

IBM

@server

iSeries

組み込み SQL プログラミング

バージョン 5 リリース 3





@server

iSeries

組み込み SQL プログラミング

バージョン 5 リリース 3

ご注意!

本書および本書で紹介する製品をご使用になる前に、211ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM OS/400 (プロダクト番号 5722-SS1) のバージョン 5、リリース 3、モディフィケーション 0 に適用されます。また、改訂版で断りがない限り、それ以降のすべてのリリースおよびモディフィケーションに適用されます。このバージョンは、すべての RISC モデルで稼働するとは限りません。また CISC モデルでは稼働しません。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

<http://www.ibm.com/jp/manuals/main/mail.html>

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

<http://www.ibm.com/jp/manuals/> の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原 典： @server
iSeries
Embedded SQL programming
Version 5 Release 3

発 行： 日本アイ・ビー・エム株式会社

担 当： ナショナル・ランゲージ・サポート

第1刷 2005.8

この文書では、平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財)日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

© Copyright International Business Machines Corporation 1998, 2005. All rights reserved.

© Copyright IBM Japan 2005

目次

第 1 章 組み込み SQL プログラミング . . . 1

第 2 章 SQL プログラミング、バージョン

5 リリース 3 の新機能 3

第 3 章 トピックの印刷 5

第 4 章 組み込み SQL の使用に関する一 般的な概念と規則 7

SQL を使用するアプリケーションの作成 7

SQL ステートメントでのホスト変数の使用 7

SQL ステートメントでのホスト変数の割り当て規
則 9

SQL を使用するアプリケーションでの標識変数 12

SQLCA を使用した SQL エラー戻りコードの処理 14

SQL 診断域を使用する 15

SQL 診断域を使用するようアプリケーションを更
新する 15

iSeries サーバー・プログラミング・モデル 16

SQL 診断域を使用するうえでの他の注意事項 16

例: SQL ルーチン例外 16

例: SQL 診断域の項目をロギングする 17

WHENEVER ステートメントによる例外条件の処理 17

第 5 章 C および C++ アプリケーショ ンでの SQL ステートメントのコーディン グ方法 21

SQL を使用する C および C++ アプリケーショ
ンでの SQL 連絡域の定義 21

SQL を使用する C および C++ アプリケーショ
ンでの SQL 記述子域の定義 23

SQL を使用する C および C++ アプリケーショ
ンへの SQL ステートメントの組み込み 25

SQL を使用する C および C++ アプリケーショ
ンでの注記 25

SQL を使用する C および C++ アプリケーショ
ンでの SQL ステートメントの継続 25

SQL を使用する C および C++ アプリケーショ
ンへのコードの組み込み 26

SQL を使用する C および C++ アプリケーショ
ンでのマージン 26

SQL を使用する C および C++ アプリケーショ
ンでの名前 26

SQL を使用する C および C++ アプリケーショ
ンでの NULL および NUL 27

SQL を使用する C および C++ アプリケーショ
ンでのステートメント・ラベル 27

SQL を使用する C および C++ アプリケーショ
ンでのプリプロセッサ順序 27

SQL を使用する C および C++ アプリケーショ
ンでの 3 文字表記 27

SQL を使用する C および C++ アプリケーショ
ンでの WHENEVER ステートメント 27

SQL を使用する C および C++ アプリケーショ
ンでのホスト変数の使用 27

SQL を使用する C および C++ アプリケーショ
ンでのホスト変数の宣言 28

SQL を使用する C および C++ アプリケーショ
ンでのホスト構造の使用 40

SQL を使用する C および C++ アプリケーショ
ンでのホスト構造宣言 41

SQL を使用する C および C++ アプリケーショ
ンでのホスト構造標識配列 44

SQL を使用する C および C++ アプリケーショ
ンでのホスト構造配列の使用 44

SQL を使用する C および C++ アプリケーショ
ンでのホスト構造配列 45

SQL を使用する C および C++ アプリケーショ
ンでのホスト構造配列標識構造 48

SQL を使用する C および C++ アプリケーショ
ンでのポインター・データ・タイプの使用 48

SQL を使用する C および C++ アプリケーショ
ンでの typedef の使用 49

SQL を使用する C および C++ アプリケーショ
ンでの ILE C コンパイラ 外部ファイル記述の使用 . 50

SQL データ・タイプおよび C または C++ デー
タ・タイプの対応関係の判別 51

C および C++ 変数宣言と使用法についての注意 53

SQL を使用する C および C++ アプリケーショ
ンでの標識変数の使用 54

第 6 章 COBOL アプリケーションでの SQL ステートメントのコーディング方法 . 55

SQL を使用する COBOL アプリケーションでの
SQL 連絡域の定義 55

SQL を使用する COBOL アプリケーションでの
SQL 記述子域の定義 56

SQL を使用する COBOL アプリケーションでの
SQL ステートメントの組み込み 58

SQL を使用する COBOL アプリケーションでの
注記 59

SQL を使用する COBOL アプリケーションでの
SQL ステートメントの継続 59

SQL を使用する COBOL アプリケーションでの
コードの組み込み 59

SQL を使用する COBOL アプリケーションでの
マージン 59

SQL を使用する COBOL アプリケーションでの
順序番号 59

SQL を使用する COBOL アプリケーションでの名前	59
SQL を使用する COBOL アプリケーションでの COBOL コンパイル時オプション	60
SQL を使用する COBOL アプリケーションでのステートメント・ラベル	60
SQL を使用する COBOL アプリケーションでの WHENEVER ステートメント	60
複数ソース COBOL プログラムおよび SQL COBOL プリコンパイラー	60
SQL を使用する COBOL アプリケーションでのホスト変数の使用	60
SQL を使用する COBOL アプリケーションでのホスト変数の宣言	61
SQL を使用する COBOL アプリケーションでのホスト構造の使用	70
SQL を使用する COBOL アプリケーションでのホスト構造	71
SQL を使用する COBOL アプリケーションでのホスト構造標識配列	74
SQL を使用する COBOL アプリケーションでのホスト構造配列の使用	75
SQL を使用する COBOL アプリケーションでのホスト構造配列	76
SQL を使用する COBOL アプリケーションでのホスト配列標識構造	79
SQL を使用する COBOL アプリケーションでの外部ファイル記述の使用	80
SQL を使用する COBOL アプリケーションでのホスト構造配列の外部ファイル記述の使用	81
SQL データ・タイプと COBOL データ・タイプの対応関係の判別	82
COBOL 変数宣言と使用上の注意事項	84
SQL を使用する COBOL アプリケーションでの標識変数の使用	85

第 7 章 PL/I アプリケーションでの SQL ステートメントのコーディング方法 . . . 87

SQL を使用する PL/I アプリケーションでの SQL 連絡域の定義	87
SQL を使用する PL/I アプリケーションでの SQL 記述子域の使用	88
SQL を使用する PL/I アプリケーションでの SQL ステートメントの組み込み	89
例: SQL を使用する PL/I ステートメントでの SQL ステートメントの組み込み	89
SQL を使用する PL/I アプリケーションでの注記	90
SQL を使用する PL/I アプリケーションでの SQL ステートメントの継続	90
SQL を使用する PL/I アプリケーションでのコードの組み込み	90
SQL を使用する PL/I アプリケーションでのマージン	90
SQL を使用する PL/I アプリケーションでの名前	90
SQL を使用する PL/I アプリケーションでのステートメント・ラベル	90

SQL を使用する PL/I アプリケーションでの WHENEVER ステートメント	90
SQL を使用する PL/I アプリケーションでのホスト変数の使用	91
SQL を使用する PL/I アプリケーションでのホスト変数の宣言	91
SQL を使用する PL/I アプリケーションでのホスト構造の使用	96
SQL を使用する PL/I アプリケーションでのホスト構造	97
SQL を使用する PL/I アプリケーションでのホスト構造標識配列	98
SQL を使用する PL/I アプリケーションでのホスト構造配列の使用	98
SQL を使用する PL/I アプリケーションでのホスト構造配列	99
SQL を使用する PL/I アプリケーションでの外部ファイル記述の使用	100
SQL データ・タイプと PL/I データ・タイプの対応関係の判別	101
SQL を使用する PL/I アプリケーションでの標識変数の使用	103
構造パラメーター受け渡し技法による PL/I での相違	104

第 8 章 RPG for iSeries アプリケーションでの SQL ステートメントのコーディング方法 . . . 105

SQL を使用する RPG for iSeries アプリケーションでの SQL 連絡域の定義	105
SQL を使用する RPG for iSeries アプリケーションでの SQL 記述子域の定義	106
SQL を使用する RPG for iSeries アプリケーションでの SQL ステートメントの組み込み	107
例: SQL を使用する RPG for iSeries アプリケーションでの SQL ステートメントの組み込み	107
SQL を使用する RPG for iSeries アプリケーションでの注記	108
SQL を使用する RPG for iSeries アプリケーションでの SQL ステートメントの継続	108
SQL を使用する RPG for iSeries アプリケーションでのコードの組み込み	108
SQL を使用する RPG for iSeries アプリケーションでの順序番号	108
SQL を使用する RPG for iSeries アプリケーションでの名前	108
SQL を使用する RPG for iSeries アプリケーションでのステートメント・ラベル	108
SQL を使用する RPG for iSeries アプリケーションでの WHENEVER ステートメント	109
SQL を使用する RPG for iSeries アプリケーションでのホスト変数の使用	109
SQL を使用する RPG for iSeries アプリケーションでのホスト変数の宣言	109
SQL を使用する RPG for iSeries アプリケーションでのホスト構造の使用	110

SQL を使用する RPG for iSeries アプリケーションでの ホスト構造配列の使用	110
SQL を使用する RPG for iSeries アプリケーションでの 外部ファイル記述の使用	111
SQL を使用する RPG for iSeries アプリケーションでの ホスト構造配列の外部ファイル記述に関する 考慮事項	112
SQL データ・タイプと RPG for iSeries データ・ タイプの対応関係の判別	113
SQL を使用する RPG for iSeries アプリケーションでの 割り当て規則	115
SQL を使用する RPG for iSeries アプリケーションでの 標識変数の使用	115
例: SQL を使用する RPG for iSeries アプリケーションでの 標識変数の使用	116
構造パラメーター受け渡し技法による RPG for iSeries での相違	116
呼び出された SQL を使用する RPG for iSeries プ ログラムの正しい終了方法	116
第 9 章 ILE RPG for iSeries アプリケーションでの SQL ステートメントのコーディング方法 119	
SQL を使用する ILE RPG for iSeries アプリケーションでの SQL 連絡域の定義	119
SQL を使用する ILE RPG for iSeries アプリケーションでの SQL 記述子域の定義	120
SQL を使用する ILE RPG for iSeries アプリケーションでの SQL ステートメントの組み込み	122
SQL を使用する ILE RPG for iSeries アプリケーションでの 注記	122
SQL を使用する ILE RPG for iSeries アプリケーションでの SQL ステートメントの継続	122
SQL を使用する ILE RPG for iSeries アプリケーションでの コードの組み込み	123
SQL を使用する ILE RPG for iSeries アプリケーションでの ディレクティブの使用	123
SQL を使用する ILE RPG for iSeries アプリケーションでの 順序番号	124
SQL を使用する ILE RPG for iSeries アプリケーションでの 名前	124
SQL を使用する ILE RPG for iSeries アプリケーションでの ステートメント・ラベル	124
SQL を使用する ILE RPG for iSeries アプリケーションでの WHENEVER ステートメント	124
SQL を使用する ILE RPG for iSeries アプリケーションでの ホスト変数の使用	124
SQL を使用する ILE RPG for iSeries アプリケーションでの ホスト変数の宣言	125
SQL を使用する ILE RPG for iSeries アプリケーションでの ホスト構造の使用	129
SQL を使用する ILE RPG for iSeries アプリケーションでの ホスト構造配列の使用	132
SQL を使用する ILE RPG for iSeries アプリケーションでの 外部ファイル記述の使用	133

SQL を使用する ILE RPG for iSeries アプリケーションでの ホスト構造配列の外部ファイル記述に関する 考慮事項	133
SQL データ・タイプと ILE RPG データ・タイプの 対応関係の判別	134
ILE RPG for iSeries 変数宣言と使用法に関する 注意事項	141
SQL を使用する ILE RPG for iSeries アプリケーションでの 標識変数の使用	141
例: SQL を使用する ILE RPG for iSeries アプリケーションでの 標識変数の使用	142
SQL を使用する ILE RPG for iSeries アプリケーションでの 複数行領域取り出し用 SQLDA の例	142
SQL を使用する ILE RPG for iSeries アプリケーションでの 動的 SQL の例	143

第 10 章 REXX アプリケーションでの SQL ステートメントのコーディング方法 145	
REXX アプリケーションでの SQL 連絡域の定義	145
REXX アプリケーションでの SQL 記述子域の使用	146
REXX アプリケーションでの SQL ステートメントの 組み込み	147
SQL を使用する REXX アプリケーションでの 注記	149
SQL を使用する REXX アプリケーションでの SQL ステートメントの継続	149
SQL を使用する REXX アプリケーションでの コードの組み込み	149
SQL を使用する REXX アプリケーションでの マージン	149
SQL を使用する REXX アプリケーションでの 名前	149
SQL を使用する REXX アプリケーションでの ヌル	150
SQL を使用する REXX アプリケーションでの ステートメント・ラベル	150
SQL を使用する REXX アプリケーションでの エラーおよび警告の処理	150
SQL を使用する REXX アプリケーションでの ホスト変数の使用	150
SQL を使用する REXX アプリケーションでの 入力ホスト変数のデータ・タイプの判別	151
SQL を使用する REXX アプリケーションでの 出力ホスト変数のフォーマット	152
SQL を使用する REXX アプリケーションでの REXX 変換の回避	153
SQL を使用する REXX アプリケーションでの 標識変数の使用	153

