이 없으므로 SQL_SUCCESS_WITH_INFO가 리턴되지 않습니다.

진단

표 74. SQLFreeStmt SQLSTATEs

SQLSTATE	설명	설명
40003 *****	명령문 완료 상태를 알 수 없음	함수가 처리를 완료하기 전에 CLI와 자료 소스와의 통신 링크가 끊어졌습니다.
58004	시스템 오류	복구할 수 없는 시스템 오류.
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*fOption* 인수에 대해 지정된 값이 SQL_CLOSE, SQL_DROP, SQL_UNBIND 또는 SQL_RESET_PARAMS가 아닙니다.

예

SQLFetch()110 페이지의 『예』를 참조하십시오.

참조

- 37 페이지의 『SQLAllocStmt - 명령문 핸들 할당』
- 39 페이지의 『SQLBindCol - 어플리케이션 변수에 열 바인드』

- 109 페이지의『SQLFetch - 다음 행 페치』
- 123 페이지의『SQLFreeConnect - 연결 핸들 해제』
- 254 페이지의『SQLSetParam - 매개변수 설정』

SQLGetCol - 결과 집합 행의 한 열을 검색

목적

SQLGetCol()은 결과 집합 행의 단일 열에 대한 자료를 검색합니다. SQLFetch()를 호출하여 직접 어플리케이션 변수로 자료를 전달하는 SQLBindCol() 대신 이 함수를 사용할 수 있습니다. 큰 문자용 자료를 나누어 검색하기 위해서도 SQLGetCol()을 사용할 수 있습니다.

SQLGetCol()을 호출하기 전에 SQLFetch()를 호출해야 합니다.

각 열에 대해 SQLGetCol()을 호출한 후에 다음 행을 검색하기 위해 SQLFetch()가 호출됩니다.

구문

```
SQLRETURN SQLGetCol  (SQLHSTMT       hstmt,
                      SQLSMALLINT    icol,
                      SQLSMALLINT    fCType,
                      SQLPOINTER     rgbValue,
                      SQLINTEGER     cbValueMax,
                      SQLINTEGER     *pcbValue);
```

함수 인수

표 75. SQLGetCol 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLSMALLINT	*icol*	입력	자료 검색이 요구된 열 번호

표 75. SQLGetCol 인수 (계속)

자료 유형	인수	사용	설명
SQLSMALLINT	*fCType*	입력	*icol*에 의해 식별된 열의 어플리케이션 자료 유형. 다음 유형들이 지원됩니다. • SQL_CHAR • SQL_VARCHAR • SQL_NUMERIC • SQL_DECIMAL • SQL_BIGINT • SQL_INTEGER • SQL_SMALLINT • SQL_FLOAT • SQL_REAL • SQL_DOUBLE • SQL_GRAPHIC • SQL_VARGRAPHIC • SQL_DATETIME • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP
SQLPOINTER	*rgbValue*	출력	검색된 열 자료가 저장될 버퍼를 가리키는 포인터.
SQLINTEGER	*cbValueMax*	입력	*rgbValue*가 가리키는 버퍼의 최대 크기. *fcType*이 SQL_DECIMAL 또는 SQL_NUMERIC이면, *cbValueMax*는 실제 정밀도와 스케일이어야 합니다. 두 값을 지정하는 방법은 (정밀도 * 256) + 스케일을 사용하는 것입니다. 이 값은 또한 SQLColAttributes()를 사용할 때 이 자료 유형의 LENGTH로 리턴된 값입니다.
SQLINTEGER *	*pcbValue*	출력	*rgbValue* 버퍼에서 리턴하기 위해 DB2 UDB CLI가 사용할 수 있는 바이트 수를 나타내는 값을 가리키는 포인터. 자료를 나눠서 검색하면, 이전의 SQLGetCol() 호출에서 얻은 열 자료의 바이트를 제외하고 아직 남아 있는 바이트 수가 여기에 있습니다. 열의 자료 값이 널이면 값은 SQL_NULL_DATA입니다. 이 포인터가 널(null)이고 SQLFetch()가 널(null) 자료가 들어 있는 열을 얻으면, 이를 보고할 수단이 없으므로 이 함수는 실패하게 됩니다. SQLFetch()가 그래픽 자료가 들어 있는 열을 페치하면 *pcbValue*를 가리키는 포인터는 널(null)이 아니어야 하며 그렇지 않으면 *rgbValue* 버퍼에 검색된 자료 길이를 어플리케이션에 알려줄 방법이 없으므로 이 함수는 실패하게 됩니다.

사용법

*icol*의 값이 바인드된 열을 지정하지 않는 한 같은 행에 대해 SQLBindCol()와 함께 SQLGetCol()을 사용할 수 있습니다. 일반적인 단계는 다음과 같습니다.

1. SQLFetch() - 커서를 첫 행으로 진행시키고 첫 행을 검색하여, 바인드된 열에 대해 자료를 전송합니다.
2. SQLGetCol() - 지정된(바인드 해제된) 열에 대해 자료를 전송합니다.
3. 필요한 각 열에 대해 2단계를 반복합니다.
4. SQLFetch() - 커서를 다음 행으로 진행시키고 다음 행을 검색하여, 바인드된 열에 대해 자료를 전송합니다.
5. 결과 집합이 더 이상 필요하지 않을 때까지 결과 집합의 각 행에 대해 2 - 4단계를 반복합니다.

C 자료 유형(*fCType*)이 SQL_CHAR이거나 *fCType*이 SQL_DEFAULT이고 열 유형이 CHAR 또는 VARCHAR인 경우 SQLGetCol()는 긴 열을 검색합니다.

각각의 SQLGetCol() 호출에서 리턴하기 위해 사용할 수 있는 자료가 *cbValueMax* 보다 크거나 같으면 자료가 절단됩니다. 자료 절단을 표시하는 SQLSTATE와 함께 쌍을 이루는 SQL_SUCCESS_WITH_INFO의 함수 리턴 코드는 절단을 표시합니다. 절단 지점에서 시작하는 같은 바인드 해제된 열에서 나중에 자료를 가져오려면 같은 *icol* 값으로 어플리케이션이 SQLGetCol()을 다시 호출할 수 있어야 합니다. 전체 열을 가져오려면 함수가 SQL_SUCCESS를 리턴할 때까지 어플리케이션은 이러한 호출을 반복해야 합니다. 다음 번의 SQLGetCol() 호출은 SQL_NO_DATA_FOUND를 리턴합니다.

검색을 통해 열 자료 부분을 삭제하기 위해 어플리케이션은 *icol*을 관심있는 다음 열 위치로 설정하여 SQLGetCol()을 호출할 수 있습니다. 전체 행에 대해 검색되지 않은 자료를 삭제하기 위해 어플리케이션은 SQLFetch()를 호출하여 커서를 다음 행으로 진행시키거나 결과 집합의 추가 자료에 관심이 없으면 SQLFreeStmt()를 호출하여 커서를 닫습니다.

열 자료가 *rgbValue*가 가리키는 기억장치 영역에 놓이기 전에 *fCType* 입력 인수는 필요한 자료 변환(있는 경우) 유형을 결정합니다.

SQL_ATTR_OUTPUT_NTS 속성을 변경하기 위해 SQLSetEnvAttr()이 사용되지 않거나 어플리케이션이 복수 청크(Chunk)에서 자료를 검색하는 경우에는 rgbValue에서 리턴된 내용은 항상 널로 종료됩니다. 어플리케이션이 복수 청크(Chunk)에서 자료를 검색하는 경우, 널 종료 바이트는 오직 자료의 마지막 부분에 추가됩니다.

절단이 소수점 오른쪽 자릿수와 관련되면 숫자 자료 유형 절단은 보고되지 않습니다. 소수점 왼쪽에서 절단이 발생하면 오류가 리턴됩니다(진단 관련 섹션 참조).

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

이전의 SQLGetCol() 호출이 이 열에 대한 모든 자료를 검색한 경우 SQL_NO_DATA_FOUND가 리턴됩니다.

SQLGetCol()에 의해 길이가 0인 스트링이 검색된 경우 SQL_SUCCESS가 리턴됩니다. 이 경우 *pcbValue*는 0을 포함하며 *rgbValue*는 널 종료자가 있습니다.

이전의 SQLFetch() 호출이 실패한 경우 결과가 정의되지 않았으므로 SQLGetCol()은 호출되지 말아야 합니다.

진단

표 76. SQLGetCol SQLSTATEs

SQLSTATE	설명	설명
07006	제한된 자료 유형 속성 위반	자료 값이 *fCType* 인수에 의해 지정된 C 자료 유형으로 변환될 수 없습니다.
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*cbValueMax* 인수의 값이 1 미만이며 *fCType* 인수가 SQL_CHAR입니다. 지정된 열 번호가 유효하지 않습니다. *rgbValue* 또는 *pcbValue* 인수가 널(null) 포인터입니다.
HY010	함수 순서 오류	지정된 *hstmt*는 커서로 위치지정된 상태가 아닙니다. 먼저 SQLFetch()를 호출하지 않고 함수가 호출됩니다.
HY013 *	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.
HYC00	드라이버가 지원되지 않음	지정된 자료 유형에 대한 SQL 자료 유형이 인식되지만 드라이버에 의해 지원되지 않습니다. SQL 자료 유형에서 어플리케이션 자료 *fCType*으로의 변환 요구가 드라이버나 자료 소스에 의해 수행될 수 없습니다.

제한사항

같은 명령문 핸들의 같은 행에 대해 SQLGetCol()에 의해 마지막으로 검색된 열보다 낮은 번호의 열을 *icol*이 지정하지 않도록 ODBC가 요구합니다. (행의 열이 바인드된 경우) 마지막으로 바인드된 열 앞에 있는 열에 대한 자료를 검색하기 위해 SQLGetCol()을 사용하는 것도 ODBC는 허용하지 않습니다.

*icol*이 바인드된 열을 지정하지 않는 경우, *icol*의 값을 바인드된 열 앞과 임의 순서로 지정하도록 하여 DB2 UDB CLI는 이 두 규칙을 해제합니다.

예

바인드시킨 열을 사용하는 것과 SQLGetCol()을 사용하는 것의 차이는 SQLFetch()110 페이지의 『예』를 참조하십시오.

다음 예에서 사용된 check_error, initialize, terminate 함수의 리스트는 300 페이지의 『예: 대화식 SQL 및 그에 해당되는 DB2 UDB CLI 함수 호출』을 참조하십시오.

```
/*****************************************************************************
** file = getcol.c
**
** Example of directly executing an SQL statement.
** Getcol is used to retrieve information from the result set.
** Compare to fetch.c
**
** Functions used:
**
**        SQLAllocConnect      SQLFreeConnect
**        SQLAllocEnv          SQLFreeEnv
**        SQLAllocStmt         SQLFreeStmt
**        SQLConnect           SQLDisconnect
**
**        SQLBindCol           SQLFetch
**        SQLTransact          SQLError
**        SQLExecDirect        SQLGetCursor
*****************************************************************************/

#include <stdio.h>
#include <string.h>
#include "sqlcli.h"

#define MAX_STMT_LEN 255

int initialize(SQLHENV *henv,
               SQLHDBC *hdbc);

int terminate(SQLHENV henv,
              SQLHDBC hdbc);

int print_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt);

int check_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt,
                 SQLRETURN  frc);

/*****************************************************************
** main
** - initialize
** - terminate
*****************************************************************/
int main()
{
    SQLHENV     henv;
    SQLHDBC     hdbc;
```

```
    SQLCHAR     sqlstmt[MAX_STMT_LEN + 1]="";
    SQLRETURN   rc;

    rc = initialize(&henv, &hdbc);
    if (rc != SQL_SUCCESS) return(terminate(henv, hdbc));

    {SQLHSTMT   hstmt;
     SQLCHAR    sqlstmt[]="SELECT deptname, location from org where division = 'Eastern'";
     SQLCHAR    deptname[15],
                location[14];
     SQLINTEGER rlength;

        rc = SQLAllocStmt(hdbc, &hstmt);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

        rc = SQLExecDirect(hstmt, sqlstmt, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt, rc);

        printf("Departments in Eastern division:\n");
        printf("DEPTNAME       Location\n");
        printf("-------------- -------------\n");

        while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS)
        {
            rc = SQLGetCol(hstmt, 1, SQL_CHAR, (SQLPOINTER) deptname, 15, &rlength);
            rc = SQLGetCol(hstmt, 2, SQL_CHAR, (SQLPOINTER) location, 14, &rlength);
            printf("%-14.14s %-13.13s \n", deptname, location);
        }
        if (rc != SQL_NO_DATA_FOUND )
            check_error (henv, hdbc, hstmt, rc);
    }

    rc = SQLTransact(henv, hdbc, SQL_COMMIT);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

    terminate(henv, hdbc);
    return (SQL_SUCCESS);

}/* end main */
```

참조

- 39 페이지의『SQLBindCol - 어플리케이션 변수에 열 바인드』
- 109 페이지의『SQLFetch - 다음 행 페치』

SQLGetConnectAttr - 연결 속성 값 얻기

목적

SQLGetConnectAttr()는 지정된 연결 옵션에 대한 현재 설정을 리턴합니다.

이 옵션은 SQLSetConnectAttr() 함수를 사용하여 설정됩니다.

구문

```
SQLRETURN SQLGetConnectAttr(   SQLHDBC      hdbc,
                               SQLINTEGER   fAttr,
                               SQLPOINTER   pvParam),;
                               SQLINTEGER   bLen,
                               SQLINTEGER   *sLen);
```

함수 인수

표 77. SQLGetConnectAttr 인수

자료 유형	인수	사용	설명
SQLHDBC	*hdbc*	입력	연결 핸들
SQLINTEGER	*fAttr*	입력	검색할 속성. 자세한 정보는 236 페이지의 표 147을 참조하십시오.
SQLPOINTER	*pvParam*	출력	*fAttr*과 연관된 값. *fAttr*의 값에 따라 32비트 정수 값이나 널로 종료되는 문자 스트링을 가리키는 포인터입니다.
SQLINTEGER	*bLen*	입력	값이 문자 스트링인 경우 pvParm에 저장되는 최대 바이트 수. 그렇지 않은 경우는 사용되지 않습니다.
SQLINTEGER *	*sLen*	출력	속성이 문자 스트링인 경우 출력 문자의 길이. 그렇지 않은 경우는 사용되지 않습니다.

사용법

SQLGetConnectAttr()이 호출되고 지정된 *fAttr*이 SQLSetConnectAttr를 통해 설정되지 않거나 디폴트 값이 없으면 SQLGetConnectAttr()은 SQL_NO_DATA_FOUND를 리턴합니다.

명령문 옵션 설정은 SQLGetConnectAttr()를 통해 검색될 수 없습니다.

진단

표 78. SQLGetConnectAttr SQLSTATEs

SQLSTATE	설명	설명
08003	연결이 열려 있지 않음	열린 연결이 필요하도록 *fAttr*을 지정하였습니다.
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	속성 유형이 범위 밖의 값임	유효하지 않은 *fAttr* 값이 지정되었습니다. *pvParam* 인수가 널(null) 포인터입니다.

표 78. SQLGetConnectAttr SQLSTATEs (계속)

SQLSTATE	설명	설명
HYC00	드라이버가 지원되지 않음	*fAttr*이 인식되었지만 지원되지 않습니다.

SQLGetConnectOption - 연결 옵션의 현재 설정을 리턴

목적

SQLGetConnectOption()은 지정된 연결 옵션에 대한 현재 설정을 리턴합니다.

이 옵션은 SQLSetConnectOption() 함수를 사용하여 설정됩니다.

구문

```
SQLRETURN SQLGetConnectOption( HDBC         hdbc,
                              SQLSMALLINT   fOption,
                              SQLPOINTER    pvParam);
```

함수 인수

표 79. SQLGetConnectOption 인수

자료 유형	인수	사용	설명
HDBC	*hdbc*	입력	연결 핸들
SQLSMALLINT	*fOption*	입력	검색할 옵션. 자세한 정보는 236 페이지의 표 147을 참조하십시오.
SQLPOINTER	*pvParam*	출력	*fOption*과 연관된 값. *fOption*의 값에 따라 32비트 정수 값이나 널로 종료되는 문자 스트링을 가리키는 포인터입니다. 리턴된 문자 스트링의 최대 길이는 SQL_MAX_OPTION_STRING_LENGTH바이트입니다(널 종료 바이트 제외).

사용법

SQLGetConnectOption()은 SQLGetConnectAttr()과 같은 함수를 제공하며 호환성을 위해 두 함수 모두 지원됩니다.

SQLGetConnectOption()이 호출되고 지정된 *fOption*이 SQLSetConnectOption을 통해 설정되지 않거나 디폴트 값이 없으면 SQLGetConnectOption()은 SQL_NO_DATA_FOUND를 리턴합니다.

명령문 옵션 설정은 SQLGetConnectOption()를 통해 검색될 수 없습니다.

진단

표 80. SQLGetConnectOption SQLSTATEs

SQLSTATE	설명	설명
08003	연결이 열려 있지 않음	열린 연결이 필요하도록 *fOption*을 지정하였습니다.
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	옵션 유형이 범위 밖의 값임	유효하지 않은 *fOption* 값이 지정되었습니다. *pvParam* 인수가 널(null) 포인터입니다.

표 80. SQLGetConnectOption SQLSTATEs (계속)

SQLSTATE	설명	설명
HYC00	드라이버가 지원되지 않음	*fOption*이 인식되었지만 지원되지 않습니다.

SQLGetCursorName - 커서명 얻기

목적

SQLGetCursorName()은 입력 명령문 핸들과 연관된 커서명을 리턴합니다. SQLSetCursorName()을 호출하여 커서명이 명시적으로 설정되면 이 이름이 리턴됩니다. 그렇지 않은 경우는 내포적으로 생성된 이름이 리턴됩니다.

구문

```
SQLRETURN SQLGetCursorName (SQLHSTMT       hstmt,
                            SQLCHAR        *szCursor,
                            SQLSMALLINT    cbCursorMax,
                            SQLSMALLINT    *pcbCursor);
```

함수 인수

표 81. SQLGetCursorName 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLCHAR *	*szCursor*	출력	커서명
SQLSMALLINT	*cbCursorMax*	입력	*szCursor* 버퍼의 길이
SQLSMALLINT *	*pcbCursor*	출력	*szCursor*에 대해 리턴하기 위해 사용할 수 있는 바이트 수.

사용법

SQLSetCursorName()을 사용하여 이름이 설정되었거나 SELECT문이 명령문 핸들에서 실행된 경우 SQLGetCursorName()은 커서명을 리턴합니다. 둘 다 참이 아닌 경우에 SQLGetCusorName()을 호출하면 오류가 발생합니다.

SQLSetCursorName()을 사용하여 명시적으로 이름이 설정되면 명령문이 제거되거나 다른 명시적 이름이 설정될 때까지 이 이름이 리턴됩니다.

명시적 이름이 설정되지 않으면, SELECT문이 실행될 때 내포적 이름이 생성되며 이 이름이 리턴됩니다. 내포적 커서명은 항상 SQLCUR로 시작됩니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 82. SQLGetCursorName SQLSTATEs

SQLSTATE	설명	설명
01004	자료가 절단됨	*szCursor*에 리턴된 커서명이 *cbCursorMax*의 값보다 길어서 *cbCursorMax* - 1바이트로 절단됩니다. *pcbCursor* 인수에는 리턴하기 위해 사용할 수 있는 전체 커서명의 길이가 있습니다. 함수는 SQL_SUCCESS_WITH_INFO를 리턴합니다.
40003 *	명령문 완료 상태를 알 수 없음	함수가 처리를 완료하기 전에 CLI와 자료 소스와의 통신 링크가 끊어졌습니다.
58004	시스템 오류	복구할 수 없는 시스템 오류
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*szCursor* 또는 *pcbCursor* 인수가 널(null) 포인터입니다. *cbCursorMax* 인수에 대해 지정된 값이 1 미만입니다.
HY010	함수 순서 오류	명령문 *hstmt*가 실행 상태가 아닙니다. SQLGetCursorName()를 호출하기 전에 SQLExecute(), SQLExecDirect() 또는 SQLSetCursorName()을 호출합니다.
HY013 *	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.
HY015	커서명이 없음	*hstmt*에 열린 커서가 없으며 SQLSetCursorName()과 함께 설정된 커서명이 없습니다. *hstmt*와 연관된 명령문은 커서 사용을 지원하지 않습니다.

제한사항

ODBC가 생성한 커서명은 SQL_CUR로 시작하며, X/Open CLI가 생성한 커서명은 SQLCUR로 시작합니다. DB2 UDB CLI는 SQLCUR을 사용합니다.

예

다음 예에서 사용된 check_error, initialize, terminate 함수의 리스트는 300 페이지의 『예: 대화식 SQL 및 그에 해당되는 DB2 UDB CLI 함수 호출』을 참조하십시오.

```
/************************************************************************
** file = getcurs.c
**
** Example of directly executing a SELECT and positioned UPDATE SQL statement.
** Two statement handles are used, and SQLGetCursor is used to retrieve the
** generated cursor name.
**
** Functions used:
**
**        SQLAllocConnect      SQLFreeConnect
**        SQLAllocEnv          SQLFreeEnv
**        SQLAllocStmt         SQLFreeStmt
**        SQLConnect           SQLDisconnect
**
**        SQLBindCol           SQLFetch
**        SQLTransact          SQLError
```

SQLGetCursorName

```
**        SQLExecDirect        SQLGetCursorName
*****************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "sqlcli.h"

#define MAX_STMT_LEN 255

int initialize(SQLHENV *henv,
               SQLHDBC *hdbc);

int terminate(SQLHENV henv,
              SQLHDBC hdbc);

int print_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt);

int check_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt,
                 SQLRETURN  frc);

/*****************************************************************
** main
** - initialize
** - terminate
*****************************************************************/
int main()
{
    SQLHENV     henv;
    SQLHDBC     hdbc;
    SQLRETURN   rc,
                rc2;

    rc = initialize(&henv, &hdbc);
    if (rc != SQL_SUCCESS) return(terminate(henv, hdbc));

    {SQLHSTMT   hstmt1,
                hstmt2;
     SQLCHAR    sqlstmt[]="SELECT name, job from staff for update of job";
     SQLCHAR    updstmt[MAX_STMT_LEN + 1];
     SQLCHAR    name[10],
                job[6],
                newjob[6],
                cursor[19];

     SQLINTEGER     rlength, attr;
     SQLSMALLINT    clength;

        rc = SQLAllocStmt(hdbc, &hstmt1);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

        /* make sure the statement is update-capable */
        attr = SQL_FALSE;
        rc = SQLSetStmtAttr(hstmt1,SQL_ATTR_FOR_FETCH_ONLY, &attr, 0);
```

```
        /* allocate second statement handle for update statement */
        rc2 = SQLAllocStmt(hdbc, &hstmt2);
        if (rc2 != SQL_SUCCESS )
            check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

        rc = SQLExecDirect(hstmt1, sqlstmt, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt1, rc);

        /* Get Cursor of the SELECT statement's handle */
        rc = SQLGetCursorName(hstmt1, cursor, 19, &clength);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt1, rc);

        /* bind name to first column in the result set */
        rc = SQLBindCol(hstmt1, 1, SQL_CHAR, (SQLPOINTER) name, 10,
                        &rlength);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt1, rc);

        /* bind job to second column in the result set */
        rc = SQLBindCol(hstmt1, 2, SQL_CHAR, (SQLPOINTER) job, 6,
                        &rlength);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt1, rc);


        printf("Job Change for all clerks\n");

        while ((rc = SQLFetch(hstmt1)) == SQL_SUCCESS)
        {
            printf("Name: %-9.9s Job: %-5.5s \n", name, job);
            printf("Enter new job or return to continue\n");
            gets(newjob);
            if (newjob[0] != '\0')
            {
                sprintf( updstmt,
                    "UPDATE staff set job = '%s' where current of %s",
                    newjob, cursor);
                rc2 = SQLExecDirect(hstmt2, updstmt, SQL_NTS);
                if (rc2 != SQL_SUCCESS )
                    check_error (henv, hdbc, hstmt2, rc);
            }
        }
        if (rc != SQL_NO_DATA_FOUND )
            check_error (henv, hdbc, hstmt1, rc);
        SQLFreeStmt(hstmt1, SQL_CLOSE);
    }

    printf("Commiting Transaction\n");
    rc = SQLTransact(henv, hdbc, SQL_COMMIT);
    if (rc != SQL_NO_DATA_FOUND )
        check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

    terminate(henv, hdbc);
    return (0);
}/* end main */
```

참조

- 104 페이지의『SQLExecute - 명령문 실행』
- 102 페이지의『SQLExecDirect - 명령문 직접 실행』
- 243 페이지의『SQLSetCursorName - 커서명 설정』

SQLGetData - 열에서 자료 얻기

목적

SQLGetData()는 결과 집합의 현재 행에서 단일 열에 대한 자료를 검색합니다. SQLFetch()를 호출하여 직접 어플리케이션 변수로 자료를 전달하는 SQLBindCol() 대신 이 함수를 사용할 수 있습니다. 큰 문자용 자료를 나누어 검색하기 위해서도 SQLGetData()를 사용할 수 있습니다.

SQLGetData()를 호출하기 전에 SQLFetch()를 호출해야 합니다.

각 열에 대해 SQLGetData()를 호출한 후에, 다음 행을 검색하기 위해 SQLFetch()가 호출됩니다.

SQLGetData()는 SQLGetCol()과 같으며, 호환성을 위해 두 함수 모두 지원됩니다.

구문

```
SQLRETURN SQLGetData (SQLHSTMT       hstmt,
                      SQLSMALLINT    icol,
                      SQLSMALLINT    fCType,
                      SQLPOINTER     rgbValue,
                      SQLINTEGER     cbValueMax,
                      SQLINTEGER     *pcbValue);
```

주: 적용가능한 섹션의 설명에 대해서는 132 페이지의 『SQLGetCol - 결과 집합 행의 한 열을 검색』을 참조하십시오.

SQLGetDescField - 설명자 필드 얻기

목적

SQLGetDescField()는 설명자에서 값을 얻습니다. SQLGetDescField()는 SQLGetDescRec() 함수를 더 확장할 수 있게 하는 대체 함수입니다.

이 함수는 SQLDescribeCol()과 비슷하지만 SQLGetDescField()는 행 설명자 뿐 아니라 매개변수 설명자에서 자료를 검색합니다.

구문

```
SQLRETURN SQLGetDescField  (SQLHDESC       hdesc,
                            SQLSMALLINT    irec,
                            SQLSMALLINT    fDescType,
                            SQLPOINTER     rgbDesc,
                            SQLINTEGER     bLen,
                            SQLINTEGER     *sLen);
```

함수 인수

표 83. SQLGetDescField 인수

자료 유형	인수	사용	설명
SQLHDESC	*hdesc*	입력	설명자 핸들
SQLSMALLINT	*irec*	입력	지정된 필드가 검색될 레코드 번호
SQLSMALLINT	*fDescType*	입력	표 84를 참조하십시오.
SQLPOINTER	*rgbDesc*	출력	버퍼를 가리키는 포인터
SQLINTEGER	*bLen*	입력	설명자 버퍼 길이(*rgbDesc*)
SQLINTEGER *	*sLen*	출력	리턴될 설명자의 실제 바이트 수. 이 인수에 *rgbDesc* 버퍼의 길이보다 크거나 같은 값이 포함되어 있으면 값이 절단됩니다.

표 84. fDescType 설명자 유형

설명자	유형	설명
SQL_DESC_COUNT	SMALLINT	설명자의 레코드 수는 *rgbDesc*에 리턴됩니다.
SQL_DESC_ALLOC_TYPE	SMALLINT	어플리케이션이 명시적으로 설명자를 할당한 경우 SQL_DESC_ALLOC_USER이고, 구현 프로그램이 자동으로 설명자를 할당한 경우 SQL_DESC_ALLOC_AUTO입니다.
SQL_DESC_NAME	CHAR(128)	*irec*의 NAME 필드를 검색합니다.
SQL_DESC_TYPE	SMALLINT	*irec*의 TYPE 필드를 검색합니다.

표 84. fDescType 설명자 유형 (계속)

설명자	유형	설명
SQL_DESC_DATETIME_INTERVAL_CODE	SMALLINT	SQL_DATETIME 유형의 레코드에 대한 간격 코드를 검색합니다. 내부 코드가 SQL_DATETIME 자료 유형을 더 자세히 정의합니다. 코드 값은 SQL_CODE_DATE, SQL_CODE_TIME, SQL_CODE_TIMESTAMP입니다.
SQL_DESC_LENGTH	INTEGER	*irec*의 LENGTH 필드를 검색합니다.
SQL_DESC_PRECISION	SMALLINT	*irec*의 PRECISION 필드를 검색합니다.
SQL_DESC_SCALE	SMALLINT	*irec*의 SCALE 필드를 검색합니다.
SQL_DESC_NULLABLE	SMALLINT	*irec*에 널이 들어갈 수 있으면 *rgbDesc*에 SQL_NULLABLE이 리턴됩니다. 그렇지 않은 경우 *rgbDesc*에 SQL_NO_NULLS이 리턴됩니다.
SQL_DESC_UNNAMED	SMALLINT	NAME 필드가 실제 이름이면 SQL_NAMED이고 NAME 필드가 구현 프로그램에서 생성한 이름이면 SQL_UNNAMED입니다.
SQL_DESC_DATA_PTR	SQLPOINTER	*irec*에 대한 자료 포인터 필드를 검색합니다.
SQL_DESC_LENGTH_PTR	SQLPOINTER	*irec*에 대한 포인터 필드의 길이를 검색합니다.
SQL_DESC_INDICATOR_PTR	SQLPOINTER	*irec*에 대한 인디케이터 포인터 필드를 검색합니다.

사용법

설명자가 행 설명자이면 설명자의 레코드 수는 결과 집합의 열 수에 대응하며, 매개변수 설명자일 경우에는 매개변수 갯수입니다.

리턴된 열이 있는지를 판별하기 위해 `SQLNumResultCols()`를 호출하는 대신 *fDescType*을 SQL_DESC_COUNT로 설정하여 `SQLGetDescField()`를 호출할 수도 있습니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

진단

표 85. SQLGetDescField SQLSTATEs

SQLSTATE	설명	설명
HY009	유효하지 않은 인수 값	*fDescType* 또는 *irec* 인수에 지정된 값이 유효하지 않습니다. *rgbDesc* 또는 *sLen* 인수가 널(null) 포인터입니다.
HY013 *	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

참조

- 39 페이지의『SQLBindCol - 어플리케이션 변수에 열 바인드』
- 84 페이지의『SQLDescribeCol - 열 속성 설명』
- 102 페이지의『SQLExecDirect - 명령문 직접 실행』
- 104 페이지의『SQLExecute - 명령문 실행』
- 212 페이지의『SQLPrepare - 명령문 준비』

SQLGetDescRec - 설명자 레코드 얻기

목적

SQLGetDescRec()은 설명자에서 전체 레코드를 가져옵니다. SQLGetDescRec()은 SQLDescField() 함수를 간단히 대체합니다.

구문

```
SQLRETURN SQLGetDescRec    (SQLHDESC       hdesc,
                            SQLSMALLINT    irec,
                            SQLCHAR        *rgbDesc,
                            SQLSMALLINT    cbDescMax,
                            SQLSMALLINT    *pcbDesc,
                            SQLSMALLINT    *type,
                            SQLSMALLINT    *subtype,
                            SQLINTEGER     *length,
                            SQLSMALLINT    *prec,
                            SQLSMALLINT    *scale,
                            SQLSMALLINT    *nullable);
```

함수 인수

표 86. SQLGetDescRec 인수

자료 유형	인수	사용	설명
SQLHDESC	*hdesc*	입력	설명자 핸들
SQLSMALLINT	*irec*	입력	정보가 검색될 레코드 번호
SQLCHAR *	*rgbDesc*	출력	레코드에 대한 NAME 필드
SQLSMALLINT	*cbDescMax*	입력	*rgbDesc*에 저장될 최대 바이트 수
SQLSMALLINT *	*pcbDesc*	출력	출력 자료의 총 길이
SQLSMALLINT *	*type*	출력	레코드에 대한 TYPE 필드
SQLSMALLINT *	*subtype*	출력	TYPE이 SQL_DATETIME인 레코드에 대한 DATETIME_INTERVAL_CODE
SQLINTEGER *	*length*	출력	레코드에 대한 LENGTH 필드
SQLSMALLINT *	*prec*	출력	레코드에 대한 PRECISION 필드
SQLSMALLINT *	*scale*	출력	레코드에 대한 SCALE 필드
SQLSMALLINT *	*nullable*	출력	레코드에 대한 NULLABLE 필드

사용법

SQLGetDescRec()를 호출하면 한 번의 호출로 설명자 레코드에서 모든 자료를 검색합니다. 설명자의 레코드 수를 판별하기 위해 SQL_DESC_COUNT와 함께 SQLGetDescField()를 호출할 수도 있습니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

SQLGetDescRec

- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

진단

표 87. SQLGetDescRec SQLSTATEs

SQLSTATE	설명	설명
HY009	유효하지 않은 인수 값	*irec* 인수에 대해 지정된 값이 유효하지 않습니다. *rgbDesc*, *pcbDesc*, *type*, *subtype*, *length*, *prec*, *scale* 또는 *nullable* 인수가 널(null) 포인터입니다.
HY013 *****	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

참조

- 39 페이지의『SQLBindCol - 어플리케이션 변수에 열 바인드』
- 84 페이지의『SQLDescribeCol - 열 속성 설명』
- 102 페이지의『SQLExecDirect - 명령문 직접 실행』
- 104 페이지의『SQLExecute - 명령문 실행』
- 212 페이지의『SQLPrepare - 명령문 준비』

SQLGetDiagField - 진단 정보(확장가능) 리턴

목적

SQLGetDiagField()는 특정 명령문, 연결 또는 환경 핸들에 대해 최근에 호출된 DB2 UDB CLI 함수와 연관된 진단 정보를 리턴합니다.

정보는 표준화된 SQLSTATE, 원래의 오류 코드, 텍스트 메세지로 구성됩니다. 자세한 정보는 18 페이지의 『DB2 UDB CLI 어플리케이션에서의 진단』을 참조하십시오.

다른 함수 호출에서 SQL_ERROR 또는 SQL_SUCCESS_WITH_INFO 리턴 코드를 수신한 후 SQLGetDiagField()를 호출합니다.

주: 명령문을 실행하여 SQL_NO_DATA_FOUND를 리턴한 후 일부 데이터베이스 서버는 특정 제품 진단 정보를 제공할 수도 있습니다.

구문

```
SQLRETURN SQLGetDiagField (SQLSMALLINT      htype,
                           SQLINTEGER       handle,
                           SQLSMALLINT      recNum,
                           SQLSMALLINT      diagId,
                           SQLPOINTER       diagInfo,
                           SQLSMALLINT      bLen,
                           SQLSMALLINT      *sLen);
```

함수 인수

표 88. SQLDiagField 인수

자료 유형	인수	사용	설명
SQLSMALLINT	*hType*	입력	핸들 유형
SQLINTEGER	*handle*	입력	진단 정보가 요구되는 핸들
SQLSMALLINT	*recNum*	입력	여러 개의 오류가 있는 경우 검색될 오류를 지정합니다. 헤더 정보가 요구되면 이 값은 0이어야 합니다. 첫 번째 오류 레코드가 1번입니다.
SQLSMALLINT	*diagId*	입력	154 페이지의 표 89를 참조하십시오.
SQLPOINTER	*diagInfo*	출력	진단 정보에 대한 버퍼.
SQLSMALLINT	*bLen*	입력	요구된 자료가 문자 스트링인 경우 diagInfo의 길이. 그렇지 않은 경우에는 사용되지 않습니다.
SQLSMALLINT *	*sLen*	출력	요구된 자료가 문자 스트링인 경우 전체 진단 정보의 길이. 그렇지 않은 경우에는 사용되지 않습니다.

표 89. diagId 유형

설명자	유형	설명
SQL_DIAG_RETURNCODE	SMALLINT	하위 함수의 리턴 코드. SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA_FOUND, 또는 SQL_ERROR입니다.
SQL_DIAG_NUMBER	INTEGER	지정된 핸들에 대해 사용할 수 있는 진단 레코드의 수.
SQL_DIAG_ROW_COUNT	INTEGER	핸들이 명령문 핸들인 경우 지정된 핸들에 대한 행 수.
SQL_DIAG_SQLSTATE	CHAR(5)	진단 레코드와 관련된 5문자의 SQLSTATE 코드. SQLSTATE 코드는 이식가능한 진단 표시를 제공합니다.
SQL_DIAG_NATIVE	INTEGER	진단 레코드와 관련된 구현 프로그램에서 정의한 오류 코드. 이식가능한 어플리케이션은 이 값을 기준으로 작동되어서는 안됩니다.
SQL_DIAG_MESSAGE_TEXT	CHAR(254)	진단 레코드와 관련된 구현 프로그램에서 정의한 메세지 텍스트.
SQL_DIAG_SERVER_NAME	CHAR(128)	연결을 설정한 SQLConnect() 명령문에 제공된 대로의, 진단 레코드와 관련된 서버명.

사용법

SQLSTATE는 X/OPEN SQL CAE와 X/Open SQL CLI 스냅샷에 의해 정의된 것과 같으나 앞에 IBM 고유의 제품별 SQLSTATE 값이 붙습니다.

한 DB2 UDB CLI 함수에 의해 생성된 진단 정보가 SQLGetDiagField() 이외의 함수가 같은 핸들로 호출되기 전에 검색되지 않으면, 이전 함수 호출에 대한 정보가 손실됩니다. 두 번째 DB2 UDB CLI 함수 호출에 대해 생성된 진단 정보가 있는지에 관계없이 항상 해당됩니다.

주어진 DB2 UDB CLI 함수 호출 후에 여러 개의 진단 메세지가 표시됩니다. 이 메세지는 SQLGetDiagField()를 반복해서 호출하여 한 번에 하나씩 검색할 수 있습니다. 검색되는 각 메세지에 대해 SQLGetDiagField()는 SQL_SUCCESS를 리턴하고 메세지 리스트에서 제거합니다. 더 이상 검색할 메세지가 없으면 SQL_NO_DATA_FOUND가 리턴됩니다.

주어진 핸들 아래에 저장된 진단 정보는 해당 핸들로 SQLGetDiagField()가 호출될 때나 해당 핸들로 다른 DB2 UDB CLI 함수가 호출될 때 지워집니다. 그러나 연관되기는 했지만 다른 핸들 유형으로 SQLGetDiagField()를 호출하여 주어진 핸들 유형과 연관된 정보는 지울 수 없습니다. 예를 들면, 연결 핸들 입력으로 SQLGetDiagField()를 호출해도 해당 연결의 명령문 핸들과 연관된 오류를 지우지 않습니다.

오류 메세지(*szDiagFieldMsg*)에 대한 버퍼가 너무 작아도 SQL_SUCCESS가 리턴됩니다. 이는 SQLGetDiagField()를 다시 호출해도 어플리케이션이 같은 오류 메세지를 검색할 수는 없기 때문입니다. 메세지 텍스트의 실제 길이가 *pcbDiagFieldMsg*에 리턴됩니다.

오류 메세지가 절단되지 않게 하려면 버퍼 길이를 SQL_MAX_MESSAGE_LENGTH + 1로 선언합니다. 메세지 텍스트는 이보다 길어질 수 없습니다.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

입력 핸들에 대한 진단 메세지가 없거나 SQLGetDiagField() 호출을 통해 모든 메세지가 검색되면 SQL_NO_DATA_FOUND가 리턴됩니다.

diagInfo 또는 sLen 인수가 널(null) 포인터인 경우에는 SQL_ERROR이 리턴됩니다.

진단

SQLGetDiagField()가 자신에 대한 진단 정보를 생성하지 않으므로 SQLSTATE가 정의되지 않습니다.

제한사항

ODBC도 X/Open SQL CAE SQLSTATE를 리턴하지만, DB2 UDB CLI만이 추가 IBM 정의 SQLSTATE를 리턴합니다. ODBC 드라이버 관리자는 표준 값 뿐만 아니라 SQLSTATE 값도 리턴합니다. ODBC에 특정한 SQLSTATE에 대한 자세한 정보는 *Microsoft ODBC Programmer's Reference*를 참조하십시오.

이로 인해 사용자는 표준 SQLSTATE에서만 종속 사항을 빌드해야 합니다. 이는 어플리케이션의 분기 논리가 표준 SQLSTATE에만 의존해야 함을 의미합니다. 접두어가 붙은 SQLSTATE는 디버깅에 아주 유용합니다.

SQLGetDiagRec - (간략한) 진단 정보 리턴

목적

SQLGetDiagRec()는 특정 명령문, 연결 또는 환경 핸들에 대해 가장 최근에 호출된 DB2 UDB CLI 함수와 연관된 진단 정보를 리턴합니다.

정보는 표준화된 SQLSTATE, 원래의 오류 코드, 텍스트 메세지로 구성됩니다. 자세한 정보는 18 페이지의 『DB2 UDB CLI 어플리케이션에서의 진단』을 참조하십시오.

다른 함수 호출에서 SQL_ERROR 또는 SQL_SUCCESS_WITH_INFO 리턴 코드를 수신한 후 SQLGetDiagRec()를 호출합니다.

주: 명령문을 실행하여 SQL_NO_DATA_FOUND를 리턴한 후 일부 데이터베이스 서버는 특정 제품 진단 정보를 제공할 수도 있습니다.

구문

```
SQLRETURN SQLGetDiagRec (SQLSMALLINT  hType,
                         SQLINTEGER   handle,
                         SQLSMALLINT  recNum,
                         SQLCHAR      *szSqlState,
                         SQLINTEGER   *pfNativeError,
                         SQLCHAR      *szErrorMsg,
                         SQLSMALLINT  cbErrorMsgMax,
                         SQLSMALLINT  *pcbErrorMsg);
```

함수 인수

표 90. SQLGetDiagRec 인수

자료 유형	인수	사용	설명
SQLSMALLINT	*hType*	입력	핸들 유형
SQLINTEGER	*handle*	입력	진단 정보가 요구되는 핸들
SQLSMALLINT	*recNum*	입력	여러 개의 오류가 있는 경우 검색될 오류를 지정합니다. 헤더 정보가 요구되면 이 값은 0이어야 합니다. 첫 번째 오류 레코드가 1번입니다.
SQLCHAR *	*szSqlState*	출력	널 문자로 종료된 5자 스트링의 SQLSTATE. 처음 2자는 오류 클래스를 표시하며, 다음 3자는 하위 클래스를 표시합니다. 이 값은 X/Open SQL CAE 스펙과 ODBC 스펙에 정의된 SQLSTATE 값에 직접 대응하며 앞에 IBM 고유의 제품별 SQLSTATE 값이 붙습니다.
SQLINTEGER *	*pfNativeError*	출력	원래의 오류 코드. DB2 UDB CLI에서 *pfNativeError* 인수에는 데이터베이스 관리 시스템(DBMS)에 의해 리턴된 SQLCODE 값이 있습니다. 오류가 DBMS가 아니라 DB2 UDB CLI에 의해 생성된 경우 이 필드는 -99999로 설정됩니다.

표 90. SQLGetDiagRec 인수 (계속)

자료 유형	인수	사용	설명
SQLCHAR *	*szErrorMsg*	출력	구현 프로그램에서 정의된 메세지 텍스트를 포함하는 버퍼를 가리키는 포인터. DB2 UDB CLI에서는 데이터베이스 관리 시스템(DBMS)이 생성한 메세지만 리턴됩니다. DB2 UDB CLI는 문제를 설명하는 어떤 메세지 텍스트도 리턴하지 않습니다.
SQLSMALLINT	*cbErrorMsgMax*	입력	*szErrorMsg* 버퍼의 최대(즉, 할당된) 길이. 권장되는 할당 길이는 SQL_MAX_MESSAGE_LENGTH + 1입니다.
SQLSMALLINT *	*pcbErrorMsg*	출력	*szErrorMsg* 버퍼에 리턴되는 사용할 수 있는 전체 바이트 수를 가리키는 포인터. 널 종료 문자를 포함하지 않습니다.

사용법

SQLSTATE는 X/OPEN SQL CAE와 X/Open SQL CLI 스냅샷에 의해 정의된 것과 같으나 앞에 IBM 고유의 제품별 SQLSTATE 값이 붙습니다.

한 DB2 UDB CLI 함수에 의해 생성된 진단 정보가 SQLGetDiagRec() 이외의 함수가 같은 핸들로 호출되기 전에 검색되지 않으면, 이전 함수 호출에 대한 정보가 유실됩니다. 두 번째 DB2 UDB CLI 함수 호출에 대해 생성된 진단 정보가 있는지에 관계없이 항상 해당됩니다.

주어진 DB2 UDB CLI 함수 호출 후에 여러 개의 진단 메세지가 표시됩니다. 이 메세지는 SQLGetDiagRec()를 반복해서 호출하여 한 번에 하나씩 검색할 수 있습니다. 검색되는 각 메세지에 대해 SQLGetDiagRec()는 SQL_SUCCESS를 리턴하고 메세지 리스트에서 제거합니다. 더 이상 검색할 메세지가 없으면, SQL_NO_DATA_FOUND가 리턴되고, SQLSTATE가 "00000"으로 설정되고, *pfNativeError*가 0으로 설정되고, *pcbErrorMsg*와 *szErrorMsg*가 정의되지 않습니다.

주어진 핸들에 저장된 진단 정보는 해당 핸들로 SQLGetDiagRec()가 호출될 때나 해당 핸들로 다른 DB2 UDB CLI 함수가 호출될 때 지워집니다. 그러나 연관되더라도 다른 핸들 유형으로 LGetdialFiRdc를 호출하여 기존 핸들 유형과 연관된 정보를 지울 수 없습니다. 예를 들면, 연결 핸들 입력으로 SQLGetDiagRec()를 호출해도 해당 연결의 명령문 핸들과 연관된 오류를 지우지 않습니다.

SQLGetDiagRec()를 다시 호출해도 어플리케이션이 같은 오류 메세지를 검색할 수 없기 때문에 오류 메세지(*szErrorMsg*)에 대한 버퍼가 너무 작아도 SQL_SUCCESS가 리턴됩니다. 메세지 텍스트의 실제 길이가 *pcbErrorMsg*에 리턴됩니다.

오류 메세지가 절단되지 않게 하려면 버퍼 길이를 SQL_MAX_MESSAGE_LENGTH + 1로 선언합니다. 메세지 텍스트는 이보다 길어질 수 없습니다.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR

SQLGetDiagRec

- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

입력 핸들에 대한 진단 메세지가 없거나 SQLGetDiagRec() 호출을 통해 모든 메세지가 검색되면 SQL_NO_DATA_FOUND가 리턴됩니다.

szSqlState, pfNativeError, szErrorMsg 또는 pcbErrorMsg 인수가 널(null) 포인터인 경우에는 SQL_ERROR이 리턴됩니다.

진단

SQLGetDiagRec()가 자신에 대한 진단 정보를 생성하지 않으므로 sqlstate가 정의되지 않습니다.

제한사항

ODBC도 X/Open SQL CAE SQLSTATE를 리턴하지만, DB2 UDB CLI만이 추가 IBM 정의 SQLSTATE를 리턴합니다. ODBC 드라이버 관리자는 표준 값 뿐만 아니라 SQLSTATE 값도 리턴합니다. ODBC에 특정한 SQLSTATE에 대한 자세한 정보는 *Microsoft ODBC Programmer's Reference*를 참조하십시오.

이로 인해 사용자는 표준 SQLSTATE에서만 종속 사항을 빌드해야 합니다. 이는 어플리케이션의 분기 논리가 표준 SQLSTATE에만 의존해야 함을 의미합니다. 접두어가 붙은 SQLSTATE는 디버깅에 아주 유용합니다.

참조

• 153 페이지의『SQLGetDiagField - 진단 정보(확장가능) 리턴』

SQLGetEnvAttr - 환경 속성의 현재 설정 리턴

목적

SQLGetEnvAttr()은 지정된 환경 속성에 대한 현재 설정을 리턴합니다.

이 옵션은 SQLSetEnvAttr() 함수를 사용하여 설정됩니다.

구문

```
SQLRETURN SQLGetEnvAttr (SQLHENV      henv,
                         SQLINTEGER   Attribute,
                         SQLPOINTER   Value,
                         SQLINTEGER   BufferLength,
                         SQLINTEGER   *StringLength);
```

함수 인수

표 91. SQLGetEnvAttr 인수

자료 유형	인수	사용	설명
SQLHENV	*henv*	입력	환경 핸들
SQLINTEGER	*Attribute*	입력	검색할 속성. 자세한 정보는 249 페이지의 표 159를 참조하십시오.
SQLPOINTER	*Value*	출력	*Attribute*와 연관된 현재 값. 리턴된 값 유형은 *Attribute*에 따라 달라집니다.
SQLINTEGER	*BufferLength*	입력	속성 값이 문자 스트링인 경우 *Value*가 가리키는 버퍼의 최대 크기. 그렇지 않은 경우는 사용되지 않습니다.
SQLINTEGER *	*StringLength*	출력	속성 값이 문자 스트링인 경우 출력 자료의 바이트 길이. 그렇지 않은 경우는 사용되지 않습니다.

*Attribute*가 스트링을 나타내지 않으면 DB2 UDB CLI는 *BufferLength*를 무시하고 *StringLength*를 설정하지 않습니다.

사용법

환경 핸들을 할당한 후부터 해제할 때까지 아무 때나 SQLGetEnvAttr()을 호출할 수 있습니다. 이 함수는 환경 속성의 현재 값을 가져옵니다.

SQLGetEnvAttr

진단

표 92. SQLGetEnvAttr SQLSTATEs

SQLSTATE	설명	설명
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	속성이 범위를 벗어났음.	유효하지 않은 *Attribute* 값이 지정되었습니다. *Value* 또는 *StringLength* 인수가 널(null) 포인터입니다.

SQLGetFunctions - 함수 얻기

목적

SQLGetFunctions()는 특정 함수가 지원되는지 조회합니다. 이 함수를 사용하여 어플리케이션은 사용하는 드라이버에 따라 다양한 레벨의 지원을 할 수 있습니다.

이 함수를 호출하기 전에 SQLConnect()를 호출하고 자료 소스(데이터베이스 서버)와 연결되어 있어야 합니다.

구문

```
SQLRETURN SQLGetFunctions (SQLHDBC        hdbc,
                           SQLSMALLINT    fFunction,
                           SQLSMALLINT    *pfSupported);
```

함수 인수

표 93. SQLGetFunctions 인수

자료 유형	인수	사용	설명
SQLHDBC	*hdbc*	입력	데이터베이스 연결 핸들
SQLSMALLINT	*fFunction*	입력	조회되는 함수.
SQLSMALLINT *	*pfSupported*	출력	조회되는 함수가 지원되는 지에 따라 SQL_TRUE 또는 SQL_FALSE를 리턴하는 위치를 가리키는 포인터.

사용법

그림 4는 *fFunction* 인수가 유효하고 대응하는 함수가 지원되는지를 표시합니다.

주: 별표(*)가 표시된 값은 리모트 서버에 연결될 때는 지원되지 않습니다.

```
SQL_API_ALLOCCONNECT      = TRUE
SQL_API_ALLOCENV          = TRUE
SQL_API_ALLOCHANDLE       = TRUE
SQL_API_ALLOCSTMT         = TRUE
SQL_API_BINDCOL           = TRUE
SQL_API_BINDFILETOCOL     = TRUE
SQL_API_BINDFILETOPARAM   = TRUE
SQL_API_BINDPARAM         = TRUE
SQL_API_BINDPARAMETER     = TRUE
SQL_API_CANCEL            = TRUE
SQL_API_CLOSECURSOR       = TRUE
```

그림 4. 지원되는 함수 (1/3)

SQLGetFunctions

```
SQL_API_COLATTRIBUTES     = TRUE
SQL_API_COLUMNS           = TRUE
SQL_API_CONNECT           = TRUE
SQL_API_COPYDESC          = TRUE
SQL_API_DATASOURCES       = TRUE
SQL_API_DESCRIBECOL       = TRUE
SQL_API_DESCRIBEPARAM     = TRUE
SQL_API_DISCONNECT        = TRUE
SQL_API_DRIVERCONNECT     = TRUE
SQL_API_ENDTRAN           = TRUE
SQL_API_ERROR             = TRUE
SQL_API_EXECDIRECT        = TRUE
SQL_API_EXECUTE           = TRUE
SQL_API_EXTENDEDFETCH     = TRUE
SQL_API_FETCH             = TRUE
SQL_API_FOREIGNKEYS       = TRUE
SQL_API_FREECONNECT       = TRUE
SQL_API_FREEENV           = TRUE
SQL_API_FREEHANDLE        = TRUE
SQL_API_FREESTMT          = TRUE
SQL_API_GETCOL            = TRUE
SQL_API_GETCONNECTATTR    = TRUE
SQL_API_GETCONNECTOPTION  = TRUE
SQL_API_GETCURSORNAME     = TRUE
SQL_API_GETDATA           = TRUE
SQL_API_GETDESCFIELD      = TRUE
SQL_API_GETDESCREC        = TRUE
SQL_API_GETDIAGFIELD      = TRUE
SQL_API_GETDIAGREC        = TRUE
SQL_API_GETENVATTR        = TRUE
SQL_API_GETFUNCTIONS      = TRUE
SQL_API_GETINFO           = TRUE
SQL_API_GETLENGTH         = TRUE
SQL_API_GETPOSITION       = TRUE
SQL_API_GETSTMTATTR       = TRUE
SQL_API_GETSTMTOPTION     = TRUE
SQL_API_GETSUBSTRING      = TRUE
SQL_API_GETTYPEINFO       = TRUE
SQL_API_LANGUAGES         = TRUE
SQL_API_MORERESULTS       = TRUE
SQL_API_NATIVESQL         = TRUE
SQL_API_NUMPARAMS         = TRUE
SQL_API_NUMRESULTCOLS     = TRUE
SQL_API_PARAMDATA         = TRUE
SQL_API_PARAMOPTIONS      = TRUE
SQL_API_PREPARE           = TRUE
SQL_API_PRIMARYKEYS       = TRUE
SQL_API_PROCEDURECOLUMNS  = TRUE
SQL_API_PROCEDURES        = TRUE
SQL_API_PUTDATA           = TRUE
SQL_API_RELEASEENV        = TRUE
SQL_API_ROWCOUNT          = TRUE
SQL_API_SETCONNECTATTR    = TRUE
SQL_API_SETCONNECTOPTION  = TRUE
SQL_API_SETCURSORNAME     = TRUE
SQL_API_SETDESCFIELD      = TRUE
SQL_API_SETDESCREC        = TRUE
SQL_API_SETENVATTR        = TRUE
```

그림 4. 지원되는 함수 (2/3)

```
SQL_API_SETPARAM          = TRUE
SQL_API_SETSTMTATTR       = TRUE
SQL_API_SETSTMTOPTION     = TRUE
SQL_API_SPECIALCOLUMNS    = TRUE *
SQL_API_STATISTICS        = TRUE *
SQL_API_TABLES            = TRUE
SQL_API_TRANSACT          = TRUE
```

그림 4. 지원되는 함수 (3/3)

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 94. SQLGetFunctions SQLSTATEs

SQLSTATE	설명	설명
40003 *****	명령문 완료 상태를 알 수 없음	함수가 처리를 완료하기 전에 CLI와 자료 소스와의 통신 링크가 끊어졌습니다.
58004	시스템 오류	복구할 수 없는 시스템 오류
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*pfSupported* 인수가 널(null) 포인터입니다.
HY010	함수 순서 오류. 아직 연결 핸들을 할당할 수 없습니다.	SQLConnect 이전에 SQLGetFunctions가 호출됩니다.
HY013 *****	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

SQLGetInfo - 일반 정보 얻기

목적

SQLGetInfo()는 현재 어플리케이션이 연결된 데이터베이스 관리 시스템(DBMS)에 대한 일반 정보(지원되는 자료 변환 포함)를 리턴합니다.

구문

```
SQLRETURN SQLGetInfo (SQLHDBC        hdbc,
                      SQLSMALLINT    fInfoType,
                      SQLPOINTER     rgbInfoValue,
                      SQLSMALLINT    cbInfoValueMax,
                      SQLSMALLINT    *pcbInfoValue);
```

함수 인수

표 95. SQLGetInfo 인수

자료 유형	인수	사용	설명
SQLHDBC	*hdbc*	입력	데이터베이스 연결 핸들
SQLSMALLINT	*fInfoType*	입력	원하는 정보 유형.
SQLPOINTER	*rgbInfoValue*	출력(또한 입력)	이 함수가 원하는 정보를 저장하는 버퍼를 가리키는 포인터. 검색되는 정보 유형에 따라 다음의 4가지 유형의 정보가 리턴될 수 있습니다. • 16비트 정수 값 • 32비트 정수 값 • 32비트 2진 값 • 널로 종료되는 문자 스트링
SQLSMALLINT	*cbInfoValueMax*	입력	*rgbInfoValue* 포인터가 가리키는 버퍼의 최대 길이.
SQLSMALLINT *	*pcbInfoValue*	출력	원하는 정보를 리턴하는데 사용할 수 있는 총 바이트 수를 리턴하는 위치를 가리키는 포인터. *pcbInfoValue*가 가리키는 위치의 값이 *cbInfoValueMax*에 지정된 *rgbInfoValue* 버퍼의 크기보다 크면 스트링 출력 정보가 *cbInfoValueMax* - 1바이트로 절단되고 함수는 SQL_SUCCESS_WITH_INFO를 리턴합니다.

사용법

165 페이지의 표 96은 *fInfoType*의 가능한 값과 SQLGetInfo()가 해당 값에 대해 리턴할 정보에 대한 설명을 나열합니다.

표 96. SQLGetInfo에 의해 리턴되는 정보

fInfoType	형식	설명과 주
SQL_ACTIVE_CONNECTIONS	Short int	어플리케이션당 지원되는 사용 중인 연결의 최대 수. 0이 리턴되면 시스템 자원에 따라 한계가 결정된다는 것을 표시합니다.
SQL_ACTIVE_STATEMENTS	Short int	연결당 사용 중인 명령문의 최대 수. 0이 리턴되면 시스템 자원에 따라 한계가 결정된다는 것을 표시합니다.
SQL_AGGREGATE_FUNCTIONS	32비트 마스크	비트 마스크가 열거하는 집합 함수에 대한 지원: • SQL_AF_ALL • SQL_AF_AVG • SQL_AF_COUNT • SQL_AF_DISTINCT • SQL_AF_MAX • SQL_AF_MIN • SQL_AF_SUM
SQL_CATALOG_NAME	스트링	문자 스트링 ″Y″ 는 서버가 카탈로그 이름을 지원함을 표시합니다. ″N″은 카탈로그 이름이 지원되지 않음을 표시합니다.
SQL_COLUMN_ALIAS	스트링	연결에서 열 별명을 지원하는지 여부. 연결에서 열 별명 개념을 지원하면 ″Y″ 값이 리턴됩니다.

표 96. SQLGetInfo에 의해 리턴되는 정보 (계속)

fInfoType	형식	설명과 주
SQL_CONVERT_BIGINT SQL_CONVERT_BINARY SQL_CONVERT_BLOB SQL_CONVERT_CHAR SQL_CONVERT_CLOB SQL_CONVERT_DATE SQL_CONVERT_DBCLOB SQL_CONVERT_DECIMAL SQL_CONVERT_DOUBLE SQL_CONVERT_FLOAT SQL_CONVERT_INTEGER SQL_CONVERT_LONGVARBINARY SQL_CONVERT_LONGVARCHAR SQL_CONVERT_NUMERIC SQL_CONVERT_REAL SQL_CONVERT_SMALLINT SQL_CONVERT_TIME SQL_CONVERT_TIMESTAMP SQL_CONVERT_VARBINARY SQL_CONVERT_VARCHAR SQL_CONVERT_WCHAR SQL_CONVERT_WLONGVARCHAR SQL_CONVERT_WVARCHAR	32비트 마스크	infoType에 명명된 유형의 자료에 대한 CONVERT 스칼라 함수로 자료소스에 의해 지원된 변환을 표시합니다. 비트 마스크가 0이면 동일한 자료 유형으로의 변환을 포함한 명명된 유형의 자료에 대한 어떠한 변환도 자료 소스는 지원하지 않습니다. 예를 들어, 자료 소스가 SQL_INTEGER 자료의 SQL_DECIMAL 자료 유형으로의 변환을 지원하는지를 알기 위해, 어플리케이션이 SQL_CONVERT_INTEGER의 finfoType으로 SQLGetInfo()를 호출합니다. 이 때 어플리케이션이 리턴된 비트 마스크와 SQL_CVT_DECIMAL를 AND합니다. 결과 값이 0이 아니면 변환이 지원됩니다. 다음의 비트 마스크를 사용하여 지원되는 변환을 판별합니다. • SQL_CVT_BIGINT • SQL_CVT_BINARY • SQL_CONVERT_BLOB • SQL_CVT_CHAR • SQL_CONVERT_CLOB • SQL_CVT_DATE • SQL_CONVERT_DBCLOB • SQL_CVT_DECIMAL • SQL_CVT_DOUBLE • SQL_CVT_FLOAT • SQL_CVT_INTEGER • SQL_CVT_LONGVARBINARY • SQL_CVT_LONGVARCHAR • SQL_CVT_NUMERIC • SQL_CVT_REAL • SQL_CONVERT_SMALLINT • SQL_CONVERT_TIME • SQL_CONVERT_TIMESTAMP • SQL_CONVERT_VARBINARY • SQL_CONVERT_VARCHAR • SQL_CONVERT_WCHAR • SQL_CONVERT_WLONGVARCHAR • SQL_CONVERT_WVARCHAR
SQL_CONVERT_FUNCTIONS	32비트 마스크	드라이버 및 관련 자료 소스에 의해 지원되는 스칼라 변환 함수를 표시합니다. • SQL_FN_CVT_CONVERT - 어떤 변환 함수가 지원되는지를 판별하는데 사용됩니다. • SQL_FN_CVT_CAST - 어떤 캐스트 함수가 지원되는지를 판별하는데 사용됩니다.

표 96. SQLGetInfo에 의해 리턴되는 정보 (계속)

fInfoType	형식	설명과 주
SQL_CORRELATION_NAME	Short int	서버에 의한 상관명 지원 정도를 표시합니다. • SQL_CN_ANY - 지원되며 어떠한 유효한 사용자 정의 이름도 될 수 있습니다. • SQL_CN_NONE - 지원되지 않는 상관명. • SQL_CN_DIFFERENT - 지원되는 상관명이지만 표시하는 테이블 이름이 아닌 다른 이름이어야 합니다.
SQL_CURSOR_COMMIT_BEHAVIOR	16비트 정수	COMMIT 조작이 커서에 미치는 영향을 표시합니다. 값: • SQL_CB_DELETE - 커서를 없애고 동적 SQL 문에 대한 액세스 계획을 드롭합니다. • SQL_CB_CLOSE - 커서를 없애지만 동적 SQL 문(비조회 명령문 포함)에 대한 액세스 계획을 보유합니다. • SQL_CB_PRESERVE - 커서 및 동적 명령문(비조회 명령문 포함)에 대한 액세스 계획을 보유합니다. 어플리케이션이 자료를 계속 페치하거나 명령문을 다시 작성하지 않고 커서를 닫아 조회를 재실행할 수 있습니다. 주: COMMIT 이후 - 위치 갱신이나 삭제와 같은 조치를 취하기 전에 FETCH를 발행하여 커서의 위치를 다시 지정하여야 합니다.
SQL_CURSOR_ROLLBACK_BEHAVIOR	16비트 정수	ROLLBACK 조작이 커서에 미치는 영향을 표시합니다. 값: • SQL_CB_DELETE - 커서를 없애고 동적 SQL 문에 대한 액세스 계획을 드롭합니다. • SQL_CB_CLOSE - 커서를 없애지만 동적 SQL 문(비조회 명령문 포함)에 대한 액세스 계획을 보유합니다. • SQL_CB_PRESERVE - 커서 및 동적 명령문(비조회 명령문 포함)에 대한 액세스 계획을 보유합니다. 어플리케이션이 자료를 계속 페치하거나 명령문을 다시 작성하지 않고 커서를 닫아 조회를 재실행할 수 있습니다. 주: DB2 서버에 SQL_CB_PRESERVE 특성이 없습니다.
SQL_DATA_SOURCE_NAME	스트링	연결 핸들에 대한 연결된 자료 소스명.
SQL_DATA_SOURCE_READ_ONLY	스트링	문자 스트링 ″Y″는 데이터베이스가 READ ONLY 모드로 설정된 것을 표시하며 ″N″은 READ ONLY 모드로 설정되지 않은 것을 표시합니다.
SQL_DBMS_NAME	스트링	액세스되는 데이터베이스 관리 시스템(DBMS) 제품명. 예를 들면 다음과 같습니다. • QSQ: ″iSeries용 DB2 UDB″ • SQL: ″OS/2용 DB2 UDB″ • DSN: ″zOS 및 OS/390용 DB2 UDB″

표 96. SQLGetInfo에 의해 리턴되는 정보 (계속)

fInfoType	형식	설명과 주
SQL_DBMS_VER	스트링	액세스되는 데이터베이스 관리 시스템(DBMS) 제품의 버전.
SQL_DEFAULT_TXN_ISOLATION	32비트 마스크	지원되는 디폴트 트랜잭션 분리 레벨. 다음 마스크 중 하나가 리턴됩니다. • SQL_TXN_READ_UNCOMMITTED - 변경사항이 모든 트랜잭션에 의해 즉각 인식됩니다(dirty 읽기, 비반복 읽기 및 phantom이 가능합니다). 이 마스크는 UR 레벨과 동일합니다. • SQL_TXN_READ_COMMITTED - 트랜잭션 1에 의한 행 읽기가 트랜잭션 2에 의해 변경되거나 확약될 수 있습니다(비반복 읽기 및 phantom이 가능합니다). 이 마스크는 CS 레벨과 동일합니다. • SQL_TXN_REPEATABLE_READ - 트랜잭션이 탐색 조건이나 지연 트랜잭션과 일치하는 행을 추가하거나 삭제할 수 있습니다 (반복 읽기, 단 phantom 가능). 이 마스크는 RS 레벨과 동일합니다. • SQL_TXN_SERIALIZABLE - 지연 트랜잭션에 의해 영향 받은 자료는 다른 트랜잭션에 사용할 수 없습니다 (반복 읽기, phantom 불가능). 이 마스크는 RR 레벨과 동일합니다. • SQL_TXN_VERSIONING - IBM DBMS에 적용할 수 없습니다. • SQL_TXN_NOCOMMIT - 조작이 성공적으로 종료시 변경사항이 효과적으로 확약됩니다. 명백한 확약이나 롤백은 허용되지 않습니다. 이 마스크는 iSeries용 DB2 UDB 분리 레벨입니다. IBM 용어는 다음과 같습니다. • SQL_TXN_READ_UNCOMMITTED는 확약되지 않은 읽기입니다. • SQL_TXN_READ_COMMITTED는 커서 안정성입니다. • SQL_TXN_REPEATABLE_READ는 읽기 안정성입니다. • SQL_TXN_SERIALIZABLE는 반복할 수 있는 읽기입니다.
SQL_DESCRIBE_PARAMETER	스트링	매개변수를 설명할 수 있으면 Y이고 설명할 수 없으면 N입니다.
SQL_DRIVER_NAME	스트링	자료 소스 액세스에 사용되는 드라이버명.

표 96. SQLGetInfo에 의해 리턴되는 정보 (계속)

fInfoType	형식	설명과 주
SQL_DRIVER_ODBC_VER	스트링	드라이버가 지원하는 ODBC 버전 번호. DB2 ODBC가 2.1을 리턴합니다.
SQL_GROUP_BY	16비트 정수	서버에 의한 GROUP BY 절에 대한 지원 정도를 표시합니다. • SQL_GB_NO_RELATION - GROUP BY와 SELECT 목록의 열은 서로 관계가 없습니다. • SQL_GB_NOT_SUPPORTED - GROUP BY가 지원되지 않습니다. • SQL_GB_GROUP_BY_EQUALS_SELECT - GROUP BY는 선택 목록의 모든 비집합 열을 포함하여야 합니다. • SQL_GB_GROUP_BY_CONTAINS_SELECT - GROUP BY 절은 SELECT 목록의 모든 비집합 열을 포함하여야 합니다.
SQL_IDENTIFIER_CASE	16비트 정수	(테이블 이름과 같은) 오브젝트명의 대소문자 구분을 표시합니다. 값: • SQL_IC_UPPER - 시스템 카탈로그내 상위 케이스에 ID 이름이 저장됩니다. • SQL_IC_LOWER - 시스템 카탈로그내 하위 케이스에 ID 이름이 저장됩니다. • SQL_IC_SENSITIVE - ID 이름에 대소문자를 구별하며 시스템 카탈로그내 혼합 케이스에 저장됩니다. • SQL_IC_MIXED - ID 이름에 대소문자를 구별하지 않으며 시스템 카탈로그내 혼합 케이스에 저장됩니다. 주: IBM DBMS의 ID 이름에는 대소문자를 구별하지 않습니다.
SQL_IDENTIFIER_QUOTE_CHAR	스트링	인용 문자열의 분리문자로 사용되는 문자.
SQL_LIKE_ESCAPE_CLAUSE	스트링	LIKE 술부의 메타문자 백분율 및 밑줄에 이탈 문자가 지원되지는지를 표시하는 문자 스트링
SQL_MAX_CATALOG_NAME_LEN	16비트 정수	카탈로그 규정자 이름 최대 길이; 3 부분으로 이루어진 테이블 이름의 첫 번째 부분(바이트)
SQL_MAX_COLUMN_NAME_LEN	Short int	열 이름 최대 길이.
SQL_MAX_COLUMNS_IN_GROUP_BY	Short int	GROUP BY 절의 최대 열 수.
SQL_MAX_COLUMNS_IN_INDEX	Short int	SQL 색인의 최대 열 수.
SQL_MAX_COLUMNS_IN_ORDER_BY	Short int	ORDER BY 절의 최대 열 수.
SQL_MAX_COLUMNS_IN_SELECT	Short int	SELECT 문의 최대 열 수.
SQL_MAX_COLUMNS_IN_TABLE	Short int	SQL 표의 최대 열 수.
SQL_MAX_CURSOR_NAME_LEN	Short int	커서명의 최대 길이.
SQL_MAX_OWNER_NAME_LEN	Short int	소유자명의 최대 길이.

표 96. SQLGetInfo에 의해 리턴되는 정보 (계속)

fInfoType	형식	설명과 주
SQL_MAX_ROW_SIZE	32비트 부호 없는 정수	서버가 기본 테이블의 단일 행에 지원하는 최대 바이트 길이를 지정합니다. 제한이 없으면 0입니다.
SQL_MAX_SCHEMA_NAME_LEN	Int	스키마명의 최대 길이.
SQL_MAX_STATEMENT_LEN	32비트 부호 없는 정수	흰색 공간 수를 포함한 SQL 문 스트링의 최대 바이트 길이를 표시합니다.
SQL_MAX_TABLE_NAME	Short int	표 이름 최대 길이.
SQL_MAX_TABLES_IN_SELECT	Short int	SELECT 문의 최대 표 수.
SQL_MULTIPLE_ACTIVE_TXN	스트링	문자 스트링 ″Y″는 다중 연결시 활동 트랜잭션이 허용됨을 표시합니다. ″N″은 한 번에 하나의 연결에만 활성 트랜잭션이 허용됨을 표시합니다.
SQL_NON_NULLABLE_COLUMNS	16비트 정수	널이 가능하지 않은 열이 지원되는지를 표시합니다. • SQL_NNC_NON_NULL - 열을 NOT NULL로 정의할 수 있습니다. • SQL_NNC_NULL - 열을 NOT NULL로 정의할 수 없습니다.
SQL_NUMERIC_FUNCTIONS	32비트 마스크	지원되는 스칼라 숫자 함수를 표시합니다. 다음의 비트 마스크를 사용하여 지원되는 숫자 함수를 판별합니다. • SQL_FN_NUM_ABS • SQL_FN_NUM_ACOS • SQL_FN_NUM_ASIN • SQL_FN_NUM_ATAN • SQL_FN_NUM_ATAN2 • SQL_FN_NUM_CEILING • SQL_FN_NUM_COS • SQL_FN_NUM_COT • SQL_FN_NUM_DEGREES • SQL_FN_NUM_EXP • SQL_FN_NUM_FLOOR • SQL_FN_NUM_LOG • SQL_FN_NUM_LOG10 • SQL_FN_NUM_MOD • SQL_FN_NUM_PI • SQL_FN_NUM_POWER • SQL_FN_NUM_RADIANS • SQL_FN_NUM_RAND • SQL_FN_NUM_ROUND • SQL_FN_NUM_SIGN • SQL_FN_NUM_SIN • SQL_FN_NUM_SQRT • SQL_FN_NUM_TAN • SQL_FN_NUM_TRUNCATE

표 96. SQLGetInfo에 의해 리턴되는 정보 (계속)

fInfoType	형식	설명과 주
SQL_ODBC_API_CONFORMANCE	16비트 정수	ODBC 준수 레벨: • SQL_OAC_NONE • SQL_OAC_LEVEL1 • SQL_OAC_LEVEL2
SQL_ODBC_SQL_CONFORMANCE	16비트 정수	값: • SQL_OSC_MINIMUM - 지원되는 최소 ODBC SQL 문법을 의미합니다 • SQL_OSC_CORE - 지원되는 핵심 ODBC SQL Grammer를 의미합니다 • SQL_OSC_EXTENDED - 지원되는 확장 ODBC SQL Grammar를 의미합니다 위의 3가지 유형의 ODBC SQL 문법 정의는 Microsoft ODBC 3.0 Software Development Kit 및 Programmer's Reference를 참조하십시오.
SQL_ORDER_BY_COLUMNS_IN_SELECT	스트링	Set to ″Y″ if columns in the ORDER BY 절의 열이 선택 목록에 반드시 있어야 할 경우 ″Y″로 설정하고 그렇지 않을 경우 ″N″으로 설정하십시오.
SQL_OUTER_JOINS	스트링	문자 스트링 • ″Y″는 외부 결합이 지원됨을 표시하며 DB2 ODBC가 ODBC 외부 결합 요구 구문을 지원합니다. • ″N″은 외부 결합이 지원되지 않음을 표시합니다.
SQL_OWNER_TERM 또는 SQL_SCHEMA_TERM	스트링	스키마에 대한 데이터베이스 벤더 전문용어(소유자).
SQL_OWNER_USAGE 또는 SQL_SCHEMA_USAGE	32비트 마스크	SQL 문이 실행될 때 이 명령문과 연관된 스키마(소유자)가 있는 SQL 문 유형을 표시합니다. 스키마 규정자(소유자)는 다음과 같습니다. • SQL_OU_DML_STATEMENTS - 모든 DML 명령문에 지원됩니다. • SQL_OU_PROCEDURE_INVOCATION - 절차 호출 명령문에 지원됩니다. • SQL_OU_TABLE_DEFINITION - 모든 테이블 정의 명령문에 지원됩니다. • SQL_OU_INDEX_DEFINITION - 모든 색인 정의 명령문에 지원됩니다. • SQL_OU_PRIVILEGE_DEFINITION - 모든 권한 정의 명령문에 지원됩니다(부여 및 호출 명령문).

표 96. SQLGetInfo에 의해 리턴되는 정보 (계속)

fInfoType	형식	설명과 주
SQL_POSITIONED_STATEMENTS	32비트 마스크	위치된 UPDATE 및 위치된 DELETE 명령문에 대한 지원 정도를 표시합니다. • SQL_PS_POSITIONED_DELETE • SQL_PS_POSITIONED_UPDATE • SQL_PS_SELECT_FOR_UPDATE, 열을 커서를 통해 갱신할 수 있도록 서버가 <조회 표현식>에 지정될 FOR UPDATE 절을 요구하는지의 여부를 표시합니다.
SQL_PROCEDURE_TERM	스트링	프로시듀어의 자료 소스명.
SQL_PROCEDURES	스트링	현재 서버에서 SQL 프로시듀어 지원 여부. 연결이 SQL 절차를 지원하면 ″Y″ 값이 리턴됩니다.
SQL_QUALIFIER_LOCATION 또는 SQL_CATALOG_LOCATION	16비트 정수	규정 테이블명에서 규정자 위치를 표시하는 16비트 정수 값. 0은 규정된 이름이 지원되지 않음을 표시합니다.
SQL_QUALIFIER_NAME_SEPARATOR 또는 SQL_CATALOG_NAME_SEPARATOR	스트링	카탈로그명과 그 뒤에 오는 규정된 이름 요소 사이의 분리자로 사용된 문자.
SQL_QUALIFIER_TERM 또는 SQL_CATALOG_TERM	스트링	규정자에 대한 데이터베이스 벤더 전문 용어. 벤더가 3 부분으로 이루어진 이름의 높은 순서 파트에 사용하는 이름. DB2 ODBC가 3 부분으로 이루어진 이름을 지원하지 않기 때문에 길이가 0인 스트링이 리턴됩니다. 비ODBC 어플리케이션에 SQL_QUALIFIER_NAME 대신 SQL_CATALOG_TERM 기호명이 사용되어야 합니다.
SQL_QUALIFIER_USAGE 또는 SQL_CATALOG_USAGE	32비트 마스크	이 마스크는 카탈로그에 사용되는 것을 제외하고 SQL_OWNER_USAGE와 유사합니다.
SQL_QUOTED_IDENTIFIER_CASE	16비트 정수	리턴 • SQL_IC_UPPER - SQL에 인용된 ID는 대소문자를 구별하며 시스템 카탈로그내 상위 케이스에 저장됩니다. • SQL_IC_LOWER - SQL에 인용된 ID는 대소문자를 구별하지 않으며 시스템 카탈로그내 하위 케이스에 저장됩니다. • SQL_IC_SENSITIVE - SQL에 인용된 ID (분리 ID)는 대소문자를 구별하며 시스템 카탈로그내 혼합 케이스에 저장됩니다. • SQL_IC_MIXED - SQL에 인용된 ID는 대소문자를 구별하지 않으며 시스템 카탈로그내 혼합 케이스에 저장됩니다. 이것은 (인용되지 않은) ID를 시스템 카탈로그에 저장하는 방법을 판별하는데 사용되는 SQL_IDENTIFIER_CASE fInfoType과 대조되어야 합니다.

표 96. SQLGetInfo에 의해 리턴되는 정보 (계속)

fInfoType	형식	설명과 주
SQL_SEARCH_PATTERN_ESCAPE	스트링	(SQLTables(), SQLColumns())과 같은 카탈로그 함수에 대한 이탈 문자로서 드라이버가 지원하는 것을 지정하는데 사용됩니다.
SQL_SQL92_PREDICATES	32비트 마스크	SQL-92가 정의하는 SELECT 명령문에 지원되는 술부를 표시합니다. • SQL_SP_BETWEEN • SQL_SP_COMPARISON • SQL_SP_EXISTS • SQL_SP_IN • SQL_SP_ISNOTNULL • SQL_SP_ISNULL • SQL_SP_LIKE • SQL_SP_MATCH_FULL • SQL_SP_MATCH_PARTIAL • SQL_SP_MATCH_UNIQUE_FULL • SQL_SP_MATCH_UNIQUE_PARTIAL • SQL_SP_OVERLAPS • SQL_SP_QUANTIFIED_COMPARISON • SQL_SP_UNIQUE
SQL_SQL92_VALUE_EXPRESSIONS	32비트 마스크	SQL-92가 정의하는 것을 지원하는 값 표현식을 표시합니다. • SQL_SVE_CASE • SQL_SVE_CAST • SQL_SVE_COALESCE • SQL_SVE_NULLIF

표 96. SQLGetInfo에 의해 리턴되는 정보 (계속)

fInfoType	형식	설명과 주
SQL_STRING_FUNCTIONS	32비트 마스크	지원되는 스트링 함수를 표시합니다. 다음의 비트 마스크는 지원되는 스트링 함수 판별에 사용됩니다. • SQL_FN_STR_ASCII • SQL_FN_STR_CHAR • SQL_FN_STR_CONCAT • SQL_FN_STR_DIFFERENCE • SQL_FN_STR_INSERT • SQL_FN_STR_LCASE • SQL_FN_STR_LEFT • SQL_FN_STR_LENGTH • SQL_FN_STR_LOCATE • SQL_FN_STR_LOCATE_2 • SQL_FN_STR_LTRIM • SQL_FN_STR_REPEAT • SQL_FN_STR_REPLACE • SQL_FN_STR_RIGHT • SQL_FN_STR_RTRIM • SQL_FN_STR_SOUNDEX • SQL_FN_STR_SPACE • SQL_FN_STR_SUBSTRING • SQL_FN_STR_UCASE 어플리케이션이 스트링1, 스트링2 및 시작 인수로 LOCATE 스칼라 함수를 호출할 수 있으면 SQL_FN_STR_LOCATE 비트 마스크가 리턴됩니다. 어플리케이션이 스트링1 및 스트링2로 LOCATE 스칼라 함수만을 호출할 수 있으면 SQL_FN_STR_LOCATE_2 bitmask가 리턴됩니다. LOCATE 스칼라 함수가 완전히 지원되면 양쪽의 비트 마스크가 리턴됩니다.

표 96. SQLGetInfo에 의해 리턴되는 정보 (계속)

fInfoType	형식	설명과 주
SQL_TIMEDATE_FUNCTIONS	32비트 마스크	지원되는 시간 및 날짜 함수를 표시합니다. 다음의 비트 마스크를 사용하여 지원되는 데이터 함수를 판별합니다. • SQL_FN_TD_CURDATE • SQL_FN_TD_CURTIME • SQL_FN_TD_DAYNAME • SQL_FN_TD_DAYOFMONTH • SQL_FN_TD_DAYOFWEEK • SQL_FN_TD_DAYOFYEAR • SQL_FN_TD_HOUR • SQL_FN_TD_JULIAN_DAY • SQL_FN_TD_MINUTE • SQL_FN_TD_MONTH • SQL_FN_TD_MONTHNAME • SQL_FN_TD_NOW • SQL_FN_TD_QUARTER • SQL_FN_TD_SECOND • SQL_FN_TD_SECONDS_SINCE_MIDNIGHT • SQL_FN_TD_TIMESTAMPADD • SQL_FN_TD_TIMESTAMPDIFF • SQL_FN_TD_WEEK • SQL_FN_TD_YEAR
SQL_TXN_CAPABLE	Short int	트랜잭션에 DDL이나 DML 또는 두 가지가 모두 있는지를 표시합니다. • SQL_TC_NONE - 지원되지 않는 트랜잭션. • SQL_TC_DML - 트랜잭션에 DML 명령문 (SELECT, INSERT, UPDATE, DELETE 등)만이 있을 수 있습니다. 트랜잭션에 인카운터된 DDL 명령문 (CREATE TABLE, DROP INDEX 등)이 오류를 일으킵니다. • SQL_TC_DDL_COMMIT - 트랜잭션에 DML 명령문만이 있을 수 있습니다. 트랜잭션에서 인카운터된 명령문이 확약될 트랜잭션을 일으킵니다. • SQL_TC_DDL_IGNORE - 트랜잭션에 DML 명령문만이 있을 수 있습니다. 트랜잭션에서 인카운트된 DDL 명령문이 무시됩니다. • SQL_TC_ALL - 순서에 상관없이 트랜잭션에 DDL 및 DML 명령문이 있을 수 있습니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 97. SQLGetInfo SQLSTATEs

SQLSTATE	설명	설명
01004	자료가 절단됨	요구된 정보가 널로 종료되는 스트링으로 리턴되고 길이가 *cbInfoValueMax*에 지정된 어플리케이션 버퍼 길이를 초과했습니다. *pcbInfoValue* 인수에 요구된 정보의 실제(절단되지 않은) 길이가 들어 있습니다.
08003	연결이 열려 있지 않음	*fInfoType*에 요구된 정보 유형이 열린 연결을 요구합니다. SQL_ODBC_VER만이 열린 연결을 요구하지 않습니다.
40003 *	명령문 완료 상태를 알 수 없음	함수가 처리를 완료하기 전에 CLI와 자료 소스와의 통신 링크가 끊어졌습니다.
58004	시스템 오류	복구할 수 없는 시스템 오류.
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*rgbInfoValue* 인수가 널(null) 포인터입니다. 유효하지 않은 *fInfoType*이 지정되었습니다.
HY013 *	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

SQLGetLength - 스트링 값의 길이 검색

목적

SQLGetLength()가 사용되어 현재 트랜잭션 동안 서버에서 리턴된(페치나 SQLGetSubString() 호출의 결과로) 큰 오브젝트 로케이터에 의해 참조되는 큰 오브젝트 값의 길이를 검색합니다.