第 11 章 SQL ステートメントを含むプログラムの準備と実行 155	
SQL プリコンパイラーの基本処理	155
SQL プリコンパイラーへの入力	156

SQL プリコンパイラーのソース・ファイル	
CCSID	157
SQL プリコンパイラーの出力	157
非 ILE SQL プリコンパイラー・コマンド	163
SQL を使用する非 ILE アプリケーション・プログラム のコンパイル	163
ILE SQL プリコンパイラー・コマンド	164
SQL を使用する ILE アプリケーション・プログラ ムのコンパイル	164
I プリコンパイラー・コマンドを使用したコンパイラ I ー・オプションの設定	165
SQL を使用するアプリケーションでのコンパ イル・エラーの解釈	166
SQL を使用するアプリケーション・プログラ ムのコンパイル時のエラーおよび警告メッセ ージ	166
SQL を使用するアプリケーションのバインド	167
SQL を使用するアプリケーションでのプログラ ム参照	168
SQL プリコンパイラー・オプションの表示	168
組み込み SQL を使用したプログラムの実行	168
組み込み SQL を使用したプログラムの実行: OS/400 DDM の考慮事項	169
組み込み SQL を使用したプログラム実行: 一時 変更に関する考慮事項	169
組み込み SQL を使用したプログラムの実行: SQL 戻りコード	169
第 12 章 DB2 UDB for iSeries ステ トメントを使用したサンプル・プログラ ム	171
例: ILE C および C++ プログラム内の SQL ステ ートメント	173

例: COBOL および ILE COBOL プログラム内の SQL ステートメント	178
例: PL/I 内の SQL ステートメント	186
例: RPG for iSeries プログラム内の SQL ステ ートメント	191
例: ILE RPG for iSeries プログラム内の SQL ステ ートメント	197
例: REXX プログラム内の SQL ステートメント	203
SQL を使用したサンプル・プログラムにより作成 される報告書	207

第 13 章 ホスト言語プリコンパイラ 用の DB2 UDB for iSeries CL コマン ドの説明	209
CRTSQCLBCL (SQL COBOL プログラム作成) コマ ンド	209
CRTSQCLBLI (SQL ILE COBOL オブジェクト作 成) コマンド	209
CRTSQCLCI (SQL ILE C オブジェクト作成) コマン ド	209
CRTSQCLCPPI (SQL C++ オブジェクト作成) コマン ド	210
CRTSQCLPLI (SQL PL/I 作成) コマンド	210
CRTSQCLRPG (SQL RPG 作成) コマンド	210
CRTSQCLRPGI (SQL ILE RPG オブジェクト作成) コマンド	210

付録. 特記事項	211
プログラミング・インターフェース情報	213
商標	213
資料に関するご使用条件	213
コードについての特記事項	214

第 1 章 組み込み SQL プログラミング

本書は、DB2® UDB for iSeries™ の SQL ステートメントおよび機能を使用したホスト言語でのデータベース・アプリケーション作成について説明しています。

組み込み SQL プログラミングについての詳細情報として、以下のトピックがあります。

V5R3 の新機能

このリリースでの組み込み SQL プログラミングに関する変更点と追加事項を確認してください。

トピックの印刷

組み込み SQL プログラミングに関するハードコピー資料を得るために、本書を印刷します。

組み込み SQL の使用に関する一般的な概念と規則

一般的な概念と規則について学習します。組み込み SQL プログラミングでのホスト変数の使用、SQL エラーおよび戻りコードの扱い方、および WHENEVER ステートメントを使った例外条件処理について説明されています。

C および C++ アプリケーションでの SQL ステートメントのコーディング方法

SQL ステートメントを C または C++ プログラムに組み込む場合に固有のアプリケーションおよびコーディング上の要件について説明します。

COBOL アプリケーションでの SQL ステートメントのコーディング方法

SQL ステートメントを COBOL プログラムに組み込む場合に固有のアプリケーションおよびコーディング上の要件について説明します。

PL/I アプリケーションでの SQL ステートメントのコーディング方法

SQL ステートメントを iSeries PL/I プログラムに組み込む場合に固有のアプリケーションおよびコーディング上の要件について説明します。

RPG for iSeries アプリケーションでの SQL ステートメントのコーディング方法

SQL ステートメントを RPG for iSeries プログラムに組み込む場合に固有のアプリケーションおよびコーディング上の要件について説明します。

ILE RPG for iSeries アプリケーションでの SQL ステートメントのコーディング方法

SQL ステートメントを ILE RPG for iSeries プログラムに組み込む場合の固有のアプリケーションおよびコーディング上の要件について説明します。

REXX アプリケーションでの SQL ステートメントのコーディング方法

SQL ステートメントを REXX アプリケーションに組み込む方法について説明します。

SQL ステートメントを含むプログラムの準備と実行

アプリケーション・プログラムの準備と実行に必要な作業をいくつか説明します。

DB2 UDB for iSeries ステートメントを使用したサンプル・プログラム

DB2 UDB for iSeries システムがサポートする各言語で SQL ステートメントをコーディングする方法を示したサンプル・アプリケーションが記載されています。

ホスト言語プリコンパイラー用の DB2 UDB for iSeries CL コマンドの説明

組み込み SQL ステートメントを使用する各プログラム言語で書かれたプログラムをプリコンパイルするためのコマンド使用方法について説明します。



注: 重要な法律的事柄に関する情報について、214 ページの『コードについての特記事項』をご覧ください。

第 2 章 SQL プログラミング、バージョン 5 リリース 3 の新機能

- このトピックは、以前は「ホスト言語での SQL プログラミング」という資料でした。
- SQL 診断域を使用して、SQL エラー戻りコードを処理します。
- ホスト変数 BINARY および VARBINARY (C および C++、COBOL、ILE RPG、および PL/I 言語)。
- ILE RPG 用の SQL 連絡域への標準的な SQL 変数名の追加。
- ILE RPG でのホスト変数の規則。
- ディレクティブを処理するための ILE RPG ソースのプリプロセス (ネストされた /COPY ステートメントのサポートを含む)。
- プリコンパイラー・コマンドを使用したコンパイラー・オプションの設定での COMPILEOPT パラメーター。

新規情報と改訂情報の見つけ方

この資料では、技術上の改訂情報を見つやすくために、次のアイコンを使っています。

-  アイコン。このアイコンの場所から、新規情報や改訂情報が始まっています。
-  アイコン。このアイコンの場所で、新規情報や改訂情報が終わっています。

このリリースの新規情報や改訂情報に関するその他の情報については、「iSeries 最初にお読みください」を参照してください。

第 3 章 トピックの印刷

この資料の PDF 版をダウンロードして表示するには、組み込み SQL プログラミング PDF (約 1168 KB) を選択します。

以下のような関連トピックを表示またはダウンロードできます。


- SQL 解説書 (5663 KB) には、次の情報が含まれています。
 - システム管理、データベース管理、アプリケーション・プログラミング、運用などのタスクに関連した参照情報。
 - iSeries システムで使用されるそれぞれの SQL ステートメントの構文、使用上の注意、キーワード、および例。
- SQL プログラミング (2149 KB) には、次の情報が含まれています。
 - DB2 UDB for iSeries ライセンス・プログラムの使用方法。
 - データベース内のデータへのアクセス方法。
 - SQL ステートメントを含むアプリケーション・プログラムを作成、実行、およびテストする方法。
- データベース・パフォーマンスおよび QUERY の最適化 (1815 KB) には、次の情報が含まれています。
 - データベース・アプリケーションのパフォーマンスを最適化するために、DB2 UDB for iSeries で使用可能なツールや機能を使用する方法。
 - DB2 UDB for iSeries 統合データベースの機能を最大限に活用する照会を実行する方法。
- SQL 呼び出しレベル・インターフェース (ODBC) (1271 KB) には、次の情報が含まれています。
 - 組み込み動的 SQL に代わって DB2 UDB CLI を使用する方法。
 - DB2 UDB CLI 機能の説明および例。

PDF ファイルの保管

表示または印刷の PDF をワークステーションに保存するには、以下のようにします。

1. ブラウザーで PDF を右マウス・ボタン・クリックする (上部のリンクを右マウス・ボタン・クリック)。
2. Internet Explorer を使用している場合は、「対象をファイルに保存...」をクリックする。Netscape Communicator を使用している場合は、「リンクを名前を付けて保存...」をクリックする。
3. PDF を保存したいディレクトリーに進む。
4. 「保存」をクリックする。

Adobe Acrobat Reader のダウンロード

これらの PDF を表示または印刷するには、Adobe Acrobat Reader が必要です。このアプリケーションは、Adobe Web サイト (www.adobe.com/products/acrobat/readstep.html)  からダウンロードできます。

第 4 章 組み込み SQL の使用に関する一般的な概念と規則

ここでは、ホスト言語での SQL ステートメントの使用における一般的な概念と規則を、次の事項に分けて説明します。

- 『SQL を使用するアプリケーションの作成』
- 『SQL ステートメントでのホスト変数の使用』
- 14 ページの『SQLCA を使用した SQL エラー戻りコードの処理』
- 15 ページの『SQL 診断域を使用する』
- 17 ページの『WHENEVER ステートメントによる例外条件の処理』

注: 重要な法律的事柄に関する情報について、214 ページの『コードについての特記事項』をご覧ください。

SQL を使用するアプリケーションの作成

DB2 UDB for iSeries SQL ステートメントおよび機能を使用したホスト言語でのデータベース・アプリケーションを作成できます。組み込み SQL を使用するには、DB2 Query Manager and SQL Development Kit がインストール済みである必要があります。さらに、使用するホスト言語に対応したコンパイラをインストールする必要もあります。各ホスト言語についての、アプリケーション要件およびコーディング要件の詳細は、以下を参照してください。

- 21 ページの『第 5 章 C および C++ アプリケーションでの SQL ステートメントのコーディング方法』
- 55 ページの『第 6 章 COBOL アプリケーションでの SQL ステートメントのコーディング方法』
- 87 ページの『第 7 章 PL/I アプリケーションでの SQL ステートメントのコーディング方法』
- 105 ページの『第 8 章 RPG for iSeries アプリケーションでの SQL ステートメントのコーディング方法』
- 119 ページの『第 9 章 ILE RPG for iSeries アプリケーションでの SQL ステートメントのコーディング方法』
- 145 ページの『第 10 章 REXX アプリケーションでの SQL ステートメントのコーディング方法』
- 155 ページの『第 11 章 SQL ステートメントを含むプログラムの準備と実行』

注: ホスト言語として Java™ を使用する場合の詳細については、「IBM® Developer Kit for Java」を参照してください。

SQL ステートメントでのホスト変数の使用

ユーザー・プログラムがデータを検索する場合、その値は、ユーザー・プログラムによって定義され、SELECT INTO ステートメントおよび FETCH ステートメントの INTO 文節で指定されたデータ項目に入れます。このようなデータ項目を**ホスト変数**と呼びます。

ホスト変数は、ユーザー・プログラムのフィールドで、通常、列の取り出し元または受け入れ先として SQL ステートメントの中で指定されるものです。ホスト変数と列は、データ・タイプに互換性がなければなりません。ホスト変数は、テーブルやビューなどの SQL オブジェクトの識別には使用できません。ただし、DESCRIBE TABLE ステートメントでは別です。

ホスト構造は、選択された一連の値 (たとえば、ある行の一連の列の値) の取り出し元または受け入れ先として使用されるホスト変数のグループです。ホスト構造配列は、複数行用 FETCH ステートメントおよびブロック化 INSERT ステートメントで使用されるホスト構造の配列です。

注: SQL ステートメントの中でリテラル値の代わりにホスト変数を使用すると、アプリケーション・プログラムがテーブルまたはビューのいくつかの異なる行を処理する際の柔軟性を高めることができます。

たとえば、WHERE 文節の中で実際の部門番号をコーディングする代わりに、その時処理を必要としている部門番号にセットされたホスト変数を使用することができます。

一般に、ホスト変数は、SQL ステートメントの中で次のように使用されます。

1. **WHERE 文節の中で:** 探索条件の述部の中の値を指定するため、または式の中のリテラル値を書き換えるために、ホスト変数を使用することができます。たとえば、社員番号を入れるための EMPID という名前のフィールドが定義されているとすれば、社員番号が 000110 である社員の名前を、次のようにして検索することができます。

```
MOVE '000110' TO EMPID.  
EXEC SQL  
  SELECT LASTNAME  
     INTO :PGM-LASTNAME  
  FROM CORPDATA.EMPLOYEE  
  WHERE EMPNO = :EMPID  
END-EXEC.
```

2. **列の値の受け入れ値として (INTO 文節で指定):** 検索した行の列の値を入れるプログラム・データ域を指定するために、ホスト変数を使用することができます。INTO 文節に、SQL から返された列の値を入れる 1 つ以上のホスト変数の名前を指定します。たとえば、CORPDATA.EMPLOYEE テーブルの行から、EMPNO、LASTNAME、および WORKDEPT の各列の値を検索するものとします。この場合、各列の値を入れるホスト変数をユーザー・プログラムで定義し、それらのホスト変数の名前を INTO 文節に指定します。以下に、例を示します。

```
EXEC SQL  
  SELECT EMPNO, LASTNAME, WORKDEPT  
     INTO :CBLEMPNO, :CBLNAME, :CBLDEPT  
  FROM CORPDATA.EMPLOYEE  
  WHERE EMPNO = :EMPID  
END-EXEC.
```

この例では、ホスト変数 CBLEMPNO は EMPNO の値を受け入れ、CBLNAME は LASTNAME の値を受け入れ、CBLDEPT は WORKDEPT の値を受け入れます。