구문

```
SQLRETURN  SQLGetLength      (SQLHSTMT          StatementHandle,
                              SQLSMALLINT       LocatorCType,
                              SQLINTEGER        Locator,
                              SQLINTEGER        *StringLength,
                              SQLINTEGER        *IndicatorValue);
```

함수 인수

표 98. SQLGetLength 인수

자료 유형	인수	사용	설명
SQLHSTMT	*StatementHandle*	입력	명령문 핸들. 할당되었지만 현재 이에 할당된 준비된 명령문이 없는 명령문 핸들일 수 있습니다.
SQLSMALLINT	*LocatorCType*	입력	소스 LOB 로케이터의 C 유형. 다음 값 중 하나일 수 있습니다. • SQL_C_BLOB_LOCATOR • SQL_C_CLOB_LOCATOR • SQL_C_DBCLOB_LOCATOR
SQLINTEGER	*Locator*	입력	LOB 로케이터 값으로 설정되어야 합니다.
SQLINTEGER *	*StringLength*	출력	지정 로케이터의 길이.[a] 포인터가 NULL로 설정되면 SQLSTATE **HY**009가 리턴됩니다.
SQLINTEGER *	*IndicatorValue*	출력	항상 0으로 설정됩니다.

주: a. DBCLOB 자료에 대해서도 바이트 단위입니다.

사용법

SQLGetLength()를 사용하여 LOB 로케이터에 의해 표시된 자료 값의 길이를 판별할 수 있습니다. 일부 또는 전체 LOB 값을 가져오기 위한 적절한 전략을 선택할 수 있도록 어플리케이션에 의해 사용되어 참조된 LOB 값의 전체 길이를 판별합니다.

로케이터가 생성된 트랜잭션이 종료되었기 때문에 FREE LOCATOR 명령문을 사용하여 명시적으로나 내재적으로 해제되지 않은 LOB 로케이터가 로케이터 인수에 있을 수 있습니다.

명령문 핸들은 준비된 명령문이나 카탈로그 함수 호출과 연관되어서는 안됩니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

오류 조건

표 99. SQLGetLength SQLSTATEs

SQLSTATE	설명	설명
07006	유효하지 않은 변환	*LocatorCType* 및 *Locator* 의 결합은 유효하지 않습니다.
58004	예기치 않은 시스템 실패	복구할 수 없는 시스템 오류
HY003	프로그램 유형이 범위 밖의 값임	*LocatorCType*이 SQL_C_CLOB_LOCATOR, SQL_C_BLOB_LOCATOR, 또는 SQL_C_DBCLOB_LOCATOR 중 하나가 아닙니다.
HY009	유효하지 않은 인수 값	*StringLength* 또는 *IndicatorValue* 인수가 널(null) 포인터입니다.
HY010	함수 순서 오류	지정된 *StatementHandle*이 *할당된* 상태가 아닙니다.
HYC00	드라이버가 지원되지 않음	어플리케이션이 큰 오브젝트를 지원하지 않는 자료 소스에 현재 연결되었습니다.
0F001	유효하지 않은 LOB 변수	*Locator*에 대해 지정된 값이 LOB 로케이터와 연관되지 않았습니다.

제한사항

큰 오브젝트를 지원하지 않는 DB2 서버에 연결될 때는 이 함수를 사용할 수 없습니다.

참조

- 39 페이지의『SQLBindCol - 어플리케이션 변수에 열 바인드』
- 109 페이지의『SQLFetch - 다음 행 페치』
- 179 페이지의『SQLGetPosition - 스트링의 시작 위치 리턴』
- 187 페이지의『SQLGetSubString - 스트링 값의 일부 검색』

SQLGetPosition - 스트링의 시작 위치 리턴

목적

SQLGetPosition()을 사용하여 LOB 값(소스) 내의 한 스트링의 시작 위치를 리턴합니다. 소스 값은 LOB 로케이터이어야 하며, 탐색 스트링은 LOB 로케이터 또는 리터럴 스트링일 수 있습니다.

소스와 탐색 LOB 로케이터는 현재 트랜잭션에서의 페치 또는 SQLGetSubString() 호출의 데이터베이스에서 리턴된 값일 수 있습니다.

구문

```
SQLRETURN   SQLGetPosition   (SQLHSTMT          StatementHandle,
                              SQLSMALLINT       LocatorCType,
                              SQLINTEGER        SourceLocator,
                              SQLINTEGER        SearchLocator,
                              SQLCHAR           *SearchLiteral,
                              SQLINTEGER        SearchLiteralLength,
                              SQLINTEGER        FromPosition,
                              SQLINTEGER        *LocatedAt,
                              SQLINTEGER        *IndicatorValue);
```

함수 인수

표 100. SQLGetPosition 인수

자료 유형	인수	사용	설명
SQLHSTMT	*StatementHandle*	입력	명령문 핸들. 할당되었지만 현재 이에 할당된 준비된 명령문이 없는 명령문 핸들일 수 있습니다.
SQLSMALLINT	*LocatorCType*	입력	소스 LOB 로케이터의 C 유형. 다음 값 중 하나일 수 있습니다. • SQL_C_BLOB_LOCATOR • SQL_C_CLOB_LOCATOR • SQL_C_DBCLOB_LOCATOR
SQLINTEGER	*SourceLocator*	입력	*SourceLocator*는 소스 LOB 로케이터로 설정되어야 합니다.
SQLINTEGER	*SearchLocator*	입력	*SearchLiteral* 포인터가 널(null)이고 *SearchLiteralLength*가 0으로 설정되면, *SearchLocator*는 탐색 스트링과 연관된 LOB 로케이터로 설정되어야 합니다. 그렇지 않으면 인수가 무시됩니다.
SQLCHAR *	*SearchLiteral*	입력	이 인수는 탐색 스트링 리터럴이 들어 있는 기억장치 영역을 가리킵니다. *SearchLiteralLength*가 0이면 이 포인터는 널(null)이어야 합니다.
SQLINTEGER	*SearchLiteralLength*	입력	*SearchLiteral*에서 스트링의 길이(바이트).[a] 이 인수 값이 0이면 *SearchLocator* 인수가 의미가 있습니다.

표 100. SQLGetPosition 인수 (계속)

자료 유형	인수	사용	설명
SQLINTEGER	*FromPosition*	입력	BLOB과 CLOB의 경우 탐색을 시작할 소스 스트링 내의 첫 바이트 위치입니다. 함수에 의해 리턴됩니다. DBCLOB의 경우 첫 문자입니다. 시작 바이트나 문자는 1로 번호가 지정됩니다.
SQLINTEGER *	*LocatedAt*	출력	BLOB과 CLOB의 경우 스트링이 위치할 바이트 위치이며, 스트링이 없다면 값은 0입니다. DBCLOB의 경우 문자 위치입니다. 소스 스트링의 길이가 0이면, 값 1이 리턴됩니다.
SQLINTEGER *	*IndicatorValue*	출력	항상 0으로 설정됩니다.

주:

a DBCLOB 자료에 대해서도 바이트 단위입니다.

사용법

SQLGetPosition()과 SQLGetSubString()을 함께 사용하여 임의로 스트링의 일부를 가져올 수 있습니다. SQLGetSubString()을 사용하려면 전체 스트링내의 서브스트링 위치를 먼저 알고 있어야 합니다. 서브스트링의 시작 위치를 탐색 스트링에 의해 찾을 수 있는 경우 SQLGetPosition()을 사용하여 해당 서브스트링의 시작 위치를 알 수 있습니다.

로케이터가 생성된 트랜잭션이 종료되었기 때문에 FREE LOCATOR 명령문을 사용하여 명시적으로나 내재적으로 해제되지 않은 유효한 LOB 로케이터가 *Locator*와 *SearchLocator*(사용된 경우) 인수에 있을 수 있습니다.

*Locator*와 *SearchLocator*는 같은 LOB 로케이터 유형을 가져야 합니다.

명령문 핸들은 준비된 명령문이나 카탈로그 함수 호출과 연관되어서는 안됩니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

오류 조건

표 101. SQLGetPosition SQLSTATEs

SQLSTATE	설명	설명
07006	유효하지 않은 변환	LOB 로케이터 값 중 하나와 *LocatorCType*과의 결합은 유효하지 않습니다.

표 101. SQLGetPosition SQLSTATEs (계속)

SQLSTATE	설명	설명
42818	유효하지 않은 길이	패턴 길이가 너무 깁니다.
58004	예기치 않은 시스템 실패	복구할 수 없는 시스템 오류
HY009	유효하지 않은 인수 값	*LocatedAt* 또는 *IndicatorValue* 인수가 널(null) 포인터입니다. *FromPosition*에 대한 인수 값이 0보다 크지 않습니다. *LocatorCType*이 SQL_C_CLOB_LOCATOR, SQL_C_BLOB_LOCATOR, 또는 SQL_C_DBCLOB_LOCATOR 중 하나가 아닙니다.
HY010	함수 순서 오류	지정된 *StatementHandle*이 *할당된* 상태가 아닙니다.
HY090	유효하지 않은 스트링 또는 버퍼 길이	*SearchLiteralLength*의 값이 1 미만이거나 SQL_NTS가 아닙니다.
HYC00	드라이버가 지원되지 않음	어플리케이션이 큰 오브젝트를 지원하지 않는 자료 소스에 현재 연결되었습니다.
0F001	유효하지 않은 LOB 변수	*Locator* 또는 *SearchLocator* 에 지정된 값이 현재 LOB 로케이터가 아닙니다.

제한사항

큰 오브젝트를 지원하지 않는 DB2 서버에 연결될 때는 이 함수를 사용할 수 없습니다.

참조

- 39 페이지의『SQLBindCol - 어플리케이션 변수에 열 바인드』
- 106 페이지의『SQLExtendedFetch - 행 배열 페치』
- 109 페이지의『SQLFetch - 다음 행 페치』
- 177 페이지의『SQLGetLength - 스트링 값의 길이 검색』
- 187 페이지의『SQLGetSubString - 스트링 값의 일부 검색』

SQLGetStmtAttr - 명령문 속성 값 얻기

목적

SQLGetStmtAttr()는 지정된 명령문 속성의 현재 설정을 리턴합니다.

이 옵션은 SQLSetStmtAttr() 함수를 사용하여 설정됩니다. 이 함수는 SQLGetStmtOption()과 유사하며 호환성을 위해 두 함수 모두 지원됩니다.

구문

```
SQLRETURN SQLGetStmtAttr( SQLHSTMT        hstmt,
                          SQLINTEGER      fAttr,
                          SQLPOINTER      pvParam,
                          SQLINTEGER      bLen,
                          SQLINTEGER      *sLen);
```

함수 인수

표 102. SQLGetStmtAttr 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLINTEGER	*fAttr*	입력	검색할 속성. 자세한 정보는 표 103을 참조하십시오.
SQLPOINTER	*pvParam*	출력	요구된 속성에 대한 버퍼를 가리키는 포인터.
SQLINTEGER	*bLen*	입력	속성이 문자 스트링인 경우 *pvParam*에 저장된 최대 바이트 수. 그렇지 않은 경우는 사용되지 않습니다.
SQLINTEGER *	*sLen*	출력	속성이 문자 스트링인 경우 출력 자료의 길이. 그렇지 않은 경우는 사용되지 않습니다.

사용법

표 103. 명령문 속성

fAttr	자료 유형	내용
SQL_ATTR_FOR_FETCH_ONLY	정수	이 명령문 핸들에 대해 열린 커서가 읽기 전용이어야 하는지를 지정합니다. • SQL_FALSE - 위치지정된 갱신과 삭제에 대해 커서를 사용할 수 있습니다. 이것이 디폴트 값입니다. • SQL_TRUE - 커서가 읽기 전용이고 위치지정된 갱신과 삭제에 대해 사용될 수 없습니다.
SQL_ATTR_APP_ROW_DESC	정수	명령문 핸들을 사용하여 행 자료를 검색하기 위한 어플리케이션에 대한 설명자 핸들.
SQL_ATTR_APP_PARAM_DESC	정수	이 명령문 핸들에 대한 매개변수 값을 제공하기 위해 어플리케이션에 의해 사용되는 설명자 핸들.

표 103. 명령문 속성 (계속)

fAttr	자료 유형	내용
SQL_ATTR_CURSOR_SCROLLABLE	정수	이 명령문 핸들에 대해 열린 커서를 스크롤할 수 있는지를 지정하는 32비트 정수 값 • SQL_FALSE - 커서를 스크롤할 수 없으며 커서에 대해 SQLFetchScroll()을 사용할 수 없습니다. 이것이 디폴트 값입니다. • SQL_TRUE - 커서를 스크롤할 수 있습니다. SQLFetchScroll()을 사용하여 이 커서의 자료를 검색할 수 있습니다.
SQL_ATTR_CURSOR_TYPE	정수	이 명령문 핸들에 대해 열린 커서의 작동을 지정하는 32비트 정수 값. • SQL_CURSOR_FORWARD_ONLY - 커서를 스크롤 할 수 없으며 커서에 대해 SQLFetchScroll()을 사용할 수 없습니다. 이것이 디폴트 값입니다. • SQL_DYNAMIC - 커서를 스크롤할 수 있습니다. SQLFetchScroll()을 사용하여 이 커서의 자료를 검색할 수 있습니다.
SQL_ATTR_IMP_ROW_DESC	정수	이 명령문 핸들을 사용하여 행 자료를 검색하기 위해 CLI 구현 프로그램에 의해 사용되는 설명자 핸들.
SQL_ATTR_IMP_PARAM_DESC	정수	이 명령문 핸들에 대한 매개변수 값을 제공하기 위해 CLI 구현 프로그램에 의해 사용되는 설명자 핸들.
SQL_ATTR_ROWSET_SIZE	정수	행 집합의 행 수를 지정하는 32비트 정수 값. 각각의 SQLExtendedFetch() 호출에 의해 리턴된 행 번호입니다. 디폴트 값은 1입니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 104. SQLStmtOption SQLSTATEs

SQLSTATE	설명	설명
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.

표 104. SQLStmtOption SQLSTATEs (계속)

SQLSTATE	설명	설명
HY009	유효하지 않은 인수 값	*pvParam* 인수가 널(null) 포인터입니다. 유효하지 않은 *fAttr* 값이 지정되었습니다.
HYC00	드라이버가 지원되지 않음	DB2 UDB CLI는 옵션을 인식했지만 지원하지 않습니다.

SQLGetStmtOption - 명령문 옵션의 현재 설정 리턴

목적

SQLGetStmtOption()은 지정된 명령문 옵션의 현재 설정을 리턴합니다.

이 옵션은 SQLSetStmtOption() 함수를 사용하여 설정됩니다.

구문

```
SQLRETURN SQLGetStmtOption( SQLHSTMT        hstmt,
                            SQLSMALLINT     fOption,
                            SQLPOINTER      pvParam);
```

함수 인수

표 105. SQLStmtOption 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	연결 핸들
SQLSMALLINT	*fOption*	입력	검색할 옵션. 자세한 정보는 182 페이지의 표 103을 참조하십시오.
SQLPOINTER	*pvParam*	출력	옵션의 값. *fOption*의 값에 따라 32비트 정수 값, 또는 널로 종료되는 문자 스트링을 가리키는 포인터일 수 있습니다.

사용법

SQLGetStmtOption()은 SQLGetStmtAttr()과 같은 기능을 제공하며 호환성을 위해 두 함수 모두 지원됩니다.

명령문 옵션 리스트에 대해서는 182 페이지의 표 103을 참조하십시오.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 106. SQLStmtOption SQLSTATEs

SQLSTATE	설명	설명
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.

SQLGetStmtOption

표 106. SQLStmtOption SQLSTATEs (계속)

SQLSTATE	설명	설명
HY009	유효하지 않은 인수 값	*pvParam* 인수가 널(null) 포인터입니다. 유효하지 않은 *fOption* 값이 지정되었습니다.
HYC00	드라이버가 지원되지 않음	DB2 UDB CLI는 옵션을 인식했지만 지원하지 않습니다.

SQLGetSubString - 스트링 값의 일부 검색

목적

SQLGetSubString()이 사용되어 현재 트랜잭션 동안 서버에서 리턴된(페치나 SQLGetSubString() 호출의 결과로) 큰 오브젝트 로케이터에 의해 참조되는 큰 오브젝트 값의 일부를 검색합니다.

구문

```
SQLRETURN  SQLGetSubString   (
                SQLHSTMT          StatementHandle,
                SQLSMALLINT       LocatorCType,
                SQLINTEGER        SourceLocator,
                SQLINTEGER        FromPosition,
                SQLINTEGER        ForLength,
                SQLSMALLINT       TargetCType,
                SQLPOINTER        DataPtr,
                SQLINTEGER        BufferLength,
                SQLINTEGER        *StringLength,
                SQLINTEGER        *IndicatorValue);
```

함수 인수

표 107. SQLGetSubString 인수

자료 유형	인수	사용	설명
SQLHSTMT	*StatementHandle*	입력	명령문 핸들. 할당되었지만 현재 이에 할당된 준비된 명령문이 없는 명령문 핸들일 수 있습니다.
SQLSMALLINT	*LocatorCType*	입력	소스 LOB 로케이터의 C 유형. 다음 값 중 하나일 수 있습니다. • SQL_C_BLOB_LOCATOR • SQL_C_CLOB_LOCATOR • SQL_C_DBCLOB_LOCATOR
SQLINTEGER	*SourceLocator*	입력	*SourceLocator*는 소스 LOB 로케이터 값으로 설정되어야 합니다.
SQLINTEGER	*FromPosition*	입력	BLOB과 CLOB의 경우 함수가 리턴할 첫 바이트의 위치입니다. DBCLOB의 경우 첫 문자입니다. 시작 바이트나 문자는 1로 번호가 지정됩니다.
SQLINTEGER	*ForLength*	입력	함수에 의해 리턴될 스트링의 길이입니다. BLOB과 CLOB의 경우 바이트 단위의 길이입니다. DBCLOB의 문자 단위의 길이입니다. *FromPosition*이 소스 스트링의 길이 미만이지만 *FromPosition* + *ForLength* - 1이 소스 스트링의 끝을 넘어 확장되면, 결과의 오른쪽에 필요한 수만큼 문자가 채워집니다(BLOB의 경우 X'00', CLOB의 경우 1바이트 공백 문자, DBCLOB의 경우 2바이트 공백 문자).
SQLSMALLINT	*TargetCType*	입력	*DataPtr*의 C 자료 유형. 목표는 C 스트링 변수(SQL_C_CHAR, SQL_C_WCHAR, SQL_C_BINARY, 또는 SQL_C_DBCHAR)여야 합니다.

SQLGetSubString

표 107. SQLGetSubString 인수 (계속)

자료 유형	인수	사용	설명
SQLPOINTER	*DataPtr*	출력	검색된 스트링 값 또는 LOB 로케이터가 저장될 버퍼를 가리키는 포인터.
SQLINTEGER	*BufferLength*	입력	*DataPtr*이 가리키는 버퍼의 최대 길이(바이트).
SQLINTEGER *	*StringLength*	출력	목표 C 버퍼 유형이 로케이터 값용이 아니라 2진이나 문자 스트링 변수용일 경우 *DataPtr*에 리턴된 정보의 길이(바이트)[a]. 포인터가 널(null)로 설정되면 아무 값도 리턴되지 않습니다.
SQLINTEGER *	*IndicatorValue*	출력	항상 0으로 설정됩니다.

주:

a DBCLOB 자료에 대해서도 바이트 단위입니다.

사용법

SQLGetSubString()을 사용하여 LOB 로케이터에 의해 표시된 스트링의 일부를 가져옵니다. 목표에 대해 두 가지를 선택할 수 있습니다.

- 목표는 적절한 C 스트링 변수일 수 있습니다.
- 서버에서 새로운 LOB 값을 작성하거나 클라이언트에서 해당 값에 대한 LOB 로케이터를 목표 어플리케이션에 할당할 수 있습니다.

자료를 조각으로 가져오기 위해 SQLGetSubString()을 SQLGetData 대신 사용할 수 있습니다. 이 경우 먼저 열이 LOB 로케이터에 바인드된 후 전체 또는 조각으로 LOB를 페치합니다.

로케이터가 생성된 트랜잭션이 종료되었기 때문에 FREE LOCATOR 명령문을 사용하여 명시적으로나 내재적으로 해제되지 않은 LOB 로케이터가 로케이터 인수에 있을 수 있습니다.

명령문 핸들은 준비된 명령문이나 카탈로그 함수 호출과 연관되어서는 안됩니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

오류 조건

표 108. SQLGetSubString SQLSTATEs

SQLSTATE	설명	설명
01004	자료가 절단됨	리턴될 자료의 양은 *BufferLength*보다 깁니다. 리턴되기 위해 사용할 수 있는 실제 길이가 *StringLength*에 저장됩니다.
07006	유효하지 않은 변환	*TargetCType*에 대해 지정된 값이 SQL_C_CHAR, SQL_C_BINARY, SQL_C_DBCHAR 또는 LOB 로케이터가 아닙니다. *TargetCType*에 대해 지정된 값이 소스에 대해 적합하지 않습니다(예를 들면 BLOB 열에 대해 SQL_C_DBCHAR).
22011	서브스트링 오류 발생	*FromPosition*이 소스 스트링 길이보다 큽니다.
58004	예기치 않은 시스템 실패	복구할 수 없는 시스템 오류
HY003	프로그램 유형이 범위 밖의 값임	*LocatorCType*이 SQL_C_CLOB_LOCATOR, SQL_C_BLOB_LOCATOR, 또는 SQL_C_DBCLOB_LOCATOR 중 하나가 아닙니다.
HY009	유효하지 않은 인수 값	*FromPosition* 또는 *ForLength*에 지정된 값이 양의 정수가 아닙니다. *DataPtr*, *StringLength* 또는 *IndicatorValue* 인수가 널(null) 포인터입니다.
HY010	함수 순서 오류	지정된 *StatementHandle*이 *할당된* 상태가 아닙니다.
HY090	유효하지 않은 스트링 또는 버퍼 길이	*BufferLength*의 값이 0 미만입니다.
HYC00	드라이버가 지원되지 않음	어플리케이션이 큰 오브젝트를 지원하지 않는 자료 소스에 현재 연결되었습니다.
0F001	현재 할당된 로케이터가 없음	*Locator*에 대해 지정된 값이 현재 LOB 로케이터가 아닙니다.

제한사항

큰 오브젝트를 지원하지 않는 DB2 서버에 연결될 때는 이 함수를 사용할 수 없습니다.

참조

- 39 페이지의『SQLBindCol - 어플리케이션 변수에 열 바인드』
- 109 페이지의『SQLFetch - 다음 행 페치』
- 147 페이지의『SQLGetData - 열에서 자료 얻기』
- 177 페이지의『SQLGetLength - 스트링 값의 길이 검색』
- 179 페이지의『SQLGetPosition - 스트링의 시작 위치 리턴』

SQLGetTypeInfo - 자료 유형 정보 얻기

목적

SQLGetTypeInfo()는 DB2 UDB CLI와 연관된 데이터베이스 관리 시스템(DBMS)에 의해 지원되는 자료 유형에 대한 정보를 리턴합니다. 정보는 SQL 결과 집합에 리턴됩니다. 조회를 처리하기 위해 사용된 함수와 같은 함수를 사용하여 열을 수신할 수 있습니다.

구문

```
SQLRETURN   SQLGetTypeInfo   (SQLHSTMT          StatementHandle,
                              SQLSMALLINT       DataType);
```

함수 인수

표 109. SQLGetTypeInfo 인수

자료 유형	인수	사용	설명
SQLHSTMT	*StatementHandle*	입력	명령문 핸들.
SQLSMALLINT	*DataType*	입력	조회되는 SQL 자료 유형. 지원되는 유형은 다음과 같습니다. • SQL_ALL_TYPES • SQL_BIGINT • SQL_CHAR • SQL_DATE • SQL_DECIMAL • SQL_DOUBLE • SQL_FLOAT • SQL_GRAPHIC • SQL_INTEGER • SQL_NUMERIC • SQL_REAL • SQL_SMALLINT • SQL_TIME • SQL_TIMESTAMP • SQL_VARCHAR • SQL_VARGRAPHIC SQL_ALL_TYPES이 지정된 경우 TYPE_NAME에 의해 모든 지원되는 자료 유형이 오름차순으로 리턴됩니다. 모든 지원되지 않는 자료 유형은 결과 집합에 포함되지 않습니다.

사용법

SQLGetTypeInfo()는 결과 집합을 생성하고 조회를 실행하는 것과 마찬가지므로, 커서를 생성하고 트랜잭션을 시작합니다. 이 명령문 핸들에서 다른 명령문을 준비하여 실행하려면 커서를 닫아야 합니다.

SQLGetTypeInfo()을 유효하지 않은 *DataType*으로 호출하면 빈 결과 집합이 리턴됩니다.

이 함수에 의해 생성된 결과 집합의 열이 아래에 설명됩니다.

새로운 열이 추가되고 기존 열의 이름이 이후의 릴리스에서 변경될 수 있지만 현재 열의 위치는 변경되지 않습니다. 리턴되는 자료 유형은 CREATE TABLE, ALTER TABLE, DDL 명령문에 사용될 수 있습니다. 지속되지 않는 자료 유형은 리턴된 결과 집합의 일부가 아닙니다. 사용자 정의 자료 유형도 역시 리턴되지 않습니다.

표 110. SQLGetTypeInfo에 의해 리턴되는 열

열 번호/이름	자료 유형	설명
1 TYPE_NAME	VARCHAR(128) NOT NULL	SQL 자료 유형명의 문자 표현(예: VARCHAR, DATE, INTEGER).
2 DATA_TYPE	SMALLINT NOT NULL	SQL 자료 유형 정의 값(예: SQL_VARCHAR, SQL_DATE, SQL_INTEGER).
3 COLUMN_SIZE	INTEGER	자료 유형이 문자이거나 2진 스트링이면 이 열에는 최대 길이(바이트)가 있습니다. 그래픽(DBCS) 스트링인 경우 이 열에 대한 2바이트 문자 수입니다. 날짜, 시간, 시간소인 자료 유형인 경우, 문자로 변경될 때 값을 표시하기 위해 필요한 총 바이트 수입니다. 숫자 자료 유형인 경우 총 자릿수입니다.
4 LITERAL_PREFIX	VARCHAR(128)	DB2이 이 자료 유형의 리터럴에 대한 접두부로 인식하는 문자. 리터럴 접두부를 적용할 수 없는 자료 유형인 경우 이 열은 널입니다.
5 LITERAL_SUFFIX	VARCHAR(128)	DB2이 이 자료 유형의 리터럴에 대한 접미부로 인식하는 문자. 리터럴 접두부를 적용할 수 없는 자료 유형인 경우 이 열은 널입니다.
6 CREATE_PARAMS	VARCHAR(128)	TYPE_NAME의 이름을 SQL에서 자료 유형으로 사용할 때 어플리케이션이 괄호로 지정하는 각 매개변수에 대응하고, 쉼표로 분리되는 키워드 리스트가 이 열의 텍스트에 있습니다. 리스트의 키워드는 LENGTH, PRECISION, SCALE 중 하나일 수 있습니다. 키워드는 사용될 때 SQL 구문에서 요구하는 순서로 나타납니다. 자료 유형 정의(INTEGER와 같은)에 대한 매개변수가 없는 경우 널(null) 인디케이터가 리턴됩니다. **주:** CREATE_PARAMS의 목적은 어플리케이션이 *DDL 빌더*를 위한 인터페이스를 사용자 정의할 수 있도록 하는 것입니다. 이를 사용하여 어플리케이션은 자료 유형을 정의하기 위해 필요한 인수의 갯수를 결정하고 편집 제어의 레이블을 정하기 위해 사용할 수 있는 텍스트를 로컬화할 수만 있습니다.

표 110. SQLGetTypeInfo에 의해 리턴되는 열 (계속)

열 번호/이름	자료 유형	설명
7 NULLABLE	SMALLINT NOT NULL	자료 유형이 널(null)값을 허용하는 지를 표시합니다. • 널(null)값이 허용되지 않으면 SQL_NO_NULLS로 설정합니다. • 널(null)값이 허용되면 SQL_NULLABLE로 설정합니다.
8 CASE_SENSITIVE	SMALLINT NOT NULL	조사하기 위한 목적으로 자료 유형을 대소문자를 구분하여 취급할 수 있는 지를 표시합니다. 유효한 값은 SQL_TRUE와 SQL_FALSE입니다.
9 SEARCHABLE	SMALLINT NOT NULL	WHERE절에서 자료 유형을 사용하는 방법을 표시합니다. 유효한 값은 다음과 같습니다. • SQL_UNSEARCHABLE - 자료 유형이 WHERE절에서 사용될 수 없는 경우. • SQL_LIKE_ONLY - 자료 유형을 WHERE절에서 LIKE 술부와 함께만 사용할 수 있는 경우. • SQL_ALL_EXCEPT_LIKE - 자료 유형을 WHERE절에서 LIKE를 제외한 모든 비교 연산자와 함께 사용할 수 있는 경우. • SQL_SEARCHABLE - 자료 유형을 WHERE절에서 비교 연산자와 함께 사용할 수 있는 경우.
10 UNSIGNED_ATTRIBUTE	SMALLINT	자료 유형에 부호가 붙지 않을 수 있는 지를 표시합니다. 유효한 값은 SQL_TRUE, SQL_FALSE 또는 널(null)입니다. 이 속성이 자료 유형에 적용될 수 없는 경우 널(null) 인디케이터가 리턴됩니다.
11 FIXED_PREC_SCALE	SMALLINT NOT NULL	자료 유형이 정확한 숫자이고 항상 같은 정밀도와 소수 자릿수(scale)이면 SQL_TRUE가 있습니다. 그렇지 않으면 SQL_FALSE가 있습니다.
12 AUTO_INCREMENT	SMALLINT	행이 삽입될 때 이 자료 유형의 열이 고유한 값으로 설정되면 SQL_TRUE가 있습니다. 그렇지 않으면 SQL_FALSE가 있습니다.
13 LOCAL_TYPE_NAME	VARCHAR(128)	자료 유형의 일반 이름과는 다른 자료 유형에 대한 로컬화된(자국어) 이름이 이 열에 있습니다. 로컬화된 이름이 없는 경우 이 열은 널(null)입니다. 이 열은 화면에만 표시하기 위한 것입니다. 스트링의 문자 집합은 지역의 영향을 받으며 일반적으로 데이터베이스의 디폴트 문자 세트입니다.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

오류 조건

표 111. SQLGetTypeInfo SQLSTATEs

SQLSTATE	설명	설명
24000	유효하지 않은 커서 상태	커서가 명령문 핸들에서 이미 열렸습니다. *StatementHandle*가 닫히지 않았습니다.
40003 **08**S01	통신 링크 실패	함수가 완료되기 전에 어플리케이션과 자료 소스와의 통신 링크가 끊어졌습니다.
HY001	메모리 할당 실패	DB2 UDB CLI가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY004	SQL 자료 유형이 범위 밖의 값임	유효하지 않은 *DataType*이 지정되었습니다.
HY010	함수 순서 오류	함수가 실행시 자료(SQLParamData(), SQLPutData()) 조작중에 호출됩니다.
HYT00	시간 종료가 만료됨	

제한사항

다음 ODBC 지정 SQL 자료 유형(및 대응하는 *DataType* 정의 값)은 IBM RDBMS에 의해 지정되지 않습니다.

자료 유형	*DataType*
TINY INT	SQL_TINYINT
BIT	SQL_BIT

예

```
/* From CLI sample typeinfo.c */
/* ... */
    rc = SQLGetTypeInfo(hstmt, SQL_ALL_TYPES);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 1, SQL_C_CHAR, (SQLPOINTER) typename.s, 128, &typename.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 2, SQL_C_DEFAULT, (SQLPOINTER) & datatype,
                    sizeof(datatype), &datatype_ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 3, SQL_C_DEFAULT, (SQLPOINTER) & precision,
                    sizeof(precision), &precision_ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 7, SQL_C_DEFAULT, (SQLPOINTER) & nullable,
                    sizeof(nullable), &nullable_ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 8, SQL_C_DEFAULT, (SQLPOINTER) & casesens,
                    sizeof(casesens), &casesens_ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    printf("Datatype                  Datatype Precision  Nullable   Case\n");
```

```
printf("Typename                  (int)                      Sensitive\n");
printf("------------------------ -------- ---------- -------- ---------\n");
/* LONG VARCHAR FOR BIT DATA      99  2147483647   FALSE    FALSE */
/* Fetch each row, and display */
while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS) {
    printf("%-25s ", typename.s);
    printf("%8d ", datatype);
    printf("%10ld ", precision);
    printf("%-8s ", truefalse[nullable]);
    printf("%-9s\n", truefalse[casesens]);
}                           /* endwhile */

if ( rc != SQL_NO_DATA_FOUND )
   CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
```

참조

- 39 페이지의『SQLBindCol - 어플리케이션 변수에 열 바인드』
- 164 페이지의『SQLGetInfo - 일반 정보 얻기』

SQLLanguages - SQL 다이얼렉트 또는 적합성 정보 얻기

목적

SQLLanguages()는 SQL 다이얼렉트 또는 적합성 정보를 리턴합니다. SELECT문에 의해 생성된 결과 집합을 페치하기 위해 사용된 함수와 같은 함수를 사용하여 검색될 수 있는 SQL 결과 집합에 이 정보가 리턴됩니다.

구문

```
SQLRETURN SQLLanguages  (SQLHSTMT       hstmt);
```

함수 인수

표 112. SQLLanguages 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.

사용법

함수는 다이얼렉트와 적합성 정보를 StatementHandle의 결과 집합 형태로 리턴합니다. 여기에는 SQL 제품에서 요구하는 모든 적합성 요구 사항에 대한 행이 있습니다(ISO와 특정 제품 버전에 대해 정의된 서브세트 포함). 이 스펙에 따르기 위해 요구사항을 제기하는 제품의 경우, 결과 집합에는 최소한 한 행이 있습니다.

ISO 표준과 특정 제품 언어를 정의하는 행이 같은 표에 있을 수 있습니다. 각 행에는 최소한 이 열들이 있으며, X/Open SQL 적합성 요구사항을 제기하는 경우 열에는 이 값이 있습니다.

표 113. SQLLanguages에 의해 리턴되는 열

열명(column name)	자료 유형	설명
SOURCE	VARCHAR(254), NOT NULL	이 SQL 버전을 정의한 조직
SOURCE_YEAR	VARCHAR(254)	관련 소스 문서가 승인된 연도
CONFORMANCE	VARCHAR(254)	구현 프로그램이 요구사항을 제기한 관련 문서에 대한 적합성 레벨
INTEGRITY	VARCHAR(254)	구현 프로그램이 IEF(Integrity Enhancement Feature)를 지원하는 지에 대한 표시.
IMPLEMENTATION	VARCHAR(254)	업체의 SQL 제품을 식별하는 업체에 의해 정의된 문자 스트링.
BINDING_SYTLE	VARCHAR(254)	'EMBEDDED', 'DIRECT', 또는 'CLI'.
PROGRAMMING_LANG	VARCHAR(254)	바인딩 방식이 제공되는 호스트 언어.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR

SQLLanguages

- SQL_INVALID_HANDLE

진단

표 114. SQLLanguages SQLSTATE

SQLSTATE	설명	설명
24000	유효하지 않은 커서 상태	커서 관련 정보가 요구되었지만 열린 커서가 없습니다.
40003 *****	명령문 완료 상태를 알 수 없음	함수가 처리를 완료하기 전에 CLI와 자료 소스와의 통신 링크가 끊어졌습니다.
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 스트링 또는 버퍼 길이	이름 길이 인수 중 하나의 값이 0 미만이지만 SQL_NTS와 같지 않습니다.
HYC00	드라이버가 지원되지 않음	DB2 UDB CLI는 *catalog*를 표 이름에 대한 규정자로 지원하지 않습니다.

SQLMoreResults - 추가 결과 집합이 있는지 판별

목적

SQLMoreResults()는 결과 집합을 리턴하는 저장된 프로시듀어와 연관된 명령문 핸들에서 사용할 수 있는 추가 정보가 더 있는지 판별합니다.

구문

```
SQLRETURN   SQLMoreResults   (SQLHSTMT          StatementHandle);
```

함수 인수

표 115. SQLMoreResults 인수

자료 유형	인수	사용	설명
SQLHSTMT	StatementHandle	입력	명령문 핸들.

사용법

이 함수를 사용하여 SQL 조회문이 들어 있는 저장된 프로시듀어 실행시 순차적 방법으로 설정된 여러 결과를 리턴합니다. 저장된 프로시듀어가 실행이 끝날 때까지 결과 집합에 액세스할 수 있도록 커서는 열려 있습니다.

처음 결과 집합을 완전히 처리한 후 어플리케이션은 SQLMoreResults()를 호출하여 다른 결과 집합을 사용할 수 있는지 판별할 수 있습니다. 현재 결과 집합에 페치되지 않은 행이 있으면 SQLMoreResults()는 커서를 닫아서 이를 삭제하며, 다른 결과 집합이 있으면 SQL_SUCCESS를 리턴합니다.

모든 결과 집합이 처리되었으면 SQLMoreResults()는 SQL_NO_DATA_FOUND를 리턴합니다.

SQL_CLOSE 또는 SQL_DROP 옵션으로 SQLFreeStmt()를 호출하면 이 명령문 핸들 가운데 지연 중인 모든 집합이 삭제됩니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

오류 조건

표 116. SQLMoreResults SQLSTATEs

SQLSTATE	설명	설명
40003 **08**S01	통신 링크 실패	함수가 완료되기 전에 어플리케이션과 자료 소스와의 통신 링크가 끊어졌습니다.
58004	예기치 않은 시스템 실패	복구할 수 없는 시스템 오류
HY001	메모리 할당 실패	DB2 UDB CLI가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY010	함수 순서 오류	함수가 실행시 자료(SQLParamData(), SQLPutData()) 조작중에 호출됩니다.
HY013	예기치 않은 메모리 처리 오류	DB2 UDB CLI가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.
HYT00	시간 종료가 만료됨	

뿐만 아니라 SQLMoreResults()는 SQLExecute()와 연관된 SQLSTATE를 리턴할 수 있습니다.

제한사항

SQLMoreResults()의 ODBC 스펙에서는 리턴될 입력 매개변수 값 배열과 함께 매개변수화된 NSERT, UPDATE 및 DELETE문의 실행과 연관된 갯수를 허용합니다. 그러나 DB2 UDB CLI는 이러한 갯수 정보 리턴을 지원하지 않습니다.

참조

- 39 페이지의『SQLBindCol - 어플리케이션 변수에 열 바인드』
- 55 페이지의『SQLBindParameter - 버퍼에 매개변수 마커 바인드』

SQLNativeSql - 원래의 SQL 텍스트 얻기

목적

SQLNativeSql()을 사용하여 DB2 UDB CLI가 vendor escape 절을 분석하는 방법을 표시합니다. 어플리케이션에 의해 전달된 원래의 SQL 스트링에 일련의 vendor escape 절이 포함되면, DB2 UDB CLI는 자료 소스에 의해 표시되는 변경된 SQL 스트링을 리턴합니다(필요한 경우 삭제되었거나 변환된 vendor escape 절과 함께).

구문

```
SQLRETURN   SQLNativeSql     (SQLHDBC           ConnectionHandle,
                              SQLCHAR           *InStatementText,
                              SQLINTEGER        TextLength1,
                              SQLCHAR           *OutStatementText,
                              SQLINTEGER        BufferLength,
                              SQLINTEGER        *TextLength2Ptr);
```

함수 인수

표 117. SQLNativeSql 인수

자료 유형	인수	사용	설명
SQLHDBC	ConnectionHandle	입력	연결 핸들
SQLCHAR *	InStatementText	입력	입력 SQL 스트링
SQLINTEGER	TextLength1	입력	*InStatementText*의 길이
SQLCHAR *	OutStatementText	출력	변경된 출력 스트링에 대한 버퍼를 가리키는 포인터
SQLINTEGER	BufferLength	입력	*OutStatementText*가 가리키는 버퍼 크기
SQLINTEGER *	TextLength2Ptr	출력	*OutStatementText*에 리턴할 수 있는 총 바이트 수. 리턴할 수 있는 바이트 수가 *BufferLength*보다 크거나 같으면, *OutStatementText*의 출력 SQl 스트링이 *BufferLength - 1* 바이트로 절단됩니다. 출력 스트링이 생성되지 않으면 SQL_NULL_DATA 값이 리턴됩니다.

사용법

DB2 UDB CLI에 의해 자료 소스에 전달될 변경된 SQL 스트링을 어플리케이션이 조사하거나 표시하려는 경우 이 함수가 호출됩니다. 입력 SQL문 스트링에 일련의 vendor escape 절이 들어 있을 때만 변환(맵핑)이 됩니다.

iSeries에는 업체가 제공하는 이탈 순서가 없습니다. 이 프로시듀어는 호환을 위해 제공되는 것입니다. 또한 이 프로시듀어는 SQL 스트링의 구문 오류를 검사하기 위해 사용될 수도 있습니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

SQLNativeSql

- SQL_ERROR
- SQL_INVALID_HANDLE

오류 조건

표 118. SQLNativeSql SQLSTATEs

SQLSTATE	설명	설명
01004	자료가 절단됨	*OutStatementText*가 전체 SQL 스트링을 포함할 정도로 크기 않으므로 절단이 일어납니다. *TextLength2Ptr* 인수에는 절단되지 않은 SQL 스트링의 총 길이가 있습니다. (함수가 SQL_SUCCESS_WITH_INFO를 리턴합니다.)
08003	연결이 닫힘	*ConnectionHandle*은 열린 데이터베이스 연결을 참조하지 않습니다.
37000	유효하지 않은 SQL 구문	*InStatementText*의 입력 SQL 스트링에 구문 오류가 있습니다.
HY001	메모리 할당 실패	DB2 UDB CLI가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*InStatementText, OutStatementText* 또는 *TextLength2Ptr* 인수가 널(null) 포인터입니다.
HY090	유효하지 않은 스트링 또는 버퍼 길이	*TextLength1* 인수가 0 미만이지만 SQL_NTS와 같지 않습니다. *BufferLength* 인수가 0 미만입니다.

제한사항

없음

예

```
/* From CLI sample native.c */
/* ... */
    SQLCHAR in_stmt[1024], out_stmt[1024] ;
    SQLSMALLINT pcPar ;
    SQLINTEGER indicator ;
/* ... */
    /* Prompt for a statement to prepare */
    printf("Enter an SQL statement: \n");
    gets((char *)in_stmt);

    /* prepare the statement */
    rc = SQLPrepare(hstmt, in_stmt, SQL_NTS);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    SQLNumParams(hstmt, &pcPar);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    SQLNativeSql(hstmt, in_stmt, SQL_NTS, out_stmt, 1024, &indicator);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    if ( indicator == SQL_NULL_DATA ) printf( "Invalid statement\n" ) ;
    else {
       printf( "Input Statement: \n %s \n", in_stmt ) ;
       printf( "Output Statement: \n %s \n", in_stmt ) ;
       printf( "Number of Parameter Markers = %d\n", pcPar ) ;
```

```
}

rc = SQLFreeHandle( SQL_HANDLE_STMT, hstmt ) ;
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
```

참조

없음

SQLNextResult - 다음 결과 세트 처리

목적

SQLNextResult()는 결과 집합을 리턴하는 저장된 프로시듀어와 연관된 명령문 핸들에서 사용할 수 있는 추가 정보가 더 있는지 판별합니다.

구문

```
SQLRETURN   SQLNextResult   (SQLHSTMT          StatementHandle,
                             SQLHSTMT          NextResultHandle);
```

함수 인수

표 119. SQLNextResult 인수

자료 유형	인수	사용	설명
SQLHSTMT	StatementHandle	입력	명령문 핸들.
SQLHSTMT	NextResultHandle	입력	다음 결과 세트의 명령문 핸들.

사용법

이 함수는 StatementHandle에서 나온 다음 결과 세트를 NextResultHandle과 연관시키기 위해 사용됩니다. 이것은 결과 세트를 동시에 처리할 수 있도록 명령문 핸들을 모두에게 허용하므로 SQLMoreResults()와는 다릅니다.

모든 결과 집합이 처리되었으면 SQLNextResult()는 SQL_NO_DATA_FOUND를 리턴합니다.

SQL_CLOSE 또는 SQL_DROP 옵션으로 SQLFreeStmt()를 호출하면 이 명령문 핸들 가운데 지연 중인 모든 집합이 삭제됩니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

오류 조건

표 120. SQLNextResult SQLSTATEs

SQLSTATE	설명	설명
40003 **08**S01	통신 링크 실패	함수가 완료되기 전에 어플리케이션과 자료 소스와의 통신 링크가 끊어졌습니다.
58004	예기치 않은 시스템 실패	복구할 수 없는 시스템 오류

표 120. SQLNextResult SQLSTATEs (계속)

SQLSTATE	설명	설명
HY001	메모리 할당 실패	DB2 UDB CLI가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY010	함수 순서 오류	함수가 실행시 자료(SQLParamData(), SQLPutData()) 조작중에 호출됩니다.
HY013	예기치 않은 메모리 처리 오류	DB2 UDB CLI가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.
HYT00	시간 종료가 만료됨	

참조

• 197 페이지의『SQLMoreResults - 추가 결과 집합이 있는지 판별』

SQLNumParams - SQL문의 매개변수 갯수 얻기

목적

SQLNumParams()은 SQL문의 매개변수 마커 수를 리턴합니다.

구문

```
SQLRETURN   SQLNumParams     (SQLHSTMT          StatementHandle,
                              SQLSMALLINT       *ParameterCountPtr);
```

함수 인수

표 121. SQLNumParams 인수

자료 유형	인수	사용	설명
SQLHSTMT	StatementHandle	입력	명령문 핸들.
SQLSMALLINT *	ParameterCountPtr	출력	명령문의 매개변수 갯수

사용법

이 함수는 *StatementHandle*과 연관된 명령문이 준비된 후에만 호출될 수 있습니다. 명령문에 매개변수 마커가 포함되어 있지 않으면 *ParameterCountPtr*이 0으로 설정됩니다.

어플리케이션은 이 함수를 호출하여 명령문 핸들과 연관된 SQL문에 필요한 SQLBindParameter() 호출의 수를 결정할 수 있습니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

오류 조건

표 122. SQLNumParams SQLSTATEs

SQLSTATE	설명	설명
40003 **08**S01	통신 링크 실패	함수가 완료되기 전에 어플리케이션과 자료 소스와의 통신 링크가 끊어졌습니다.
HY001	메모리 할당 실패	DB2 UDB CLI가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY008	조작 취소	
HY009	유효하지 않은 인수 값	*ParameterCountPtr*이 널입니다.

표 122. SQLNumParams SQLSTATEs (계속)

SQLSTATE	설명	설명
HY010	함수 순서 오류	지정된 *StatementHandle*에 SQLPrepare()가 호출되기 전에 이 함수가 호출됩니다. 함수가 실행시 자료(SQLParamData(), SQLPutData()) 조작중에 호출됩니다.
HY013	예기치 않은 메모리 처리 오류	DB2 UDB CLI가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.
HYT00	시간 종료가 만료됨	

제한사항

없음

예

SQLNativeSql() 200 페이지의 『예』를 참조하십시오.

참조

- 50 페이지의 『SQLBindParam - 매개변수 마커에 버퍼 바인드』
- 212 페이지의 『SQLPrepare - 명령문 준비』

SQLNumResultCols - 결과 열의 수 얻기

목적

SQLNumResultCols()은 입력 명령문 핸들과 연관된 결과 집합의 열 수를 리턴합니다.

이 함수를 호출하기 전에 SQLPrepare() 또는 SQLExecDirect()를 호출해야 합니다.

이 함수를 호출한 후에 SQLDescribeCol(), SQLColAttributes(), SQLBindCol() 또는 SQLGetData()를 호출할 수 있습니다.

구문

```
SQLRETURN SQLNumResultCols (SQLHSTMT       hstmt,
                            SQLSMALLINT    *pccol);
```

함수 인수

표 123. SQLNumResultCols 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLSMALLINT *	*pccol*	출력	결과 집합의 열 수

사용법

입력 명령문 핸들에서 실행된 마지막 명령문이 SELECT문이 아닌 경우 함수는 출력 인수를 0으로 설정합니다.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 124. SQLNumResultCols SQLSTATEs

SQLSTATE	설명	설명
40003 *****	명령문 완료 상태를 알 수 없음	함수가 처리를 완료하기 전에 CLI와 자료 소스와의 통신 링크가 끊어졌습니다.
58004	시스템 오류	복구할 수 없는 시스템 오류
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*pcbCol*가 널(null) 포인터입니다.
HY010	함수 순서 오류	*hstmt*에 대해 SQLPrepare 또는 SQLExecDirect를 호출하기 전에 함수가 호출됩니다.

표 124. SQLNumResultCols SQLSTATEs (계속)

SQLSTATE	설명	설명
S1013 *****	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

참조

- 39 페이지의『SQLBindCol - 어플리케이션 변수에 열 바인드』
- 65 페이지의『SQLColAttributes - 열 속성』
- 84 페이지의『SQLDescribeCol - 열 속성 설명』
- 102 페이지의『SQLExecDirect - 명령문 직접 실행』
- 132 페이지의『SQLGetCol - 결과 집합 행의 한 열을 검색』
- 212 페이지의『SQLPrepare - 명령문 준비』

SQLParamData - 자료 값이 필요한 다음 매개변수 얻기

목적

SQLParamData()을 SQLPutData()와 함께 사용하여 긴 자료를 나누어 송신합니다. 고정된 길이의 자료를 송신할 수도 있습니다.

구문

```
SQLRETURN SQLParamData  (SQLHSTMT     hstmt,
                         SQLPOINTER   *prgbValue);
```

함수 인수

표 125. SQLParamData 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLPOINTER *	*prgbValue*	출력	SQLSetParam 호출에 지정된 *rgbValue* 인수의 값을 가리키는 포인터.

사용법

아직 자료가 할당되지 않은 최소한 하나의 SQL_DATA_AT_EXEC가 있는 경우 SQLParamData()는 SQL_NEED_DATA를 리턴합니다. 이 함수는 이전 SQLBindParam() 호출에서 어플리케이션에 의해 제공된 *prgbValue*에 어플리케이션이 정의한 값을 리턴합니다. SQLPutData()이 한 번 이상 호출되어 매개변수 자료를 송신합니다. SQLParamData()이 호출되어 현재 매개변수에 대해 모든 자료가 송신되었음을 알리고 다음 SQL_DATA_AT_EXEC 매개변수로 진행합니다. 모든 매개변수가 자료 값을 할당하고 연관된 명령문이 성공적으로 실행되면 SQL_SUCCESS가 리턴됩니다. 실제 명령문 실행 동안이나 전에 오류가 발생하면 SQL_ERROR가 리턴됩니다.

SQLParamData()가 SQL_NEED_DATA를 리턴하면, SQLPutData() 또는 SQLCancel() 호출이 발생합니다. 이 명령문 핸들을 사용한 모든 다른 함수 호출은 실패합니다. 뿐만 아니라, *hstmt*의 상위 *hdbc*를 참조하는 모든 함수 호출이 속성이나 해당 연결 상태의 변경과 관련되면 실패합니다. 상위 *hdbc*의 다음 함수 호출도 허용되지 않습니다.

- SQLAllocConnect()
- SQLAllocHandle()
- SQLAllocStmt()
- SQLSetConnectOption()

SQL_NEED_DATA 순서 중에 호출되어야 하는 경우 이 함수는 **HY**010의 SQLSTATE와 함께 SQL_ERROR를 리턴하며 SQL_DATA_AT_EXEC 매개변수 처리는 영향을 받지 않습니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NEED_DATA

진단

SQLParamData()는 SQLExecDirect()와 SQLExecute() 함수에 의해 리턴된 SQLSTATE를 리턴할 수 있습니다. 뿐만 아니라 다음 진단 정보도 생성될 수 있습니다.

표 126. SQLParamData SQLSTATEs

SQLSTATE	설명	설명
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*prgbValue* 인수가 널(null) 포인터입니다.
HY010	함수 순서 오류	SQLParamData()가 올바르지 않은 순서로 호출됩니다. 이 호출은 SQLExecDirect() 또는 SQLExecute() 뒤 또는 SQLPutData() 호출 뒤에만 유효합니다.
HYDE0	지연 중인 실행 값에 자료가 없음	SQLExecDirect() 또는 SQLExecute() 호출 뒤에 이 함수가 호출되었지만 처리할 (남은) SQL_DATA_AT_EXEC 매개변수가 없습니다.

SQLParamOptions - 매개변수에 대한 입력 배열 지정

목적

SQLParamOptions()을 사용하여 SQLBindParameter()에 의해 설정된 각 매개변수에 대한 여러 값을 설정할 수 있습니다. 따라서 어플리케이션은 SQLExecute() 또는 SQLExecDirect()의 단일 호출로 표에 여러 행을 삽입할 수 있습니다.

구문

```
SQLRETURN   SQLParamOptions  (SQLHSTMT          StatementHandle,
                              SQLINTEGER        Crow,
                              SQLINTEGER        *FetchOffsetPtr);
```

함수 인수

표 127. SQLParamOptions 인수

자료 유형	인수	사용	설명
SQLHSTMT	*StatementHandle*	입력	명령문 핸들.
SQLINTEGER	*Crow*	입력	각 매개변수에 대한 값의 갯수. 1보다 크면 SQLBindParameter()의 *rgbValue*가 매개변수 값 배열을 가리키고 *pcbValue*는 길이 배열을 가리킵니다.
SQLINTEGER *	*FetchOffsetPtr*	출력(지연됨)	현재 사용되지 않음

사용법

이 함수는 SQLBindParameter()와 함께 사용되어 여러 행의 INSERT문을 설정할 수 있습니다. 이를 위해 어플리케이션은 삽입되는 모든 자료를 위한 기억장치를 할당해야 합니다. 이 자료는 행 방식 형태로 구성되어야 합니다. 이는 첫 행에 대한 모든 자료가 연속적으로 다음 행에 대한 모든 자료가 뒤이어 나와야 함을 의미합니다. 여러 행의 INSERT문일 경우 SQLBindParameter()에 제공된 주소가 자료의 첫 행을 참조하기 위해 사용됩니다. 전체 행 길이만큼 주소를 증가하여 모든 후속 자료 행이 참조됩니다.

예를 들어, 어플리케이션이 표에 100행의 자료를 삽입하려 하는데 각 행에는 4바이트 정수 값이 포함되어 있고 그 뒤에 10바이트 문자 값이 있습니다. 어플리케이션은 1400바이트의 기억장치를 할당하고 각각의 14바이트씩의 기억장치 조각을 행에 대해 적절한 자료로 채웁니다.

또한 SQLBindParameter()에 전달된 인디케이터 포인터는 800바이트의 기억장치 일부를 참조해야 합니다. 이 기억장치는 널 인디케이터 값을 전달하는데 사용됩니다. 이 저장장치도 행 방식이므로, 처음 8바이트는 첫 행에 대한 2개의 인디케이터이며 그 뒤에 계속 다음 행에 대한 인디케이터가 나옵니다. 어플리케이션에 의해 SQLParamOptions() 함수가 사용되어 다음 번 INSERT문 실행시 명령문 핸들을 사용하여 삽입되는 행 수를 지정합니다. INSERT문은 복수 행 형식이어야 합니다.

예: INSERT INTO CORPDATA.NAMES ? ROWS VALUES(?, ?)

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

오류 조건

표 128. SQLParamOptions SQLSTATEs

SQLSTATE	설명	설명
HY009	유효하지 않은 인수 값	*Crow* 인수의 값이 1 미만입니다.
HY010	함수 순서 오류	함수가 실행시 자료(SQLParamData(), SQLPutData()) 조작중에 호출됩니다.

제한사항

없음

참조

- 50 페이지의『SQLBindParam - 매개변수 마커에 버퍼 바인드』
- 197 페이지의『SQLMoreResults - 추가 결과 집합이 있는지 판별』

SQLPrepare - 명령문 준비

목적

SQLPrepare()는 SQL문을 입력 명령문 핸들과 연관시키고 명령문을 준비할 데이터베이스 관리 시스템(DBMS)으로 송신합니다. 명령문 핸들을 다른 함수에 전달하여 어플리케이션은 이 준비된 명령문을 참조할 수 있습니다.

명령문 핸들이 SELECT문과 함께 사용되면 SQLPrepare()를 호출하기 전에 SQLFreeStmt()를 호출하여 커서를 닫아야 합니다.

구문

```
SQLRETURN SQLPrepare (SQLHSTMT       hstmt,
                      SQLCHAR        *szSqlStr,
                      SQLINTEGER     cbSqlStr);
```

함수 인수

표 129. SQLPrepare 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들. *hstmt*와 연관된 열린 커서가 없어야 합니다.
SQLCHAR *	*szSqlStr*	입력	SQL문 스트링.
SQLINTEGER	*cbSqlStr*	입력	*szSqlStr* 인수의 내용 길이. 길이는 *szSqlstr*의 SQL문의 정확한 길이 또는 명령문이 널로 종료될 경우 SQL_NTS로 설정되어야 합니다.

사용법

SQLPrepare()를 사용하여 일단 명령문이 준비되면, 어플리케이션은 다음 함수를 호출하여 결과 집합(SELECT문이었던 경우)의 형식에 대한 정보를 요구할 수 있습니다.

- SQLNumResultCols()
- SQLDescribeCol()
- SQLColAttributes()

준비된 명령문은 SQLExecute()를 호출하여 여러 번 또는 한 번 실행될 수 있습니다. SQL문은 명령문 핸들이 다른 SQLPrepare(), SQLExecDirect(), SQLColumns(), SQLSpecialColumns(), SQLStatistics() 또는 SQLTables()와 함께 사용될 때까지 명령문 핸들과 연관되어 있습니다.

SQL문 스트링에는 매개변수 마커가 있을 수 있습니다. 매개변수 마커는 "?" 문자로 표시되며 SQLExecute() 가 호출될 때 어플리케이션 변수 값이 대체되어야 할 명령문의 위치를 지정합니다. SQLBindParam()은 어플리케이션 변수를 각 매개변수 마커에 바인드(연관)시켜서 자료가 전송될 때 자료 변환이 수행되어야 하는지를 표시합니다.

SQL문은 COMMIT 또는 ROLLBACK문일 수 없습니다. 대신, COMMIT 또는 ROLLBACK을 발행하기 위해 SQLTransact()가 호출되어야 합니다.

SQL문이 위치지정된 DELETE 또는 위치지정된 UPDATE문이면 명령문에 의해 참조된 커서는 같은 연결 핸들 아래의 분리된 명령문 핸들에 정의되어야 합니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 130. SQLPrepare SQLSTATEs

SQLSTATE	설명	설명
24000	유효하지 않은 커서 상태	지정된 *hstmt*에 열린 커서가 있습니다.
37xxx	구문 오류 또는 액세스 위반	*szSqlStr*에 다음 중 하나 이상이 있습니다. • COMMIT • ROLLBACK • 연결된 데이터베이스 서버가 준비할 수 없는 SQL문 • 구문 오류가 있는 명령문
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*szSqlStr*이 널(null) 포인터입니다. *cbSqlStr* 인수가 1 미만이었으나 SQL_NTS와 같지 않습니다.
HY013 *	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

주: 준비시에 모든 데이터베이스 관리 시스템(DBMS)이 모든 위의 진단 메세지를 보고하는 것은 아닙니다. 그러므로 SQLExecute()를 호출할 때 어플리케이션은 이 상태도 핸들할 수 있어야 합니다.

예

다음 예에서 사용된 check_error, initialize, terminate 함수의 리스트는 300 페이지의 『예: 대화식 SQL 및 그에 해당되는 DB2 UDB CLI 함수 호출』을 참조하십시오.

SQLPrepare

```
/*****************************************************************************
** file = prepare.c
**
** Example of preparing then repeatedly executing an SQL statement.
**
** Functions used:
**
**        SQLAllocConnect      SQLFreeConnect
**        SQLAllocEnv          SQLFreeEnv
**        SQLAllocStmt         SQLFreeStmt
**        SQLConnect           SQLDisconnect
**
**        SQLBindCol           SQLFetch
**        SQLTransact          SQLError
**        SQLPrepare           SQLSetParam
**        SQLExecute
*****************************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "sqlcli.h"

#define MAX_STMT_LEN 255

int initialize(SQLHENV *henv,
               SQLHDBC *hdbc);

int terminate(SQLHENV henv,
              SQLHDBC hdbc);

int print_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt);

int check_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt,
                 SQLRETURN  rc);

/*****************************************************************
** main
** - initialize
** - terminate
*****************************************************************/
int main()
{
    SQLHENV     henv;
    SQLHDBC     hdbc;
    SQLCHAR     sqlstmt[MAX_STMT_LEN + 1]="";
    SQLRETURN   rc;

    rc = initialize(&henv, &hdbc);
    if (rc == SQL_ERROR) return(terminate(henv, hdbc));

    {SQLHSTMT   hstmt;
     SQLCHAR    sqlstmt[]="SELECT deptname, location from org where division = ?";
     SQLCHAR    deptname[15],
```

```
            location[14],
            division[11];

 SQLINTEGER rlength,
            plength;

    rc = SQLAllocStmt(hdbc, &hstmt);
    if (rc != SQL_SUCCESS )
    check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

    /* prepare statement for multiple use */
    rc = SQLPrepare(hstmt, sqlstmt, SQL_NTS);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, hstmt, rc);

    /* bind division to parameter marker in sqlstmt */
    rc = SQLSetParam(hstmt, 1, SQL_CHAR, SQL_CHAR, 10, 10, division,
                &plength);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, hstmt, rc);

    /* bind deptname to first column in the result set */
    rc = SQLBindCol(hstmt, 1, SQL_CHAR, (SQLPOINTER) deptname, 15,
                    &rlength);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, hstmt, rc);
    rc = SQLBindCol(hstmt, 2, SQL_CHAR, (SQLPOINTER) location, 14,
                    &rlength);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, hstmt, rc);

    printf("\nEnter Division Name or 'q' to quit:\n");
    printf("(Eastern, Western, Midwest, Corporate)\n");
    gets(division);
    plength = SQL_NTS;

    while(division[0] != 'q')
    {
        rc = SQLExecute(hstmt);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt, rc);

        printf("Departments in %s Division:\n", division);
        printf("DEPTNAME       Location\n");
        printf("-------------- -------------\n");

        while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS)
        {
            printf("%-14.14s %-13.13s \n", deptname, location);
        }
        if (rc != SQL_NO_DATA_FOUND )
            check_error (henv, hdbc, hstmt, rc);
        SQLFreeStmt(hstmt, SQL_CLOSE);
        printf("\nEnter Division Name or 'q' to quit:\n");
        printf("(Eastern, Western, Midwest, Corporate)\n");
        gets(division);
    }
}
```

```
    rc = SQLTransact(henv, hdbc, SQL_ROLLBACK);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

    terminate(henv, hdbc);
    return (0);
}/* end main */
```

참조

- 65 페이지의『SQLColAttributes - 열 속성』
- 84 페이지의『SQLDescribeCol - 열 속성 설명』
- 102 페이지의『SQLExecDirect - 명령문 직접 실행』
- 104 페이지의『SQLExecute - 명령문 실행』
- 206 페이지의『SQLNumResultCols - 결과 열의 수 얻기』

SQLPrimaryKeys - 표의 1차 키 열 얻기

목적

SQLPrimaryKeys()는 표에 대한 1차 키를 의미하는 열명(column name) 리스트를 리턴합니다. 조회에 의해 생성된 결과 집합을 처리하기 위해 사용된 함수와 같은 함수를 사용하여 검색될 수 있는 SQL 결과 집합에 이 정보가 리턴됩니다.

구문

```
SQLRETURN   SQLPrimaryKeys   (SQLHSTMT          StatementHandle,
                              SQLCHAR           *CatalogName,
                              SQLSMALLINT       NameLength1,
                              SQLCHAR           *SchemaName,
                              SQLSMALLINT       NameLength2,
                              SQLCHAR           *TableName,
                              SQLSMALLINT       NameLength3);
```

함수 인수

표 131. SQLPrimaryKeys 인수

자료 유형	인수	사용	설명
SQLHSTMT	StatementHandle	입력	명령문 핸들.
SQLCHAR *	CatalogName	입력	세 부분으로 된 표 이름의 카탈로그 규정자. 이 포인터는 널(null) 포인터이거나 길이가 0인 스트링이어야 합니다.
SQLSMALLINT	NameLength1	입력	*CatalogName*의 길이.
SQLCHAR *	SchemaName	입력	표 이름의 스키마 규정자.
SQLSMALLINT	NameLength2	입력	*SchemaName*의 길이.
SQLCHAR *	TableName	입력	표 이름.
SQLSMALLINT	NameLength3	입력	*TableName*의 길이.

사용법

SQLPrimaryKeys()는 하나의 표에서 1차 키 열을 리턴하며 탐색 패턴은 스키마 규정자 또는 표 이름을 지정하기 위해 사용될 수 없습니다.

결과 집합에는 218 페이지의 표 132에 나열된 열을 있으며 TABLE_CAT, TABLE_SCHEM, TABLE_NAME, ORDINAL_POSITION에 의해 순서가 정해집니다.

많은 경우에 있어서 SQLPrimaryKeys() 호출은 시스템 카탈로그에 반해 복잡하고 비용이 많이 드는 조회에 맵핑되므로 자주 사용하지 않는 것이 좋으며 반복 호출이 아닌 결과를 저장합니다.

새로운 열이 추가되고 기존 열의 이름이 이후의 릴리스에서 변경될 수 있지만 현재 열의 위치는 변경되지 않습니다.

SQLPrimaryKeys

표 132. SQLPrimaryKeys에 의해 리턴되는 열

열 번호/이름	자료 유형	설명
1 TABLE_CAT	VARCHAR(128)	현재 서버.
2 TABLE_SCHEM	VARCHAR(128)	TABLE_NAME이 있는 스키마명.
3 TABLE_NAME	VARCHAR(128)는 널(null)이 아님	지정된 표의 이름.
4 COLUMN_NAME	VARCHAR(128)는 널(null)이 아님	1차 키 열명.
5 ORDINAL_POSITION	SMALLINT는 널(null)이 아님	1부터 시작하는 1차 키의 열 순서 번호.
6 PK_NAME	VARCHAR(128)	1차 키 ID. 자료 소스에 적용할 수 없는 경우 널(null).

주: DB2 UDB CLI에 의해 사용된 열명(column name)은 X/Open CLI CAE 스펙 방식을 준수합니다. 열 유형, 내용과 순서는 ODBC의 SQLPrimaryKeys() 결과 집합에 대해 정의된 값과 같습니다.

지정된 표에 1차 키가 없으면 빈 결과 집합이 리턴됩니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

오류 조건

표 133. SQLPrimaryKeys SQLSTATEs

SQLSTATE	설명	설명
24000	유효하지 않은 커서 상태	커서가 명령문 핸들에서 이미 열렸습니다.
40003 **08**S01	통신 링크 실패	함수가 완료되기 전에 어플리케이션과 자료 소스와의 통신 링크가 끊어졌습니다.
HY001	메모리 할당 실패	DB2 UDB CLI가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY008	조작 취소	
HY010	함수 순서 오류	함수가 실행시 자료(SQLParamData(), SQLPutData()) 조작중에 호출됩니다.
HY014	더 이상의 핸들이 없음	DB2 UDB CLI는 내부 자원 때문에 핸들을 할당할 수 없습니다.
HY090	유효하지 않은 스트링 또는 버퍼 길이	이름 길이 인수 중 하나의 값이 0 미만이지만 SQL_NTS와 같지 않습니다.
HYC00	드라이버가 지원되지 않음	DB2 UDB CLI는 *catalog*를 표 이름에 대한 규정자로 지원하지 않습니다.
HYT00	시간 종료가 만료됨	

제한사항

없음

참조

- 118 페이지의『SQLForeignKeys - 외부 키 열 리스트 얻기』
- 264 페이지의『SQLStatistics - 기본 표에 대한 색인과 통계 정보 얻기』

SQLProcedureColumns - 프로시듀어에 대한 입/출력(I/O) 매개변수 정보 얻기

목적

SQLProcedureColumns()는 프로시듀어와 연관된 입력과 출력 매개변수 목록을 리턴합니다. 조회에 의해 생성된 결과 집합을 처리하기 위해 사용된 함수와 같은 함수를 사용하여 검색될 수 있는 SQL 결과 집합에 이 정보가 리턴됩니다.

구문

```
SQLRETURN SQLProcedureColumns(SQLHSTMT          StatementHandle,
                              SQLCHAR           *CatalogName,
                              SQLSMALLINT       NameLength1,
                              SQLCHAR           *SchemaName,
                              SQLSMALLINT       NameLength2,
                              SQLCHAR           *ProcName,
                              SQLSMALLINT       NameLength3,
                              SQLCHAR           *ColumnName,
                              SQLSMALLINT       NameLength4);
```

함수 인수

표 134. SQLProcedureColumns 인수

자료 유형	인수	사용	설명
SQLHSTMT	StatementHandle	입력	명령문 핸들.
SQLCHAR *	CatalogName	입력	세 부분으로 된 프로시듀어명의 카탈로그 규정자. 이 포인터는 널(null) 포인터이거나 길이가 0인 스트링이어야 합니다.
SQLSMALLINT	NameLength1	입력	*CatalogName*의 길이. 0으로 설정되어야 합니다.
SQLCHAR *	SchemaName	입력	스키마명에 의해 결과 집합을 규정하기 위해 *패턴 값*을 포함하는 버퍼. zOS 및 OS/390용 DB2 UDB V 4.1의 경우 모든 저장 프로시듀어가 하나의 스키마에 있으며 *SchemaName* 인수에 허용되는 유일한 값이 널(null) 포인터입니다. DB2 UDB의 경우™*SchemaName*에 유효한 패턴 값이 있을 수 있습니다.
SQLSMALLINT	NameLength2	입력	*SchemaName*의 길이.
SQLCHAR *	ProcName	입력	프로시듀어명에 의해 결과 집합을 규정하기 위해 *패턴 값*을 포함하는 버퍼.
SQLSMALLINT	NameLength3	입력	*ProcName*의 길이.
SQLCHAR *	ColumnName	입력	매개변수명에 의해 결과 집합을 규정하기 위해 *패턴 값*을 포함하는 버퍼. ProcName 또는 SchemaName에 대한 비어 있지 않은 값을 지정하여 이미 제한된 결과 집합을 더 규정하기 위해 이 인수가 사용됩니다.
SQLSMALLINT	NameLength4	입력	*ColumnName*의 길이.

사용법

DB2 UDB CLI는 프로시듀어와 연관된 입력, 입력과 출력, 출력 매개변수 정보를 리턴하지만 리턴된 결과 집합에 대한 설명자 정보에 대한 정보는 리턴할 수 없습니다.

SQLProcedureColumns()는 PROCEDURE_CAT, PROCEDURE_SCHEM, PROCEDURE_NAME, COLUMN_TYPE에 의해 순서가 정해진 결과 집합에 정보를 리턴합니다. 표 135는 결과 집합의 열을 나열합니다. 어플리케이션은 마지막 열 이후의 열이 향후 릴리스시에 정의될 수 있음을 알아야 합니다.

많은 경우에서 SQLProcedureColumns() 호출은 시스템 카탈로그에 반해 복잡하고 비용이 많이 드는 조회에 맵핑되므로 자주 사용하지 않는 것이 좋으며 반복 호출이 아닌 결과를 저장합니다.

표 135. SQLProcedureColumns에 의해 리턴되는 열

열 번호/이름	자료 유형	설명
1 PROCEDURE_CAT	VARCHAR(128)	현재 서버.
2 PROCEDURE_SCHEM	VARCHAR(128)	PROCEDURE_NAME이 들어 있는 스키마명.
3 PROCEDURE_NAME	VARCHAR(128)	프로시듀어명
4 COLUMN_NAME	VARCHAR(128)	매개변수명
5 COLUMN_TYPE	SMALLINT는 널(null)이 아님	이 행과 연관된 유형 정보를 식별합니다. 다음과 같은 값이 될 수 있습니다. • SQL_PARAM_TYPE_UNKNOWN - 매개변수 유형을 알 수 없습니다. **주:** 리턴되지 않습니다. • SQL_PARAM_INPUT - 이 매개변수는 입력 매개변수입니다. • SQL_PARAM_INPUT_OUTPUT - 이 매개변수는 입/출력 매개변수입니다. • SQL_PARAM_OUTPUT - 이 매개변수는 출력 매개변수입니다. • SQL_RETURN_VALUE - 프로시듀어 열은 프로시듀어의 리턴 값입니다. **주:** 리턴되지 않습니다. • SQL_RESULT_COL - 이 매개변수는 실제로는 결과 집합의 열입니다. **주:** 리턴되지 않습니다.
6 DATA_TYPE	SMALLINT는 널(null)이 아님	SQL 자료 유형
7 TYPE_NAME	VARCHAR(128)는 널(null)이 아님	DATA_TYPE에 대응하는 자료 유형명을 나타내는 문자 스트링.

SQLProcedureColumns

표 135. SQLProcedureColumns에 의해 리턴되는 열 (계속)

열 번호/이름	자료 유형	설명
8 COLUMN_SIZE	INTEGER	DATA_TYPE 열 값이 문자이거나 2진 스트링을 의미하면, 이 열에는 최대 길이(바이트)가 있습니다. 그래픽(DBCS) 스트링인 경우 이 매개변수에 대한 2바이트 문자 수입니다. 날짜, 시간, 시간소인 자료 유형인 경우, 문자로 변경될 때 값을 표시하기 위해 필요한 총 바이트 수입니다. 숫자 자료 유형인 경우, 결과 집합의 NUM_PREC_RADIX 열의 값에 따라 열에 허용된 총 비트 수이거나 총 자릿수입니다.
9 BUFFER_LENGTH	INTEGER	SQL_C_DEFAULT가 SQLBindCol(), SQLGetData()와 SQLBindParameter() 호출에서 지정된 경우, 이 매개변수에서부터 자료를 저장하는 연관된 C 버퍼에 대한 최대 바이트 수. 이 길이에 널 종료자는 제외됩니다. 정확한 숫자 자료 유형을 위해 길이는 십진수와 부호만 계산합니다.
10 DECIMAL_DIGITS	SMALLINT	매개변수의 소수 자릿수(scale). 소수 자릿수를 적용할 수 없는 자료 유형에 대해서는 널(null)이 리턴됩니다.
11 NUM_PREC_RADIX	SMALLINT	10, 2 또는 널(null)입니다. DATA_TYPE이 대략적인 숫자 자료 유형인 경우 이 열에는 값 2가 있으며 COLUMN_SIZE 열에는 이 매개변수에 허용된 비트 수가 있습니다. DATA_TYPE이 정확한 숫자 자료 유형인 경우 이 열에는 값 10이 있으며 COLUMNP_SIZE와 DECIMAL_DIGITS 열에는 이 매개변수에 허용된 십진 자릿수가 있습니다. 숫자 자료 유형의 경우 데이터베이스 관리 시스템(DBMS)은 10이나 2의 NUM_PREC_RADIX를 리턴할 수 있습니다. 기수(radix)를 적용할 수 없는 자료 유형의 경우 널(null)이 리턴됩니다.
12 NULLABLE	VARCHAR(3)	매개변수가 NULL을 허용하지 않으면 'NO'입니다. 매개변수가 NULL을 허용하면 'YES'입니다.
13 REMARKS	VARCHAR(254)	매개변수에 대한 설명 정보를 포함할 수 있습니다.
14 COLUMN_DEF	VARCHAR	열의 디폴트 값입니다. NULL이 디폴트 값으로 지정되면 이 열은 작은 따옴표로 묶지 않은 단어 NULL이 있습니다. 자르지 않고 디폴트 값을 나타낼 수 없으면 작은 따옴표로 묶지 않은 TRUNCATED가 있습니다. 디폴트 값이 지정되지 않으면 이 열은 NULL입니다. TRUNCATED 값을 제외하고 COLUMN_DEF 값이 새로운 열 정의를 작성할 때 사용됩니다.
15 SQL_DATA_TYPE	SMALLINT는 널(null)이 아님	설명자의 SQL_DESC_TYPE 필드에 나오는 SQL 자료 유형의 값. datetime 자료 유형을 제외하고 이 열은 DATA_TYPE 열과 같습니다. (DB2 UDB CLI는 interval 자료 유형을 지원하지 않습니다.) datetime 자료 유형의 경우 결과 세트에서 SQL_DATA_TYPE 필드가 SQL_DATETIME이며, SQL_DATETIME_SUB 필드가 특정 datetime 자료 유형(SQL_CODE_DATE, SQL_CODE_TIME 또는 SQL_CODE_TIMESTAMP)에 대해 서브코드를 리턴합니다.

표 135. SQLProcedureColumns에 의해 리턴되는 열 (계속)

열 번호/이름	자료 유형	설명
16 SQL_DATETIME_SUB	SMALLINT	datetime 자료 유형의 subtype 코드. 다른 모든 자료 유형의 경우 이 열이 NULL을 리턴합니다(DB2 UDB CLI가 지원하지 않는 interval 자료 유형 포함).
17 CHAR_OCTET_LENGTH	INTEGER	문자 자료 유형 열에 대한 바이트 단위의 최대 길이. 다른 모든 자료 유형의 경우 이 열이 NULL을 리턴합니다.
18 ORDINAL_POSITION	INTEGER NOT NULL	이 결과 세트에 COLUMN_NAME이 제공하는 매개변수의 서수 (ordinal) 위치를 포함하고 있습니다. 이것은 CALL 명령문에 제공될 인수의 서수 위치입니다. 왼쪽 끝의 인수가 서수 자리 1을 사용합니다.
19 IS_NULLABLE	VARCHAR	• "NO": 열에 NULL이 없을 경우. • "YES": 열에 NULL이 있을 경우. • 0 길이 스트링: NULL 사용 가능 여부를 알 수 없을 경우. ISO 규칙에 따라 NULL 사용 가능 여부가 결정됩니다. ISO SQL-준수 DBMS는 빈 스트링을 리턴할 수 없습니다. 이 열에 리턴되는 값은 NULLABLE 열에 리턴되는 값과 다릅니다. (NULLABLE 열의 설명을 참조하십시오.)

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

오류 조건

표 136. SQLProcedureColumns SQLSTATEs

SQLSTATE	설명	설명
24000	유효하지 않은 커서 상태	커서가 명령문 핸들에서 이미 열렸습니다.
40003 **08**S01	통신 링크 실패	함수가 완료되기 전에 어플리케이션과 자료 소스와의 통신 링크가 끊어졌습니다.
42601	PARMLIST 구문 오류	저장된 프로시듀어 카탈로그 표의 PARMLIST 값에 구문 오류가 있습니다.
HY001	메모리 할당 실패	DB2 UDB CLI가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY008	조작 취소	
HY010	함수 순서 오류	
HY014	더 이상의 핸들이 없음	DB2 UDB CLI는 내부 자원 때문에 핸들을 할당할 수 없습니다.

표 136. SQLProcedureColumns SQLSTATEs (계속)

SQLSTATE	설명	설명
HY090	유효하지 않은 스트링 또는 버퍼 길이	이름 길이 인수 중 하나의 값이 0 미만이지만 SQL_NTS와 같지 않습니다.
HYC00	드라이버가 지원되지 않음	DB2 UDB CLI는 *catalog*를 표 이름에 대한 규정자로 지원하지 않습니다. 연결된 서버는 *schema*를 프로시듀어명에 대한 규정자로 지원하지 않습니다.
HYT00	시간 종료가 만료됨	

제한사항

SQLProcedureColumns()는 저장된 프로시듀어에서 리턴될 결과 집합의 속성에 대한 정보를 리턴하지 않습니다.

어플리케이션이 저장된 프로시듀어 카탈로그를 지원하지 않거나 저장된 프로시듀어를 지원하지 않는 DB2 서버에 연결되면, SQLProcedureColumns()는 빈 결과 집합을 리턴합니다.

예

```
/* From CLI sample proccols.c */
/* ... */

    printf("Enter Procedure Schema Name Search Pattern:\n");
    gets((char *)proc_schem.s);

    printf("Enter Procedure Name Search Pattern:\n");
    gets((char *)proc_name.s);

    rc = SQLProcedureColumns(hstmt, NULL, 0, proc_schem.s, SQL_NTS,
                             proc_name.s, SQL_NTS, (SQLCHAR *)"%", SQL_NTS);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 2, SQL_C_CHAR, (SQLPOINTER) proc_schem.s, 129,
                    &proc_schem.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 3, SQL_C_CHAR, (SQLPOINTER) proc_name.s, 129,
                    &proc_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 4, SQL_C_CHAR, (SQLPOINTER) column_name.s, 129,
                    &column_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 5, SQL_C_SHORT, (SQLPOINTER) &arg_type,
                    0, &arg_type_ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 7, SQL_C_CHAR, (SQLPOINTER) type_name.s, 129,
                    &type_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
```

```
rc = SQLBindCol(hstmt, 8, SQL_C_LONG, (SQLPOINTER) & length,
                0, &length_ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

rc = SQLBindCol(hstmt, 10, SQL_C_SHORT, (SQLPOINTER) &scale,
                0, &scale_ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

rc = SQLBindCol(hstmt, 13, SQL_C_CHAR, (SQLPOINTER) remarks.s, 255,
                &remarks.ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

/* Fetch each row, and display */
while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS) {
    sprintf((char *)cur_name, "%s.%s", proc_schem.s, proc_name.s);
    if (strcmp((char *)cur_name, (char *)pre_name) != 0) {
        printf("\n%s\n", cur_name);
    }
    strcpy((char *)pre_name, (char *)cur_name);
    printf("   %s", column_name.s);
    switch (arg_type)
    { case SQL_PARAM_INPUT : printf(", Input"); break;
      case SQL_PARAM_OUTPUT : printf(", Output"); break;
      case SQL_PARAM_INPUT_OUTPUT : printf(", Input_Output"); break;
    }
    printf(", %s", type_name.s);
    printf(" (%ld", length);
    if (scale_ind != SQL_NULL_DATA) {
        printf(", %d)\n", scale);
    } else {
        printf(")\n");
    }
    if (remarks.ind > 0 ) {
        printf("(remarks), %s)\n", remarks.s);
    }
}                           /* endwhile */
```

참조

• 226 페이지의『SQLProcedures - 프로시듀어명 리스트 얻기』

SQLProcedures - 프로시듀어명 리스트 얻기

목적

SQLProcedures()는 서버에 등록되고 지정된 탐색 패턴과 일치하는 프로시듀어명 리스트를 리턴합니다.