3. **SELECT 文節の値として:** SELECT 文節に項目のリストを指定する場合、指定できるのは、テーブルやビューの列名に限定されているわけではありません。ユーザー・プログラムは、ホスト変数の値およびリテラル定数を混在させて一連の列の値を返すことができます。以下に、例を示します。

```
MOVE '000220' TO PERSON.  
EXEC SQL  
  SELECT "A", LASTNAME, SALARY, :RAISE,  
     SALARY + :RAISE  
     INTO :PROCESS, :PERSON-NAME, :EMP-SAL,  
     :EMP-RAISE, :EMP-TTL  
  FROM CORPDATA.EMPLOYEE  
  WHERE EMPNO = :PERSON  
END-EXEC.
```


結果は次のようになります。

PROCESS	PERSON-NAME	EMP-SAL (給与)	EMP-RAISE (昇給額)	EMP-TTL (合計)
A	LUTZ	29840	4476	34316

4. SQL ステートメントの他の文節の値として:

- UPDATE ステートメントの SET 文節
- INSERT ステートメントの VALUES 文節
- CALL ステートメント

これらのステートメントの詳細については、「SQL 解説書」を参照してください。

ホスト変数の使用法の詳細については、次のセクションを参照してください。

- 『SQL ステートメントでのホスト変数の割り当て規則』
- 13 ページの『ホスト構造で使用される標識変数』

SQL ステートメントでのホスト変数の割り当て規則

SQL 値は、FETCH、SELECT INTO、SET、および VALUES INTO ステートメントの実行時に、ホスト変数に割り当てられます (代入されます)。SQL 値は、INSERT、UPDATE、および CALL ステートメントの実行時に、ホスト変数から割り当てられます。すべての割り当て操作は次の規則に従って行われます。

- 数値と文字列 (ストリング) の間には、次のような互換性があります。

数値を、文字ストリングまたは漢字ストリングの列またはホスト変数に割り当てることができます。
文字ストリングまたは漢字ストリングを、数値列または数値ホスト変数に割り当てることができます。

- すべての文字ストリングおよび DBCS 漢字ストリングは、各 CCSID 間で変換がサポートされている場合、UCS-2 および UTF-16 グラフィック列と互換性があります。CCSID に互換性があれば、すべての漢字ストリングは互換性があります。すべての数値は互換性があります。必要があれば、SQL により変換が行われます。すべての文字ストリングおよび DBCS 漢字ストリングは、各 CCSID 間で変換がサポートされている場合は、割り当て操作の UCS-2 および UTF-16 グラフィック列と互換性があります。CALL ステートメントに関しては、変換がサポートされている場合は、DBCS グラフィック・パラメーターは UCS-2 および UTF-16 パラメーターと互換性があります。
- バイナリー・ストリングは、バイナリー・ストリングとだけ互換性があります。
- 関連する標識変数をもっていないホスト変数にヌル値を割り当てすることはできません。
- タイプが異なる日付 / 時刻値の間には互換性がありません。日付は、日付または日付の文字表現とだけ互換性があります。時刻は、時刻または時刻の文字表現とだけ互換性があります。タイム・スタンプは、タイム・スタンプまたはタイム・スタンプの文字表現とだけ互換性があります。

日付は、日付列、文字列、DBCS-open または DBCS-either 列または変数、または文字変数¹にだけ割り当てることができます。日付列の挿入値または更新値は、日付または日付の文字表現でなければなりません。

1. DBCS-open または DBCS-either 変数とは、外部記述ファイルの定義を組み入れることによってホスト言語で宣言された変換です。DBCS-open 変数は、そのジョブの CCSID が MIXED (混合) データを示している場合や、DECLARE VARIABLE ステートメントが使用され、MIXED CCSID または FOR MIXED DATA 文節が指定されている場合にも、宣言されます。「SQL 解説書」の『DECLARE VARIABLE』を参照してください。

時刻は、時刻列、文字列、DBCS-open または DBCS-either 列または変数、または文字変数にだけ割り当てることができます。時刻列の挿入値または更新値は、時刻または時刻の文字表現でなければなりません。

タイム・スタンプは、タイム・スタンプ列、文字列、DBCS-open または DBCS-either 列または変数、または文字変数にだけ割り当てることができます。タイム・スタンプ列の挿入値または更新値は、タイム・スタンプまたはタイム・スタンプの文字表現でなければなりません。

割り当て規則の詳細については、以下のトピックを参照してください。

- 『SQL ステートメントでのホスト変数の文字列割り当て規則』
- 『SQL ステートメントでのホスト変数の CCSID 規則』
- 11 ページの 『SQL ステートメントでのホスト変数の数値割り当て規則』
- 11 ページの 『SQL ステートメントでのホスト変数の日付、時刻、およびタイム・スタンプの割り当て規則』
- 12 ページの 『SQL を使用するアプリケーションでの標識変数』

SQL ステートメントでのホスト変数の文字列割り当て規則

文字ストリングの割り当てに関する規則は次のとおりです。

- 文字ストリングまたは漢字ストリングを列に割り当てる場合、そのストリングの長さはその列の長さ属性より小さくなければなりません。(末尾ブランクは、通常文字列の長さに含まれます。ただし、ストリング割り当ての場合、末尾ブランクはストリングの長さに含まれません。)
- バイナリー・ストリングを列に割り当てる場合、そのストリングの長さはその列の長さ属性より小さくなければなりません。(16 進ゼロは、通常、ストリングの長さに含まれます。ただし、ストリング割り当ての場合、16 進ゼロはストリングの長さに含まれません。)
- MIXED 文字結果列を MIXED 列に割り当てる場合は、MIXED 文字結果列の値は有効な MIXED 文字列でなければなりません。
- 結果列の値をホスト変数に割り当てる場合、結果列の文字列値の方がホスト変数の長さ属性より長いと、文字列の右側の部分が余分の桁数だけ切り捨てられます。その場合には、SQLWARN0 および SQLWARN1 (SQLCA の中の) が W にセットされます。
- 結果列の値を文字または漢字の固定長ホスト変数に割り当てる場合、またはホスト変数の値を文字または漢字の固定長結果列に割り当てる場合に、ストリング値の長さが受け入れ先の長さ属性より小さいと、ストリングの右側の部分に余分な桁数だけブランクが埋め込まれます。
- 結果列の値を固定長バイナリー・ホスト変数に割り当てる場合、またはホスト変数の値を固定長バイナリー結果列に割り当てる場合に、ストリング値の長さが受け入れ先の長さ属性より小さいと、ストリングの右側の部分に余分な桁数だけ 16 進ゼロが埋め込まれます。
- MIXED 文字結果列の値を入れたホスト変数の長さが文字列の長さより短かったために、その文字列の末尾が切り捨てられる場合も、文字列の終わりのシフトイン文字は残されます。したがって、結果は有効な MIXED 文字ストリングのまま残っています。

SQL ステートメントでのホスト変数の CCSID 規則

ある文字値またはグラフィック値を別の文字値またはグラフィック値に割り当てる時、CCSID を考慮する必要があります。これには、ホスト変数の割り当てが含まれます。データベース・マネージャーは、SBCS データ、DBCS データ、MIXED データ、およびグラフィック・データを変換するとき、共通のシステム・サービスを使用します。

CCSID に関する規則は次のとおりです。

- 取り出し元の CCSID が受け入れ先の CCSID と一致するときは、変換することなく値が割り当てられます。
- 取り出し元または受け入れ先のサブタイプが BIT のときは、変換することなく値が割り当てられます。
- 値がヌルかヌル・ストリングのときは、変換することなく値が割り当てられます。
- 特定の CCSID 相互間で変換することが定義されていないときは、値は割り当てられず、エラー・メッセージが出されます。
- 変換が定義されていて、変換が必要な場合は、取り出し元の値が受け入れ先の CCSID に変換されてから割り当てが行われます。

CCSID の詳細については、Information Center の『グローバルゼーション』を参照してください。

SQL ステートメントでのホスト変数の数値割り当て規則

数値の割り当てに関する規則は次のとおりです。

- 数値を浮動小数点数に変換するとき、その数値の整数部分が変更されることがあります。単精度の浮動小数点フィールドには 7 桁の 10 進数しか入れることができません。7 桁を超える数値の整数部分は、丸めによって変更されます。倍精度の浮動小数点フィールドには 16 桁の 10 進数しか入れることはできません。16 桁を超える数値の整数部分は、丸めによって変更されます。
- 数値の整数部分が切り捨てられることはありません。必要ならば、ステートメント内の小数部が切り捨てられます。変換後の数値が受け入れ先のホスト変数または列に収まらない場合には、負の SQLCODE が返されます。
- **10 進数、数字、または整数**が 10 進数、数字、または整数の列またはホスト変数に割り当てられる場合、必要に応じて、その数値は受け入れ先の精度および位取りに変換されます。その際に、必要数の先行ゼロが付加または削除されます。数値の小数部においては、必要数の後続ゼロが付加されるか、または末尾の必要数の数字が除去されます。
- **整数または浮動小数点数**が 10 進数または数字の列またはホスト変数に割り当てられる場合、その数値はまず一時的に 10 進数または数字に変換され、その後、必要に応じて受け入れ先の精度および位取りに変換されます。
 - 位取りが 0 の **ハーフワード 2 進数 (SMALLINT)** が 10 進数または数字に変換される場合には、一時的な結果は精度が 5 で位取りが 0 となります。
 - **フルワード 2 進数 (INTEGER)** が 10 進数または数字に変換される場合には、一時的な結果は精度が 11 で位取りが 0 となります。
 - **ダブル・フルワード 2 進数 (BIGINT)** が 10 進数または数字に変換される場合には、一時的な結果は精度が 19 で位取りが 0 となります。
 - **浮動小数点数**が 10 進数または数字に変換される場合には、一時的な結果は精度が 31 となり、最大の位取りは、有効数字も正確度も失わずに数値の整数部分全体を表すものになります。

SQL ステートメントでのホスト変数の日付、時刻、およびタイム・スタンプの割り当て規則

日付がホスト変数に割り当てられるときは、その日付は CRTSQLxxx コマンドの DATFMT パラメーターおよび DATSEP パラメーターで指定された文字表現に変換されます。先行ゼロは文字表現のどの部分からも省かれません。ホスト変数は固定長または可変長の文字ストリング変数であって、その長さは *USA、*EUR、*JIS、または *ISO の日付形式の場合は少なくとも 10 バイト、*MDY、*DMY、または *YMD の日付形式の場合は 8 バイト、*JUL の日付形式の場合は 6 バイトでなければなりません。長さが 10 を超えるときは、文字列の右側にブランクが埋められます。ILE RPG および ILE COBOL では、ホスト変数は日付変数にも使用できます。

時刻がホスト変数に割り当てられると、その時刻は、CRTSQLxxx コマンドの TIMFMT パラメーターおよび TIMSEP パラメーターで指定された文字表現に変換されます。先行ゼロは省かれませんが、ホスト変数は固定長または可変長の文字ストリング変数でなければなりません。ホスト変数の長さが時刻の文字表現よりも大きいときは、その文字列は右側にブランクが埋められます。ILE RPG および ILE COBOL では、ホスト変数は時刻変数にも使用できます。

- *USA 形式を使用するときは、ホスト変数の長さは 8 以上でなければなりません。
- *HMS、*ISO、*EUR、または *JIS 形式を使用する場合は、ホスト変数の長さは、秒を含める場合は少なくとも 8 バイトに、時と分だけが必要な場合は 5 バイトにしなければなりません。この場合は、SQLWARN0 および SQLWARN1 (SQLCA の中の) は W にセットされ、標識変数の指定があれば、切り捨てられた実際の秒数にセットされます。

タイム・スタンプがホスト変数に割り当てられるときは、そのタイム・スタンプはその文字表現に変換されます。先行ゼロはどの部分からも省かれませんが、ホスト変数は固定長または可変長の文字ストリング変数であって、その長さは少なくとも 19 バイトでなければなりません。長さが 26 未満のときは、ホスト変数にはマイクロ秒の全桁が収まりません。長さが 26 より大きいときは、ホスト変数は右側にブランクが埋められます。ILE RPG および ILE COBOL では、ホスト変数はタイム・スタンプ変数にも使用できます。

SQL を使用するアプリケーションでの標識変数

標識変数はハーフワードの整数変数であり、関連するホスト変数にヌル値が割り当てられているかどうかを示すために使用されます。

- 結果の列の値がヌルである場合には、SQL は標識変数に -1 を入れます。
- 標識変数を使用していないときに結果列がヌル値の場合には、負の SQLCODE が返されます。
- 結果の列が原因でデータ・マッピング・エラーが起こった場合には、SQL は標識変数を -2 にセットします。

標識変数を使用すると、検索された文字列が切り捨てられたかどうかを確認することができます。切り捨てが行われた場合は、標識変数には文字列の元の長さを指定する正の整数値が入っています。文字列がラージ・オブジェクト (LOB) を表し、その文字列の元の長さが 32767 より長い場合は、標識変数に保管されている値は 32767 になります。これは、32767 より長い値をハーフワードの整数に保管できないからです。

結果列の値がデータベース・マネージャーから返されたときは、標識変数を調べることができます。標識変数の値がゼロより小さければ、結果列の値がヌルであることが分かります。データベース・マネージャーがヌル値を返したときは、ホスト変数は結果列の省略時値にセットされます。

標識変数は、ホスト変数の直後 (前にコロンを付けて) か、キーワード INDICATOR の直後に指定します。以下に、例を示します。

```
EXEC SQL
  SELECT COUNT(*), AVG(SALARY)
  INTO :PLICNT, :PLISAL:INDNULL
  FROM CORPDATA.EMPLOYEE
  WHERE EDLEVEL < 18
END-EXEC.
```