조회에 의해 생성된 결과 집합을 처리하기 위해 사용된 함수와 같은 함수를 사용하여 검색될 수 있는 SQL 결과 집합에 이 정보가 리턴됩니다.

구문

```
SQLRETURN   SQLProcedures    (SQLHSTMT          StatementHandle,
                              SQLCHAR           *CatalogName,
                              SQLSMALLINT       NameLength1,
                              SQLCHAR           *SchemaName,
                              SQLSMALLINT       NameLength2,
                              SQLCHAR           *ProcName,
                              SQLSMALLINT       NameLength3);
```

함수 인수

표 137. SQLTables 인수

자료 유형	인수	사용	설명
SQLHSTMT	StatementHandle	입력	명령문 핸들.
SQLCHAR *	CatalogName	입력	세 부분으로 된 프로시듀어명의 카탈로그 규정자. 이 포인터는 널(null) 포인터이거나 길이가 0인 스트링이어야 합니다.
SQLSMALLINT	NameLength1	입력	*CatalogName*의 길이. 0으로 설정되어야 합니다.
SQLCHAR *	SchemaName	입력	스키마명에 의해 결과 집합을 규정하기 위해 *패턴 값*을 포함하는 버퍼. zOS 및 OS/390용 DB2 UDB V 4.1의 경우 모든 저장 프로시듀어가 하나의 스키마에 있으며 *SchemaName* 인수에 허용되는 유일한 값이 널(null) 포인터입니다. DB2 UDB의 경우*SchemaName*에 유효한 패턴 값이 있을 수 있습니다.
SQLSMALLINT	NameLength2	입력	*SchemaName*의 길이
SQLCHAR *	ProcName	입력	프로시듀어명에 따라 결과 집합을 규정하기 위한 *패턴 값*이 있는 버퍼.
SQLSMALLINT	NameLength3	입력	*ProcName*의 길이

사용법

SQLProcedures()에 의해 리턴된 결과 집합에는 주어진 순서로 227 페이지의 표 138에 나열된 열이 있습니다. 열은 PROCEDURE_CAT, PROCEDURE_SCHEMA, PROCEDURE_NAME에 의해 순서가 정해집니다.

많은 경우에 있어서 SQLProcedures() 호출은 시스템 카탈로그에 반해 복잡하고 비용이 많이 드는 조회에 맵핑되므로 자주 사용하지 않는 것이 좋으며 반복 호출이 아닌 결과를 저장합니다.

새로운 열이 추가되고 기존 열의 이름이 이후의 릴리스에서 변경될 수 있지만 현재 열의 위치는 변경되지 않습니다.

표 138. SQLProcedures에 의해 리턴되는 열

1	PROCEDURE_CAT	VARCHAR(128)	현재 서버.
2	PROCEDURE_SCHEM	VARCHAR(128)	PROCEDURE_NAME이 들어 있는 스키마명.
3	PROCEDURE_NAME	VARCHAR(128) NOT NULL	프로시듀어명
4	NUM_INPUT_PARAMS	INTEGER는 널(null)이 아님	입력 매개변수의 수 이 열은 사용되어서는 안됩니다. 향후에 ODBC에 의해 사용되도록 예약되었습니다. 버전 5 이전의 DB2 UDB CLI 버전에서 사용되었습니다. 호환성을 위해 이전의 DB2CLI.PROCEDURES 의사 카탈로그 표와 함께 사용될 수 있습니다(PATCH1 CLI/ODBC 구성 키워드를 설정하여).
5	NUM_OUTPUT_PARAMS	INTEGER는 널(null)이 아님	출력 매개변수의 수 이 열은 사용되어서는 안됩니다. 향후에 ODBC에 의해 사용되도록 예약되었습니다. 버전 5 이전의 DB2 UDB CLI 버전에서 사용되었습니다. 호환성을 위해 이전의 DB2CLI.PROCEDURES 의사 카탈로그 표와 함께 사용될 수 있습니다(PATCH1 CLI/ODBC 구성 키워드를 설정하여).
6	NUM_RESULT_SETS	INTEGER는 널(null)이 아님	프로시듀어에 의해 리턴되는 결과 집합의 수. 이 열은 사용되어서는 안됩니다. 향후에 ODBC에 의해 사용되도록 예약되었습니다. 버전 5 이전의 DB2 UDB CLI 버전에서 사용되었습니다. 호환성을 위해 이전의 DB2CLI.PROCEDURES 의사 카탈로그 표와 함께 사용될 수 있습니다(PATCH1 CLI/ODBC 구성 키워드를 설정하여).
7	REMARKS	VARCHAR(254)	프로시듀어에 대한 설명 정보가 있습니다.

주: DB2 UDB CLI에 의해 사용된 열명(column name)은 X/Open CLI CAE 스펙 방식을 준수합니다. 열 유형, 내용과 순서는 ODBC의 SQLProcedures() 결과 집합에 대해 정의된 값과 같습니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

오류 조건

표 139. SQLProcedures SQLSTATEs

SQLSTATE	설명	설명
24000	유효하지 않은 커서 상태	커서가 명령문 핸들에서 이미 열렸습니다.
40003 **08**S01	통신 링크 실패	함수가 완료되기 전에 어플리케이션과 자료 소스와의 통신 링크가 끊어졌습니다.
HY001	메모리 할당 실패	DB2 UDB CLI가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY008	조작 취소	
HY010	함수 순서 오류	
HY014	더 이상의 핸들이 없음	DB2 UDB CLI는 내부 자원 때문에 핸들을 할당할 수 없습니다.
HY090	유효하지 않은 스트링 또는 버퍼 길이	이름 길이 인수 중 하나의 값이 0 미만이었으나 SQL_NTS와 같지 않습니다.
HYC00	드라이버가 지원되지 않음	DB2 UDB CLI는 *catalog*를 표 이름에 대한 규정자로 지원하지 않습니다. 연결된 서버는 스키마를 프로시듀어명에 대한 규정자로 지원하지 않습니다.
HYT00	시간 종료가 만료됨	

제한사항

어플리케이션이 저장된 프로시듀어 카탈로그를 지원하지 않거나 저장된 프로시듀어를 지원하지 않는 DB2 서버에 연결되면, SQLProcedureColumns()는 빈 결과 집합을 리턴합니다.

예

```
/* From CLI sample procs.c */
/* ... */

    printf("Enter Procedure Schema Name Search Pattern:\n");
    gets((char *)proc_schem.s);

    rc = SQLProcedures(hstmt, NULL, 0, proc_schem.s, SQL_NTS, (SQLCHAR *)"%", SQL_NTS);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 2, SQL_C_CHAR, (SQLPOINTER) proc_schem.s, 129,
                    &proc_schem.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 3, SQL_C_CHAR, (SQLPOINTER) proc_name.s, 129,
                    &proc_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 7, SQL_C_CHAR, (SQLPOINTER) remarks.s, 255,
                    &remarks.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    printf("PROCEDURE SCHEMA          PROCEDURE NAME            \n");
    printf("------------------------- ------------------------- \n");
    /* Fetch each row, and display */
    while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS) {
```

```
        printf("%-25s %-25s\n", proc_schem.s, proc_name.s);
        if (remarks.ind != SQL_NULL_DATA) {
            printf("  (Remarks) %s\n", remarks.s);
        }
    }                           /* endwhile */
```

참조

- 220 페이지의『SQLProcedureColumns - 프로시듀어에 대한 입/출력(I/O) 매개변수 정보 얻기』

SQLPutData - 매개변수에 대한 자료 값 전달

목적

SQLPutData()는 SQL_NEED_DATA를 리턴하는 SQLParamData() 호출 다음에 호출되어 매개변수 자료 값을 제공합니다. 이 함수는 큰 매개변수 값을 나누어 송신하는 데 사용될 수 있습니다.

구문

```
SQLRETURN SQLPutData  (SQLHSTMT     hstmt,
                       SQLPOINTER   rgbValue,
                       SQLINTEGER   cbValue);
```

함수 인수

표 140. SQLPutData 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLPOINTER	*rgbValue*	입력	매개변수에 대한 실제 자료 또는 자료 일부를 가리키는 포인터. 자료는 어플리케이션이 매개변수를 지정할 때 사용하는 SQLBindParam() 호출에 지정된 형식이어야 합니다.
SQLINTEGER	*cbValue*	입력	*rgbValue*의 길이. SQLPutData() 호출에서 송신된 자료의 양을 지정합니다. 자료의 양은 주어진 매개변수에 대한 호출마다 달라질 수 있습니다. 어플리케이션은 *cbValue*에 대해 SQL_NTS 또는 SQL_NULL_DATA를 지정할 수 있습니다. 모든 날짜, 시간, 시간소인 자료 유형 그리고 SQL_NUMERIC과 SQL_DECIMAL을 제외한 모든 숫자 자료 유형에 대해 *cbValue*가 무시됩니다. C 버퍼 유형이 SQL_CHAR 또는 SQL_BINARY이거나 SQL_DEFAULT가 C 버퍼 유형으로 지정되고, C 버퍼 유형 디폴트가 SQL_CHAR 또는 SQL_BINARY인 경우, 이것은 *rgbValue* 버퍼에 있는 자료의 바이트 수입니다.

사용법

SQL_DATA_AT_EXEC 매개변수에 대한 자료 값을 제공하기 위해 SQL_NEED_DATA 상태의 명령문에서 SQLParamData()을 호출한 후 어플리케이션은 SQLPutData()를 호출합니다. 긴 자료는 SQLPutData()를 반복 호출하여 나누어서 송신됩니다. 매개변수에 대한 모든 자료가 나누어서 송신된 후 어플리케이션은 다시 SQLParamData()를 호출합니다. SQLParamData()는 다음 SQL_DATA_AT_EXEC 매개변수로 진행하거나 모든 매개변수가 자료 값을 가지면 이 명령문을 실행합니다.

SQLPutData()는 고정 길이 매개변수에 대해 한 번 이상 호출될 수 없습니다.

SQLPutData() 호출 후에, 입력 자료가 문자 또는 2진 자료인 경우 가능한 호출은 SQLParamData(), SQLCancel() 또는 다른 SQLPutData() 호출 뿐입니다. SQLParamData()와 마찬가지로, 이 명령문 핸들을 사용한 모든 다른 함수 호출은 실패합니다. 뿐만 아니라, *hstmt*의 상위 *hdbc*를 참조하는 모든 함수 호출이 속성이나 해당 연결 상태의 변경과 관련되면 실패합니다. 이 함수의 리스트는 208 페이지의 『SQLParamData - 자료 값이 필요한 다음 매개변수 얻기』에 대한 사용 섹션을 참조하십시오.

SQL_SUCCESS에서 단일 매개변수 결과에 대해 한 번 이상 SQLPutData()를 호출한 경우, 같은 매개변수에 대해 *cbValue*를 SQL_NULL_DATA로 설정하고 SQLPutData()를 호출하려고 시도하면 **HY**011의 SQLSTATE 오류가 발생합니다. 이 오류로 상태는 변하지 않지만 명령문 핸들은 계속 *Need Data* 상태에 있게 되며 어플리케이션은 매개변수 자료를 계속 송신할 수 있습니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

다음 중 일부 진단 상태는 SQLPutData()가 호출될 때가 아니라 마지막 SQLParamData() 호출에서 보고될 수 있습니다.

표 141. SQLPutData SQLSTATEs

SQLSTATE	설명	설명
22001	자료가 너무 많음	SQLPutData()에서 현재의 매개변수로 제공한 자료 *크기*가 매개변수 *크기*를 초과합니다. SQLPutData()에 대한 마지막 호출에서 제공된 자료는 무시됩니다.
01004	자료가 절단됨	숫자 매개변수에 대해 송신된 자료가 유효 숫자의 손실없이 절단되었습니다. 날짜나 시간 열에 대해 송신된 시간소인 자료가 절단되었습니다. 함수가 SQL_SUCCESS_WITH_INFO를 리턴합니다.
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*rgbValue* 인수는 널(null) 포인터입니다. *rgbValue* 인수가 널(null) 포인터가 아니고, *cbValue* 인수는 0보다 작지만 SQL_NTS 또는 SQL_NULL_DATA와 같지 않습니다.
HY010	함수 순서 오류	명령문 핸들 *hstmt*는 **자료가 필요한** 상태에 있어야 하며 이전 SQLParamData() 호출을 통해 SQL_DATA_AT_EXEC 매개변수로 위치가 지정되어야 합니다.

SQLReleaseEnv - 모든 환경 자원 해제

목적

SQLReleaseEnv()는 환경 핸들을 무효화하고 해제합니다. 환경 핸들과 연관된 모든 DB2 UDB CLI 자원이 해제됩니다.

이 함수를 호출하기 전에 SQLFreeConnect()를 호출해야 합니다.

이 함수는 종료하기 전에 어플리케이션에서 필요로 하는 마지막 DB2 UDB CLI단계입니다.

구문

```
SQLRETURN SQLReleaseEnv (SQLHENV    henv);
```

함수 인수

표 142. SQLReleaseEnv 인수

자료 유형	인수	사용	설명
SQLHENV	*henv*	입력	환경 핸들

사용법

유효한 연결 핸들이 아직 있는 상태에서 이 함수를 호출하면 SQL_ERROR가 리턴되고 환경 핸들은 계속 유효합니다.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 143. SQLReleaseEnv SQLSTATEs

SQLSTATE	설명	설명
58004	시스템 오류	복구할 수 없는 시스템 오류
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY010	함수 순서 오류	할당되었거나 연결된 상태의 *hdbc*가 있습니다. SQLReleaseEnv를 호출하기 전에 *hdbc*에 대해 SQLDisconnect와 SQLFreeConnect를 호출합니다.
HY013 *	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

예

SQLAllocEnv()33 페이지의 『예』를 참조하십시오.

참조

- 123 페이지의 『SQLFreeConnect - 연결 핸들 해제』

SQLRowCount - 행 갯수 얻기

목적

SQLRowCount()는 표에 대해 실행된 UPDATE, INSERT, 또는 DELETE문에 의해 영향을 받는 표 또는 표를 바탕으로 한 뷰의 행 수를 리턴합니다.

이 함수를 호출하기 전에 SQLExecute() 또는 SQLExecDirect()를 호출해야 합니다.

구문

```
SQLRETURN SQLRowCount (SQLHSTMT      hstmt,
                       SQLINTEGER    *pcrow);
```

함수 인수

표 144. SQLRowCount 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLINTEGER *	*pcrow*	출력	영향을 받는 행의 수가 저장된 위치를 가리키는 포인터.

사용법

입력 명령문 핸들에 의해 참조된 마지막으로 실행된 명령문이 UPDATE, INSERT 또는 DELETE문이 아니거나 성공적으로 실행되지 않은 경우 함수는 *pcrow*의 내용을 0으로 설정합니다.

명령문에 의해 영향을 받는 다른 표의 행(예를 들면, 연속되는 삭제)은 갯수에 포함되지 않습니다.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 145. SQLRowCount SQLSTATEs

SQLSTATE	설명	설명
40003 *****	명령문 완료 상태를 알 수 없음	함수가 처리를 완료하기 전에 CLI와 자료 소스와의 통신 링크가 끊어졌습니다.
58004	시스템 오류	복구할 수 없는 시스템 오류
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*pcrow*가 널(null) 포인터입니다.

표 145. SQLRowCount SQLSTATEs (계속)

SQLSTATE	설명	설명
HY010	함수 순서 오류	*hstmt*에 대해 SQLExecute 또는 SQLExecDirect를 호출하기 전에 이 함수가 호출됩니다.
HY013 *****	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

참조

- 102 페이지의『SQLExecDirect - 명령문 직접 실행』
- 104 페이지의『SQLExecute - 명령문 실행』
- 206 페이지의『SQLNumResultCols - 결과 열의 수 얻기』

SQLSetConnectAttr - 연결 속성 설정

목적

SQLSetConnectAttr()는 특정 연결에 대한 연결 속성을 설정합니다.

구문

```
SQLRETURN SQLSetConnectAttr  (SQLHDBC     hdbc,
                              SQLINTEGER     fAttr,
                              SQLPOINTER     vParam,
                              SQLINTEGER     sLen);
```

함수 인수

표 146. SQLSetConnectAttr 인수

자료 유형	인수	사용	설명
SQLHDBC	*hdbc*	입력	연결 핸들
SQLINTEGER	*fAttr*	입력	설정할 연결 속성. 자세한 내용은 표 147을 참조하십시오.
SQLPOINTER	*vParam*	입력	*fAttr*과 연관된 값. 옵션에 따라 32비트 정수 값 또는 문자 스트링을 가리키는 포인터가 될 수 있습니다.
SQLINTEGER	*sLen*	입력	문자 스트링인 경우 입력 값의 길이. 그렇지 않은 경우는 사용되지 않습니다.

사용법

SQLSetConnectAttr()를 통해 설정된 모든 연결과 명령문 옵션은 SQLFreeConnect()가 호출되거나 다음 번에 SQLSetConnectAttr()이 호출될 때까지 보존됩니다.

*vParam*을 통해 설정된 정보 형식은 *fAttr*에 따라 달라집니다. 옵션 정보는 32비트 정수 또는 널로 종료되는 문자를 가리키는 포인터일 수 있습니다.

표 147. 연결 옵션

fAttr	내용
SQL_ATTR_AUTOCOMMIT	연결에 대한 확약 작동을 설정하는 32비트 값. 가능한 값은 다음과 같습니다. • SQL_TRUE - 각각의 SQL문은 실행시 자동으로 확약됩니다. • SQL_FALSE - SQL문이 자동으로 확약되지 않습니다. 확약 제어와 함께 실행되고 있다면, 변경된 내용은 SQLEndTran() 또는 SQLTransact()를 사용하여 명시적으로 확약되거나 롤백되어야 합니다.

표 147. 연결 옵션 (계속)

fAttr	내용
SQL_ATTR_COMMIT 또는 SQL_TXN_ISOLATION	*hdbc*에 의해 참조되는 현재 연결에 대한 트랜잭션 분리 레벨을 설정하는 32비트 값. 다음 값들은 DB2 UDB CLI의해 허용되지만 각 서버는 이 분리 레벨 중 일부만을 지원할 수도 있습니다. • SQL_TXN_NO_COMMIT - 확약 제어가 사용되지 않습니다. • SQL_TXN_READ_UNCOMMITTED - 더티(dirty) 읽기, 비반복(nonrepeatable) 읽기, 팬텀(phantom)이 가능합니다. • SQL_TXN_READ_COMMITTED - 더티(dirty) 읽기가 가능하지 않습니다. 비반복(nonrepeatable) 읽기, 팬텀(phantom)이 가능합니다. • SQL_TXN_REPEATABLE_READ - 더티(dirty) 읽기와 비반복(nonrepeatable) 읽기가 가능하지 않습니다. 팬텀(phantom)은 가능합니다. • SQL_TXN_SERIALIZABLE - 트랜잭션을 순차화(serializable)할 수 있습니다. 더티(dirty) 읽기, 비반복(non-repeatable) 읽기, 팬텀(phantom)이 가능하지 않습니다. IBM 용어는 다음과 같습니다. • SQL_TXN_READ_UNCOMMITTED는 UR(Uncommitted Read)입니다. • SQL_TXN_READ_COMMITTED는 CS(Cursor Stability)입니다. • SQL_TXN_REPEATABLE_READ는 RS(Read Stability)입니다. • SQL_TXN_SERIALIZABLE은 RR(Repeatable Read)입니다. 분리 레벨에 대한 자세한 설명은 IBM SQL Reference를 참조하십시오. SQL_ATTR_COMMIT 속성은 SQLConnect() 전에 설정되어야 합니다. 연결이 설정된 후 값이 변경되고 연결이 리모트 자료 소스와의 연결인 경우, 연결 핸들에 대한 다음 번 SQLConnect()가 성공할 때까지 변경은 유효하지 않습니다.

표 147. 연결 옵션 (계속)

fAttr	내용
SQL_ATTR_DATE_FMT	32비트 정수 값은 다음과 같습니다. • SQL_FMT_ISO - ISO(International Organization for Standardization) 날짜 형식 yyyy-mm-dd가 사용됩니다. 이것이 디폴트 값입니다. • SQL_FMT_USA - 미국 날짜 형식 mm/dd/yyyy가 사용됩니다. • SQL_FMT_EUR - 유럽 날짜 형식 dd.mm.yyyy가 사용됩니다. • SQL_FMT_JIS - 일본 산업 표준 날짜 형식 yyyy-mm-dd가 사용됩니다. • SQL_FMT_MDY - 날짜 형식 mm/dd/yyyy가 사용됩니다. • SQL_FMT_DMY - 날짜 형식 dd/mm/yyyy가 사용됩니다. • SQL_FMT_YMD - 날짜 형식 yy/mm/dd가 사용됩니다. • SQL_FMT_JUL - 율리우스력 날짜 형식 yy/ddd가 사용됩니다. • SQL_FMT_JOB - 작업 디폴트가 사용됩니다.
SQL_ATTR_DATE_SEP	32비트 정수 값은 다음과 같습니다. • SQL_SEP_SLASH - 슬래시(/)가 날짜 분리자로 사용됩니다. 이것이 디폴트 값입니다. • SQL_SEP_DASH - 대시(-)가 날짜 분리자로 사용됩니다. • SQL_SEP_PERIOD - 마침표(.)가 사용됩니다. • SQL_SEP_COMMA - 쉼표(,)가 날짜 분리자로 사용됩니다. • SQL_SEP_BLANK - 공백이 날짜 분리자로 사용됩니다. • SQL_SEP_JOB - 작업 디폴트가 사용됩니다.
SQL_ATTR_DBC_DEFAULT_LIB	규정되지 않은 파일 참조를 분석하기 위해 사용될 디폴트 라이브러리를 표시하는 문자 값. 이 값은 연결이 시스템 명명 모드를 사용하는 경우 유효하지 않습니다.
SQL_ATTR_DBC_SYS_NAMING	32비트 정수 값은 다음 중 하나일 수 있습니다. • SQL_TRUE - DB2 UDB CLI는 iSeries 시스템 명명 모드를 사용합니다. 파일은 슬래시(/) 분리문자를 사용하여 규정됩니다. 규정되지 않은 파일은 작업에 대한 라이브러리 리스트를 사용하여 분석됩니다. • SQL_FALSE - DB2 UDB CLI는 디폴트 명명 모드를 사용하는데 이는 SQL 명명 모드입니다. 파일은 마침표(.) 분리문자를 사용하여 규정됩니다. 규정되지 않은 파일은 디폴트 라이브러리 또는 현재 사용자 ID를 사용하여 분석됩니다.
SQL_ATTR_DECIMAL_SEP	32비트 정수 값은 다음과 같습니다. • SQL_SEP_PERIOD - 마침표(.)가 사용됩니다. 이것이 디폴트 값입니다. • SQL_SEP_COMMA - 쉼표(,)가 날짜 분리자로 사용됩니다. • SQL_SEP_JOB - 작업 디폴트가 사용됩니다.

표 147. 연결 옵션 (계속)

fAttr	내용
SQL_ATTR_EXTENDED_COL_INFO	32비트 정수 값은 다음 중 하나일 수 있습니다. • SQL_TRUE - 이 연결 핸들에 반하여 할당된 명령문 핸들을 SQLColAttributes()에 사용하여 기본 표, 기본 열, 레이블과 같은 확장 열 정보를 검색할 수 있습니다. • SQL_FALSE -이 연결 핸들에 반하여 할당된 명령문 핸들을 SQLColAttributes() 함수에 사용하여 확장 열 정보를 검색할 수 있습니다. 이것이 디폴트 값입니다.
SQL_ATTR_TIME_FMT	32비트 정수 값은 다음과 같습니다. • SQL_FMT_ISO - ISO(International Organization for Standardization) 시간 형식 hh.mm.ss가 사용됩니다. 이것이 디폴트 값입니다. • SQL_FMT_USA - 미국 시간 형식 hh:mmxx(XX는 AM 또는 PM)가 사용됩니다. • SQL_FMT_EUR - 유럽 시간 형식 hh.mm.ss가 사용됩니다. • SQL_FMT_JIS - 일본 산업 표준 시간 형식 hh:mm:ss가 사용됩니다. • SQL_FMT_HMS - hh:mm:ss 형식이 사용됩니다.
SQL_ATTR_TIME_SEP	32비트 정수 값은 다음과 같습니다. • SQL_SEP_COLON - 콜론(:)이 시간 분리자로 사용됩니다. 이것이 디폴트 값입니다. • SQL_SEP_PERIOD - 마침표(.)가 사용됩니다. • SQL_SEP_COMMA - 쉼표(,) 가 시간 분리자로 사용됩니다. • SQL_SEP_BLANK - 공백이 시간 분리자로 사용됩니다. • SQL_SEP_JOB - 작업 디폴트가 사용됩니다.
SQL_SAVEPOINT_NAME	SQL_SAVEPOINT_NAME_ROLLBACK이나 SQL_SAVEPOINT_NAME_RELEASE 함수에서 SQLEndTran()이 사용할 savepoint 이름을 나타내는 문자 값.
SQL_2ND_LEVEL_TEXT	32비트 정수 값은 다음 중 하나일 수 있습니다. • SQL_TRUE - SQLError()를 호출하여 발생된 오류 텍스트에 오류와 관련된 모든 설명이 있습니다. • SQL_FALSE - SQLError()를 호출하여 발생된 오류 텍스트에 오류와 관련된 1단계 설명만 있습니다. 이것이 디폴트 값입니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR

- SQL_INVALID_HANDLE

진단

표 148. SQLSetConnectAttr SQLSTATEs

SQLSTATE	설명	설명
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*fAttr* 값의 경우 유효하지 않은 값이 *vParam*에 대해 지정되었습니다. 유효하지 않은 *fAttr* 값이 지정되었습니다.

SQLSetConnectOption - 연결 옵션 설정

목적

SQLSetConnectOption()은 특정 연결에 대한 연결 속성을 설정합니다.

구문

```
SQLRETURN SQLSetConnectOption  (SQLHDBC hdbc,
                                SQLSMALLINT fOption,
                                SQLPOINTER  vParam);
```

함수 인수

표 149. SQLSetConnectOption 인수

자료 유형	인수	사용	설명
SQLHDBC	*hdbc*	입력	연결 핸들
SQLSMALLINT	*fOption*	입력	설정할 연결 옵션. 자세한 정보는 236 페이지의 표 147을 참조하십시오.
SQLPOINTER	*vParam*	입력	*fOption*과 연관된 값. 옵션에 따라 32비트 정수 값 또는 문자 스트링을 가리키는 포인터가 될 수 있습니다.

사용법

SQLSetConnectOption()은 SQLSetConnectAttr()과 같은 기능을 제공하며 호환성을 위해 두 함수 모두 지원됩니다.

SQLSetConnectOption()을 통해 설정된 모든 연결과 명령문 옵션은 SQLFreeConnect()가 호출되거나 다음 번 SQLSetConnectOption() 호출때 까지 보존됩니다.

*vParam*을 통해 설정된 정보 형식은 지정된 *fOption*에 따라 달라집니다. 옵션 정보는 32비트 정수 또는 널로 종료되는 문자를 가리키는 포인터일 수 있습니다.

적절한 연결 옵션에 대해서는 236 페이지의 표 147을 참조하십시오.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

SQLSetConnectOption

진단

표 150. SQLSetConnectOption SQLSTATEs

SQLSTATE	설명	설명
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*fOption* 값의 경우 *vParam* 인수에 대해 유효하지 않은 값이 지정되었습니다. 유효하지 않은 *fOption* 값이 지정되었습니다.
HYC00	드라이버가 지원되지 않음	지정된 *fOption*은 DB2 UDB CLI 또는 서버에 의해 지원되지 않습니다. 지정된 *fOption* 값의 경우 *vParam*에 대해 지정된 값이 지원되지 않습니다.

SQLSetCursorName - 커서명 설정

목적

SQLSetCursorName()은 커서명과 명령문 핸들을 연관시킵니다. DB2 UDB CLI는 내포적으로 필요할 때 커서명을 생성하므로 이 함수는 생략가능합니다.

구문

```
SQLRETURN SQLSetCursorName (SQLHSTMT       hstmt,
                            SQLCHAR        *szCursor,
                            SQLSMALLINT    cbCursor);
```

함수 인수

표 151. SQLSetCursorName 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLCHAR *	*szCursor*	입력	커서명
SQLSMALLINT	*cbCursor*	입력	*szCursor* 인수의 내용 길이

사용법

SELECT문이 직접 실행되거나 준비될 때 DB2 UDB CLI는 항상 내부적으로 생성된 커서명을 생성하여 사용합니다SQLSetCursorName()을 사용하여 어플리케이션이 지정한 커서명을 SQL문(위치지정된 UPDATE 또는 DELETE문)에서 사용할 수 있습니다. DB2 UDB CLI는 이 이름을 내부 이름에 맵핑합니다.
SQLSetCursorName()은 내부 이름이 생성되기 전에 호출되어야 합니다. 핸들이 제거될 때까지 이름은 명령문 핸들과 연관되어 남아 있습니다. 이름은 트랜잭션이 종료된 후에도 남아 있지만 이 때 SQLSetCursorName()이 호출되어 이 명령문 핸들에 대해 다른 이름이 설정될 수 있습니다.

커서명은 다음 규칙을 따라야 합니다.

- 연결 내의 모든 커서명은 고유해야 합니다.
- 각 커서명은 길이가 18바이트 이하여야 합니다. 커서명을 18바이트 보다 길게 설정하려고 하면 커서명이 18바이트로 절단됩니다. (경고는 없습니다.)
- 커서명은 SQL에서 ID로 간주되므로 영문자(a-z, A-Z)로 시작하고 숫자(0-9)와 영문자 또는 밑줄 문자(_)의 조합이 사용됩니다.
- 입력 커서명에 큰 따옴표가 없으면 입력 커서명에서 선행 공백과 후미 공백이 제거됩니다.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 152. SQLSetCursorName SQLSTATEs

SQLSTATE	설명	설명
34000	유효하지 않은 커서명	*szCursor* 인수에 의해 지정된 커서명이 유효하지 않습니다. 커서명은 ″SQLCUR″ 또는 ″SQL_CUR″로 시작해야 하며 그렇지 않으면 드라이버 또는 자료 소스 커서 명명 규칙(a-z 또는 A-Z으로 시작하고 영문자, 숫자, '_' 문자의 조합이 사용 가능함)을 위반하게 됩니다. *szCursor* 인수에 의해 지정된 커서명이 존재합니다.
58004	시스템 오류	복구할 수 없는 시스템 오류
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*szCursor*가 널(null) 포인터입니다. *cbCursor* 인수가 1 미만이었으나 SQL_NTS와 같지 않습니다.
HY010	함수 순서 오류	명령문 핸들이 할당된 상태가 아닙니다. `SQLPrepare()` 또는 `SQLExecDirect()`가 `SQLSetCursorName()` 전에 호출됩니다.
HY013 *****	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

참조

• 142 페이지의『SQLGetCursorName - 커서명 얻기』

SQLSetDescField - 설명자 필드 설정

목적

SQLSetDescField()는 설명자의 필드를 설정합니다. SQLSetDescField()는 SQLSetDescRec() 함수를 더 확장할 수 있는 대체 함수입니다.

구문

```
SQLRETURN SQLSetDescField  (SQLHDESC       hdesc,
                            SQLSMALLINT    irec,
                            SQLSMALLINT    fDescType,
                            SQLPOINTER     rgbDesc,
                            SQLINTEGER     bLen);
```

함수 인수

표 153. SQLSetDescField 인수

자료 유형	인수	사용	설명
SQLHDESC	*hdesc*	입력	설명자 핸들
SQLSMALLINT	*irec*	입력	지정된 필드가 검색될 레코드 번호
SQLSMALLINT	*fDescType*	입력	표 154를 참조하십시오.
SQLPOINTER	*rgbDesc*	입력	버퍼를 가리키는 포인터
SQLINTEGER	*bLen*	입력	설명자 버퍼 길이(*rgbDesc*)

표 154. fDescType 설명자 유형

설명자	유형	설명
SQL_DESC_COUNT	SMALLINT	설명자의 레코드 수를 설정합니다. *irec*은 무시됩니다.
SQL_DESC_TYPE	SMALLINT	*irec*의 유형 필드를 설정합니다.
SQL_DESC_DATETIME_INTERVAL_CODE	SMALLINT	레코드에 대한 간격 코드를 SQL_DATETIME 유형으로 설정합니다.
SQL_DESC_LENGTH	INTEGER	*irec*의 길이 필드를 설정합니다.
SQL_DESC_PRECISION	SMALLINT	*irec*의 정밀도 필드를 설정합니다.
SQL_DESC_SCALE	SMALLINT	*irec*의 소수 자릿수(scale) 필드를 설정합니다.
SQL_DESC_DATA_PTR	SQLPOINTER	*irec*에 대한 자료 포인터 필드를 설정합니다.
SQL_DESC_LENGTH_PTR	SQLPOINTER	*irec*에 대한 길이 포인터 필드를 설정합니다.
SQL_DESC_INDICATOR_PTR	SQLPOINTER	*irec*에 대한 인디케이터 포인터 필드를 설정합니다.

사용법

SQLSetDescRec()과 같은 전체 인수 집합을 요구하는 대신, SQLSetDescField()는 특정 설명자 레코드에 대해 사용자가 설정하려는 속성을 지정합니다.

SQLSetDescField()는 나중 확장에 대해 허용하지만 각 설명자 레코드에 대해 SQLSetDescRec()보다 동일 자료를 설정하기 위한 더 많은 호출을 필요로 합니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 155. SQLGetDescField SQLSTATEs

SQLSTATE	설명	설명
HY009	유효하지 않은 인수 값	*fDescType* 또는 *irec* 인수에 지정된 값이 유효하지 않습니다. *rgbValue* 인수가 널(null) 포인터입니다.
HY013 *****	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

참조

- 39 페이지의『SQLBindCol - 어플리케이션 변수에 열 바인드』
- 84 페이지의『SQLDescribeCol - 열 속성 설명』
- 102 페이지의『SQLExecDirect - 명령문 직접 실행』
- 104 페이지의『SQLExecute - 명령문 실행』
- 212 페이지의『SQLPrepare - 명령문 준비』

SQLSetDescRec - 설명자 레코드 설정

목적

SQLSetDescRec()는 설명자 레코드에 대한 모든 속성을 설정합니다. SQLSetDescRec()는 SQLDescField() 함수를 간단히 대체합니다.

구문

```
SQLRETURN SQLSetDescRec (SQLHDESC       hdesc,
                         SQLSMALLINT    irec,
                         SQLSMALLINT    type,
                         SQLSMALLINT    subtype,
                         SQLINTEGER     length,
                         SQLSMALLINT    prec,
                         SQLSMALLINT    scale,
                         SQLPOINTER     data,
                         SQLINTEGER     *sLen,
                         SQLINTEGER    *indic);
```

함수 인수

표 156. SQLSetDescRec 인수

자료 유형	인수	사용	설명
SQLDESC	*hdesc*	입력	설명자 핸들
SQLSMALLINT	*irec*	입력	설명자 내의 레코드 번호
SQLSMALLINT	*type*	입력	레코드에 대한 TYPE 필드
SQLSMALLINT	*subtype*	입력	TYPE이 SQL_DATETIME인 레코드에 대한 DATETIME_INTERVAL_CODE 필드
SQLINTEGER	*length*	입력	레코드에 대한 LENGTH 필드
SQLSMALLINT	*prec*	입력	레코드에 대한 PRECISION 필드
SQLSMALLINT	*scale*	입력	레코드에 대한 SCALE 필드
SQLPOINTER	*data*	입력(지연됨)	레코드에 대한 DATA_PTR 필드
SQLINTEGER *	*sLen*	입력(지연됨)	레코드에 대한 LENGTH_PTR 필드
SQLINTEGER *	*indic*	입력(지연됨)	레코드에 대한 INDICATOR_PTR 필드

사용법

SQLSetDescRec()를 호출하면 한 호출의 설명자 레코드의 모든 필드가 설정됩니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 157. SQLSetDescRec SQLSTATEs

SQLSTATE	설명	설명
HY009	유효하지 않은 인수 값	*irec* 인수에 대해 지정된 값이 1 미만입니다. 다른 인수에 대해 유효하지 않은 값이 지정되었습니다.
HY016	유효하지 않은 설명자	설명자 핸들이 구현 프로그램 행 설명자를 참조했습니다.

참조

- 39 페이지의『SQLBindCol - 어플리케이션 변수에 열 바인드』
- 84 페이지의『SQLDescribeCol - 열 속성 설명』
- 102 페이지의『SQLExecDirect - 명령문 직접 실행』
- 104 페이지의『SQLExecute - 명령문 실행』
- 212 페이지의『SQLPrepare - 명령문 준비』

SQLSetEnvAttr - 환경 속성 설정

목적

SQLSetEnvAttr()는 현재 환경에 대한 환경 속성을 설정합니다.

구문

```
SQLRETURN SQLSetEnvAttr  (SQLHENV      henv,
                          SQLINTEGER   Attribute,
                          SQLPOINTER   Value,
                          SQLINTEGER   StringLength);
```

함수 인수

표 158. SQLSetEnvAttr 인수

자료 유형	인수	사용	설명
SQLHENV	*henv*	입력	환경 핸들
SQLINTEGER	*Attribute*	입력	설정할 환경 속성. 자세한 정보는 표 159를 참조하십시오.
SQLPOINTER	*pValue*	입력	*Attribute*에 대해 원하는 값
SQLINTEGER	*StringLength*	입력	속성 값이 문자 스트링인 경우 *Value*의 길이(바이트). *Attribute*가 스트링을 의미하지 않는 경우 DB2 UDB CLI는 *StringLength*를 무시합니다.

사용법

표 159. 환경 속성

Attribute	내용
SQL_ATTR_DATE_FMT	32비트 정수 값은 다음과 같습니다. • SQL_FMT_ISO - ISO(International Organization for Standardization) 날짜 형식 yyyy-mm-dd가 사용됩니다. 이것이 디폴트 값입니다. • SQL_FMT_USA - 미국 날짜 형식 mm/dd/yyyy가 사용됩니다. • SQL_FMT_EUR - 유럽 날짜 형식 dd.mm.yyyy가 사용됩니다. • SQL_FMT_JIS - 일본 산업 표준 날짜 형식 yyyy-mm-dd가 사용됩니다. • SQL_FMT_MDY - 날짜 형식 mm/dd/yyyy가 사용됩니다. • SQL_FMT_DMY - 날짜 형식 dd/mm/yyyy가 사용됩니다. • SQL_FMT_YMD - 날짜 형식 yy/mm/dd가 사용됩니다. • SQL_FMT_JUL - 율리우스력 날짜 형식 yy/ddd가 사용됩니다. • SQL_FMT_JOB - 작업 디폴트가 사용됩니다.

SQLSetEnvAttr

표 159. 환경 속성 (계속)

Attribute	내용
SQL_ATTR_DATE_SEP	32비트 정수 값은 다음과 같습니다. • SQL_SEP_SLASH - 슬래시(/)가 날짜 분리자로 사용됩니다. 이것이 디폴트 값입니다. • SQL_SEP_DASH - 대시(-)가 날짜 분리자로 사용됩니다. • SQL_SEP_PERIOD - 마침표(.)가 사용됩니다. • SQL_SEP_COMMA - 쉼표(,)가 날짜 분리자로 사용됩니다. • SQL_SEP_BLANK - 공백이 날짜 분리자로 사용됩니다. • SQL_SEP_JOB - 작업 디폴트가 사용됩니다.
SQL_ATTR_DECIMAL_SEP	32비트 정수 값은 다음과 같습니다. • SQL_SEP_PERIOD - 마침표(.)가 사용됩니다. 이것이 디폴트 값입니다. • SQL_SEP_COMMA - 쉼표(,)가 날짜 분리자로 사용됩니다. • SQL_SEP_JOB - 작업 디폴트가 사용됩니다.
SQL_ATTR_DEFAULT_LIB	규정되지 않은 파일 참조를 분석하기 위해 사용될 디폴트 라이브러리를 표시하는 문자 값. 이 값은 환경이 시스템 명명 모드를 사용하는 경우 유효하지 않습니다.
SQL_ATTR_ENVHNDL_COUNTER	32비트 정수 값은 다음 중 하나일 수 있습니다. • SQL_FALSE - DB2 CLI는 환경 핸들 할당 횟수를 세지 않습니다. 따라서 환경을 해제하는 첫 번째 호출이 핸들 및 모든 연관된 자원을 해제합니다. • SQL_TRUE - DB2 CLI는 환경 핸들 할당 횟수를 나타내는 카운터를 유지합니다. 환경 핸들이 해제될 때마다 카운터는 감소합니다. DB2 CLI는 카운터가 0이 될 때만 핸들 및 모든 연관된 자원을 실제로 해제합니다. 이렇게 하면 CLI 환경 핸들을 할당하고 해제하는 CLI를 사용하는 프로그램에 대한 내포된 호출이 가능해집니다.
SQL_ATTR_ESCAPE_CHAR	SQLColumns( ) 또는 SQLTables( )로 지정할 때 사용될 이탈 문자를 표시하는 문자 값.
SQL_ATTR_FOR_FETCH_ONLY	32비트 정수 값은 다음 중 하나일 수 있습니다. • SQL_TRUE - 커서가 읽기 전용이고 위치지정된 갱신과 삭제에 대해 사용될 수 없습니다. 이것이 디폴트 값입니다. • SQL_FALSE - 위치지정된 갱신과 삭제에 대해 커서를 사용할 수 있습니다. SQLSetStmtAttr()을 사용하여 SQL_ATTR_FOR_FETCH_ONLY 속성도 개별 명령문에 대해 설정할 수 있습니다.

표 159. 환경 속성 (계속)

Attribute	내용
SQL_ATTR_JOB_SORT_SEQUENCE	32비트 정수 값은 다음 중 하나일 수 있습니다. • SQL_TRUE - DB2 UDB CLI는 작업에 대해 설정된 정렬 순서를 사용합니다. • SQL_FALSE - DB2 UDB CLI는 *HEX인 디폴트 정렬 순서를 사용합니다.
SQL_ATTR_OUTPUT_NTS	32비트 정수 값은 다음 중 하나일 수 있습니다. • SQL_TRUE - DB2 UDB CLI는 출력 문자 스트링의 길이를 표시하기 위해 널 종료를 사용합니다. • SQL_FALSE - DB2 UDB CLI는 널 종료를 사용하지 않습니다. 이 속성에 의해 영향을 받는 CLI 함수는 모두 문자 스트링 매개변수를 가진 환경에 대해 (그리고 환경 하에서 할당된 연결에 대해) 호출된 함수입니다.
SQL_ATTR_SERVER_MODE	32비트 정수 값은 다음 중 하나일 수 있습니다. • SQL_FALSE - DB2 CLI는 같은 작업 내의 모든 연결의 SQL 문을 처리합니다. 모든 변경은 하나의 트랜잭션을 구성합니다. 이것은 디폴트 처리 모드입니다. • SQL_TRUE - DB2 CLI는 분리된 작업 내의 각 연결의 SQL 문을 처리합니다. 이렇게 하면 각 연결에 대해 다른 사용자 ID로 같은 자료 소스에 여러 번 연결할 수 있습니다. 또한 각 연결 핸들에서 이루어진 변경을 자체의 트랜잭션으로 분리합니다. 이렇게 하여 각 연결 핸들은 다른 연결 핸들에서 이루어진 지연 중인 변경에 영향을 주지 않고 확약되거나 롤백될 수 있습니다. 자세한 정보는 309 페이지의 부록 D『서버 모드로 DB2 UDB CLI 실행』을 참조하십시오.
SQL_ATTR_SYS_NAMING	32비트 정수 값은 다음 중 하나일 수 있습니다. • SQL_TRUE - DB2 UDB CLI는 iSeries 시스템 명명 모드를 사용합니다. 파일은 슬래시(/) 분리문자를 사용하여 규정됩니다. 규정되지 않은 파일은 작업에 대한 라이브러리 리스트를 사용하여 분석됩니다. • SQL_FALSE - DB2 UDB CLI는 디폴트 명명 모드를 사용하는데 이는 SQL 명명 모드입니다. 파일은 마침표(.) 분리문자를 사용하여 규정됩니다. 규정되지 않은 파일은 디폴트 라이브러리 또는 현재 사용자 ID를 사용하여 분석됩니다.

표 159. 환경 속성 (계속)

Attribute	내용
SQL_ATTR_TIME_FMT	32비트 정수 값은 다음과 같습니다. • SQL_FMT_ISO - ISO (International Organization for Standardization) 시간 형식 hh.mm.ss가 사용됩니다. 이것이 디폴트 값입니다. • SQL_FMT_USA - 미국 시간 형식 hh:mmxx(XX는 AM 또는 PM)가 사용됩니다. • SQL_FMT_EUR - 유럽 시간 형식 hh.mm.ss가 사용됩니다. • SQL_FMT_JIS - 일본 산업 표준 시간 형식 hh:mm:ss가 사용됩니다. • SQL_FMT_HMS - hh:mm:ss 형식이 사용됩니다.
SQL_ATTR_TIME_SEP	32비트 정수 값은 다음과 같습니다. • SQL_SEP_COLON - 콜론(:)이 시간 분리자로 사용됩니다. 이것이 디폴트 값입니다. • SQL_SEP_PERIOD - 마침표(.)가 사용됩니다. • SQL_SEP_COMMA - 쉼표(,) 가 시간 분리자로 사용됩니다. • SQL_SEP_BLANK - 공백이 시간 분리자로 사용됩니다. • SQL_SEP_JOB - 작업 디폴트가 사용됩니다.
SQL_ATTR_UTF8	32비트 정수 값은 다음 중 하나일 수 있습니다. • SQL_FALSE - 문자 자료가 디폴트 작업 CSID에서와 마찬가지로 처리됩니다. 이것이 디폴트 값입니다. • SQL_TRUE - 문자 자료가 UTF-8 CCSID(1208)에서와 마찬가지로 처리됩니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 160. SQLSetEnvAttr SQLSTATEs

SQLSTATE	설명	설명
HY009	유효하지 않은 매개변수 값	지정된 *Attribute*는 DB2 UDB CLI에 의해 지원되지 않습니다. 지정된 *Attribute* 값의 경우 *Value* 인수에 대해 지정된 값이 지원되지 않습니다. *pValue* 인수가 널(null) 포인터입니다.

표 160. SQLSetEnvAttr SQLSTATEs (계속)

SQLSTATE	설명	설명
HY010	함수 순서 오류	연결 핸들이 이미 할당되어 있습니다.

SQLSetParam - 매개변수 설정

목적

SQLSetParam()은 어플리케이션 변수를 SQL문의 매개변수 마커와 연관(바인드)시킵니다. 명령문이 실행되면 바인드된 변수의 내용이 데이터베이스 서버에 송신됩니다. 이 함수는 필요한 자료 변환을 지정하기 위해서도 사용됩니다.

구문

```
SQLRETURN SQLSetParam (SQLHSTMT       hstmt,
                       SQLSMALLINT    ipar,
                       SQLSMALLINT    fCType,
                       SQLSMALLINT    fSqlType,
                       SQLINTEGER     cbParamDef,
                       SQLSMALLINT    ibScale,
                       SQLPOINTER     rgbValue,
                       SQLINTEGER     *pcbValue);
```

주: 이 함수에 대한 설명은 50 페이지의 『SQLBindParam - 매개변수 마커에 버퍼 바인드』를 참조하십시오. 함수는 호환성을 위해 지원됩니다.

SQLSetStmtAttr - 명령문 속성 설정

목적

SQLSetStmtAttr()는 특정 명령문 핸들의 속성을 설정합니다. 연결 핸들과 연관된 모든 명령문 핸들에 대한 옵션을 설정하기 위해 어플리케이션은 SQLSetConnectOption()을 호출할 수 있습니다. (추가 세부사항에 대해서는 241 페이지의 『SQLSetConnectOption - 연결 옵션 설정』을 참조하십시오.)

구문

```
SQLRETURN SQLSetStmtAttr  (SQLHSTMT       hstmt,
                           SQLINTEGER     fAttr,
                           SQLPOINTER     vParam,
                           SQLINTEGER     sLen);
```

함수 인수

표 161. SQLSetStmtAttr 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLINTEGER	*fAttr*	입력	설정할 속성. 설정할 수 있는 명령문 속성 리스트에 대해서는 표 162를 참조하십시오.
SQLPOINTER	*vParam*	입력	*fAttr*과 연관된 값. *vParam*은 32비트 정수 값 또는 문자 스트링일 수 있습니다.
SQLINTEGER	*sLen*	입력	자료가 문자 스트링인 경우 자료의 길이. 그렇지 않은 경우는 사용되지 않습니다.

사용법

다른 SQLSetStmtAttr() 호출에 의해 옵션이 변경되거나 SQL_DROP과 함께 SQLFreeStmt()를 호출하여 *hstmt*가 제거될 때까지 *hstmt*에 대한 명령문 옵션은 유효합니다. SQL_CLOSE, SQL_UNBIND, 또는 SQL_RESET_PARAMS 옵션으로 SQLFreeStmt()를 호출해도 명령문 옵션을 재설정하지는 않습니다.

*vParam*을 통해 설정된 정보 형식은 지정된 *fOption*에 따라 달라집니다. 각각의 형식은 표 162에 나와 있습니다.

표 162. 명령문 속성

fAttr	내용
SQL_ATTR_APP_PARAM_DESC	*VParam*은 설명자 핸들이어야 합니다. 지정된 설명자는 명령문 핸들에서의 이후의 SQLExecDirect() 호출에 대한 어플리케이션 매개변수 설명자로 작용합니다.
SQL_ATTR_APP_ROW_DESC	*VParam*은 설명자 핸들이어야 합니다. 지정된 설명자는 명령문 핸들에서의 이후의 SQLFetch() 호출에 대한 어플리케이션 행 설명자로 작용합니다.

SQLSetStmtAttr

표 162. 명령문 속성 (계속)

fAttr	내용
SQL_ATTR_CURSOR_HOLD	이 명령문 핸들에 대해 열린 커서를 보유할 수 있는지를 지정하는 32비트 정수 값 • SQL_FALSE - 확약 또는 롤백 조작 시 이 명령문에 대해 열린 커서가 닫힙니다. 이것이 디폴트 값입니다. • SQL_TRUE - 확약 또는 롤백 조작 시 이 명령문에 대해 열린 커서가 닫히지 않습니다.
SQL_ATTR_CURSOR_SCROLLABLE	이 명령문 핸들에 대해 열린 커서를 스크롤할 수 있는지를 지정하는 32비트 정수 값 • SQL_FALSE - 커서를 스크롤할 수 없으며 커서에 대해 SQLFetchScroll()을 사용할 수 없습니다. 이것이 디폴트 값입니다. • SQL_TRUE - 커서를 스크롤할 수 있습니다. SQLFetchScroll()을 사용하여 이 커서의 자료를 검색할 수 있습니다.
SQL_ATTR_CURSOR_TYPE	이 명령문 핸들에 대해 열린 커서의 작동을 지정하는 32비트 정수 값. • SQL_CURSOR_FORWARD_ONLY - 커서를 스크롤 할 수 없으며 커서에 대해 SQLFetchScroll()을 사용할 수 없습니다. 이것이 디폴트 값입니다. • SQL_DYNAMIC - 커서를 스크롤할 수 있습니다. SQLFetchScroll()을 사용하여 이 커서의 자료를 검색할 수 있습니다.
SQL_ATTR_EXTENDED_COL_INFO	이 명령문 핸들에 대해 열린 커서가 확장 열 정보를 제공하는지를 지정하는 32비트 정수 값 • SQL_FALSE - 확장 열 정보를 검색하기 위해 SQLColAttributes() 함수에 이 명령문 핸들을 사용할 수 없습니다. 이것이 디폴트 값입니다. 명령문 레벨에서 이 속성을 설정하면 속성의 연결 레벨 설정이 대체됩니다. • SQL_TRUE - 이 명령문 핸들을 SQLColAttributes()에 사용하여 기본 표, 기본 열, 레이블과 같은 확장 정보를 검색할 수 있습니다.
SQL_ATTR_FOR_FETCH_ONLY	이 명령문 핸들에 대해 열린 커서가 읽기 전용이어야 하는지를 지정하는 32비트 정수 값 • SQL_TRUE - 커서가 읽기 전용이고 위치지정된 갱신과 삭제에 대해 사용될 수 없습니다. SQL_ATTR_FOR_FETCH_ONLY 환경이 SQL_FALSE로 설정되지 않은 경우 기본값입니다. • SQL_FALSE - 위치지정된 갱신과 삭제에 대해 커서를 사용할 수 있습니다.

표 162. 명령문 속성 (계속)

fAttr	내용
SQL_ATTR_FULL_OPEN	이 명령문 핸들에 대해 열린 커서를 열어 놓을 수 있는지를 지정하는 32비트 정수 값 • SQL_FALSE - 이 명령문에 대해 열린 커서가 성능 상의 이유로 캐시 처리된 커서를 사용할 수 있습니다. 이것이 디폴트 값입니다. • SQL_TRUE - 이 명령문에 대해 열린 커서가 항상 새로운 커서 열기를 강제로 시행합니다.
SQL_ATTR_ROWSET_SIZE	행 집합의 행 수를 지정하는 32비트 정수 값. 각각의 `SQLExtendedFetch()` 호출에 의해 리턴된 행 번호입니다. 디폴트 값은 1입니다.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 163. SQLStmtAttr SQLSTATEs

SQLSTATE	설명	설명
40003 *****	명령문 완료 상태를 알 수 없음	함수가 처리를 완료하기 전에 CLI와 자료 소스와의 통신 링크가 끊어졌습니다.
HY000	일반 오류	특정한 SQLSTATE가 없거나 구현 프로그램에서 정의한 SQLSTATE가 정의되지 않은 오류가 발생했습니다. *szErrorMsg* 인수에서 SQLError()에 의해 리턴된 오류 메세지는 오류와 그 원인을 설명합니다.
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*fAttr* 값의 경우 vParam 인수에 대해 유효하지 않은 값이 지정되었습니다. 유효하지 않은 *fAttr* 값이 지정되었습니다. *vParam* 인수가 널(null) 포인터입니다.
HY010	함수 순서 오류	함수가 올바르지 않은 순서로 호출됩니다.
HYC00	드라이버가 지원되지 않음	드라이버 또는 자료 소스가 지정된 옵션을 지원하지 않습니다.

SQLSetStmtOption - 명령문 옵션 설정

목적

SQLSetStmtOption()는 특정 명령문 핸들의 속성을 설정합니다. 연결 핸들과 연관된 모든 명령문 핸들에 대한 옵션을 설정하기 위해 어플리케이션은 SQLSetConnectOption()을 호출할 수 있습니다. (추가 세부사항에 대해서는 241 페이지의 『SQLSetConnectOption - 연결 옵션 설정』을 참조하십시오.)

구문

```
SQLRETURN SQLSetStmtOption  (SQLHSTMT       hstmt,
                             SQLSMALLINT    fOption,
                             SQLPOINTER     vParam);
```

함수 인수

표 164. SQLSetStmtOption 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLSMALLINT	*fOption*	입력	설정할 옵션. 설정할 수 있는 명령문 옵션 리스트에 대해서는 255 페이지의 표 162를 참조하십시오.
SQLPOINTER	*vParam*	입력	*fOption*과 연관된 값. *vParam*은 32비트 정수 값 또는 문자 스트링을 가리키는 포인터일 수 있습니다.

사용법

SQLSetStmtOption()은 SQLSetStmtAttr()과 같은 기능을 제공하며 호환성을 위해 두 함수 모두 지원됩니다.

다른 SQLSetStmtOption() 호출에 의해 옵션이 변경되거나 SQL_DROP과 함께 SQLFreeStmt()를 호출하여 *hstmt*가 제거될 때까지 *hstmt*에 대한 명령문 옵션은 유효합니다. SQL_CLOSE, SQL_UNBIND, 또는 SQL_RESET_PARAMS 옵션으로 SQLFreeStmt()를 호출해도 명령문 옵션을 재설정하지는 않습니다.

*vParam*을 통해 설정된 정보 형식은 지정된 *fOption*에 따라 달라집니다. 각각의 형식은 255 페이지의 표 162에 나와 있습니다.

적절한 명령문 옵션에 대해서는 255 페이지의 표 162를 참조하십시오.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 165. SQLStmtOption SQLSTATEs

SQLSTATE	설명	설명
40003 *	명령문 완료 상태를 알 수 없음	함수가 처리를 완료하기 전에 CLI와 자료 소스와의 통신 링크가 끊어졌습니다.
HY000	일반 오류	특정한 SQLSTATE가 없거나 구현 프로그램에서 정의한 SQLSTATE가 정의되지 않은 오류가 발생했습니다. *szErrorMsg* 인수에서 SQLError()에 의해 리턴된 오류 메세지는 오류와 그 원인을 설명합니다.
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 값	*fOption* 값의 경우 vParam 인수에 대해 유효하지 않은 값이 지정되었습니다. 유효하지 않은 *fOption* 값이 지정되었습니다. *szSchemaName* 또는 *szTableName* 인수가 널(null) 포인터입니다.
HY010	함수 순서 오류	함수가 올바르지 않은 순서로 호출됩니다.
HYC00	드라이버가 지원되지 않음	드라이버 또는 자료 소스가 지정된 옵션을 지원하지 않습니다.

SQLSpecialColumns - 특별한(행 ID) 열 얻기

목적

SQLSpecialColumns()는 표에 대한 고유한 행 ID 정보(1차 키 또는 고유 색인)를 리턴합니다. 예를 들면, 고유 색인 또는 1차 키 정보입니다. SELECT문에 의해 생성된 결과 집합을 페치하기 위해 사용된 함수와 같은 함수를 사용하여 검색될 수 있는 SQL 결과 집합에 이 정보가 리턴됩니다.

구문

```
SQLRETURN SQLSpecialColumns  (SQLHSTMT       hstmt,
                              SQLSMALLINT    fColType,
                              SQLCHAR        *szCatalogName,
                              SQLSMALLINT    cbCatalogName,
                              SQLCHAR        *szSchemaName,
                              SQLSMALLINT    cbSchemaName,
                              SQLCHAR        *szTableName,
                              SQLSMALLINT    cbTableName,
                              SQLSMALLINT    fScope,
                              SQLSMALLINT    fNullable);
```

함수 인수

표 166. SQLSpecialColumns 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLSMALLINT	*fColType*	입력	추가 유형의 특별한 열을 지원하기 위해 이후에 사용할 수 있도록 예약됨. 이 자료 유형은 현재 무시됩니다.
SQLCHAR *	*szCatalogName*	입력	3부분으로 이루어진 표 이름의 카탈로그 규정자. 이 포인터가 널(null) 포인터이거나 길이가 0인 스트링이어야 합니다.
SQLSMALLINT	*cbCatalogName*	입력	*szCatalogName*의 길이. 0으로 설정되어야 합니다.
SQLCHAR *	*szSchemaName*	입력	지정된 표의 스키마 규정자.
SQLSMALLINT	*cbSchemaName*	입력	*szSchemaName*의 길이.
SQLCHAR *	*szTableName*	입력	표 이름.
SQLSMALLINT	*cbTableName*	입력	*cbTableName*의 길이.

표 166. SQLSpecialColumns 인수 (계속)

자료 유형	인수	사용	설명
SQLSMALLINT	*fScope*	입력	고유한 행 ID가 유효한 최소한의 요구 기간. *fScope*는 다음 중 하나의 값이어야 합니다. • SQL_SCOPE_CURROW - 행 ID가 해당 행에 위치하는 동안만 유효함을 보장합니다. 행이 다른 트랜잭션에 의해 갱신되거나 삭제된 경우 같은 행 ID 값을 사용하여 나중에 다시 선택하면 행을 리턴하지 않을 수도 있습니다. • SQL_SCOPE_TRANSACTION - 행 ID가 현재 트랜잭션 동안만 유효함을 보장합니다. • SQL_SCOPE_SESSION - 행 ID가 연결된 동안만 유효함을 보장합니다. 행 ID의 유효성이 보장되는 기간 이상의 기간은 현재 트랜잭션의 분리 레벨에 따라 달라집니다. 분리 레벨과 관련된 시나리오와 정보는 IBM DB2 SQL Reference를 참조하십시오.
SQLSMALLINT	*fNullable*	입력	널(null)값을 가질 수 있는 특별한 열을 리턴할 지를 판별합니다. 다음 값 중 하나이어야 합니다. • SQL_NO_NULLS 리턴된 행 ID 열 집합은 널(null)값을 가질 수 없습니다. • SQL_NULLABLE 리턴된 행 ID 열 집합은 널(null)값이 허용되는 열을 포함할 수 있습니다.

사용법

표에서 행을 고유하게 식별하는 방법이 여러 가지일 경우(예를 들어, 지정된 표에 복수 고유 색인이 있는 경우) DB2 UDB CLI는 내부 기준에 의해 *최선의* 행 ID 열 집합을 리턴합니다.

표의 행을 고유하게 식별할 수 있는 열 집합이 없다면 빈 결과 집합이 리턴됩니다.

고유 행 ID 정보는 행 ID의 각 열이 결과 집합의 한 행에 의해 표시되는 결과 집합 형태로 리턴됩니다. `SQLSpecialColumns()`에 의해 리턴된 결과 집합에는 다음 순서로 열이 있습니다.

SQLSpecialColumns

표 167. SQLSpecialColumns에 의해 리턴되는 열

열명(column name)	자료 유형	설명
SCOPE	SMALLINT는 널(null)이 아님	rowid의 실제 범위. 다음 값 중 하나가 있어야 합니다. • SQL_SCOPE_CURROW • SQL_SCOPE_TRANSACTION • SQL_SCOPE_SESSION 각 값에 대한 설명은 260 페이지의 표 166의 *fScope*를 참조하십시오.
COLUMN_NAME	VARCHAR(128)는 널(null)이 아님	행 ID 열명
DATA_TYPE	SMALLINT는 널(null)이 아님	열의 SQL 자료 유형
TYPE_NAME	VARCHAR(128)는 널(null)이 아님	DATA_TYPE 열 값과 연관된 이름을 표시하는 데이터베이스 관리 시스템(DBMS) 문자 스트링
LENGTH_PRECISION	INTEGER	열의 정밀도. 정밀도를 적용할 수 없는 자료 유형에 대해서는 널(null)이 리턴됩니다.
BUFFER_LENGTH	INTEGER	디폴트 C 유형에 리턴되는 자료의 길이(바이트). CHAR 자료 유형의 경우 LENGTH_PRECISION 열의 값과 같습니다.
SCALE	SMALLINT	열의 소수 자릿수(scale). 소수 자릿수를 적용할 수 없는 자료 유형에 대해서는 널(null)이 리턴됩니다.
PSEUDO_COLUMN	SMALLINT	열이 의사 열인지를 표시하며 DB2 UDB CLI는 다음 사항만 리턴합니다. • SQL_PC_NOT_PSEUDO

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 168. SQLSpecialColumns SQLSTATEs

SQLSTATE	설명	설명
24000	유효하지 않은 커서 상태	커서 관련 정보가 요구되었지만 열린 커서가 없습니다.
40003 *****	명령문 완료 상태를 알 수 없음	함수가 처리를 완료하기 전에 CLI와 자료 소스와의 통신 링크가 끊어졌습니다.
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.

표 168. SQLSpecialColumns SQLSTATEs (계속)

SQLSTATE	설명	설명
HY009	유효하지 않은 인수 길이	길이 인수 중 하나의 값이 0 미만이지만 SQL_NTS와 같지 않습니다.
HYC00	드라이버가 지원되지 않음	자료 소스는 세 부분 표 이름의 *catalog* 부분(첫 번째 부분)을 지원하지 않습니다.

SQLStatistics - 기본 표에 대한 색인과 통계 정보 얻기

목적

SQLStatistics()는 주어진 표에 대한 색인 정보를 검색합니다. 또한 표와 표의 색인과 연관된 페이지 번호와 기수(cardinality)도 리턴합니다. SELECT문에 의해 생성된 결과 집합을 페치하기 위해 사용된 함수와 같은 함수를 사용하여 검색될 수 있는 결과 집합에 이 정보가 리턴됩니다.

구문

```
SQLRETURN SQLStatistics  (SQLHSTMT       hstmt,
                          SQLCHAR        *szCatalogName,
                          SQLSMALLINT    cbCatalogName,
                          SQLCHAR        *szSchemaName,
                          SQLSMALLINT    cbSchemaName,
                          SQLCHAR        *szTableName,
                          SQLSMALLINT    cbTableName,
                          SQLSMALLINT    fUnique,
                          SQLSMALLINT    fAccuracy);
```

함수 인수

표 169. SQLStatistics 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLCHAR *	*szCatalogName*	입력	3부분으로 이루어진 표 이름의 카탈로그 규정자. 이 포인터가 널(null) 포인터이거나 길이가 0인 스트링이어야 합니다.
SQLSMALLINT	*cbCatalogName*	입력	*cbCatalogName*의 길이. 0으로 설정되어야 합니다.
SQLCHAR *	*szSchemaName*	입력	지정된 표의 스키마 규정자.
SQLSMALLINT	*cbSchemaName*	입력	*szSchemaName*의 길이.
SQLCHAR *	*szTableName*	입력	표 이름.
SQLSMALLINT	*cbTableName*	입력	*cbTableName*의 길이.
SQLSMALLINT	*fUnique*	입력	리턴할 색인 정보 유형. • SQL_INDEX_UNIQUE 고유 색인만이 리턴됩니다. • SQL_INDEX_ALL 모든 색인이 리턴됩니다.
SQLSMALLINT	*fAccuracy*	입력	현재 사용되지 않습니다. 0으로 설정되어야 합니다.

사용법

SQLStatistics()는 다음 정보 유형을 리턴합니다.

- 표에 대한 통계 정보(사용할 수 있는 경우):
  - 다음 표의 TYPE 열이 SQL_TABLE_STAT로 설정된 경우, 표의 행 수와 표를 저장하기 위해 사용된 페이지 수.

– TYPE 열이 색인을 나타내는 경우, 색인의 고유 값 수와 색인을 저장하기 위해 사용된 페이지 수.
– 각 색인 열이 결과 집합의 한 행에 의해 표시되는 각 색인에 대한 정보. 결과 집합 열은 다음 표에 표시된 순서로 저장됩니다. 결과 집합의 행은 NON_UNIQUE, TYPE, INDEX_QUALIFIER, INDEX_QUALIFIER, INDEX_NAME와 ORDINAL_POSITION에 의해 순서가 정해집니다.

표 170. SQLStatistics에 의해 리턴되는 열

열명(column name)	자료 유형	설명
TABLE_CAT	VARCHAR(128)	TABLE_SCHEM이 들어 있는 카탈로그명. 널(null)로 설정됩니다.
TABLE_SCHEM	VARCHAR(128)	TABLE_NAME이 있는 스키마명.
TABLE_NAME	VARCHAR(128)는 널(null)이 아님	표의 이름.
NON_UNIQUE	SMALLINT	색인이 중복 값을 금지하는 지를 표시합니다. • 색인이 중복 값을 허용하는 경우 TRUE • 색인이 고유한 값이어야 하는 경우 FALSE • 이 행이 SQL_TABLE_STAT(표 자체의 통계 정보)임을 TYPE 열이 표시하는 경우 널(null)
INDEX_QUALIFIER	VARCHAR(128)	색인명을 규정하기 위해 사용된 ID. TYPE 열이 SQL_TABLE_STAT를 표시하면 널(null)입니다.
INDEX_NAME	VARCHAR(128)	색인명. TYPE 열의 값이 SQL_TABLE_STAT이면, 이 열은 널(null) 값을 가집니다.
TYPE	SMALLINT는 널(null)이 아님	결과 집합의 이 행에 있는 정보 유형을 표시합니다. • SQL_TABLE_STAT 이 행에 표 자체의 통계 정보가 있는지를 표시합니다. • SQL_INDEX_CLUSTERED 이 행에 색인에 대한 정보가 들어 있고, 색인 유형이 클러스터 색인인지를 표시합니다. • SQL_INDEX_HASHED 이 행에 색인에 대한 정보가 들어 있고, 색인 유형이 해쉬 색인인지를 표시합니다. • SQL_INDEX_OTHER 이 행에 색인에 대한 정보가 들어 있고, 색인 유형이 클러스터나 해쉬 이외의 색인인지를 표시합니다. **주:** 현재, SQL_INDEX_OTHER가 유일한 가능한 유형입니다.
ORDINAL_POSITION	SMALLINT	이름이 INDEX_NAME 열에 주어진 색인 내의 열의 순서적 위치. TYPE 열의 값이 SQL_TABLE_STAT이면 이 열에 대해 널(null) 값이 리턴됩니다.
COLUMN_NAME	VARCHAR(128)	색인의 열명.
COLLATION	CHAR(1)	열에 대한 정렬 순서. 오름차순인 경우 ″**A**″, 내림차순인 경우 ″**D**″. TYPE 열의 값이 SQL_TABLE_STAT인 경우 널(null) 값이 리턴됩니다.

SQLStatistics

표 170. SQLStatistics에 의해 리턴되는 열 (계속)

열명(column name)	자료 유형	설명
CARDINALITY	INTEGER	• TYPE 열에 SQL_TABLE_STAT 값이 있으면 이 열에 표의 행 번호가 있습니다. • TYPE 열 값이 SQL_TABLE_STAT가 아니면 이 열에 색인의 고유한 값 번호가 있습니다. • 정보를 데이터베이스 관리 시스템(DBMS)에 제공하지 않으면 널(null) 값이 리턴됩니다.
PAGES	INTEGER	• TYPE 열에 SQL_TABLE_STAT 값이 있으면 이 열에 표를 저장하기 위해 사용된 페이지 수가 있습니다. • TYPE 열 값이 SQL_TABLE_STAT가 아니면 이 열에 색인을 저장하기 위해 사용된 페이지 수가 있습니다. • 정보를 데이터베이스 관리 시스템(DBMS)에 제공하지 않으면 널(null) 값이 리턴됩니다.

표 통계가 있는 결과 집합의 행에 대해서는(TYPE이 SQL_TABLE_STAT로 설정), 열 값 NON_UNIQUE, INDEX_QUALIFIER, INDEX_NAME, ORDINAL_POSITION, COLUMN_NAME, COLLATION이 널(null)로 설정됩니다. CARDINALITY 또는 PAGES 정보가 판별될 수 없는 경우 이 열에 대해 널(null)이 리턴됩니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 171. SQLStatistics SQLSTATEs

SQLSTATE	설명	설명
24000	유효하지 않은 커서 상태	커서 관련 정보가 요구되었지만 열린 커서가 없습니다.
40003 *	명령문 완료 상태를 알 수 없음	함수가 처리를 완료하기 전에 CLI와 자료 소스와의 통신 링크가 끊어졌습니다.
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 또는 버퍼 길이	이름 길이 인수 중 하나의 값이 0 미만이었으나 SQL_NTS와 같지 않습니다.
HYC00	드라이버가 지원되지 않음	자료 소스는 세 부분 표 이름의 카탈로그 부분(첫 번째 부분)을 지원하지 않습니다.

SQLTablePrivileges - 표와 연관된 권한 확보

목적

SQLTablePrivileges() 는 표 목록 및 각 표에 대한 연관된 권한을 리턴합니다. 조회에서 생성된 결과 집합을 프로세스하는 데 사용된 동일한 함수를 사용하여 검색할 수 있는 SQL 결과 집합에 이 정보가 리턴됩니다.

구문

```
SQLRETURN SQLTablePrivileges  (SQLHSTMT         StatementHandle,
                               SQLCHAR          *CatalogName,
                               SQLSMALLINT      NameLength1,
                               SQLCHAR          *SchemaName,
                               SQLSMALLINT      NameLength2,
                               SQLCHAR          *TableName,
                               SQLSMALLINT      NameLength3);
```

함수 인수

표 172. SQLTablePrivileges Arguments

자료 유형	인수	사용	설명
SQLHSTMT	*StatementHandle*	입력	명령문 핸들.
SQLCHAR *	*szTableQualifier*	입력	세 부분으로 된 표 이름의 카탈로그 규정자. 이 포인터가 널(null) 포인터이거나 길이가 0인 스트링이어야 합니다.
SQLSMALLINT	*cbTableQualifier*	입력	*CatalogName.*의 길이가 0으로 설정되어야 합니다.
SQLCHAR *	*SchemaName*	입력	스키마명에 의해 결과 집합을 규정하기 위해 *패턴 값*을 포함하는 버퍼
SQLSMALLINT	*NameLength2*	입력	*SchemaName.*의 길이
SQLCHAR *	*TableName*	입력	표 이름에 의해 결과 집합을 규정하기 위한 *패턴 값*이 있는 버퍼.
SQLSMALLINT	*NameLength3*	입력	*TableName.*의 길이

사용법

다음 표에 나열된 열이 있는 표준 결과 집합으로서 결과가 리턴됩니다. 결과 집합이 TABLE_CAT, TABLE_SCHEM, TABLE_NAME, COLUMN_NAME 및 PRIVILEGE에 의해 주문됩니다. 다중 권한이 지정된 열과 연관된 경우, 각 권한은 분리 행으로서 리턴됩니다.

여기에 보고된 각 권한의 그래뉴래리티는 열 레벨에서 적용될 수도 있고 적용되지 않을 수도 있습니다. 예를 들어 일부 자료 소스의 경우 표 갱신이 가능하면 그 표의 모든 열도 갱신할 수 있습니다. 다른 자료 소스의 경우 어플리케이션이 SQLColumnPrivileges()를 호출하여 각 열에 동일한 표 권한이 있는지 알아내야 합니다.

많은 경우에서 SQLColumnPrivileges() 호출은 시스템 카탈로그에 반해 복잡하고 비용이 많이 드는 조회에 맵핑되므로 자주 사용하지 않는 것이 좋으며 반복 호출이 아닌 결과를 저장합니다.

SQLTablePrivileges

카탈로그 함수 결과 집합의 VARCHAR 열이 SQL 92 제한 길이와 일치하도록 최대 길이 인수인 128로 선언됩니다. DB2 이름이 128 보다 적기 때문에 어플리케이션이 출력 버퍼에 128문자(및 널 터미네이터)를 항상 별개로 설정하거나, SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_TABLE_NAME_LEN 및 SQL_MAX_COLUMN_NAME_LEN을 사용해서 SQLGetInfo()를 호출하여 연결된 DBMS에 의해 지원되는 TABLE_CAT, TABLE_SCHEM, TABLE_NAME 및 COLUMN_NAME 열의 실제 길이를 각각 판별하도록 선택할 수 있습니다.

새로운 열이 추가되고 기존 열의 이름이 이후의 릴리스에서 변경될 수 있지만 현재 열의 위치는 변경되지 않습니다.

표 173. SQLTablePrivileges에 의해 리턴되는 열

열명(column name)	자료 유형	설명
TABLE_CAT	VARCHAR(128)	이 열은 항상 널(null)입니다.
TABLE_SCHEM	VARCHAR(128)	TABLE_NAME이 있는 스키마명.
TABLE_NAME	VARCHAR(128)는 널(null)이 아님	표 이름
GRANTOR	VARCHAR(128)	권한을 부여한 사용자의 권한 부여 ID
GRANTEE	VARCHAR(128)	권한을 부여받은 사용자의 권한 부여 ID
PRIVILEGE	VARCHAR(128)	표 권한. 다음 스트링 중 하나입니다. • ALTER • CONTROL • INDEX • DELETE • INSERT • REFERENCES • SELECT • UPDATE
IS_GRANTABLE	VARCHAR(3)	권한 수여자가 권한을 다른 사용자에게 부여하는 지를 표시합니다. ″YES″, ″NO″ 또는 ″NULL″이 될 수 있습니다.

주: DB2 CLI에 의해 사용된 열 이름은 X/Open CLI CAE 스펙 방식을 따릅니다. 열 유형, 내용 및 순서는 ODBC의 SQLProcedures() 결과 집합에 정의된 값과 같습니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 174. SQLTablePrivileges SQLSTATEs

SQLSTATE	설명	설명
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 스트링 또는 버퍼 길이	이름 길이 인수 중 하나의 값이 0 미만이지만 SQL_NTS와 같지 않습니다.
HY010	함수 순서 오류	명령문 핸들에 대한 커서가 열려 있습니다. 이 명령문 핸들에 대한 연결이 없습니다.

제한사항

없음

예

```
/* From the CLI sample TBINFO.C */
/* ... */

    /* call SQLTablePrivileges */
    printf("\n    Call SQLTablePrivileges for:\n");
    printf("        tbSchemaPattern = %s\n", tbSchemaPattern);
    printf("        tbNamePattern = %s\n", tbNamePattern);
    sqlrc = SQLTablePrivileges( hstmt, NULL, 0,
                                tbSchemaPattern, SQL_NTS,
                                tbNamePattern, SQL_NTS);
    STMT_HANDLE_CHECK( hstmt, sqlrc);
```

참조

SQLTables - 표 정보 얻기

목적

SQLTables()는 연결된 자료 소스의 시스템 카탈로그에 저장된 연관 정보와 표 이름 리스트를 리턴합니다. 표 이름 리스트는 결과 집합으로 리턴되면 SELECT문에 의해 생성된 결과 집합을 검색하기 위해 사용된 함수와 같은 함수를 사용하여 검색될 수 있습니다.

구문

```
SQLRETURN SQLTables  (SQLHSTMT       hstmt,
                      SQLCHAR        *szCatalogName,
                      SQLSMALLINT    cbCatalogName,
                      SQLCHAR        *szSchemaName,
                      SQLSMALLINT    cbSchemaName,
                      SQLCHAR        *szTableName,
                      SQLSMALLINT    cbTableName,
                      SQLCHAR        *szTableType,
                      SQLSMALLINT    cbTableType);
```

함수 인수

표 175. SQLTables 인수

자료 유형	인수	사용	설명
SQLHSTMT	*hstmt*	입력	명령문 핸들.
SQLCHAR *	*szCatalogName*	입력	결과 집합을 규정하기 위한 *패턴 값*이 있는 버퍼. *카탈로그*는 3부분으로 이루어진 표 이름의 첫 번째 부분입니다. 이 포인터는 널(null) 포인터이거나 길이가 0인 스트링이어야 합니다.
SQLSMALLINT	*cbCatalogName*	입력	*szCatalogName*의 길이. 0으로 설정되어야 합니다.
SQLCHAR *	*szSchemaName*	입력	스키마명에 의해 결과 집합을 규정하기 위해 *패턴 값*을 포함하는 버퍼
SQLSMALLINT	*cbSchemaName*	입력	*szSchemaName*의 길이.
SQLCHAR *	*szTableName*	입력	표 이름에 의해 결과 집합을 규정하기 위한 *패턴 값*이 있는 버퍼
SQLSMALLINT	*cbTableName*	입력	*szTableName*의 길이.

표 175. SQLTables 인수 (계속)

자료 유형	인수	사용	설명
SQLCHAR *	*szTableType*	입력	표 유형에 따라 결과 집합을 규정하기 위한 *값 리스트*가 있는 버퍼. 값 리스트는 관심있는 유형에 대해 쉼표로 분리된 값의 리스트입니다. 유효한 표 유형 ID에는 ALL, BASE TABLE, TABLE, VIEW, SYSTEM TABLE이 있습니다. *szTableType* 인수가 널(null) 포인터이거나 길이가 0인 스트링이면 표 유형 ID에 대한 모든 가능성을 지정하는 것과 같습니다. SYSTEM TABLE이 지정되면 시스템 표와 시스템 뷰(있는 경우) 모두가 리턴됩니다. 표 유형은 인용 부호안에 넣을 수도 있고 넣지 않을 수도 있습니다.
SQLSMALLINT	*cbTableType*	입력	*szTableType*의 길이.

szCatalogName, szSchemaName, szTableName 인수는 탐색 패턴을 허용합니다.

이탈 문자는 실제 문자가 탐색 패턴에서 사용되도록 와일드카드 문자와 연계하여 지정될 수 있습니다. 이탈 문자는 SQL_ATTR_ESCAPE_CHAR 환경 속성에서 지정됩니다.

사용법

각각의 표가 결과 집합의 한 행에 의해 표시되는 결과 집합에 표 정보가 리턴됩니다.

SQLTables()에 의해 리턴되는 결과 집합에는 주어진 순서로 다음 표에 나열된 열이 있습니다.

표 176. SQLTables에 의해 리턴되는 열

열명(column name)	자료 유형	설명
TABLE_CAT	VARCHAR(128)	현재 서버.
TABLE_SCHEM	VARCHAR(128)	TABLE_NAME이 있는 스키마명.
TABLE_NAME	VARCHAR(128)	표, 보기, 별명, 또는 동의어 이름.
TABLE_TYPE	VARCHAR(128)	TABLE_NAME 열의 이름에 의해 주어진 유형을 표시합니다. 'TABLE', 'VIEW', 'BASE TABLE' 또는 'SYSTEM TABLE' 스트링 값을 가질 수 있습니다.
REMARKS	VARCHAR(254)	표에 대한 설명 정보가 있습니다.

리턴 코드

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

SQLTables

진단

표 177. SQLTables SQLSTATE

SQLSTATE	설명	설명
24000	유효하지 않은 커서 상태	커서 관련 정보가 요구되었지만 열린 커서가 없습니다.
40003 *****	명령문 완료 상태를 알 수 없음	함수가 처리를 완료하기 전에 CLI와 자료 소스와의 통신 링크가 끊어졌습니다.
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY009	유효하지 않은 인수 또는 버퍼 길이	이름 길이 인수 중 하나의 값이 0 미만이었으나 SQL_NTS와 같지 않습니다.
HYC00	드라이버가 지원되지 않음	자료 소스는 세 부분 표 이름의 카탈로그 부분(첫 번째 부분)을 지원하지 않습니다.

SQLTransact - 트랜잭션 관리

목적

SQLTransact()는 연결의 현재 트랜잭션을 확약하거나 롤백합니다.

연결 시간이나 이전 SQLTransact() 호출(최근) 이후의 연결에서 수행된 데이터베이스에 대한 모든 변경이 확약되거나 롤백됩니다.

트랜잭션이 연결에서 사용 중이면 어플리케이션은 데이터베이스를 단절하기 전에 SQLTransact()를 호출해야 합니다.