次に INDNULL に負の値が入っているかどうかを検証することができます。負の値が入っていれば、SQL からヌルの値が返されたことが分かります。

列が NULL であるかどうかを検査するときは、必ず **IS NULL** 述部を使用してください。以下に、例を示します。

WHERE expression IS NULL

次の方法で NULL を検査しないでください。

```
MOVE -1 TO HUIND.  
EXEC SQL...WHERE column-name = :HUI :HUIND
```

EQUAL 述部は、ヌル値を比較するときは必ず偽と評価されます。この例の結果では、行は選択されません。

- | ヌル値が存在する可能性のある場合、DISTINCT 述部を使用して比較を実行することができます。詳細については、「SQL 解説書」の『Predicates』を参照してください。

標識変数についての詳細は、以下のトピックを参照してください。

- 『ホスト構造で使われる標識変数』
- 『ヌル値をセットするために使用する標識変数』

ホスト構造で使われる標識変数

ホスト構造をサポートするために**標識構造** (ハーフワードの整数変数の配列として定義されているもの) を指定することもできます。ホスト構造に返される結果列の値がヌルになり得る場合には、ホスト構造名に標識構造名を付加することができます。これにより、SQL は、ホスト構造内のホスト変数にヌル値が返されるたびに、それをユーザーのプログラムに知らせることができます。

次は COBOL で書いた例です。

```
01 SAL-REC.  
  10 MIN-SAL          PIC S9(6)V99 USAGE COMP-3.  
  10 AVG-SAL          PIC S9(6)V99 USAGE COMP-3.  
  10 MAX-SAL          PIC S9(6)V99 USAGE COMP-3.  
01 SALTABLE.  
02 SALIND            PIC S9999 USAGE COMP-4 OCCURS 3 TIMES.  
01 EDUC-LEVEL        PIC S9999 COMP-4.  
...  
  MOVE 20 TO EDUC-LEVEL.  
...  
  EXEC SQL  
    SELECT MIN(SALARY), AVG(SALARY), MAX(SALARY)  
      INTO :SAL-REC:SALIND  
      FROM CORPDATA.EMPLOYEE  
      WHERE EDLEVEL>:EDUC-LEVEL  
  END-EXEC.
```

この例では、SALIND は 3 つの値を含む配列であり、それらの値のおのおのについてそれが負の値であるかどうかをテストすることができます。たとえば、SALIND(1) に負の値が入っていれば、ホスト構造の中の対応するホスト変数 (すなわち MIN-SAL) は、選択された行については変更されません。

上記の例では、SQL は、行の列の値を選択してホスト構造に入れます。したがって、選択されたどの列の値がヌルであるかを判別するためには、ユーザーは対応する構造を標識変数として使用しなければなりません。

ヌル値をセットするために使用する標識変数

標識変数を使用すると、列にヌル値をセットすることができます。UPDATE ステートメントまたは INSERT ステートメントを処理するとき、SQL は標識変数 (存在する場合) を検査します。標識変数に負の値が入っていれば、列の値はヌルにセットされます。-1 より大きい値が入っていれば、それに対応するホスト変数には列の値が入っています。

たとえば、値を列に入れること (INSERT ステートメントまたは UPDATE ステートメントを使用して) を指定できますが、入力データにその値が指定されていたかどうか確かでないことがあります。列にヌル値をセットできるようにするには、次のようなステートメントを書くことができます。

```
EXEC SQL
  UPDATE CORPDATA.EMPLOYEE
    SET PHONENO = :NEWPHONE:PHONEIND
    WHERE EMPNO = :EMPID
END-EXEC.
```

NEWPHONE にヌル値以外の値が入っている場合は、ステートメントの前に次のものを置くことによって PHONEIND をゼロにセットします。

```
MOVE 0 to PHONEIND.
```

そうでない場合は、NEWPHONE にヌル値が入っていることを SQL に伝えるには、次のように PHONEIND を負の値にセットします。

```
MOVE -1 TO PHONEIND.
```

SQLCA を使用した SQL エラー戻りコードの処理

ユーザーのプログラムで SQL ステートメントが処理されると、SQL は SQLCODE フィールドと SQLSTATE フィールドに戻りコードを入れます。戻りコードは、ステートメントの実行が正常に完了したか、失敗したかを示します。SQL がステートメントを処理している途中でエラーを見つけると、SQLCODE は負の値となり、SUBSTR(SQLSTATE,1,2) は '00'、'01'、または '02' のいずれでもなくなりません。SQL がステートメントを処理している途中で例外条件を見つけたものの、それが有効な条件であれば、SQLCODE は正の数となり、SUBSTR(SQLSTATE,1,2) は '01' または '02' となります。SQL ステートメントの処理中にエラー条件も、警告条件も見つからなければ、SQLCODE はゼロになり、SQLSTATE は '00000' になります。

注: ユーザーのプログラムにゼロの SQLCODE が返された場合でも、結果が満足すべきものでないことがあります。たとえば、あるプログラムの実行の結果、ある一部が切り捨てられたとしても、プログラムに返される SQLCODE はゼロです。ただし、SQL 警告標識の 1 つ (SQLWARN1) に、切り捨てが生じたことが示されます。この場合、SQLSTATE は '00000' ではありません。

重要: SQLCODE が負かどうかをテストしない場合や、WHENEVER SQLERROR ステートメントの指定がない場合は、プログラムは次のステートメントに進みます。エラーがあったあと実行を続けると、予期しない結果が生じることがあります。

SQLSTATE の主な目的は、さまざまな IBM リレーショナル・データベース・システム間で共通の戻り条件が発生したとき、共通の戻りコードを出すことです。SQLSTATE は、分散データベース操作で問題を処理するときを使用すると、特に便利です。詳細については、「SQL 解説書」を参照してください。

SQLCA は有用な問題診断ツールであるので、SQLCA に入っている情報の一部を表示するために必要な命令をアプリケーション・プログラムに組み込んでおくと便利です。特に重要なのは、次の SQLCA フィールドです。

SQLCODE	戻りコード。
SQLSTATE	戻りコード。
SQLERRD(3)	SQL により更新、挿入、または削除された行数。
SQLWARN0	W にセットされた場合、SQL 警告フラグ (SQLWARN1 ~ SQLWARNA) の少なくとも 1 つがセットされます。

SQLCA についての詳細は、「SQL 解説書」の『SQL 連絡域』を参照してください。特定の SQLCODE または SQLSTATE を検索するには、SQL メッセージ検索機能を使用してください。DB2 UDB for iSeries の SQLCODE と SQLSTATE の一覧については、「SQL メッセージおよびコード」を参照してください。

SQL 診断域を使用する

SQL 診断域は、プログラム内で実行された SQL ステートメントから戻される情報を保持するために使用されます。ここには、アプリケーション・プログラマーが SQLCA を使って利用できるすべての情報が格納されます。さらに、接続情報など、SQL ステートメントに関するより詳細で役立つ追加情報が提供されます。1 つの SQL ステートメントに対して、複数の条件が戻される可能性があります。次の SQL ステートメントが実行されるまでは、直前の SQL ステートメントが SQL 診断域の情報を利用できます。

診断域の情報にアクセスするには、GET DIAGNOSTICS ステートメントを使用します。このステートメントでは、直前に実行された SQL ステートメントに関する複数の情報を同時に要求できます。各項目はホスト変数として戻されます。また、利用可能なすべての診断情報を含むストリングを取得するよう要求できます。GET DIAGNOSTICS ステートメントを実行しても、診断域は消去されません。

以下のトピックを参照してください。

- 『SQL 診断域を使用するようアプリケーションを更新する』
- 16 ページの『iSeries サーバー・プログラミング・モデル』
- 16 ページの『SQL 診断域を使用するうえでの他の注意事項』
- 16 ページの『例: SQL ルーチン例外』
- 17 ページの『例: SQL 診断域の項目をロギングする』

SQL 診断域を使用するようアプリケーションを更新する

アプリケーション内の SQLCA を置換すべき理由は多数あります。1 つの大きな理由は、SQLCA の SQLERRM フィールドの長さが 70 バイトしかないことです。意味のあるエラー情報を呼び出し側アプリケーションに戻すために、これでは不十分な場合が少なくありません。SQL 診断域の使用を考慮すべき他の理由は、複数行の操作、および列名とオブジェクト名にロング・ネームを使用できることです。136 バイトの制限がある SQLCA では、簡単な警告を報告することさえ難しい場合があります。SQLCA の制限に適合するために、戻されるトークンが切り捨てられることも少なくありません。

現在のアプリケーションは、以下のようにして SQLCA 定義を組み込みます。

```
EXEC SQL INCLUDE SQLCA; /* Existing SQLCA */
```

SQL 診断域を使用するようアプリケーションを変換する場合、まず独立型の SQLSTATE 変数をたとえば以下のように宣言します。

```
char SQLSTATE[6]; /* Stand-alone sqlstate */
```

さらに、独立型の SQLCODE 変数を以下のように宣言することもできます。

```
long int SQLCODE; /* Stand-alone sqlcode */
```

SQL ステートメントの完了状況を検査するには、独立型の SQLSTATE 変数をチェックします。現在の SQL ステートメントの完了時に診断情報を検索する場合、アプリケーションで SQL GET DIAGNOSTICS ステートメントを以下のように実行します。

```
| char hv1[256];
| long int hv2;
|
| EXEC SQL GET DIAGNOSTICS :hv1 = COMMAND_FUNCTION,
| :hv2 = COMMAND_FUNCTION_CODE;
```

| iSeries サーバー・プログラミング・モデル

| iSeries 統合化言語環境 (Integrated Language Environment®) (ILE) では、SQL 診断域がスレッドおよび活
| 動化グループを扱います。つまり、スレッドが SQL ステートメントを実行するそれぞれの活動化グループ
| ごとに、その活動化に関する個別の診断域が存在します。

| SQL 診断域を使用するうえでの他の注意事項

| アプリケーション・プログラムでは、SQLCA を暗黙的な、または独立型の SQLSTATE 変数に置換します
| (この変数は、プログラム内で宣言する必要があります)。SQL 診断域には複数の条件域が存在し、ほとん
| どの重大エラーまたは警告は最初の診断域の中に戻されます。複数の条件の間には特定の順序付けはありま
| せんが、最初の診断域には、SQLSTATE 変数に戻されるのと同じ SQLSTATE に関する情報が含まれま
| す。

| SQLCA の場合、SQL ステートメント実行結果を通信するために使われる SQLCA のストレージは、アプ
| リケーション・プログラムによって提供されます。SQL 診断域の場合、診断情報のストレージはデータベ
| ース・マネージャーによって管理され、診断域の内容を検索するために GET DIAGNOSTICS ステートメ
| ントを使用できます。

| なお、アプリケーション・プログラムでは引き続き SQLCA を使用できることに注意してください。さら
| に、SQLCA を使用するアプリケーション・プログラムで、GET DIAGNOSTICS ステートメントを使用す
| ることもできます。

| 例: SQL ルーチン例外

| 次のアプリケーションの例では、入力値が適切な範囲を超えている場合、ストアード・プロシージャがエ
| ラーを発信します。

```
| EXEC SQL CREATE PROCEDURE check_input (IN p1 INT)
| LANGUAGE SQL READS SQL DATA
| test: BEGIN
|   IF p1 < 0 THEN
|     SIGNAL SQLSTATE VALUE '99999'
|     SET MESSAGE_TEXT = 'Bad input value';
|   END IF
| END test;
```

| 呼び出し側アプリケーションは障害をチェックし、障害に関する情報を SQL 診断域から検索します。

```
| char SQLSTATE[6]; /* Stand-alone sqlstate */
| long int SQLCODE; /* Stand-alone sqlcode */
|
| long int hv1;
| char hv2[6];
| char hv3[256];
|
| hv1 = -1;
| EXEC SQL CALL check_input(:hv1);
|
| if (strncmp(SQLSTATE, "99999", 5) == 0)
| {
|   EXEC SQL GET DIAGNOSTICS CONDITION 1
|   :hv2 = RETURNED_SQLSTATE,
|   :hv3 = MESSAGE_TEXT;
```



```
| }
| else
| {
| }
```

例: SQL 診断域の項目をロギングする

この例では、セキュリティ上の理由で、アプリケーションがすべてのエラーをログに記録する必要があります。ログを使用して、システムの正常性や、データベースの不適切な使用を監視することができます。

発生するそれぞれの SQL ごとに、1 つの項目がログに記録されます。各項目には、エラーが発生した日時、ユーザーが使用していたアプリケーション、実行された SQL ステートメントの種類、戻された SQLSTATE の値、およびメッセージ番号とそれに対応する完全なメッセージ・テキストが含まれます。

それぞれの診断項目のデータ・タイプについては、「SQL 解説書」の『GET DIAGNOSTICS』に関するセクションの表 55 を参照してください。

```
| char stmt_command[256];
| long int error_count;
| long int condition_number;
| char auth_id[256];
| char error_state[6];
| char msgid[128];
| char msgtext[1024];
|
| EXEC SQL WHENEVER SQLERROR GOTO error;
|
| (application code)
|
| error:
| EXEC SQL GET DIAGNOSTICS :stmt_command = COMMAND_FUNCTION,
|                          :error_count = NUMBER;
|
| for (condition_number=1;i<=error_count;++condition_number)
| {
|   EXEC SQL GET DIAGNOSTICS CONDITION :condition_number
|   :auth_id = DB2_AUTHORIZATION_ID,
|   :error_state = RETURNED_SQLSTATE,
|   :msgid = DB2_MESSAGE_ID,
|   :msgtext = DB2_MESSAGE_TEXT;
|
|   EXEC SQL INSERT INTO error_log VALUES(CURRENT_TIMESTAMP,
|   :stmt_command,
|   :condition_number,
|   :auth_id,
|   :error_state,
|   :msgid,
|   :msgtext);
| }
```

WHENEVER ステートメントによる例外条件の処理

WHENEVER ステートメントが現れると、SQL は SQLSTATE と SQLCODE を検査して、ユーザーのプログラムの処理を続けるか、あるいは SQL ステートメントの実行の結果としてエラー、例外、または警告が生じていれば、プログラム内の別の個所に分岐します。例外条件処理サブルーチン (ユーザーのプログラムの一部) は、SQLCODE フィールドまたは SQLSTATE フィールドを調べて、発生したエラーまたは例外条件に見合った処置を取ることができます。

注: WHENEVER ステートメントは REXX プロシージャーでは使用できません。REXX での例外条件の処理については、『第 10 章 REXX アプリケーションでの SQL ステートメントのコーディング方法』を参照してください。

WHENEVER ステートメントを使用すると、一般的な条件に該当するときどのような処置をとるべきか指定することができます。同じ条件について複数の WHENEVER ステートメントを指定できます。その場合は、最初の WHENEVER ステートメントは、別の WHENEVER ステートメントが指定されるまでソース・プログラム内の後続ステートメントすべてに適用されます。

WHENEVER ステートメントは次のようになっています。

```
EXEC SQL  
WHENEVER condition action  
END-EXEC.
```

条件として指定できるのは次の 3 つです。

SQLWARNING SQLWARNING は、SQLWARN0 = W のとき、または SQLCODE が 100 (SUBSTR(SQLSTATE,1,2) = '01') 以外の正の値であるとき、どのような処置をとるかを指示するために指定します。

注: SQLWARN0 がセットされる理由はいくつか考えられます。たとえば、ある列の値がホスト変数に入れられるとき、その一部が切り捨てられたとしても、ユーザーのプログラムはそれをエラーと見なさないことがあります。

SQLERROR SQLERROR は、SQL ステートメントの結果としてコードが返されたとき ((SQLCODE < 0) (SUBSTR(SQLSTATE,1,2) > '02')), どのような処置をとるかを指示するために指定します。

NOT FOUND NOT FOUND は、次のいずれかの理由で +100 の SQLCODE と '02000' の SQLSTATE が返されたとき、どのような処置をとるかを指定するために使用します。

- 単一行の SELECT が出された後、またはカーソルについて最初の FETCH が出された後で、プログラムで指定したデータが存在しない。
- 2 番目以降の FETCH の後で、カーソル選択ステートメントを満たす検索対象行が残っていない。
- UPDATE、DELETE、または INSERT の後で、探索条件を満たす行がない。

処置として指定できる値は次のいずれかです。

CONTINUE この値を指定すると、ユーザーのプログラムは次のステートメントに進みます。

GO TO ラベル これを指定すると、ユーザーのプログラムはプログラム内の別の個所に分岐します。その個所を示すラベルの前にはコロンを付けることができます。WHENEVER ... GO TO ステートメントは次のように指定します。

- COBOL の場合は、セクション名または非修飾段落名でなければなりません。
- PL/I と C の場合は、ラベルです。
- RPG の場合は、TAG のラベルです。

たとえば、カーソルを使用して行を検索しているときに、場合によっては FETCH ステートメントが出されたのに SQL が新たな行を見つけることができないことがあります。このような場合に備えて、WHENEVER NOT FOUND GO TO ... ステートメントを指定しておけば、プログラム内の CLOSE ステートメントが出される個所に SQL を分岐させて、カーソルを正しくクローズすることができます。

注: WHENEVER ステートメントは、別の WHENEVER が現れるまでの間の、後続のすべてのソース SQL ステートメントに影響します。

すなわち、2 つの WHENEVER ステートメントの間にコーディングされているすべての SQL ステートメント (WHENEVER が 1 つしかない場合にはその後のすべてのステートメント) は、プログラムが進む経路とは関係なく、最初の WHENEVER ステートメントの影響を受けます。

したがって、WHENEVER ステートメントは、その影響を受ける最初の SQL ステートメントの前に置かなければなりません。WHENEVER が SQL ステートメントのあとに置かれていると、その SQL ステートメントによってセットされた SQLCODE と SQLSTATE の値に基づく分岐は行われません。ただし、ユーザーのプログラムで SQLCODE または SQLSTATE を直接検査する場合は、その検査は SQL ステートメントの実行後に行う必要があります。

WHENEVER ステートメントには、サブルーチン・オプションに対する CALL を行う働きはありません。そのため、WHENEVER ステートメントを使用するよりも、各 SQL ステートメントの実行後に SQLCODE または SQLSTATE の値を検査してサブルーチンを呼び出すようにした方がよい場合があります。

第 5 章 C および C++ アプリケーションでの SQL ステートメントのコーディング方法

ここでは、SQL ステートメントを C または C++ プログラムに組み込む場合に固有のアプリケーションおよびコーディング上の要件について説明します。C プログラムとは、ILE C for iSeries プログラムを意味します。C++ プログラムとは、ILE C++ プログラムを意味します。さらに、ホスト構造およびホスト変数に関する要件についても説明します。詳細については、以下のセクションを参照してください。

- 『SQL を使用する C および C++ アプリケーションでの SQL 連絡域の定義』
- 23 ページの『SQL を使用する C および C++ アプリケーションでの SQL 記述子域の定義』
- 25 ページの『SQL を使用する C および C++ アプリケーションへの SQL ステートメントの組み込み』
- 27 ページの『SQL を使用する C および C++ アプリケーションでのホスト変数の使用』
- 40 ページの『SQL を使用する C および C++ アプリケーションでのホスト構造の使用』
- 44 ページの『SQL を使用する C および C++ アプリケーションでのホスト構造配列の使用』
- 48 ページの『SQL を使用する C および C++ アプリケーションでのポインター・データ・タイプの使用』
- 49 ページの『SQL を使用する C および C++ アプリケーションでの typedef の使用』
- 50 ページの『SQL を使用する C および C++ アプリケーションでの ILE C コンパイラー 外部ファイル記述の使用』
- 51 ページの『SQL データ・タイプおよび C または C++ データ・タイプの対応関係の判別』
- 54 ページの『SQL を使用する C および C++ アプリケーションでの標識変数の使用』

SQL ステートメントの使い方を示した詳しいサンプル C プログラムは、『第 12 章 DB2 UDB for iSeries ステートメントを使用したサンプル・プログラム』に記載されています。

注: 重要な法律的事柄に関する情報について、214 ページの『コードについての特記事項』をご覧ください。

SQL を使用する C および C++ アプリケーションでの SQL 連絡域の定義

C または C++ プログラムでは、SQLCA を使って組み込み SQL ステートメントの戻される状況をチェックしたり、SQL 診断域を使用して戻される状況をチェックすることができます。詳しくは、『SQL 診断域を使用する』を参照してください。プログラムで SQL ステートメント GET DIAGNOSTICS を使用する場合には、SQLCA は必要ありません。

SQLCA を使用する場合、SQL ステートメントを組み込む C または C++ プログラムは、次のいずれかまたは両方を含んでいなければなりません。

- long SQLCODE として宣言されている SQLCODE 変数
- char SQLSTATE[6] として宣言されている SQLSTATE 変数

または、

- SQLCA (SQLCODE および SQLSTATE 変数が入っている)

SQLCODE 値および SQLSTATE 値は、各 SQL ステートメントが実行された後、データベース・マネージャーによって設定されます。アプリケーションは、SQLCODE 値または SQLSTATE 値を調べて、最後の SQL ステートメントが正しく実行されたかどうかを判定することができます。

SQLCA は、直接または、SQL の INCLUDE ステートメントを使用して、C または C++ プログラムの中にコーディングすることができます。SQL の INCLUDE ステートメントを使用するときは、次のような標準の宣言を含める必要があります。

```
EXEC SQL INCLUDE SQLCA ;
```

標準の宣言には、構造の定義と 'sqlca' という名前の静的データ域が含まれます。

SQLCODE、SQLSTATE、および SQLCA の各変数は、実行可能ステートメントの前に現れなければなりません。宣言の有効範囲には、プログラムの中のすべての SQL ステートメントの有効範囲が含まれていなければなりません。

SQLCA を組み込んだ C または C++ ソース・ステートメントは次のとおりです。

```
#ifndef SQLCODE
struct sqlca {
    unsigned char sqlcaid[8];
    long          sqlcabc;
    long          sqlcode;
    short         sqlerrml;
    unsigned char sqlerrmc[70];
    unsigned char sqlerrrp[8];
    long          sqlerrrd[6];
    unsigned char sqlwarn[11];
    unsigned char sqlstate[5];
};
#define SQLCODE sqlca.sqlcode
#define SQLWARN0 sqlca.sqlwarn[0]
#define SQLWARN1 sqlca.sqlwarn[1]
#define SQLWARN2 sqlca.sqlwarn[2]
#define SQLWARN3 sqlca.sqlwarn[3]
#define SQLWARN4 sqlca.sqlwarn[4]
#define SQLWARN5 sqlca.sqlwarn[5]
#define SQLWARN6 sqlca.sqlwarn[6]
#define SQLWARN7 sqlca.sqlwarn[7]
#define SQLWARN8 sqlca.sqlwarn[8]
#define SQLWARN9 sqlca.sqlwarn[9]
#define SQLWARNA sqlca.sqlwarn[10]
#define SQLSTATE sqlca.sqlstate
#endif
struct sqlca sqlca = {0x0000000000000000};
```

SQLCODE の宣言がプログラムの中にあり、プリコンパイラーが SQLCA を提供している場合、SQLCODE の個所は SQLCADE で置き換えられます。SQLSTATE の宣言がプログラムの中にあり、プリコンパイラーが SQLCA を提供している場合、SQLSTATE の個所は SQLSTOTE によって置き換えられます。

注: SQL エラー・メッセージの多くは、可変長のメッセージ・データを含みます。これらのデータ・フィールド長は、SQLCA の sqlerrmc フィールドの値に組み込まれます。この長さが原因で、C または C++ プログラムからの sqlerrmc の値の印刷が予測不可能な結果となる場合があります。

SQLCA の詳細については、「SQL 解説書」の『付録 B. SQL 連絡域』を参照してください。

SQL を使用する C および C++ アプリケーションでの SQL 記述子域の定義

SQLDA を必要とするステートメントには、次のものがあります。

```
EXECUTE...USING DESCRIPTOR 記述子名
FETCH...USING DESCRIPTOR 記述子名
OPEN...USING DESCRIPTOR 記述子名
DESCRIBE ステートメント名 INTO 記述子名
DESCRIBE TABLE ホスト変数 INTO 記述子名
PREPARE ステートメント名 INTO 記述子名
CALL...USING DESCRIPTOR 記述子名
```

SQLCA と異なり、SQLDA を 2 つ以上プログラムの中に置くことができ、また SQLDA の名前は有効であれば、どの名前でも使えます。SQLDA は、直接または、SQL の INCLUDE ステートメントを使用して、C または C++ プログラムの中にコーディングすることができます。SQL の INCLUDE ステートメントを使用するときは、標準の SQLDA 宣言を組み込む必要があります。

```
EXEC SQL INCLUDE SQLDA;
```

標準の宣言には構造の定義だけが含まれ、その名前は 'sqlda' です。

SQLDA 用として組み込まれる C および C++ 宣言は次のとおりです。

```
#ifndef SQLDASIZE
struct sqlda {
    unsigned char sqldaid[8];
    long sqldabc;
    short sqln;
    short sqld;
    struct sqlvar {
        short sqltype;
        short sqllen;
        unsigned char *sqldata;
        short *sqlind;
        struct sqlname {
            short length;
            unsigned char data[30];
        } sqlname;
    } sqlvar[1];
};
#define SQLDASIZE(n) (sizeof(struct sqlda) + (n-1)* sizeof(struct sqlvar))
#endif
```

INCLUDE SQLDA SQL ステートメントを使用すると、下記のマクロ定義も得られるという利点があります。

```
#define SQLDASIZE(n) (sizeof(struct sqlda) + (n-1)* sizeof(struct sqlvar))
```

このマクロを使用すると、SQLVAR 要素の個数を指定して、SQLDA 用の記憶域を割り振ることが簡単になります。次の例では、20 個の SQLVAR 要素を持つ SQLDA 用の記憶域を割り振るために SQLDASIZE マクロが使用されます。

```
#include <stdlib.h>
EXEC SQL INCLUDE SQLDA;

struct sqlda *mydaptr;
short numvars = 20;
```

```
mydaptr = (struct sqlda *) malloc(SQLDASIZE(numvars));  
mydaptr->sqln = 20;
```

INCLUDE SQLDA ステートメントに含まれる、その他のマクロ定義を以下に示します。

GETSQLDOUBLED(daptr) daptr がポイントする SQLDA が double である場合に 1、そうでない場合に 0 を返します。SQLDA は、SQLDAID フィールドの 7 番目のバイトが '2' に設定されると double となります。

SETSQLDOUBLED(daptr, newvalue)
SQLDAID の 7 番目のバイトを newvalue に設定します。

GETSQLDALONGLLEN(daptr,n)
daptr がポイントする SQLDA の n 番目の項目の長さ属性を返します。これは、SQLDA が double で、n 番目の SQLVAR 項目が LOB データ・タイプを持つ場合のみ使用します。

SETSQLDALONGLLEN(daptr,n,len)
daptr がポイントする SQLDA の SQLLONGLEN フィールドの n 番目の項目を len に設定します。これは、SQLDA が double で、n 番目の SQLVAR 項目が LOB データ・タイプを持つ場合のみ使用します。

GETSQLDALENPTR(daptr,n) daptr がポイントする SQLDA の n 番目の項目のデータの実際の長さへのポインターを返します。SQLDATALEN ポインター・フィールドは、long (4 バイト) 整数へのポインターを返します。SQLDATALEN ポインターがゼロの場合、NULL ポインターが返ります。これは、SQLDA が double の場合のみ使用します。