구문

```
SQLRETURN SQLTransact (SQLHENV        henv,
                       SQLHDBC        hdbc,
                       SQLSMALLINT    fType);
```

함수 인수

표 178. SQLTransact 인수

자료 유형	인수	사용	설명
SQLHENV	*henv*	입력	환경 핸들 *hdbc*가 유효한 연결 핸들이면 *henv*가 무시됩니다.
SQLHDBC	*hdbc*	입력	데이터베이스 연결 핸들 *hdbc*가 SQL_NULL_HDBC로 설정되면 *henv*에 연결과 연관된 환경 핸들이 있어야 합니다.
SQLSMALLINT	*fType*	입력	트랜잭션에 대해 원하는 조치. 이 인수에 대한 값은 다음 중 하나여야 합니다. • SQL_COMMIT • SQL_ROLLBACK • SQL_COMMIT_HOLD • SQL_ROLLBACK_HOLD

사용법

트랜잭션을 SQL_COMMIT 또는 SQL_ROLLBACK으로 완료하면 다음과 같은 효과가 있습니다.

- SQLTransact()를 호출한 후에도 명령문 핸들이 유효합니다.
- 커서명, 바인드 매개변수, 열 바인딩이 트랜잭션보다 오래 살아 있습니다.
- 열린 커서가 닫히고 검색을 지연 중인 결과 집합이 삭제됩니다.

트랜잭션을 SQL_COMMIT_HOLD 또는 SQL_ROLLBACK_HOLD로 종료하면 데이터베이스 변경을 확약하거나 롤백하지만 커서가 닫히지 않습니다.

SQLTransact

연결에서 현재 사용 중인 트랜잭션이 없으면 SQLTransact()를 호출해도 데이터베이스 서버에 아무런 영향을 주지 않으며 SQL_SUCCESS를 리턴합니다.

연결 손실로 인해 COMMIT 또는 ROLLBACK을 실행하는 동안 SQLTransact()가 실패할 수 있습니다. 이 경우에는 어플리케이션이 COMMIT 또는 ROLLBACK이 처리되었는지 판별할 수 없으므로 데이터베이스 관리자에게 도움을 요청해야 합니다. 트랜잭션 기록부와 기타 트랜잭션 관리 타스크에 대한 자세한 정보는 데이터베이스 관리 시스템(DBMS) 제품 정보를 참조하십시오.

리턴 코드

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

진단

표 179. SQLTransact SQLSTATE

SQLSTATE	설명	설명
08003	연결이 열려 있지 않음	*hdbc*가 연결 상태가 아닙니다.
08007	트랜잭션이 실행되는 동안 연결 실패	*hdbc*와 연관된 연결이 함수가 실행되는 동안 실패하였으며 실패하기 전에 요구된 COMMIT 또는 ROLLBACK이 처리되었는지 판별할 수 없습니다.
58004	시스템 오류	복구할 수 없는 시스템 오류.
HY001	메모리 할당 실패	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리를 할당할 수 없습니다.
HY012	유효하지 않은 트랜잭션 조작 상태	*fType* 인수에 대해 지정된 값이 SQL_COMMIT나 SQL_ROLLBACK이 아닙니다.
HY013 *****	메모리 관리 문제	드라이버가 함수의 실행 또는 완료를 지원하는데 필요한 메모리에 액세스할 수 없습니다.

예

110 페이지의 『예』를 참조하십시오.

부록 A. DB2 UDB CLI 일반 진단 정보

이 부록은 이 책의 여러 섹션이 참조하고 있는 정보 표를 수록하고 있습니다.

DB2 UDB CLI 함수 리턴 코드

리턴 코드	값	설명
SQL_SUCCESS	0	함수가 성공적으로 완료되었고, SQLSTATE 정보는 추가로 제공되지 않습니다.
SQL_SUCCESS_WITH_INFO	1	경고나 기타 정보와 함께 함수 실행이 성공적으로 완료되었습니다. SQLSTATE 및 기타 오류 정보를 수신하려면 SQLError()를 호출하십시오.
SQL_NO_DATA_FOUND	100	함수가 성공적으로 리턴되었지만 관련 정보를 찾을 수 없습니다.
SQL_ERROR	-1	함수가 실패하였습니다. SQLSTATE 및 기타 오류 정보를 수신하려면 SQLError()를 호출하십시오.
SQL_INVALID_HANDLE	-2	입력 인수로서 전달된 핸들(환경, 연결 또는 명령문 핸들)이 유효하지 않아 함수가 실패하였습니다.

부록 B. DB2 UDB CLI 포함 파일

DB2 UDB CLI에 사용되는 유일한 포함 파일이 sqlcli.h입니다.

```
/*** START HEADER FILE SPECIFICATIONS ****************************/
/*                                                               */
/* Header File Name: SQLCLI                                      */
/*                                                               */
/* Descriptive Name: Structured Query Language (SQL) Call Level  */
/*                   Interface.                                  */
/*                                                               */
/* 5716-SS1  (C) Copyright IBM Corp. 1995,1995                   */
/* All rights reserved.                                          */
/* US Government Users Restricted Rights -                       */
/* Use, duplication or disclosure restricted                     */
/* by GSA ADP Schedule Contract with IBM Corp.                   */
/*                                                               */
/* Licensed Materials-Property of IBM                            */
/*                                                               */
/*                                                               */
/* Description: The SQL Call Level Interface provides access to  */
/*              most SQL functions, without the need for a       */
/*              precompiler.                                     */
/*                                                               */
/* Header Files Included: SQLCLI                                 */
/*                                                               */
/* Function Prototype List:  SQLAllocConnect                     */
/*                           SQLAllocEnv                         */
/*                           SQLAllocHandle                      */
/*                           SQLAllocStmt                        */
/*                           SQLBindCol                          */
/*                           SQLBindFileToCol                    */
/*                           SQLBindFileToParam                  */
/*                           SQLBindParam                        */
/*                           SQLBindParameter                    */
/*                           SQLCancel                           */
/*                           SQLCloseCursor                      */
/*                           SQLColAttributes                    */
/*                           SQLColumns                          */
/*                           SQLConnect                          */
/*                           SQLCopyDesc                         */
/*                           SQLDataSources                      */
/*                           SQLDescribeCol                      */
/*                           SQLDescribeParam                    */
/*                           SQLDisconnect                       */
/*                           SQLDriverConnect                    */
/*                           SQLEndTran                          */
/*                           SQLError                            */
/*                           SQLExecDirect                       */
/*                           SQLExecute                          */
/*                           SQLExtendedFetch                    */
/*                           SQLFetch                            */
/*                           SQLFetchScroll                      */
/*                           SQLForeignKeys                      */
/*                           SQLFreeConnect                      */
/*                           SQLFreeEnv                          */
```

```
/*                          SQLFreeHandle                         */
/*                          SQLFreeStmt                           */
/*                          SQLGetCol                             */
/*                          SQLGetConnectOption                   */
/*                          SQLGetCursorName                      */
/*                          SQLGetConnectAttr                     */
/*                          SQLGetData                            */
/*                          SQLGetDescField                       */
/*                          SQLGetDescRec                         */
/*                          SQLGetDiagField                       */
/*                          SQLGetDiagRec                         */
/*                          SQLGetEnvAttr                         */
/*                          SQLGetFunctions                       */
/*                          SQLGetInfo                            */
/*                          SQLGetLength                          */
/*                          SQLGetPosition                        */
/*                          SQLGetStmtAttr                        */
/*                          SQLGetStmtOption                      */
/*                          SQLGetSubString                       */
/*                          SQLGetTypeInfo                        */
/*                          SQLLanguages                          */
/*                          SQLMoreResults                        */
/*                          SQLNativeSql                          */
/*                          SQLNumParams                          */
/*                          SQLNumResultCols                      */
/*                          SQLParamData                          */
/*                          SQLParamOptions                       */
/*                          SQLPrepare                            */
/*                          SQLPrimaryKeys                        */
/*                          SQLProcedureColumns                   */
/*                          SQLProcedures                         */
/*                          SQLPutData                            */
/*                          SQLReleaseEnv                         */
/*                          SQLRowCount                           */
/*                          SQLSetConnectAttr                     */
/*                          SQLSetConnectOption                   */
/*                          SQLSetCursorName                      */
/*                          SQLSetDescField                       */
/*                          SQLSetDescRec                         */
/*                          SQLSetEnvAttr                         */
/*                          SQLSetParam                           */
/*                          SQLSetStmtAttr                        */
/*                          SQLSetStmtOption                      */
/*                          SQLSpecialColumns                     */
/*                          SQLStatistics                         */
/*                          SQLTables                             */
/*                          SQLTransact                           */
/*                                                                */
/* Change Activity:                                               */
/*                                                                */
/* CFD List:                                                      */
/*                                                                */
/* FLAG REASON       LEVEL DATE   PGMR      CHANGE DESCRIPTION    */
/* ---- ------------ ----- ------ --------- ----------------------*/
/* $A0= D91823       3D60  941206 MEGERIAN  New Include           */
/* $A1= D94881       4D20  960816 MEGERIAN  V4R2M0 enhancements   */
/* $A2= D95600       4D30  970910 MEGERIAN  V4R3M0 enhancements   */
/* $A3= P3682850     4D40  981030 MEGERIAN  V4R4M0 enhancements   */
```

```
 /* $A4= D97596       4D50  990326 LJAMESON  V4R5M0 enhancements   */
 /*                                                               */
 /* End CFD List.                                                 */
 /*                                                               */
 /*  Additional notes about the Change Activity                   */
 /* End Change Activity.                                          */
 /*** END HEADER FILE SPECIFICATIONS ******************************/

#ifndef SQL_H_SQLCLI
   #define SQL_H_SQLCLI            /* Permit duplicate Includes */

#ifndef __SQL_EXTERN
    #ifdef __ILEC400__
        #define SQL_EXTERN extern
    #else
        #ifdef __cplusplus
            #define SQL_EXTERN extern "C nowiden"
        #else
            #define SQL_EXTERN extern "C"
        #endif
    #endif
    #define __SQL_EXTERN
#endif

/* generally useful constants */
#define  SQL_FALSE                 0
#define  SQL_TRUE                  1
#define  SQL_NTS                  -3  /* NTS = Null Terminated String    */
#define  SQL_SQLSTATE_SIZE         5   /* size of SQLSTATE, not including
                                          null terminating byte           */
#define  SQL_MAX_MESSAGE_LENGTH    512

/* RETCODE values             */
#define  SQL_SUCCESS              0
#define  SQL_SUCCESS_WITH_INFO    1
#define  SQL_NO_DATA_FOUND        100
#define  SQL_NEED_DATA            99
#define  SQL_NO_DATA              SQL_NO_DATA_FOUND
#define  SQL_ERROR               -1
#define  SQL_INVALID_HANDLE      -2

/* SQLFreeStmt option values  */
#define  SQL_CLOSE                0
#define  SQL_DROP                 1
#define  SQL_UNBIND               2
#define  SQL_RESET_PARAMS         3

/* SQLSetParam defines        */
#define  SQL_C_DEFAULT           99

/* SQLTransact option values  */
#define  SQL_COMMIT               0
#define  SQL_ROLLBACK             1
#define  SQL_COMMIT_HOLD          2
#define  SQL_ROLLBACK_HOLD        3

/* SQLDriverConnect option values  */
#define  SQL_DRIVER_COMPLETE          1
```

```
#define  SQL_DRIVER_COMPLETE_REQUIRED 1
#define  SQL_DRIVER_NOPROMPT          1

/*  Valid option codes for GetInfo procedure               */
#define  SQL_ACTIVE_CONNECTIONS    0
#define  SQL_ACTIVE_STATEMENTS     1
#define  SQL_PROCEDURES            2
#define  SQL_DBMS_NAME             17
#define  SQL_DBMS_VER              18
#define  SQL_MAX_COLUMN_NAME_LEN   30
#define  SQL_MAX_CURSOR_NAME_LEN   31
#define  SQL_MAX_OWNER_NAME_LEN    32
#define  SQL_MAX_SCHEMA_NAME_LEN   33
#define  SQL_MAX_TABLE_NAME_LEN    35

/* Standard SQL data types */
#define  SQL_CHAR                1
#define  SQL_NUMERIC             2
#define  SQL_DECIMAL             3
#define  SQL_INTEGER             4
#define  SQL_SMALLINT            5
#define  SQL_FLOAT               6
#define  SQL_REAL                7
#define  SQL_DOUBLE              8
#define  SQL_DATETIME            9
#define  SQL_VARCHAR            12
#define  SQL_BLOB               13
#define  SQL_CLOB               14
#define  SQL_DBCLOB             15
#define  SQL_DATALINK           16
#define  SQL_WCHAR              17
#define  SQL_WVARCHAR           18
#define  SQL_BIGINT             19
#define  SQL_BLOB_LOCATOR       20
#define  SQL_CLOB_LOCATOR       21
#define  SQL_DBCLOB_LOCATOR     22
#define  SQL_WLONGVARCHAR       SQL_WVARCHAR
#define  SQL_LONGVARCHAR        SQL_VARCHAR
#define  SQL_GRAPHIC            95
#define  SQL_VARGRAPHIC         96
#define  SQL_LONGVARGRAPHIC     SQL_VARGRAPHIC
#define  SQL_BINARY             97
#define  SQL_VARBINARY          98
#define  SQL_LONGVARBINARY      SQL_VARBINARY
#define  SQL_DATE               91
#define  SQL_TYPE_DATE          91
#define  SQL_TIME               92
#define  SQL_TYPE_TIME          92
#define  SQL_TIMESTAMP          93
#define  SQL_TYPE_TIMESTAMP     93
#define  SQL_CODE_DATE           1
#define  SQL_CODE_TIME           2
#define  SQL_CODE_TIMESTAMP      3
#define  SQL_ALL_TYPES           0

/*
 * NULL status defines; these are used in SQLColAttributes, SQLDescribeCol,
 * to describe the nullability of a column in a table.
```

```
 */
#define  SQL_UNUSED                          0
#define  SQL_HANDLE_ENV                      1
#define  SQL_HANDLE_DBC                      2
#define  SQL_HANDLE_STMT                     3
#define  SQL_HANDLE_DESC                     4
#define  SQL_NULL_HANDLE                     0

#define  SQL_NO_NULLS         0
#define  SQL_NULLABLE         1
#define  SQL_NULLABLE_UNKNOWN 2

/* Special length values  */
#define  SQL_NULL_DATA       -1
#define  SQL_DATA_AT_EXEC    -2
#define  SQL_BIGINT_PREC     19
#define  SQL_INTEGER_PREC    10
#define  SQL_SMALLINT_PREC    5


/* SQLColAttributes defines */
#define  SQL_ATTR_READONLY            0
#define  SQL_ATTR_WRITE               1
#define  SQL_ATTR_READWRITE_UNKNOWN   2

/* Valid concurrency values */
#define  SQL_CONCUR_LOCK              0
#define  SQL_CONCUR_READ_ONLY         1

/*  Valid environment attributes                    */
#define SQL_ATTR_OUTPUT_NTS          10001
#define SQL_ATTR_SYS_NAMING          10002
#define SQL_ATTR_DEFAULT_LIB         10003
#define SQL_ATTR_SERVER_MODE         10004
#define SQL_ATTR_JOB_SORT_SEQUENCE   10005
#define SQL_ATTR_ENVHNDL_COUNTER     10009
#define SQL_ATTR_ESCAPE_CHAR         10010

/*  Valid environment/connection attributes         */
#define SQL_ATTR_DATE_FMT            10020
#define SQL_ATTR_DATE_SEP            10021
#define SQL_ATTR_TIME_FMT            10022
#define SQL_ATTR_TIME_SEP            10023
#define SQL_ATTR_DECIMAL_SEP         10024

/*  Valid environment/connection values             */
#define SQL_FMT_ISO                  1
#define SQL_FMT_USA                  2
#define SQL_FMT_EUR                  3
#define SQL_FMT_JIS                  4
#define SQL_FMT_MDY                  5
#define SQL_FMT_DMY                  6
#define SQL_FMT_YMD                  7
#define SQL_FMT_JUL                  8
#define SQL_FMT_HMS                  9
#define SQL_FMT_JOB                  10
#define SQL_SEP_SLASH                1
#define SQL_SEP_DASH                 2
```

```
#define SQL_SEP_PERIOD               3
#define SQL_SEP_COMMA                4
#define SQL_SEP_BLANK                5
#define SQL_SEP_COLON                6
#define SQL_SEP_JOB                  7

/*  Valid values for type in GetCol                                  */
#define SQL_DEFAULT              99
#define SQL_ARD_TYPE            -99

/*  Valid values for UPDATE_RULE and DELETE_RULE in SQLForeignKeys  */
#define SQL_CASCADE              1
#define SQL_RESTRICT             2
#define SQL_NO_ACTION            3
#define SQL_SET_NULL             4
#define SQL_SET_DEFAULT          5

/*  Valid values for COLUMN_TYPE in SQLProcedureColumns  */
#define SQL_PARAM_INPUT             1
#define SQL_PARAM_OUTPUT            2
#define SQL_PARAM_INPUT_OUTPUT      3

/* statement attributes */
#define SQL_ATTR_APP_ROW_DESC      10010
#define SQL_ATTR_APP_PARAM_DESC    10011
#define SQL_ATTR_IMP_ROW_DESC      10012
#define SQL_ATTR_IMP_PARAM_DESC    10013
#define SQL_ATTR_FOR_FETCH_ONLY    10014
#define SQL_ATTR_CONCURRENCY       10014
#define SQL_CONCURRENCY            10014
#define SQL_ATTR_CURSOR_SCROLLABLE 10015
#define SQL_ATTR_ROWSET_SIZE       10016
#define SQL_ROWSET_SIZE            10016

 /*  Codes used in FetchScroll                                        */
#define SQL_FETCH_NEXT                  1
#define SQL_FETCH_FIRST                 2
#define SQL_FETCH_LAST                  3
#define SQL_FETCH_PRIOR                 4
#define SQL_FETCH_ABSOLUTE              5
#define SQL_FETCH_RELATIVE              6

/* SQLColAttributes defines */
#define SQL_DESC_COUNT                  1
#define SQL_DESC_TYPE                   2
#define SQL_DESC_LENGTH                 3
#define SQL_DESC_LENGTH_PTR             4
#define SQL_DESC_PRECISION              5
#define SQL_DESC_SCALE                  6
#define SQL_DESC_DATETIME_INTERVAL_CODE  7
#define SQL_DESC_NULLABLE               8
#define SQL_DESC_INDICATOR_PTR          9
#define SQL_DESC_DATA_PTR               10
#define SQL_DESC_NAME                   11
#define SQL_DESC_UNNAMED                12
#define SQL_DESC_DISPLAY_SIZE           13
#define SQL_DESC_ALLOC_TYPE             99
#define SQL_DESC_ALLOC_AUTO             1
```

```
#define SQL_DESC_ALLOC_USER             2

#define SQL_COLUMN_COUNT                  1
#define SQL_COLUMN_TYPE                   2
#define SQL_COLUMN_LENGTH                 3
#define SQL_COLUMN_LENGTH_PTR             4
#define SQL_COLUMN_PRECISION              5
#define SQL_COLUMN_SCALE                  6
#define SQL_COLUMN_DATETIME_INTERVAL_CODE  7
#define SQL_COLUMN_NULLABLE               8
#define SQL_COLUMN_INDICATOR_PTR          9
#define SQL_COLUMN_DATA_PTR               10
#define SQL_COLUMN_NAME                   11
#define SQL_COLUMN_UNNAMED                12
#define SQL_COLUMN_DISPLAY_SIZE           13
#define SQL_COLUMN_ALLOC_TYPE             99
#define SQL_COLUMN_ALLOC_AUTO             1
#define SQL_COLUMN_ALLOC_USER             2

/*  Valid codes for SpecialColumns procedure              */
#define SQL_SCOPE_CURROW         0
#define SQL_SCOPE_TRANSACTION    1
#define SQL_SCOPE_SESSION        2
#define SQL_PC_UNKNOWN           0
#define SQL_PC_NOT_PSEUDO        1
#define SQL_PC_PSEUDO            2

/*  Valid values for connect attribute                    */
#define SQL_ATTR_AUTO_IPD       10001
#define SQL_ATTR_ACCESS_MODE    10002
#define SQL_ACCESS_MODE         10002
#define SQL_ATTR_AUTOCOMMIT     10003
#define SQL_AUTOCOMMIT          10003
#define SQL_ATTR_DBC_SYS_NAMING 10004
#define SQL_ATTR_DBC_DEFAULT_LIB 10005
#define SQL_ATTR_COMMIT             0
#define SQL_MODE_READ_ONLY          0
#define SQL_MODE_READ_WRITE         1
#define SQL_MODE_DEFAULT            1
#define SQL_AUTOCOMMIT_OFF          0
#define SQL_AUTOCOMMIT_ON           1
#define SQL_TXN_ISOLATION           0
#define SQL_COMMIT_NONE             1
#define SQL_TXN_NO_COMMIT           1
#define SQL_TXN_NOCOMMIT            1
#define SQL_COMMIT_CHG              2
#define SQL_COMMIT_UR               2
#define SQL_TXN_READ_UNCOMMITTED    2
#define SQL_COMMIT_CS               3
#define SQL_TXN_READ_COMMITTED      3
#define SQL_COMMIT_ALL              4
#define SQL_COMMIT_RS               4
#define SQL_TXN_REPEATABLE_READ     4
#define SQL_COMMIT_RR               5
#define SQL_TXN_SERIALIZABLE        5

/*  Valid index flags                         */
#define SQL_INDEX_UNIQUE            0
```

```
#define SQL_INDEX_ALL               1
#define SQL_INDEX_OTHER             3

/*  Valid File Options                        */
#define SQL_FILE_READ               2
#define SQL_FILE_CREATE             8
#define SQL_FILE_OVERWRITE         16
#define SQL_FILE_APPEND            32

/* Valid types for GetDiagField                            */
#define SQL_DIAG_RETURNCODE         1
#define SQL_DIAG_NUMBER             2
#define SQL_DIAG_ROW_COUNT          3
#define SQL_DIAG_SQLSTATE           4
#define SQL_DIAG_NATIVE             5
#define SQL_DIAG_MESSAGE_TEXT       6
#define SQL_DIAG_DYNAMIC_FUNCTION   7
#define SQL_DIAG_CLASS_ORIGIN       8
#define SQL_DIAG_SUBCLASS_ORIGIN    9
#define SQL_DIAG_CONNECTION_NAME    10
#define SQL_DIAG_SERVER_NAME        11

/*
 * SQLColAttributes defines
 * These are also used by SQLGetInfo
 */
#define  SQL_UNSEARCHABLE            0
#define  SQL_LIKE_ONLY               1
#define  SQL_ALL_EXCEPT_LIKE         2
#define  SQL_SEARCHABLE              3

/* GetFunctions() values to identify CLI functions   */
#define  SQL_API_SQLALLOCCONNECT        1
#define  SQL_API_SQLALLOCENV            2
#define  SQL_API_SQLALLOCHANDLE      1001
#define  SQL_API_SQLALLOCSTMT           3
#define  SQL_API_SQLBINDCOL             4
#define  SQL_API_SQLBINDFILETOCOL    2002
#define  SQL_API_SQLBINDFILETOPARAM  2003
#define  SQL_API_SQLBINDPARAM        1002
#define  SQL_API_SQLBINDPARAMETER    1023
#define  SQL_API_SQLCANCEL              5
#define  SQL_API_SQLCLOSECURSOR      1003
#define  SQL_API_SQLCOLATTRIBUTES       6
#define  SQL_API_SQLCOLUMNS            40
#define  SQL_API_SQLCONNECT             7
#define  SQL_API_SQLCOPYDESC         1004
#define  SQL_API_SQLDATASOURCES        57
#define  SQL_API_SQLDESCRIBECOL         8
#define  SQL_API_SQLDESCRIBEPARAM      58
#define  SQL_API_SQLDISCONNECT          9
#define  SQL_API_SQLDRIVERCONNECT      68
#define  SQL_API_SQLENDTRAN          1005
#define  SQL_API_SQLERROR              10
#define  SQL_API_SQLEXECDIRECT         11
#define  SQL_API_SQLEXECUTE            12
#define  SQL_API_SQLEXTENDEDFETCH    1022
#define  SQL_API_SQLFETCH              13
```

```
#define  SQL_API_SQLFETCHSCROLL      1021
#define  SQL_API_SQLFOREIGNKEYS        60
#define  SQL_API_SQLFREECONNECT        14
#define  SQL_API_SQLFREEENV            15
#define  SQL_API_SQLFREEHANDLE       1006
#define  SQL_API_SQLFREESTMT           16
#define  SQL_API_SQLGETCOL             43
#define  SQL_API_SQLGETCONNECTATTR   1007
#define  SQL_API_SQLGETCONNECTOPTION   42
#define  SQL_API_SQLGETCURSORNAME      17
#define  SQL_API_SQLGETDATA            43
#define  SQL_API_SQLGETDESCFIELD     1008
#define  SQL_API_SQLGETDESCREC       1009
#define  SQL_API_SQLGETDIAGFIELD     1010
#define  SQL_API_SQLGETDIAGREC       1011
#define  SQL_API_SQLGETENVATTR       1012
#define  SQL_API_SQLGETFUNCTIONS       44
#define  SQL_API_SQLGETINFO            45
#define  SQL_API_SQLGETLENGTH        2004
#define  SQL_API_SQLGETPOSITION      2005
#define  SQL_API_SQLGETSTMTATTR      1014
#define  SQL_API_SQLGETSTMTOPTION      46
#define  SQL_API_SQLGETSUBSTRING     2006
#define  SQL_API_SQLGETTYPEINFO        47
#define  SQL_API_SQLLANGUAGES        2001
#define  SQL_API_SQLMORERESULTS        61
#define  SQL_API_SQLNATIVESQL          62
#define  SQL_API_SQLNUMPARAMS          63
#define  SQL_API_SQLNUMRESULTCOLS      18
#define  SQL_API_SQLPARAMDATA          48
#define  SQL_API_SQLPARAMOPTIONS     2007
#define  SQL_API_SQLPREPARE            19
#define  SQL_API_SQLPRIMARYKEYS        65
#define  SQL_API_SQLPROCEDURECOLUMNS   66
#define  SQL_API_SQLPROCEDURES         67
#define  SQL_API_SQLPUTDATA            49
#define  SQL_API_SQLRELEASEENV       1015
#define  SQL_API_SQLROWCOUNT           20
#define  SQL_API_SQLSETCONNECTATTR   1016
#define  SQL_API_SQLSETCONNECTOPTION   50
#define  SQL_API_SQLSETCURSORNAME      21
#define  SQL_API_SQLSETDESCFIELD     1017
#define  SQL_API_SQLSETDESCREC       1018
#define  SQL_API_SQLSETENVATTR       1019
#define  SQL_API_SQLSETPARAM           22
#define  SQL_API_SQLSETSTMTATTR      1020
#define  SQL_API_SQLSETSTMTOPTION      51
#define  SQL_API_SQLSPECIALCOLUMNS     52
#define  SQL_API_SQLSTATISTICS         53
#define  SQL_API_SQLTABLES             54
#define  SQL_API_SQLTRANSACT           23

/* NULL handle defines    */
#define  SQL_NULL_HENV                0L
#define  SQL_NULL_HDBC                0L
#define  SQL_NULL_HSTMT               0L

#if !defined(SDWORD)
```

```
typedef long int            SDWORD;
#endif
#if !defined(UDWORD)
typedef unsigned long int   UDWORD;
#endif
#if !defined(UWORD)
typedef unsigned short int  UWORD;
#endif
#if !defined(SWORD)
typedef signed short int    SWORD;
#endif

typedef  char               SQLCHAR;
typedef  long     int       SQLINTEGER;
typedef  short    int       SQLSMALLINT;
typedef  UWORD              SQLUSMALLINT;
typedef  UDWORD             SQLUINTEGER;
typedef  double             SQLDOUBLE;
typedef  float              SQLREAL;

typedef  void *             PTR;
typedef  PTR                SQLPOINTER;
typedef  long               HENV;
typedef  long               HDBC;
typedef  long               HSTMT;
typedef  long               HDESC;
typedef  HENV               SQLHENV;
typedef  HDBC               SQLHDBC;
typedef  HSTMT              SQLHSTMT;
typedef  HDESC              SQLHDESC;

typedef  SQLINTEGER         RETCODE;
typedef  RETCODE            SQLRETURN;

typedef  float              SFLOAT;



/*
 * DATE, TIME, and TIMESTAMP structures.  These are for compatibility
 * purposes only.  When actually specifying or retrieving DATE, TIME,
 * and TIMESTAMP values, character strings must be used.
 */


typedef  struct DATE_STRUCT
  {
    SQLSMALLINT   year;
    SQLSMALLINT   month;
    SQLSMALLINT   day;
  } DATE_STRUCT;

typedef  struct TIME_STRUCT
  {
    SQLSMALLINT   hour;
    SQLSMALLINT   minute;
    SQLSMALLINT   second;
  } TIME_STRUCT;
```

```
typedef  struct TIMESTAMP_STRUCT
  {
    SQLSMALLINT   year;
    SQLSMALLINT   month;
    SQLSMALLINT   day;
    SQLSMALLINT   hour;
    SQLSMALLINT   minute;
    SQLSMALLINT   second;
    SQLINTEGER    fraction;     /* fraction of a second */
  } TIMESTAMP_STRUCT;
```

```
SQL_EXTERN SQLRETURN  SQLAllocConnect (SQLHENV             henv,
                                       SQLHDBC            *phdbc);


SQL_EXTERN SQLRETURN  SQLAllocEnv     (SQLHENV             *phenv);

SQL_EXTERN SQLRETURN  SQLAllocHandle  (SQLSMALLINT          htype,
                                       SQLINTEGER           ihnd,
                                       SQLINTEGER          *ohnd);

SQL_EXTERN SQLRETURN  SQLAllocStmt    (SQLHDBC             hdbc,
                                       SQLHSTMT           *phstmt);

SQL_EXTERN SQLRETURN  SQLBindCol      (SQLHSTMT            hstmt,
                                       SQLSMALLINT         icol,
                                       SQLSMALLINT         iType,
                                       SQLPOINTER          rgbValue,
                                       SQLINTEGER          cbValueMax,
                                       SQLINTEGER         *pcbValue);

SQL_EXTERN SQLRETURN  SQLBindFileToCol (SQLHSTMT           hstmt,
                                       SQLSMALLINT         icol,
                                       SQLCHAR            *fName,
                                       SQLSMALLINT        *fNameLen,
                                       SQLINTEGER         *fOptions,
                                       SQLSMALLINT         fValueMax,
                                       SQLINTEGER         *sLen,
                                       SQLINTEGER         *pcbValue);

SQL_EXTERN SQLRETURN  SQLBindFileToParam (SQLHSTMT         hstmt,
                                       SQLSMALLINT         ipar,
                                       SQLSMALLINT         iType,
                                       SQLCHAR            *fName,
                                       SQLSMALLINT        *fNameLen,
                                       SQLINTEGER         *fOptions,
                                       SQLSMALLINT         fValueMax,
                                       SQLINTEGER         *pcbValue);

SQL_EXTERN SQLRETURN  SQLBindParam    (SQLHSTMT            hstmt,
                                       SQLSMALLINT         iparm,
                                       SQLSMALLINT         iType,
```

```
                                SQLSMALLINT       pType,
                                SQLINTEGER        pLen,
                                SQLSMALLINT       pScale,
                                SQLPOINTER        pData,
                                SQLINTEGER       *pcbValue);

SQL_EXTERN SQLRETURN  SQLBindParameter (SQLHSTMT          hstmt,
                                SQLSMALLINT       ipar,
                                SQLSMALLINT       fParamType,
                                SQLSMALLINT       fCType,
                                SQLSMALLINT       fSQLType,
                                SQLINTEGER        pLen,
                                SQLSMALLINT       pScale,
                                SQLPOINTER        pData,
                                SQLINTEGER        cbValueMax,
                                SQLINTEGER       *pcbValue);

SQL_EXTERN SQLRETURN  SQLCancel       (SQLHSTMT          hstmt);

SQL_EXTERN SQLRETURN  SQLCloseCursor  (SQLHSTMT          hstmt);

SQL_EXTERN SQLRETURN  SQLColAttributes (SQLHSTMT          hstmt,
                                 SQLSMALLINT       icol,
                                 SQLSMALLINT       fDescType,
                                 SQLCHAR          *rgbDesc,
                                 SQLINTEGER        cbDescMax,
                                 SQLINTEGER       *pcbDesc,
                                 SQLINTEGER       *pfDesc);

SQL_EXTERN SQLRETURN   SQLColumns    (SQLHSTMT       hstmt,
                              SQLCHAR        *szTableQualifier,
                              SQLSMALLINT    cbTableQualifier,
                              SQLCHAR        *szTableOwner,
                              SQLSMALLINT    cbTableOwner,
                              SQLCHAR        *szTableName,
                              SQLSMALLINT    cbTableName,
                              SQLCHAR        *szColumnName,
                              SQLSMALLINT    cbColumnName);

SQL_EXTERN SQLRETURN  SQLConnect    (SQLHDBC           hdbc,
                             SQLCHAR          *szDSN,
                             SQLSMALLINT       cbDSN,
                             SQLCHAR          *szUID,
                             SQLSMALLINT       cbUID,
                             SQLCHAR          *szAuthStr,
                             SQLSMALLINT       cbAuthStr);

SQL_EXTERN SQLRETURN  SQLCopyDesc   (SQLHDESC   sDesc,
                             SQLHDESC   tDesc);

SQL_EXTERN SQLRETURN  SQLDataSources (SQLHENV       henv,
                                 SQLSMALLINT   fDirection,
                                 SQLCHAR      *szDSN,
                                 SQLSMALLINT   cbDSNMax,
                                 SQLSMALLINT  *pcbDSN,
                                 SQLCHAR      *szDescription,
                                 SQLSMALLINT   cbDescriptionMax,
                                 SQLSMALLINT  *pcbDescription);
```

```
SQL_EXTERN SQLRETURN  SQLDescribeCol (SQLHSTMT          hstmt,
                                      SQLSMALLINT       icol,
                                      SQLCHAR          *szColName,
                                      SQLSMALLINT       cbColNameMax,
                                      SQLSMALLINT      *pcbColName,
                                      SQLSMALLINT      *pfSqlType,
                                      SQLINTEGER       *pcbColDef,
                                      SQLSMALLINT      *pibScale,
                                      SQLSMALLINT      *pfNullable);

SQL_EXTERN SQLRETURN  SQLDescribeParam (SQLHSTMT         hstmt,
                                      SQLSMALLINT       ipar,
                                      SQLSMALLINT      *pfSqlType,
                                      SQLINTEGER       *pcbColDef,
                                      SQLSMALLINT      *pibScale,
                                      SQLSMALLINT      *pfNullable);

SQL_EXTERN SQLRETURN  SQLDisconnect  (SQLHDBC             hdbc);

SQL_EXTERN SQLRETURN  SQLDriverConnect (SQLHDBC             hdbc,
                                        SQLPOINTER          hwnd,
                                        SQLCHAR      *szConnStrIn,
                                        SQLSMALLINT   cbConnStrin,
                                        SQLCHAR      *szConnStrOut,
                                        SQLSMALLINT  cbConnStrOutMax,
                                        SQLSMALLINT  *pcbConnStrOut,
                                        SQLSMALLINT fDriverCompletion);

SQL_EXTERN SQLRETURN  SQLEndTran     (SQLSMALLINT         htype,
                                      SQLHENV             henv,
                                      SQLSMALLINT         ctype);

SQL_EXTERN SQLRETURN  SQLError       (SQLHENV             henv,
                                      SQLHDBC             hdbc,
                                      SQLHSTMT            hstmt,
                                      SQLCHAR          *szSqlState,
                                      SQLINTEGER       *pfNativeError,
                                      SQLCHAR          *szErrorMsg,
                                      SQLSMALLINT       cbErrorMsgMax,
                                      SQLSMALLINT      *pcbErrorMsg);

SQL_EXTERN SQLRETURN  SQLExecDirect  (SQLHSTMT            hstmt,
                                      SQLCHAR          *szSqlStr,
                                      SQLINTEGER        cbSqlStr);

SQL_EXTERN SQLRETURN  SQLExecute     (SQLHSTMT            hstmt);

SQL_EXTERN SQLRETURN  SQLExtendedFetch (SQLHSTMT            hstmt,
                                        SQLSMALLINT         fOrient,
                                        SQLINTEGER          fOffset,
                                        SQLINTEGER         *pcrow,
                                        SQLSMALLINT        *rgfRowStatus);

SQL_EXTERN SQLRETURN  SQLFetch       (SQLHSTMT            hstmt);

SQL_EXTERN SQLRETURN  SQLFetchScroll (SQLHSTMT            hstmt,
                                      SQLSMALLINT         fOrient,
```

```
                                    SQLINTEGER          fOffset);

SQL_EXTERN SQLRETURN  SQLForeignKeys (SQLHSTMT          hstmt,
                                SQLCHAR          *szPkTableQualifier,
                                SQLSMALLINT       cbPkTableQualifier,
                                SQLCHAR          *szPkTableOwner,
                                SQLSMALLINT       cbPkTableOwner,
                                SQLCHAR          *szPkTableName,
                                SQLSMALLINT       cbPkTableName,
                                SQLCHAR          *szFkTableQualifier,
                                SQLSMALLINT       cbFkTableQualifier,
                                SQLCHAR          *szFkTableOwner,
                                SQLSMALLINT       cbFkTableOwner,
                                SQLCHAR          *szFkTableName,
                                SQLSMALLINT       cbFkTableName);

SQL_EXTERN SQLRETURN  SQLFreeConnect (SQLHDBC             hdbc);

SQL_EXTERN SQLRETURN  SQLFreeEnv     (SQLHENV             henv);

SQL_EXTERN SQLRETURN  SQLFreeStmt    (SQLHSTMT            hstmt,
                                SQLSMALLINT         fOption);

SQL_EXTERN SQLRETURN  SQLFreeHandle  (SQLSMALLINT         htype,
                                SQLINTEGER          hndl);

SQL_EXTERN SQLRETURN  SQLGetCol      (SQLHSTMT            hstmt,
                                SQLSMALLINT         icol,
                                SQLSMALLINT         itype,
                                SQLPOINTER          tval,
                                SQLINTEGER          blen,
                                SQLINTEGER         *olen);

SQL_EXTERN SQLRETURN  SQLGetConnectAttr (SQLHDBC             hdbc,
                                   SQLINTEGER          attr,
                                   SQLPOINTER          oval,
                                   SQLINTEGER          ilen,
                                   SQLINTEGER         *olen);

SQL_EXTERN SQLRETURN  SQLGetConnectOption (SQLHDBC             hdbc,
                                     SQLSMALLINT          iopt,
                                     SQLPOINTER          oval);

SQL_EXTERN SQLRETURN  SQLGetCursorName (SQLHSTMT            hstmt,
                                  SQLCHAR          *szCursor,
                                  SQLSMALLINT       cbCursorMax,
                                  SQLSMALLINT      *pcbCursor);

SQL_EXTERN SQLRETURN  SQLGetData     (SQLHSTMT          hstmt,
                                SQLSMALLINT       icol,
                                SQLSMALLINT       fCType,
                                SQLPOINTER        rgbValue,
                                SQLINTEGER        cbValueMax,
                                SQLINTEGER       *pcbValue);

SQL_EXTERN SQLRETURN SQLGetDescField (SQLHDESC            hdesc,
                                SQLSMALLINT          rcdNum,
                                SQLSMALLINT          fieldID,
```

```
                                SQLPOINTER           fValue,
                                SQLINTEGER           fLength,
                                SQLINTEGER          *stLength);

SQL_EXTERN SQLRETURN SQLGetDescRec   (SQLHDESC             hdesc,
                                SQLSMALLINT          rcdNum,
                                SQLCHAR             *fname,
                                SQLSMALLINT          bufLen,
                                SQLSMALLINT         *sLength,
                                SQLSMALLINT         *sType,
                                SQLSMALLINT         *sbType,
                                SQLINTEGER          *fLength,
                                SQLSMALLINT         *fprec,
                                SQLSMALLINT         *fscale,
                                SQLSMALLINT         *fnull);

SQL_EXTERN SQLRETURN SQLGetDiagField (SQLSMALLINT         hType,
                                SQLINTEGER          hndl,
                                SQLSMALLINT         rcdNum,
                                SQLSMALLINT         diagID,
                                SQLPOINTER          dValue,
                                SQLSMALLINT         bLength,
                                SQLSMALLINT        *sLength);

SQL_EXTERN SQLRETURN SQLGetDiagRec   (SQLSMALLINT         hType,
                                SQLINTEGER          hndl,
                                SQLSMALLINT         rcdNum,
                                SQLCHAR            *SQLstate,
                                SQLINTEGER         *SQLcode,
                                SQLCHAR            *msgText,
                                SQLSMALLINT         bLength,
                                SQLSMALLINT        *SLength);

SQL_EXTERN SQLRETURN SQLGetEnvAttr  (SQLHENV    hEnv,
                               SQLINTEGER fAttribute,
                               SQLPOINTER pParam,
                               SQLINTEGER cbParamMax,
                               SQLINTEGER * pcbParam);

SQL_EXTERN SQLRETURN  SQLGetFunctions  (SQLHDBC        hdbc,
                                  SQLSMALLINT    fFunction,
                                  SQLSMALLINT    *pfExists);

SQL_EXTERN SQLRETURN  SQLGetInfo       (SQLHDBC        hdbc,
                                  SQLSMALLINT    fInfoType,
                                  SQLPOINTER     rgbInfoValue,
                                  SQLSMALLINT    cbInfoValueMax,
                                  SQLSMALLINT    *pcbInfoValue);

SQL_EXTERN SQLRETURN  SQLGetLength (SQLHSTMT       hstmt,
                                  SQLSMALLINT    locType,
                                  SQLINTEGER     locator,
                                  SQLINTEGER    *sLength,
                                  SQLINTEGER    *ind);

SQL_EXTERN SQLRETURN  SQLGetPosition (SQLHSTMT       hstmt,
                                  SQLSMALLINT    locType,
                                  SQLINTEGER     srceLocator,
```

```
                                    SQLINTEGER     srchLocator,
                                    SQLCHAR       *srchLiteral,
                                    SQLINTEGER     srchLiteralLen,
                                    SQLINTEGER     fPosition,
                                    SQLINTEGER    *located,
                                    SQLINTEGER    *ind);

SQL_EXTERN SQLRETURN  SQLGetStmtAttr   (SQLHSTMT       hstmt,
                                    SQLINTEGER     fAttr,
                                    SQLPOINTER     pvParam,
                                    SQLINTEGER     bLength,
                                    SQLINTEGER    *SLength);

SQL_EXTERN SQLRETURN  SQLGetStmtOption (SQLHSTMT       hstmt,
                                    SQLSMALLINT    fOption,
                                    SQLPOINTER     pvParam);

SQL_EXTERN SQLRETURN  SQLGetSubString (SQLHSTMT       hstmt,
                                    SQLSMALLINT    locType,
                                    SQLINTEGER     srceLocator,
                                    SQLINTEGER     fPosition,
                                    SQLINTEGER     length,
                                    SQLSMALLINT    tType,
                                    SQLPOINTER     rgbValue,
                                    SQLINTEGER     cbValueMax,
                                    SQLINTEGER    *StringLength,
                                    SQLINTEGER    *ind);

SQL_EXTERN SQLRETURN  SQLGetTypeInfo   (SQLHSTMT       hstmt,
                                    SQLSMALLINT    fSqlType);

SQL_EXTERN SQLRETURN  SQLLanguages   (SQLHSTMT          hstmt);

SQL_EXTERN SQLRETURN  SQLMoreResults (SQLHSTMT          hstmt);

SQL_EXTERN SQLRETURN  SQLNativeSql   (SQLHDBC           hdbc,
                                     SQLCHAR          *szSqlStrIn,
                                     SQLINTEGER        cbSqlStrIn,
                                     SQLCHAR          *szSqlStr,
                                     SQLINTEGER        cbSqlStrMax,
                                     SQLINTEGER       *pcbSqlStr);

SQL_EXTERN SQLRETURN  SQLNumParams (SQLHSTMT            hstmt,
                                    SQLSMALLINT      *pcpar);

SQL_EXTERN SQLRETURN  SQLNumResultCols (SQLHSTMT        hstmt,
                                    SQLSMALLINT      *pccol);

SQL_EXTERN SQLRETURN  SQLParamData   (SQLHSTMT          hstmt,
                                  SQLPOINTER          *Value);

SQL_EXTERN SQLRETURN  SQLParamOptions (SQLHSTMT         hstmt,
                                  SQLINTEGER          crow,
                                  SQLINTEGER          *pirow);

SQL_EXTERN SQLRETURN  SQLPrepare     (SQLHSTMT          hstmt,
                                  SQLCHAR           *szSqlStr,
                                  SQLSMALLINT        cbSqlStr);
```

```
SQL_EXTERN SQLRETURN  SQLPrimaryKeys (SQLHSTMT          hstmt,
                                      SQLCHAR          *szTableQualifier,
                                      SQLSMALLINT       cbTableQualifier,
                                      SQLCHAR          *szTableOwner,
                                      SQLSMALLINT       cbTableOwner,
                                      SQLCHAR          *szTableName,
                                      SQLSMALLINT       cbTableName);

SQL_EXTERN SQLRETURN  SQLProcedureColumns (SQLHSTMT          hstmt,
                                      SQLCHAR          *szProcQualifier,
                                      SQLSMALLINT       cbProcQualifier,
                                      SQLCHAR          *szProcOwner,
                                      SQLSMALLINT       cbProcOwner,
                                      SQLCHAR          *szProcName,
                                      SQLSMALLINT       cbProcName,
                                      SQLCHAR          *szColumnName,
                                      SQLSMALLINT       cbColumnName);

SQL_EXTERN SQLRETURN  SQLProcedures (SQLHSTMT          hstmt,
                                      SQLCHAR          *szProcQualifier,
                                      SQLSMALLINT       cbProcQualifier,
                                      SQLCHAR          *szProcOwner,
                                      SQLSMALLINT       cbProcOwner,
                                      SQLCHAR          *szProcName,
                                      SQLSMALLINT       cbProcName);

SQL_EXTERN SQLRETURN  SQLPutData     (SQLHSTMT            hstmt,
                                      SQLPOINTER          Data,
                                      SQLINTEGER          SLen);

SQL_EXTERN SQLRETURN  SQLReleaseEnv     (SQLHENV          henv);

SQL_EXTERN SQLRETURN  SQLRowCount    (SQLHSTMT          hstmt,
                                      SQLINTEGER        *pcrow);

SQL_EXTERN SQLRETURN  SQLSetConnectAttr   (SQLHDBC       hdbc,
                                           SQLINTEGER    attrib,
                                           SQLPOINTER    vParam,
                                           SQLINTEGER    inlen);

SQL_EXTERN SQLRETURN  SQLSetConnectOption (SQLHDBC       hdbc,
                                           SQLSMALLINT   fOption,
                                           SQLPOINTER    vParam);

SQL_EXTERN SQLRETURN  SQLSetCursorName (SQLHSTMT            hstmt,
                                        SQLCHAR          *szCursor,
                                        SQLSMALLINT       cbCursor);

SQL_EXTERN SQLRETURN  SQLSetDescField  (SQLHDESC         hdesc,
                                        SQLSMALLINT       rcdNum,
                                        SQLSMALLINT       fID,
                                        SQLPOINTER        Value,
                                        SQLINTEGER        buffLen);

SQL_EXTERN SQLRETURN  SQLSetDescRec    (SQLHDESC         hdesc,
                                        SQLSMALLINT       rcdNum,
                                        SQLSMALLINT       Type,
```

```
                                    SQLSMALLINT       subType,
                                    SQLINTEGER        fLength,
                                    SQLSMALLINT       fPrec,
                                    SQLSMALLINT       fScale,
                                    SQLPOINTER        Value,
                                    SQLINTEGER       *sLength,
                                    SQLSMALLINT      *indic);

SQL_EXTERN SQLRETURN  SQLSetEnvAttr( SQLHENV hEnv,
                                  SQLINTEGER fAttribute,
                                  SQLPOINTER pParam,
                                  SQLINTEGER cbParam);

SQL_EXTERN SQLRETURN  SQLSetParam      (SQLHSTMT             hstmt,
                                    SQLSMALLINT       ipar,
                                    SQLSMALLINT       fCType,
                                    SQLSMALLINT       fSqlType,
                                    SQLINTEGER        cbColDef,
                                    SQLSMALLINT       ibScale,
                                    SQLPOINTER        rgbValue,
                                    SQLINTEGER       *pcbValue);

SQL_EXTERN SQLRETURN  SQLSetStmtAttr (SQLHSTMT       hstmt,
                                   SQLINTEGER     fAttr,
                                   SQLPOINTER     pParam,
                                   SQLINTEGER     vParam);

SQL_EXTERN SQLRETURN  SQLSetStmtOption (SQLHSTMT         hstmt,
                                    SQLSMALLINT    fOption,
                                    SQLPOINTER     vParam);

SQL_EXTERN SQLRETURN SQLSpecialColumns (SQLHSTMT       hstmt,
                                    SQLSMALLINT    fColType,
                                    SQLCHAR        *szTableQual,
                                    SQLSMALLINT    cbTableQual,
                                    SQLCHAR        *szTableOwner,
                                    SQLSMALLINT    cbTableOwner,
                                    SQLCHAR        *szTableName,
                                    SQLSMALLINT    cbTableName,
                                    SQLSMALLINT    fScope,
                                    SQLSMALLINT    fNullable);

SQL_EXTERN SQLRETURN  SQLStatistics    (SQLHSTMT       hstmt,
                                    SQLCHAR        *szTableQualifier,
                                    SQLSMALLINT    cbTableQualifier,
                                    SQLCHAR        *szTableOwner,
                                    SQLSMALLINT    cbTableOwner,
                                    SQLCHAR        *szTableName,
                                    SQLSMALLINT    cbTableName,
                                    SQLSMALLINT    fUnique,
                                    SQLSMALLINT    fres);

SQL_EXTERN SQLRETURN SQLTables        (SQLHSTMT       hstmt,
                                   SQLCHAR        *szTableQualifier,
                                   SQLSMALLINT    cbTableQualifier,
                                   SQLCHAR        *szTableOwner,
                                   SQLSMALLINT    cbTableOwner,
                                   SQLCHAR        *szTableName,
```

```
                                    SQLSMALLINT    cbTableName,
                                    SQLCHAR        *szTableType,
                                    SQLSMALLINT    cbTableType);

SQL_EXTERN SQLRETURN  SQLTransact      (SQLHENV              henv,
                                     SQLHDBC             hdbc,
                                     SQLSMALLINT       fType);


#define FAR
#define  SQL_SQLSTATE_SIZE         5   /* size of SQLSTATE, not including
                                          null terminating byte              */
#define  SQL_MAX_DSN_LENGTH       18   /* maximum data source name size      */
#define  SQL_MAX_ID_LENGTH        18   /* maximum identifier name size,
                                          e.g. cursor names                  */

#define SQL_MAX_STMT_SIZ       32767   /* Maximum statement size             */

#define SQL_MAXRECL            32766   /* Maximum record length              */

#define SQL_SMALL_LENGTH       2       /* Size of a SMALLINT                 */
#define SQL_MAXSMALLVAL        32767   /* Maximum value of a SMALLINT        */
#define SQL_MINSMALLVAL (-(SQL_MAXSMALLVAL)-1) /* Minimum value of a SMALLINT */
#define SQL_INT_LENGTH         4       /* Size of an INTEGER                 */
#define SQL_MAXINTVAL          2147483647 /* Maximum value of an INTEGER     */
#define SQL_MININTVAL (-(SQL_MAXINTVAL)-1)/* Minimum value of an INTEGER     */
#define SQL_FLOAT_LENGTH       8       /* Size of a FLOAT                    */
#define SQL_DEFDEC_PRECISION   5       /* Default precision for DECIMAL      */
#define SQL_DEFDEC_SCALE       0       /* Default scale for DECIMAL          */
#define SQL_MAXDECIMAL         31      /* Maximum scale/prec. for DECIMAL    */
#define SQL_DEFCHAR            1       /* Default length for a CHAR          */
#define SQL_DEFWCHAR           1       /* Default length for a wchar_t       */
#define SQL_MAXCHAR            32766   /* Maximum length of a CHAR           */
#define SQL_MAXLSTR            255     /* Maximum length of an LSTRING       */
#define SQL_MAXVCHAR           32740   /* Maximum length of a         */
                                       /* VARCHAR                            */
#define SQL_MAXVGRAPH          16370   /* Maximum length of a VARGRAPHIC     */
#define SQL_MAXBLOB            15728640 /* Max. length of a BLOB host var   */
#define SQL_MAXCLOB            15728640 /* Max. length of a CLOB host var   */
#define SQL_MAXDBCLOB          7864320 /* Max. length of an DBCLOB host    */
                                       /* var                                */
#define SQL_LONGMAX            32740   /* Maximum length of a LONG VARCHAR   */
#define SQL_LONGGRMAX          16370   /* Max. length of a LONG VARGRAPHIC   */
#define SQL_LVCHAROH           26      /* Overhead for LONG VARCHAR in       */
                                       /* record                             */
#define SQL_LOBCHAROH          312     /* Overhead for LOB in record         */
#define SQL_BLOB_MAXLEN        15728640 /* BLOB maximum length, in bytes    */
#define SQL_CLOB_MAXLEN        15728640 /* CLOB maximum length, in chars    */
#define SQL_DBCLOB_MAXLEN      7864320 /* maxlen for dbcs lobs             */
#define SQL_TIME_LENGTH        3       /* Size of a TIME field               */
#define SQL_TIME_STRLEN        8       /* Size of a TIME field output        */
#define SQL_TIME_MINSTRLEN     5       /* Size of a non-USA TIME field       */
                                       /* output without seconds             */
#define SQL_DATE_LENGTH        4       /* Size of a DATE field               */
#define SQL_DATE_STRLEN        10      /* Size of a DATE field output        */
#define SQL_STAMP_LENGTH       10      /* Size of a TIMESTAMP field          */
#define SQL_STAMP_STRLEN       26      /* Size of a TIMESTAMP field output   */
#define SQL_STAMP_MINSTRLEN    19      /* Size of a TIMESTAMP field output   */
```

```
                                    /* without microseconds               */
#define SQL_BOOLEAN_LENGTH     1    /* Size of a BOOLEAN field            */
#define SQL_IND_LENGTH         2    /* Size of an indicator value         */

#define SQL_MAX_PNAME_LENGTH   254  /* Max size of Stored Proc Name       */
#define SQL_LG_IDENT           18   /* Maximum length of Long Identifer   */
#define SQL_SH_IDENT           8    /* Maximum length of Short Identifer  */
#define SQL_MN_IDENT           1    /* Minimum length of Identifiers      */
#define SQL_MAX_VAR_NAME       30   /* Max size of Host Variable Name     */
#define SQL_KILO_VALUE         1024 /* # of bytes in a kilobyte           */
#define SQL_MEGA_VALUE         1048576 /* # of bytes in a megabyte        */
#define SQL_GIGA_VALUE         1073741824 /* # of bytes in a gigabyte     */

/* SQL extended data types (negative means unsupported) */
#define  SQL_TINYINT            -6
#define  SQL_BIT                -7

/* C data type to SQL data type mapping */
#define  SQL_C_CHAR       SQL_CHAR      /* CHAR, VARCHAR, DECIMAL, NUMERIC */
#define  SQL_C_LONG       SQL_INTEGER   /* INTEGER                         */
#define  SQL_C_SHORT      SQL_SMALLINT  /* SMALLINT                        */
#define  SQL_C_FLOAT      SQL_REAL      /* REAL                            */
#define  SQL_C_DOUBLE     SQL_DOUBLE    /* FLOAT, DOUBLE                   */
#define  SQL_C_DATE       SQL_DATE      /* DATE                            */
#define  SQL_C_TIME       SQL_TIME      /* TIME                            */
#define  SQL_C_TIMESTAMP  SQL_TIMESTAMP /* TIMESTAMP                       */
#define  SQL_C_BINARY     SQL_BINARY    /* BINARY, VARBINARY               */
#define  SQL_C_BIT        SQL_BIT
#define  SQL_C_TINYINT    SQL_TINYINT
#define  SQL_C_BIGINT     SQL_BIGINT
#define  SQL_C_DBCHAR     SQL_DBCLOB
#define  SQL_C_WCHAR      SQL_WCHAR     /* UNICODE                         */
#define  SQL_C_DATETIME   SQL_DATETIME  /* DATETIME                        */
#define  SQL_C_BLOB           SQL_BLOB
#define  SQL_C_CLOB           SQL_CLOB
#define  SQL_C_DBCLOB         SQL_DBCLOB
#define  SQL_C_BLOB_LOCATOR   SQL_BLOB_LOCATOR
#define  SQL_C_CLOB_LOCATOR   SQL_CLOB_LOCATOR
#define  SQL_C_DBCLOB_LOCATOR SQL_DBCLOB_LOCATOR



#define SQL_WARN_VAL_TRUNC                 "01004"


#endif /* SQL_H_SQLCLI */
```