SETSQLDALENPTR(daptr,n,ptr)
daptr がポイントする SQLDA の n 番目の項目のデータの実際の長さへのポインターを設定します。これは、SQLDA が double の場合のみ使用します。

SQLDA をポインターとして宣言したときは、ポインターとして宣言したホスト変数の場合と全く同じように、それを SQL ステートメントの中で使用するとき宣言したとおりにそれを参照しなければなりません。コンパイラー・エラーを防止するには、SQLDA の sqldata フィールドに割り当てられた値のタイプを、符号なしのポインターにしなければなりません。このように指定すると、コンパイラー・エラーの防止に役立ちます。このようなタイプの指定が必要になるのは、アプリケーション・プログラムがプログラムの中でホスト変数のアドレスを引き渡す時に使用する EXECUTE、OPEN、CALL、および FETCH ステートメントの場合に限られます。たとえば、mydaptr と名付けた SQLDA にポインターを宣言する場合は、PREPARE ステートメントの中で次のように使用します。

```
EXEC SQL PREPARE mysname INTO :*mydaptr FROM :mysqlstring;
```

SQLDA 宣言は、構造の定義が許される場所ならば、どこにでも置くことができます。通常の C 有効範囲規則が適用されます。

動的 SQL は高度なプログラミング技法です。これについては、「DB2 UDB for iSeries SQL プログラミング 概念」の『動的 SQL アプリケーション』で説明されています。動的 SQL を使用すると、ユーザーのプログラムはその実行と平行して SQL ステートメントを作成し、実行させることができます。変数 SELECT リスト (すなわち、照会の一部として返されるデータのリスト) を指定する SELECT ステートメントは、SQL 記述子域 (SQLDA) を動的に必要とします。これは、SELECT の結果を受け入れるために割り振るべき変数の数とタイプが事前に予測できないからです。

SQLDA の詳細については、「SQL 解説書」のトピック『SQL 記述子域』を参照してください。

SQL を使用する C および C++ アプリケーションへの SQL ステートメントの組み込み

SQL ステートメントは、実行可能なステートメントを置くことができる個所であれば、C または C++ プログラム内のどこにでもコーディングできます。

各 SQL ステートメントは EXEC SQL で始まり、セミコロン (;) で終わらなければなりません。EXEC SQL キーワードは 1 行でなければなりません。SQL ステートメントの残りの部分は、2 行以上にまたがっても構いません。

例：C または C++ プログラムでコーディングされた UPDATE ステートメントは、次のようになります。

```
EXEC SQL
  UPDATE DEPARTMENT
  SET MGRNO = :MGR_NUM
  WHERE DEPTNO = :INT_DEPT ;
```

詳細については、以下のセクションを参照してください。

- 『SQL を使用する C および C++ アプリケーションでの注記』
- 『SQL を使用する C および C++ アプリケーションでの SQL ステートメントの継続』
- 26 ページの『SQL を使用する C および C++ アプリケーションへのコードの組み込み』
- 26 ページの『SQL を使用する C および C++ アプリケーションでのマージン』
- 26 ページの『SQL を使用する C および C++ アプリケーションでの名前』
- 27 ページの『SQL を使用する C および C++ アプリケーションでの NULL および NUL』
- 27 ページの『SQL を使用する C および C++ アプリケーションでのステートメント・ラベル』
- 27 ページの『SQL を使用する C および C++ アプリケーションでのプリプロセッサ順序』
- 27 ページの『SQL を使用する C および C++ アプリケーションでの 3 文字表記』
- 27 ページの『SQL を使用する C および C++ アプリケーションでの WHENEVER ステートメント』

SQL を使用する C および C++ アプリケーションでの注記

SQL の注記 (--) を使用する他に、ブランクを入れることができる場合はいつでも組み込み SQL ステートメントの中に C の注記 (/* ...*/) を入れることができます。ただし、キーワードの EXEC と SQL の間には入れられません。注記は何行にもまたがることができます。注記をネストすることはできません。C++ では単一行注記 (// で開始する注記) を使用できますが、C では使用できません。

SQL を使用する C および C++ アプリケーションでの SQL ステートメントの継続

SQL ステートメントは 1 行または 2 行以上に継続することができます。ブランクを置くことができるならば、どこでも SQL ステートメントを分割することができます。円記号 (¥) を使用すると、文字列定数または区切り文字付き識別コードを継続することができます。区切り文字が付いていない識別コードは、継続することができません。

DBCS データを含む定数を複数の行にわたって継続する方法としては、次の 2 とおりが可能です。

- 継続された行の右マージンにある文字がシフトイン文字で、継続行の左マージンにある文字がシフトアウト文字の場合には、左右のマージンにあるシフト文字が除去されます。

この SQL ステートメントの G'<AABBCCDDEEFFGGHHIIJJKK>' はグラフィック定数として有効です。マージンの余分なシフトは除去されます。

```
*...+...1...+...2...+...3...+...4...+...5...+...6...+...7...*...8
EXEC SQL SELECT * FROM GRAPHTAB          WHERE GRAPHCOL = G'<AABBCCDDEEFFGGHH>
<IIJJKK>';
```

- マージンの外側にシフト文字を置くことができます。下記の例では、マージンが 5 と 75 であるとし、この SQL ステートメントの G'<AABBCCDDEEFFGGHHIIJJKK>' はグラフィック定数として有効です。

```
*...(1...+...2...+...3...+...4...+...5...+...6...+...7...)...8
EXEC SQL SELECT * FROM GRAPHTAB          WHERE GRAPHCOL = G'<AABBCCDD>
<EEFFGGHHIIJJKK>';
```

SQL を使用する C および C++ アプリケーションへのコードの組み込み

SQL ステートメント、C ステートメント、または C++ ステートメントは、ソース・コード内に次の SQL ステートメントを組み込むことによって組み入れることができます。

```
EXEC SQL INCLUDE member-name;
```

C または C++ の #include ステートメントは、SQL ステートメントを組み込んだり、SQL ステートメントの中で参照される C または C++ のホスト変数の宣言を組み込んだりするためには使用できません。

SQL を使用する C および C++ アプリケーションでのマージン

SQL ステートメントは、CRTSQLCI、または CRTSQLCPPI コマンドの MARGINS パラメーターで指定したマージンの範囲内にコーディングしなければなりません。MARGINS パラメーターが *SRCFILE として指定される場合は、ソース・ファイルのレコード長が使用されます。右マージンについて値が指定され、この値がソース・レコード長よりも大きい場合は、レコード全体が読み取られます。この値は任意の組み込みメンバーにも適用されます。たとえば、右マージンが 200 に指定され、ソース・ファイルのレコード長が 80 である場合、データの 80 列までだけがソース・ファイルから読み取られます。同じプリコンパイルに組み込まれるソース・メンバーのレコード長が 200 である場合は、組み込まれたソース・メンバー 200 がすべて読み取られます。

EXEC SQL が指定のマージン内で始まっていないときは、SQL プリコンパイラーはそれを SQL ステートメントと認識しません。CRTSQLCI または CRTSQLCPPI の詳細については、『第 13 章 ホスト言語プリコンパイラー用の DB2 UDB for iSeries CL コマンドの説明』を参照してください。

SQL を使用する C および C++ アプリケーションでの名前

ホスト変数には、任意の有効な C または C++ 変数名を使用することができます。この変数は、次の制限に従ってなければなりません。

'SQL'、'RDI'、または 'DSN' で始まるホスト変数名や外部入り口名は、大文字や小文字をどのように組み合わせても、使用してはなりません。これらの名前はデータベース・マネージャー用に予約されています。ホスト変数名の長さは 128 までです。

SQL を使用する C および C++ アプリケーションでの NULL および NUL

C、C++ および SQL では、ヌル値という語を使用していますが、意味が異なります。C および C++ 言語には、ヌル文字 (NUL)、ヌル・ポインター (NULL)、およびヌル・ステートメント (セミコロンだけ) があります。C の NUL は、0 に相当する単一文字です。C の NULL は特殊な予備のポインター値であり、有効などのデータ・オブジェクトも指していません。SQL のヌル値は特殊な値であり、これはすべての非ヌル値と区別され、(非ヌル) 値の不在を示します。

SQL を使用する C および C++ アプリケーションでのステートメント・ラベル

実行可能な SQL ステートメントの前には、ラベルを付けることができます。

SQL を使用する C および C++ アプリケーションでのプリプロセッサ順序

C または C++ プリプロセッサの前に、SQL プリプロセッサを実行しなければなりません。SQL ステートメント内で C または C++ プリプロセッサ・ディレクティブを使用することはできません。

SQL を使用する C および C++ アプリケーションでの 3 文字表記

C および C++ 文字セットの文字の中には、一部のキーボードに付いていないものがあります。これらの文字は、3 文字表記 と呼ばれる 3 文字の列を使用して、C または C++ ソース・プログラムの中に入れることができます。次の 3 文字表記の列は、ホスト変数宣言の中で使用できます。

- ??(左大括弧
- ??) 右大括弧
- ??< 左中括弧
- ??> 右中括弧
- ??= ポンド
- ??/ バックスラッシュ

SQL を使用する C および C++ アプリケーションでの WHENEVER ステートメント

SQL の WHENEVER ステートメントの中の GOTO 文節の対象となるものは、WHENEVER ステートメントの影響が及ぶ SQL ステートメントの有効範囲内になければなりません。

SQL を使用する C および C++ アプリケーションでのホスト変数の使用

SQL ステートメントの中で使用するホスト変数はいずれも明示的に宣言しなければなりません。SQL ステートメントの中で使用するホスト変数は、SQL ステートメントの中でホスト変数を初めて使用する前に、宣言しておかなければなりません。

C では、ホスト変数を定義するために使用される C ステートメントは、その前に BEGIN DECLARE SECTION ステートメントを置き、その後には END DECLARE SECTION ステートメントを置く必要があります。BEGIN DECLARE SECTION と END DECLARE SECTION を指定する場合、SQL ステートメントで使用するすべてのホスト変数宣言は、BEGIN DECLARE SECTION ステートメントと END DECLARE SECTION ステートメントの間になければなりません。typedef 識別コードを使用して宣言され

たホスト変数でも BEGIN DECLARE SECTION と END DECLARE SECTION が必要ですが、typedef 宣言がこの 2 つのセクションの間にある必要はありません。

C++ では、ホスト変数を定義するために使用される C++ ステートメントは、その前に BEGIN DECLARE SECTION ステートメントを置き、その後で END DECLARE SECTION ステートメントを置く必要があります。BEGIN DECLARE SECTION ステートメントと END DECLARE SECTION ステートメントの間には変数は、ホスト変数として使用することはできません。

SQL ステートメントの中のホスト変数はいずれも、その前にコロン (:) を付けなければなりません。

ホスト変数の名前は、ホスト変数がそれぞれ別のブロックやプロシージャの中にある場合であっても、1 つのプログラム内では固有になっていなければなりません。

ホスト変数を使用する SQL ステートメントは、その変数が宣言されたステートメントの有効範囲内になければなりません。

ホスト変数を結合要素にすることはできません。

- 1 ホスト変数の名前には、継続文字を含めることができません。

詳細については、『SQL を使用する C および C++ アプリケーションでのホスト変数の宣言』を参照してください。

SQL を使用する C および C++ アプリケーションでのホスト変数の宣言

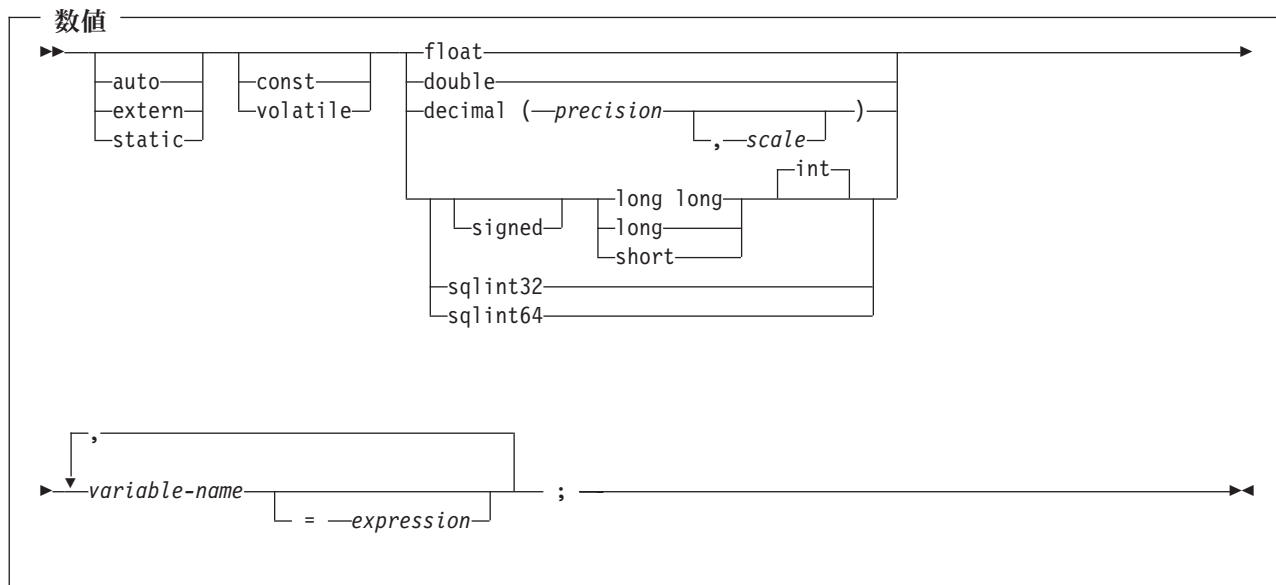
C および C++ プリコンパイラーは、有効な C および C++ 宣言のサブセットだけを有効なホスト変数宣言として認識します。