부록 C. DB2 UDB CLI 어플리케이션 코드 리스팅 예

이 부록은 책에서 전반적으로 사용된 예에 대한 전체 코드 리스트입니다.

이 예들은 모든 조건에서 완벽하게 테스트한 것이 아닙니다.

다음을 참조하십시오.

- 『예: 내장 SQL 및 그에 해당되는 DB2 UDB CLI 함수 호출』
- 300 페이지의 『예: 대화식 SQL 및 그에 해당되는 DB2 UDB CLI 함수 호출』

예: 내장 SQL 및 그에 해당되는 DB2 UDB CLI 함수 호출

주석 안에 내장 명령문과 그와 동등한 DB2 UDB CLI 함수 호출을 나타내고 있습니다.

코드 예와 관련된 정보는 viii 페이지의 『코드 면책사항 관련 정보』를 참조하십시오.

```
/************************************************************************
** file = embedded.c
**
** Example of executing an SQL statement using CLI.
** The equivalent embedded SQL statements are shown in comments.
**
** Functions used:
**
**        SQLAllocConnect      SQLFreeConnect
**        SQLAllocEnv          SQLFreeEnv
**        SQLAllocStmt         SQLFreeStmt
**        SQLConnect           SQLDisconnect
**
**        SQLBindCol           SQLFetch
**        SQLSetParam          SQLTransact
**        SQLError             SQLExecDirect
**
************************************************************************/
#include <stdio.h>
#include <string.h>
#include "sqlcli.h"

#ifndef NULL
#define NULL   0
#endif

int print_err (SQLHDBC    hdbc,
               SQLHSTMT   hstmt);

int main ()
{
    SQLHENV        henv;
    SQLHDBC        hdbc;
    SQLHSTMT       hstmt;

    SQLCHAR     server[] = "sample";
```

```
SQLCHAR     uid[30];
SQLCHAR     pwd[30];

SQLINTEGER     id;
SQLCHAR        name[51];
SQLINTEGER     namelen, intlen;
SQLSMALLINT    scale;

scale = 0;


/* EXEC SQL CONNECT TO :server USER :uid USING :authentication_string; */
SQLAllocEnv (&henv);                /* allocate an environment handle */

SQLAllocConnect (henv, &hdbc);      /* allocate a connection handle   */


/* Connect to database indicated by "server" variable with            */
/*    authorization-name given in "uid", authentication-string given  */
/*    in "pwd". Note server, uid, and pwd contain null-terminated     */
/*    strings, as indicated by the 3 input lengths set to SQL_NTS     */
if (SQLConnect (hdbc, server, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS)
               != SQL_SUCCESS)
    return (print_err (hdbc, SQL_NULL_HSTMT));

SQLAllocStmt (hdbc, &hstmt);        /* allocate a statement handle    */


/* EXEC SQL CREATE TABLE NAMEID (ID integer, NAME varchar(50));       */
{
   SQLCHAR create[] = "CREATE TABLE NAMEID (ID integer, NAME varchar(50))";

/* execute the sql statement                                          */
   if (SQLExecDirect (hstmt, create, SQL_NTS) != SQL_SUCCESS)
       return (print_err (hdbc, hstmt));
}


/* EXEC SQL COMMIT WORK;                                              */
SQLTransact (henv, hdbc, SQL_COMMIT);          /* commit create table */


/* EXEC SQL INSERT INTO NAMEID VALUES ( :id, :name         */
{
   SQLCHAR insert[] = "INSERT INTO NAMEID VALUES (?, ?)";


/* show the use of SQLPrepare/SQLExecute method                       */
/* prepare the insert                                                 */

   if (SQLPrepare (hstmt, insert, SQL_NTS) != SQL_SUCCESS)
      return (print_err (hdbc, hstmt));

/* Set up the first input parameter "id"                              */
   intlen = sizeof (SQLINTEGER);
   SQLSetParam (hstmt, 1,
                SQL_C_LONG, SQL_INTEGER,
               (SQLINTEGER) sizeof (SQLINTEGER),
```

```
               scale, (SQLPOINTER) &id,
              (SQLINTEGER *) &intlen);

   namelen = SQL_NTS;
/* Set up the second input parameter "name"                          */
   SQLSetParam (hstmt, 2,
               SQL_C_CHAR, SQL_VARCHAR,
               50,
               scale, (SQLPOINTER) name,
               (SQLINTEGER *) &namelen);

/* now assign parameter values and execute the insert               */
   id=500;
   strcpy (name, "Babbage");

   if (SQLExecute (hstmt) != SQL_SUCCESS)
      return (print_err (hdbc, hstmt));
}


/* EXEC SQL COMMIT WORK;                                            */
SQLTransact (henv, hdbc, SQL_COMMIT);        /* commit inserts      */


/* EXEC SQL DECLARE c1 CURSOR FOR SELECT ID, NAME FROM NAMEID;      */
/* EXEC SQL OPEN c1;                                                */
/* The application doesn't specify "declare c1 cursor for"          */
{
   SQLCHAR select[] = "select ID, NAME from NAMEID";
   if (SQLExecDirect (hstmt, select, SQL_NTS) != SQL_SUCCESS)
      return (print_err (hdbc, hstmt));
}


/* EXEC SQL FETCH c1 INTO :id, :name;                    */
/* Binding first column to output variable "id"                     */
SQLBindCol (hstmt, 1,
            SQL_C_LONG, (SQLPOINTER) &id,
           (SQLINTEGER) sizeof (SQLINTEGER),
           (SQLINTEGER *) &intlen);

/* Binding second column to output variable "name"                  */
SQLBindCol (hstmt, 2,
            SQL_C_CHAR, (SQLPOINTER) name,
           (SQLINTEGER) sizeof (name),
            &namelen);

SQLFetch (hstmt);                     /* now execute the fetch      */
printf("Result of Select: id = %ld name = %s\n", id, name);

/* finally, we should commit, discard hstmt, disconnect             */
/* EXEC SQL COMMIT WORK;                                            */
SQLTransact (henv, hdbc, SQL_COMMIT);  /* commit the transaction    */

/* EXEC SQL CLOSE c1;                                               */
SQLFreeStmt (hstmt, SQL_DROP);         /* free the statement handle */

/* EXEC SQL DISCONNECT;                                             */
```

```
    SQLDisconnect (hdbc);                  /* disconnect from the database */

    SQLFreeConnect (hdbc);                 /* free the connection handle   */
    SQLFreeEnv (henv);                     /* free the environment handle  */

    return (0);
}

int print_err (SQLHDBC    hdbc,
               SQLHSTMT   hstmt)
{
SQLCHAR        buffer[SQL_MAX_MESSAGE_LENGTH + 1];
SQLCHAR        sqlstate[SQL_SQLSTATE_SIZE + 1];
SQLINTEGER     sqlcode;
SQLSMALLINT    length;


        while ( SQLError(SQL_NULL_HENV, hdbc, hstmt,
                 sqlstate,
                 &sqlcode,
                 buffer,
                 SQL_MAX_MESSAGE_LENGTH + 1,
                 &length) == SQL_SUCCESS )
        {
                printf("SQLSTATE: %s Native Error Code: %ld\n",
                       sqlstate, sqlcode);
                printf("%s \n", buffer);
                printf("---------------------------- \n");
         };

        return(SQL_ERROR);

}
```

예: 대화식 SQL 및 그에 해당되는 DB2 UDB CLI 함수 호출

이 예는 대화식 SQL문의 실행을 나타내고 있으며 9 페이지의 제 2 장『DB2 UDB CLI 어플리케이션 작성』에서 설명하고 있는 흐름을 따릅니다.

코드 예와 관련된 정보는 viii 페이지의『코드 면책사항 관련 정보』를 참조하십시오.

```
/*****************************************************************************
** file = typical.c
**
** Example of executing interactive SQL statements, displaying result sets
** and simple transaction management.
**
** Functions used:
**
**        SQLAllocConnect      SQLFreeConnect
**        SQLAllocEnv          SQLFreeEnv
**        SQLAllocStmt         SQLFreeStmt
**        SQLConnect           SQLDisconnect
**
**        SQLBindCol           SQLFetch
**        SQLDescribeCol       SQLNumResultCols
**        SQLError             SQLRowCount
```

```
**        SQLExecDirect        SQLTransact
**
******************************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "sqlcli.h"

#define MAX_STMT_LEN 255
#define  MAXCOLS   100

#define  max(a,b) (a > b ? a : b)

int initialize(SQLHENV *henv,
               SQLHDBC *hdbc);

int process_stmt(SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLCHAR        *sqlstr);

int terminate(SQLHENV henv,
              SQLHDBC hdbc);

int print_error(SQLHENV    henv,
                SQLHDBC    hdbc,
                SQLHSTMT   hstmt);

int check_error(SQLHENV    henv,
                SQLHDBC    hdbc,
                SQLHSTMT   hstmt,
                SQLRETURN  frc);

void display_results(SQLHSTMT hstmt,
                     SQLSMALLINT nresultcols);

/*****************************************************************
** main
** - initialize
** - start a transaction
**   - get statement
**   - another statement?
** - COMMIT or ROLLBACK
** - another transaction?
** - terminate
*****************************************************************/
int main()
{
    SQLHENV     henv;
    SQLHDBC     hdbc;
    SQLCHAR     sqlstmt[MAX_STMT_LEN + 1]="";
    SQLCHAR     sqltrans[sizeof("ROLLBACK")];
    SQLRETURN   rc;

    rc = initialize(&henv, &hdbc);
    if (rc == SQL_ERROR) return(terminate(henv, hdbc));

    printf("Enter an SQL statement to start a transaction(or 'q' to Quit):\n");
```

```
        gets(sqlstmt);

    while (sqlstmt[0] !='q')
    {
        while (sqlstmt[0] != 'q')
        {   rc = process_stmt(henv, hdbc, sqlstmt);
            if (rc == SQL_ERROR) return(SQL_ERROR);
            printf("Enter an SQL statement(or 'q' to Quit):\n");
        gets(sqlstmt);
        }

        printf("Enter 'c' to COMMIT or 'r' to ROLLBACK the transaction\n");
        fgets(sqltrans, sizeof("ROLLBACK"), stdin);

        if (sqltrans[0] == 'c')
        {
          rc = SQLTransact (henv, hdbc, SQL_COMMIT);
          if (rc == SQL_SUCCESS)
            printf ("Transaction commit was successful\n");
            else
            check_error (henv, hdbc, SQL_NULL_HSTMT, rc);
        }

        if (sqltrans[0] == 'r')
        {
          rc = SQLTransact (henv, hdbc, SQL_ROLLBACK);
          if (rc == SQL_SUCCESS)
            printf ("Transaction roll back was successful\n");
            else
            check_error (henv, hdbc, SQL_NULL_HSTMT, rc);
        }

        printf("Enter an SQL statement to start a transaction or 'q' to quit\n");
        gets(sqlstmt);
    }

        terminate(henv, hdbc);

            return (SQL_SUCCESS);
}/* end main */

/*****************************************************************
** process_stmt
** - allocates a statement handle
** - executes the statement
** - determines the type of statement
**   - if there are no result columns, therefore non-select statement
**      - if rowcount > 0, assume statement was UPDATE, INSERT, DELETE
**   else
**      - assume a DDL, or Grant/Revoke statement
**   else
**      - must be a select statement.
**      - display results
** - frees the statement handle
*****************************************************************/

int process_stmt (SQLHENV    henv,
                  SQLHDBC    hdbc,
```

```
                  SQLCHAR    *sqlstr)
{
SQLHSTMT        hstmt;
SQLSMALLINT     nresultcols;
SQLINTEGER      rowcount;
SQLRETURN   rc;


    SQLAllocStmt (hdbc, &hstmt);       /* allocate a statement handle */

    /* execute the SQL statement in "sqlstr"    */

    rc = SQLExecDirect (hstmt, sqlstr, SQL_NTS);
        if (rc != SQL_SUCCESS)
        if (rc == SQL_NO_DATA_FOUND) {
            printf("\nStatement executed without error, however,\n");
            printf("no data was found or modified\n");
            return (SQL_SUCCESS);
        }
            else
      check_error (henv, hdbc, hstmt, rc);

    SQLRowCount (hstmt, &rowcount);
    rc = SQLNumResultCols (hstmt, &nresultcols);
        if (rc != SQL_SUCCESS)
      check_error (henv, hdbc, hstmt, rc);

    /* determine statement type */
    if (nresultcols == 0) /* statement is not a select statement */
    {
        if (rowcount > 0 ) /* assume statement is UPDATE, INSERT, DELETE */
        {
            printf ("Statement executed, %ld rows affected\n", rowcount);
        }
      else  /* assume statement is GRANT, REVOKE or a DLL statement */
        {
              printf ("Statement completed successful\n");
        }
    }
    else /* display the result set */
    {
        display_results(hstmt, nresultcols);
    } /* end determine statement type */

    SQLFreeStmt (hstmt, SQL_DROP );       /* free statement handle */

    return (0);
}/* end process_stmt */

/*****************************************************************
** initialize
**  - allocate environment handle
**  - allocate connection handle
**  - prompt for server, user id, & password
**  - connect to server
*****************************************************************/

int initialize(SQLHENV *henv,
```

```
              SQLHDBC *hdbc)
{
SQLCHAR     server[18],
            uid[10],
            pwd[10];
SQLRETURN   rc;

    rc = SQLAllocEnv (henv);         /* allocate an environment handle   */
        if (rc != SQL_SUCCESS )
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);

    rc = SQLAllocConnect (*henv, hdbc);  /* allocate a connection handle     */
        if (rc != SQL_SUCCESS )
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);

    printf("Enter Server Name:\n");
    gets(server);
    printf("Enter User Name:\n");
    gets(uid);
    printf("Enter Password Name:\n");
    gets(pwd);

    if (uid[0] == '\0')
    {   rc = SQLConnect (*hdbc, server, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
    }
            else
    {   rc = SQLConnect (*hdbc, server, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
    }
}/* end initialize */

/*****************************************************************
** terminate
**  - disconnect
**  - free connection handle
**  - free environment handle
*****************************************************************/
int terminate(SQLHENV henv,
              SQLHDBC hdbc)
{
SQLRETURN   rc;

    rc = SQLDisconnect (hdbc);          /* disconnect from database      */
        if (rc != SQL_SUCCESS )
        print_error (henv, hdbc, SQL_NULL_HSTMT);
    rc = SQLFreeConnect (hdbc);         /* free connection handle        */
        if (rc != SQL_SUCCESS )
        print_error (henv, hdbc, SQL_NULL_HSTMT);
    rc = SQLFreeEnv (henv);             /* free environment handle       */
        if (rc != SQL_SUCCESS )
        print_error (henv, SQL_NULL_HDBC, SQL_NULL_HSTMT);

}/* end terminate */

/*****************************************************************
```

```
** display_results - displays the selected character fields
**
**  - for each column
**      - get column name
**      - bind column
**  - display column headings
**  - fetch each row
**      - if value truncated, build error message
**      - if column null, set value to "NULL"
**      - display row
**      - print truncation message
**  - free local storage
**
*********************************************************************/
void display_results(SQLHSTMT hstmt,
                SQLSMALLINT nresultcols)
{
SQLCHAR         colname[32];
SQLSMALLINT     coltype[MAXCOLS];
SQLSMALLINT     colnamelen;
SQLSMALLINT     nullable;
SQLINTEGER      collen[MAXCOLS];
SQLSMALLINT     scale;
SQLINTEGER      outlen[MAXCOLS];
SQLCHAR *       data[MAXCOLS];
SQLCHAR         errmsg[256];
SQLRETURN   rc;
SQLINTEGER      i;
SQLINTEGER      displaysize;

        for (i = 0; i < nresultcols; i++)
    {
        SQLDescribeCol (hstmt, i+1, colname, sizeof (colname),
        &colnamelen, &coltype[i], &collen[i], &scale, &nullable);

        /* get display length for column */
        SQLColAttributes (hstmt, i+1, SQL_DESC_PRECISION, NULL, 0     ,
            NULL, &displaysize);

        /* set column length to max of display length, and column name
           length.  Plus one byte for null terminator       */
        collen[i] = max(displaysize, collen[i]);
        collen[i] = max(collen[i], strlen((char *) colname) ) + 1;

        printf ("%-*.*s", collen[i], collen[i], colname);

        /* allocate memory to bind column                              */
        data[i] = (SQLCHAR *) malloc (collen[i]);

        /* bind columns to program vars, converting all types to CHAR */
        SQLBindCol (hstmt, i+1, SQL_C_CHAR, data[i], collen[i], &outlen[i]);
    }
    printf("\n");

    /* display result rows                                             */
    while ((rc = SQLFetch (hstmt)) != SQL_NO_DATA_FOUND)
    {
        errmsg[0] = '\0';
```

```
        for (i = 0; i < nresultcols; i++)
        {
            /* Build a truncation message for any columns truncated */
            if (outlen[i] >= collen[i])
            {    sprintf ((char *) errmsg + strlen ((char *) errmsg),
                         "%d chars truncated, col %d\n",
                          outlen[i]-collen[i]+1, i+1);
            }
            if (outlen[i] == SQL_NULL_DATA)
                printf ("%-*.*s", collen[i], collen[i], "NULL");
            else
                printf ("%-*.*s", collen[i], collen[i], data[i]);
        } /* for all columns in this row  */

        printf ("\n%s", errmsg);  /* print any truncation messages    */
    } /* while rows to fetch */

    /* free data buffers                                              */
        for (i = 0; i < nresultcols; i++)
    {
        free (data[i]);
    }

}/* end display_results

/*********************************************************************
** SUPPORT FUNCTIONS
**  - print_error   - call SQLError(), display SQLSTATE and message
**  - check_error     - call print_error
**                    - check severity of Return Code
**                    - rollback & exit if error, continue if warning
*********************************************************************/

/*********************************************************************/
int print_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt)
{
SQLCHAR     buffer[SQL_MAX_MESSAGE_LENGTH + 1];
SQLCHAR     sqlstate[SQL_SQLSTATE_SIZE + 1];
SQLINTEGER  sqlcode;
SQLSMALLINT length;

    while ( SQLError(henv, hdbc, hstmt, sqlstate, &sqlcode, buffer,
                     SQL_MAX_MESSAGE_LENGTH + 1, &length) == SQL_SUCCESS )
    {
        printf("\n **** ERROR *****\n");
        printf("         SQLSTATE: %s\n", sqlstate);
        printf("Native Error Code: %ld\n", sqlcode);
        printf("%s \n", buffer);
    };
    return;
}

/*********************************************************************/
int check_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt,
```

```
                SQLRETURN  frc)
{
SQLRETURN   rc;

    print_error(henv, hdbc, hstmt);

    switch (frc){
    case SQL_SUCCESS : break;
    case SQL_ERROR :
    case SQL_INVALID_HANDLE:
        printf("\n ** FATAL ERROR, Attempting to rollback transaction **\n");
        rc = SQLTransact(henv, hdbc, SQL_ROLLBACK);
        if (rc != SQL_SUCCESS)
            printf("Rollback Failed, Exiting application\n");
            else
            printf("Rollback Successful, Exiting application\n");
        terminate(henv, hdbc);
        exit(frc);
        break;
    case SQL_SUCCESS_WITH_INFO :
        printf("\n ** Warning Message, application continuing\n");
        break;
    case SQL_NO_DATA_FOUND :
        printf("\n ** No Data Found ** \n");
        break;
    default :
        printf("\n ** Invalid Return Code ** \n");
        printf(" ** Attempting to rollback transaction **\n");
        SQLTransact(henv, hdbc, SQL_ROLLBACK);
        terminate(henv, hdbc);
        exit(frc);
        break;
    }
    return(SQL_SUCCESS);

}
```

부록 D. 서버 모드로 DB2 UDB CLI 실행

『DB2 UDB CLI를 SQL 서버 모드로 실행하는 이유』를 참조하십시오.

DB2 UDB CLI를 SQL 서버 모드로 실행하는 이유

SQL 서버 모드로 실행하는 이유는 여러 어플리케이션이 데이터베이스 서버 역할을 해야 하기 때문입니다. 즉, 한 작업이 여러 사용자를 대신하여 SQL 요구를 처리합니다. SQL 서버 모드를 사용하지 않으면 어플리케이션은 다음 세 가지 제한사항 중 하나 또는 그 이상에 직면할 수 있습니다.

1. 하나의 작업에는 각 활성화 그룹마다 확약 트랜잭션이 단 하나 있습니다.
2. 하나의 작업은 RDB에 단 한 번만 연결될 수 있습니다.
3. 모든 SQL문은 연결 상태에서 전달된 사용자 ID에 상관없이 해당 작업의 사용자 프로파일에서 실행됩니다.

SQL 서버 모드는 모든 SQL문을 각각의 작업에 라우팅하여 이 제한사항의 영향을 받지 않도록 합니다. 자체 작업에서 연결이 실행됩니다. 시스템은 각 연결의 시작 시간을 최소화하기 위해 QSYSWRK 서브 시스템의 사전시작 작업을 사용합니다. SQLConnect에 대한 각 호출이 서로 다른 사용자 프로파일을 수용할 수 있으므로 각 작업에도 자체 확약 트랜잭션이 있습니다. 일단 SQLDisconnect가 실행되면 작업은 재설정되어 사용할 수 있는 작업 풀로 돌아옵니다.

SQL 서버 모드로 DB2 UDB CLI를 실행하는 것에 관한 정보는 다음을 참조하십시오.

- 『SQL 서버 모드로 DB2 UDB CLI 시작』
- 310 페이지의『서버 모드로 DB2 UDB CLI 실행 시 제한사항』

SQL 서버 모드로 DB2 UDB CLI 시작

작업을 SQL 서버 모드로 설정하는 방법은 두 가지입니다.

1. 가장 많이 사용하는 경우는 CLI 함수 및 SQLSetEnvAttr을 사용하는 것입니다. SQL 서버 모드는 CLI 어플리케이션에 가장 적합한데, 그 이유는 CLI 어플리케이션은 복수 연결 핸들 개념을 사용하고 있기 때문입니다. CLI 환경을 지정한 후 바로 이 모드를 설정하십시오. 또한 이 모드를 설정하기 전에는 작업이 SQL을 실행하거나 확약 제어를 시작해서는 안됩니다. 이 중 어느 하나라도 발생한다면, 모드는 서버 모드로 바뀌지 않으며 SQL은 "인라인"으로 계속 실행될 것입니다.

   EXAMPLE.

   ```
   .
   SQLAllocEnv(&henv);
   long  attr;
   attr  =  SQL_TRUE
   SQLSetEnvAttr(henv,SQL_ATTR_SERVER_MODE,&attr,0);
   ```

```
SQLAllocConnect(henv,&hdbc);
.
.
```

2. 서버 모드를 설정하는 두 번째 방법은 QWTCHGJB(작업 변경) API를 사용하는 것입니다. iSeries Information Center에서 API 주제를 참조하십시오.

SQL 서버 모드가 설정되고 나면, 모든 SQL 연결과 SQL문은 서버 모드로 실행됩니다. 모드의 전환은 일어나지 않습니다. 작업이 서버 모드에 있게 되면 확약 제어를 시작할 수 없으며 대화식 SQL도 사용할 수 없습니다.

서버 모드로 DB2 UDB CLI 실행 시 제한사항

- 다른 일을 수행하려면 작업은 프로세스가 시작되면 바로 서버 모드를 설정해야 합니다. CLI를 엄격하게 사용하는 작업 사용자는 SQLSetEnvAttr 호출을 사용하여 서버 모드를 켜야 합니다. 여기에서 기억해야 할 점은 SQLAllocEnv 호출 직후이지만 다른 어떤 호출도 있기 전에 해야 한다는 점입니다. 서버 모드가 한 번 켜지면 끌 수 없습니다.
- 모든 SQL 함수는 사전시작 작업과 확약 제어에서 실행됩니다. 서버 모드로 들어가기 전이나 후에는 처음 시작한 작업에서 확약 제어를 시작하지 마십시오.
- SQL은 사전시작 작업에서 처리되므로 처음 시작한 작업의 특정 변경사항으로부터 영향을 받지 않습니다. 그러한 변경사항에는 라이브러리 리스트, 작업 우선순위, 메세지 기록 등이 있습니다. 사전시작은 처음 시작한 작업에서 CCSID 값이 변경되는 것에 영향을 받는데 이것은 데이터가 사용자의 프로그램에 다시 맵핑되는 방식에 영향을 주기 때문입니다.
- 서버 모드를 실행할 때, 어플리케이션은 내장된 것이든 SQL CLI에 의한 것이든 확약과 롤백을 사용해야 합니다. 처음 시작한 작업에서 실행 중인 확약 제어가 없기 때문에 CL 명령을 사용할 수 없습니다. 이 작업은 연결을 끊기 전에 확약을 발행해야 합니다. 그렇지 않으면, 내재적 자동 롤백이 일어날 것입니다.
- 서버 모드에 있는 작업에서 대화식 SQL을 사용할 수 없습니다. 서버 모드에서 STRSQL을 사용하면 SQL6141 메세지가 표시됩니다.
- 서버 모드에 있는 동안 SQL 컴파일을 수행할 수도 없습니다. 서버 모드는 컴파일이 완료된 SQL 프로그램을 실행할 때 사용될 수 있지만, 컴파일에는 사용할 수 없습니다. 작업이 서버 모드이면, 컴파일은 실패할 것입니다.
- SQLDataSources는 연결 핸들없이 실행할 수 있다는 것이 특징입니다. 서버 모드인 경우, 프로그램은 SQLDataSources를 사용하기 전에, 로컬 데이터베이스에 이미 연결되어 있어야 합니다. DataSources는 연결을 위한 RDB명을 찾는 데 사용되므로 IBM은 SQLConnect의 RDB 이름에 널(null) 포인터를 전달하도록 지원합니다. 이렇게 하면 로컬로 연결되므로 시스템명에 대한 사전 지식이 없더라도 일반 프로그램을 작성할 수 있습니다.
- CLI를 통해 확약과 롤백을 수행하는 경우, SQLEndTran과 SQLTransact 호출에는 연결 핸들이 있어야 합니다. 서버 모드에서 실행되지 않을 때에는, 모든 것을 확약하는 연결 핸들을 생략할 수 있습니다. 그러나 서버 모드에서는 각 연결(또는 스레드)에 자체 트랜잭션 범위(scoping)가 있기 때문에 생략할 수 없습니다.

- SQL 서버 모드에서 실행될 때, 여러 스레드에서 연결 핸들을 공유하지 않는 것이 좋습니다. 한 스레드가 아직 처리하지 않은 리턴 자료나 오류 정보를 다른 스레드가 덮어쓸 수 있기 때문입니다.

색인

[가]

[나]

[다]

[라]

[마]

[바]

[사]

[아]

[자]

[차]

[카]

[타]

[파]

[하]

[숫자]

B

C

D

F

G

I

N

O

S

Printed in U.S.A