以下を参照してください。

- 『SQL を使用する C および C++ アプリケーションでのホスト変数の数値』
- 29 ページの『SQL を使用する C および C++ アプリケーションでのホスト変数の文字』
- 33 ページの『SQL を使用する C および C++ アプリケーションでのグラフィック・ホスト変数』
- 35 ページの『SQL を使用する C および C++ アプリケーションでのバイナリー・ホスト変数』
- 39 ページの『SQL を使用する C および C++ アプリケーションでの ROWID ホスト変数』
- 36 ページの『SQL を使用する C および C++ アプリケーションでの LOB ホスト変数』

SQL を使用する C および C++ アプリケーションでのホスト変数の数値

下図は、有効な数値ホスト変数宣言の構文を示しています。



注:

1. precision (精度) および scale (位取り) は整数定数でなければなりません。precision の範囲は 1 から 63 までです。scale の範囲は 0 から精度の値までです。
2. 10 進データ・タイプを使用している場合は、見出しファイルの decimal.h を取り込む必要があります。
3. sqlint32 または sqlint64 を使用する場合は、ヘッダー・ファイルの sqlsystem.h が組み込まれている必要があります。

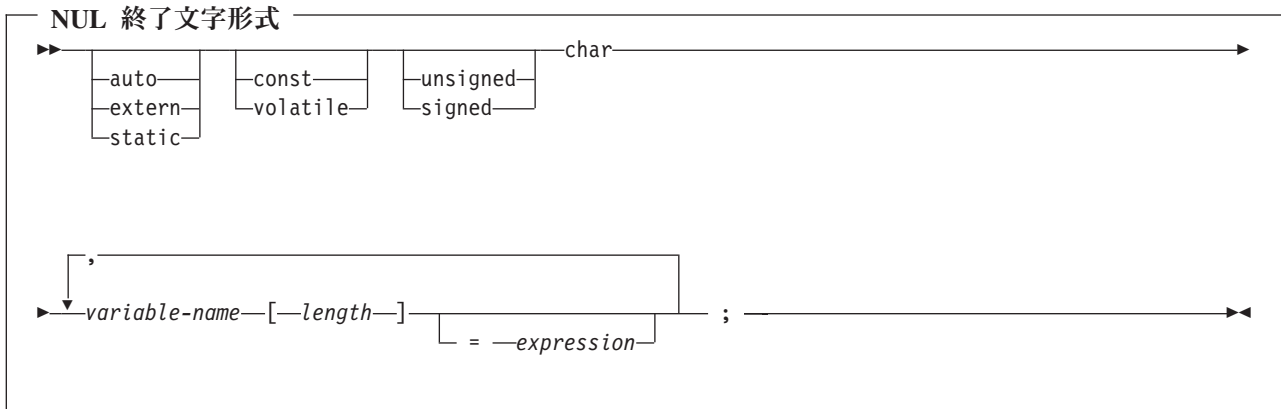
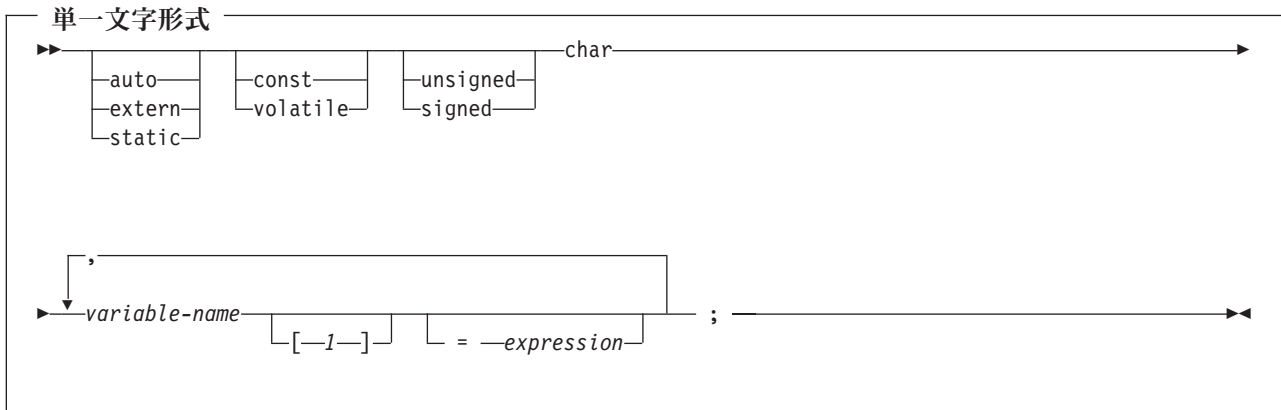
SQL を使用する C および C++ アプリケーションでのホスト変数の文字

文字ホスト変数には、有効な次の 3 つの形式があります。

- 単一文字形式
- NUL 終了文字形式
- VARCHAR 構造化形式

さらに、SQL VARCHAR 宣言は varchar ホスト変数を定義するために使用できます。

文字タイプはすべて符号なしとして扱われます。



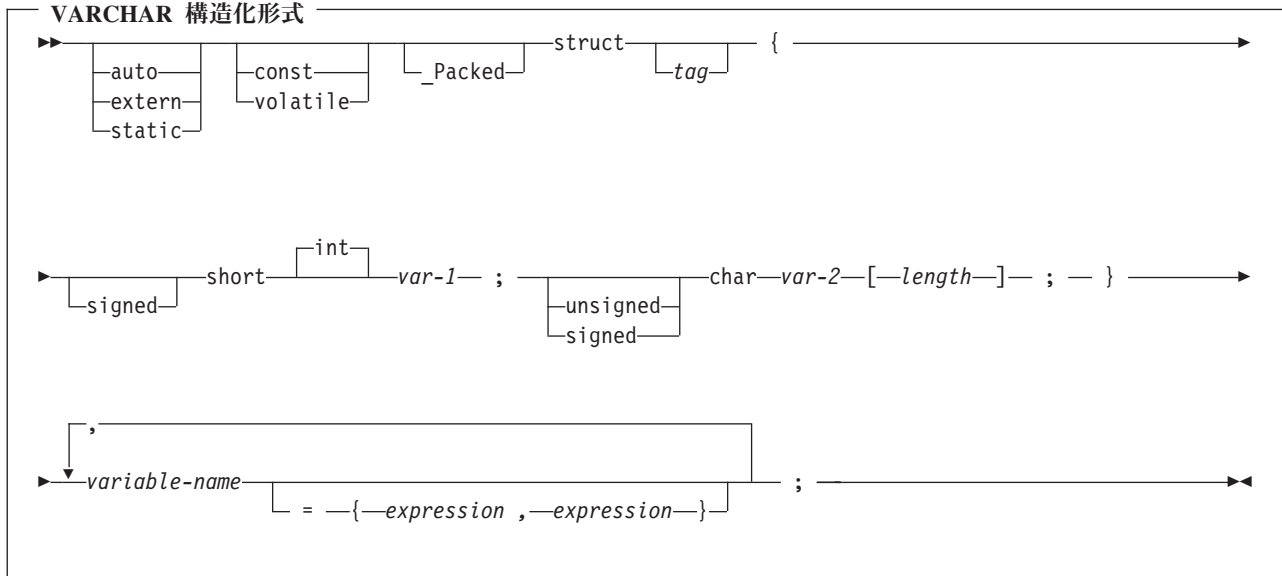
注:

- length (長さ) は、1 より大きく、32741 以下の整数定数でなければなりません。
- CRTSQLCI、または CRTSQLCPPI コマンドで *CNULRQD オプションの指定がある場合は、入力ホスト変数に NUL 終了文字を入れる必要があります。出力ホスト変数は空白で埋め込まれ、最後の文字が NUL 終了文字になります。出力ホスト変数がデータと NUL 終了文字がともに入るだけの大きさになっていないと、次の処置がとられます。
 - データが切り捨てられます。
 - 最後の文字は NUL 終了文字となります。
 - SQLWARN1 は 'W' にセットされます。
- CRTSQLCI または CRTSQLCPPI コマンドで *NOCNULRQD オプションの指定がある場合は、入力変数に NUL 終了文字を入れる必要はありません。

出力ホスト変数には以下が適用されます。

- ホスト変数がデータと NUL 終了文字がともに入る大きさである場合、次の処置がとられます。
 - データは返されますが、空白での埋め込みは行われません。
 - データのすぐ後に NUL 終了文字が入ります。
- ホスト変数がデータが入る大きさであるが、NUL 終了文字が入る大きさではない場合、次の処置がとられます。
 - データが返されます。

- NUL 終了文字は返されません。
- SQLWARN1 は 'N' にセットされます。
- ホスト変数がデータが入る大きさでない場合、次の処置がとられます。
 - データが切り捨てられます。
 - NUL 終了文字は返されません。
 - SQLWARN1 は 'W' にセットされます。



注:

1. *length* (長さ) は、0 より大きく、32740 以下の整数定数でなければなりません。
2. *var-1* (変数 1) と *var-2* (変数 2) は、単純変数参照にしなければならず、整数ホスト変数および文字ホスト変数として個々に使用することはできません。
3. *struct* (構造) タグを使用すると、他のデータ域が定義できますが、これらをホスト変数として使用することはできません。
4. VARCHAR 構造化形式は、ヌル値を含む場合のあるビット・データについて使用する必要があります。VARCHAR 構造化形式は、ヌル終了文字を使用して終了されることはありません。
5. *_Packed* は、C++ では使用してはいけません。代わりに、宣言の前に `#pragma pack(1)` を、宣言の後に `#pragma pack()` を指定します。

注: `#pragma pack (reset)` を `#pragma pack()` の代わりに使用できます。これらは同じものです。

```

#pragma pack(1)
struct VARCHAR {
    short len;
    char s[10];
} vstring;
#pragma pack()
  
```

例:

```
EXEC SQL BEGIN DECLARE SECTION;

/* valid declaration of host variable vstring */

struct VARCHAR {
    short len;
    char s[10];
} vstring;

/* invalid declaration of host variable wstring */

struct VARCHAR wstring;
```

SQL VARCHAR 形式

The diagram illustrates the SQL VARCHAR syntax. It shows the sequence: `VARCHAR`, a space, `variable-name`, a space, an optional `[-length-]` in brackets, a space, an optional `[= 'init-data']` in brackets, and a semicolon `;`. Arrows indicate the flow from left to right. A vertical line with a downward arrow points from the `variable-name` to the `[-length-]` part, and another vertical line with a downward arrow points from the `[-length-]` to the `[= 'init-data']` part.

注:

1. VARCHAR は、大文字小文字混合にすることができます。
2. length (長さ) は、0 より大きく、32740 以下の整数定数でなければなりません。
3. SQL VARCHAR 形式は、NULL 値を含む場合のあるビット・データについて使用する必要があります。SQL VARCHAR 形式は、ヌル終了文字を使用して終了されることはありません。

例:

次のように宣言すると、

```
VARCHAR vstring[528]="mydata";
```

以下の構造を生成します。

```
_Packed struct { short len;
                  char data[528];}
vstring={6, "mydata"};
```

次のように宣言すると、

```
VARCHAR vstring1[111],
        vstring2[222]="mydata",
        vstring3[333]="more data";
```

以下の構造を生成します。

```
_Packed struct { short len;
                  char data[111];}
vstring1;

_Packed struct { short len;
                  char data[222];}
vstring2={6,"mydata"};
```

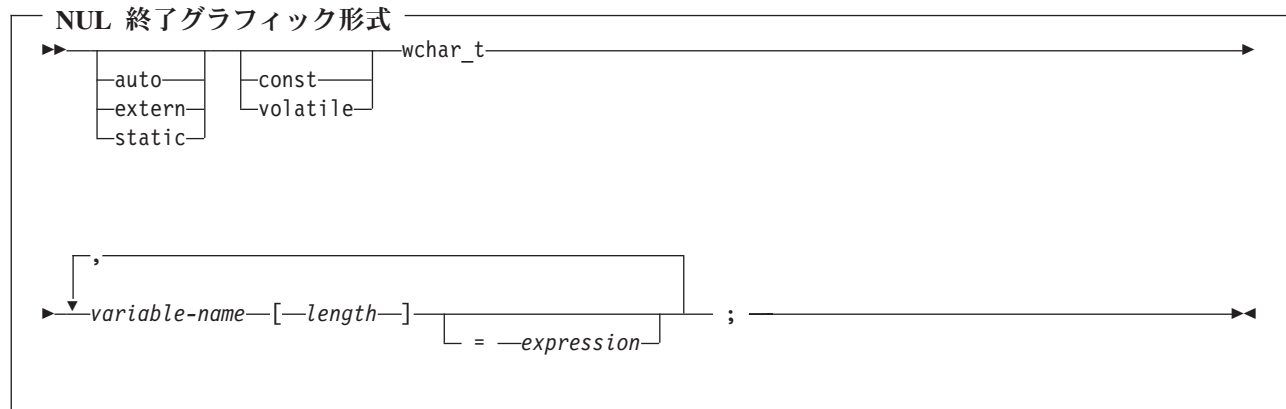
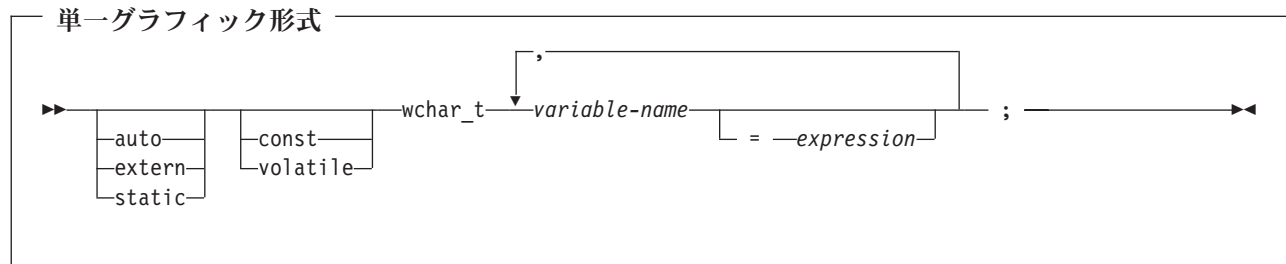


```
_Packed struct { short len;
                  char data[333];}
vstring3={9,"more data"};
```

SQL を使用する C および C++ アプリケーションでのグラフィック・ホスト変数

グラフィック・ホスト変数の有効な形式としては、次の 3 つがあります。

- 単一グラフィック形式
- NUL 終了グラフィック形式
- VARGRAPHIC 構造化形式

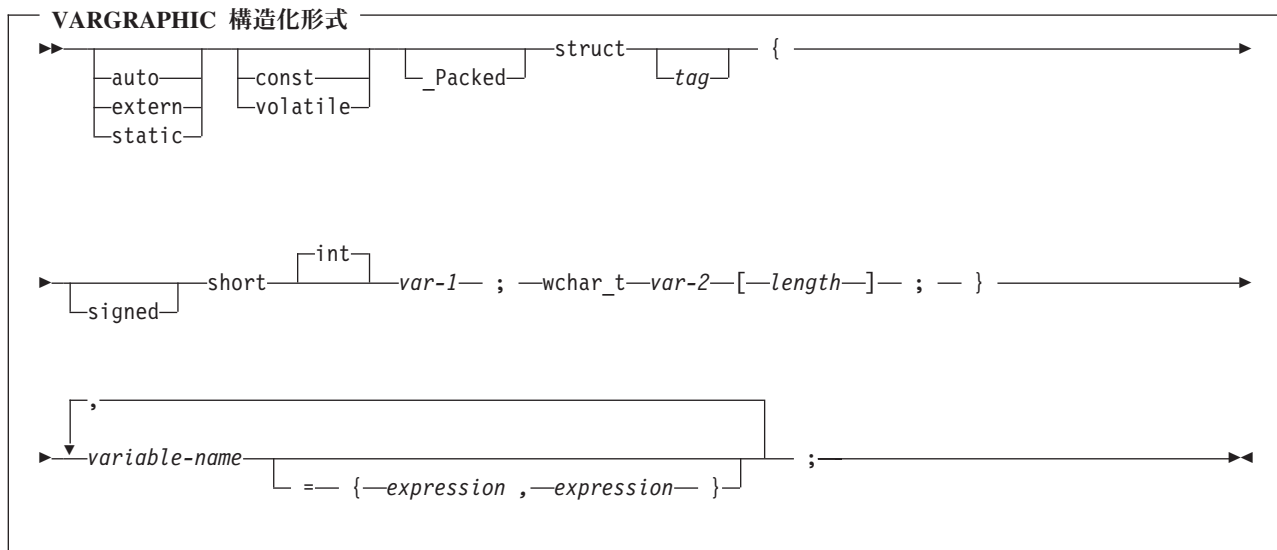


注:

1. `length` (長さ) は、1 より大きく、16371 以下の整数定数でなければなりません。
2. `CRTSQLCI` または `CRTSQPCPI` コマンドで `*CNULRQD` オプションの指定がある場合は、入力ホスト変数にグラフィック NUL 終了文字 (`/0/0`) を入れる必要があります。出力ホスト変数は `DBCS` ブランクで埋め込まれ、最後の文字がグラフィック NUL 終了文字になります。出力ホスト変数がデータと NUL 終了文字がともに入るだけの大きさになっていないと、次の処置がとられます。
 - データが切り捨てられます。
 - 最後の文字はグラフィック NUL 終了文字になります。
 - `SQLWARN1` は 'W' にセットされます。

`CRTSQLCI` または `CRTSQPCPI` コマンドで `*NOCNULRQD` オプションの指定がある場合は、入力ホスト変数にグラフィック NUL 終了文字を入れる必要はありません。出力ホスト変数の処理は以下のとおりです。

- ホスト変数がデータとグラフィック NUL 終了文字がともに入る大きさである場合、次の処置がとられます。
 - データは返されますが、DBCS ブランクでの埋め込みは行われません。
 - データのすぐ後に、グラフィック NUL 終了文字が入ります。
- ホスト変数がデータが入る大きさであるが、グラフィック NUL 終了文字が入る大きさではない場合、次の処置がとられます。
 - データが返されます。
 - グラフィック NUL 終了文字は返されません。
 - SQLWARN1 は 'N' にセットされます。
- ホスト変数がデータが入る大きさでない場合、次の処置がとられます。
 - データが切り捨てられます。
 - グラフィック NUL 終了文字は返されません。
 - SQLWARN1 は 'W' にセットされます。



注:

1. *length* (長さ) は、0 より大きく、16370 以下の整数定数でなければなりません。
2. *var-1* (変数 1) と *var-2* (変数 2) は、単純変数参照にしなければならず、ホスト変数として使用できません。
3. **struct** (構造) タグを使用すると、他のデータ域が定義できますが、これらをホスト変数として使用することはできません。
4. **_Packed** は、C++ では使用してはいけません。代わりに、宣言の前に `#pragma pack(1)` を、宣言の後に `#pragma pack()` を指定します。

```

#pragma pack(1)
struct VARGRAPH {
    short len;
    wchar_t s[10];
} vstring;
#pragma pack()

```

例:

```
EXEC SQL BEGIN DECLARE SECTION;
```

```
/* valid declaration of host variable graphic string */
```

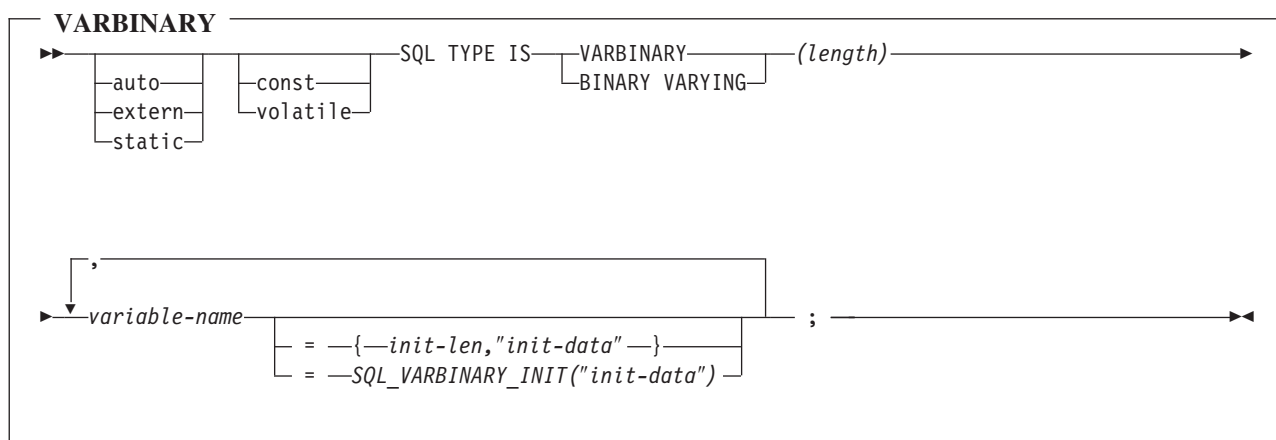
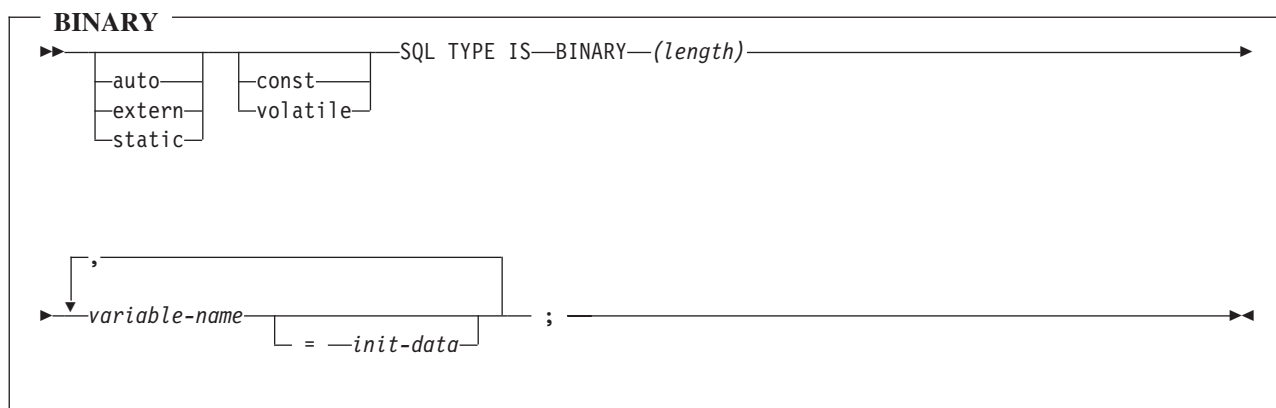
```
struct VARGRAPH {  
    short len;  
    wchar_t s[10];  
} vstring;
```

```
/* invalid declaration of host variable wstring */
```

```
struct VARGRAPH wstring;
```

SQL を使用する C および C++ アプリケーションでのバイナリー・ホスト変数

C および C++ には、SQL バイナリー・データ・タイプに対応する変数がありません。これらのデータ・タイプで使用するホスト変数を作成するには、SQL TYPE IS 文節を使用します。SQL プリコンパイラーは、この宣言を出力ソース・メンバー内で、C 言語構造に置き換えます。



注:

1. BINARY ホスト変数では、length (長さ) の範囲は 1 から 32766 まででなければなりません。
2. VARBINARY および BINARY VARYING ホスト変数では、length (長さ) の範囲は 1 から 32740 まででなければなりません。

| 3. SQL TYPE IS、BINARY、VARBINARY、および BINARY VARYING は大文字小文字混合にすることが
| できます。

| BINARY の例

| 次のように宣言すると、

| SQL TYPE IS BINARY(4) myBinField;

| 以下のような結果コードが生成されます。

| unsigned char myBinField[4];

| VARBINARY の例

| 次のように宣言すると、

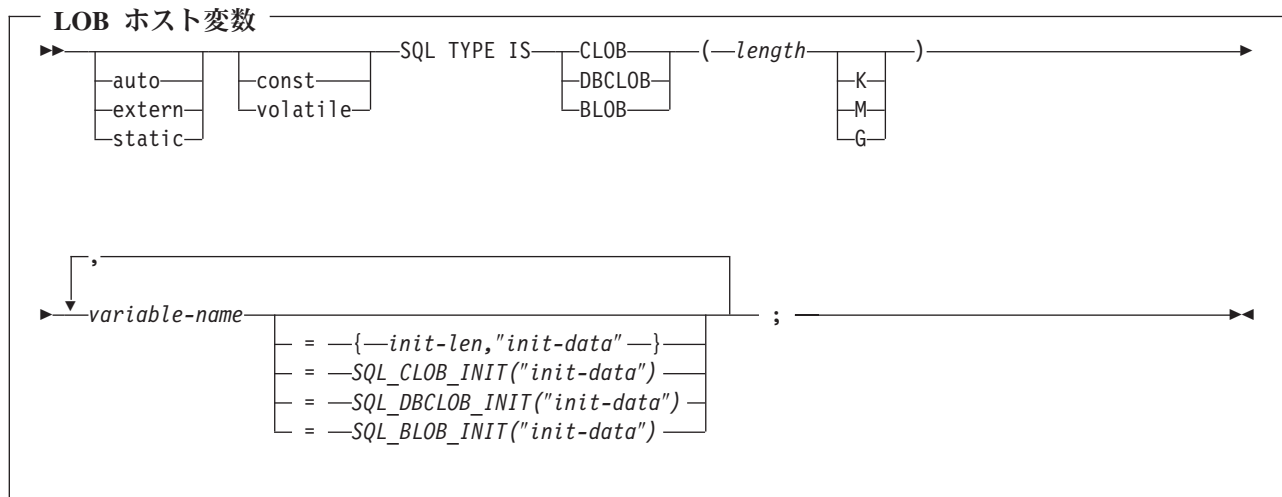
| SQL TYPE IS VARBINARY(12) myVarBinField;

| 以下の構造を生成します。

```
| _Packed struct myVarBinField_t {  
|   short length;  
|   char data[12]; }  
| myVarBinField;
```

| SQL を使用する C および C++ アプリケーションでの LOB ホスト変数

C および C++ には、LOB (ラージ・オブジェクト) の SQL データ・タイプに対応する変数があります。これらのデータ・タイプで使用するホスト変数を作成するには、SQL TYPE IS 文節を使用します。SQL プリコンパイラは、この宣言を出力ソース・メンバー内で、C 言語構造に置き換えます。



注:

1. K は length に 1024 を掛けます。M は length に 1,048,576 を掛けます。G は length に 1,073,741,824 を掛けます。
2. BLOB および CLOB の場合、 $1 \leq length \leq 2,147,483,647$ です。
3. DBCLOB の場合、 $1 \leq length \leq 1,073,741,823$ です。
4. SQL TYPE IS、BLOB、CLOB、DBCLOB、K、M、G は大文字小文字混合で構いません。
5. 初期設定文字列に使用できる最大長は 32,766 バイトです。

6. 初期設定の長さ *init-len* は、数値定数でなければなりません (すなわち、K、M、または G を含むことはできません)。
7. LOB が宣言内で初期設定されない場合は、初期設定は、プリコンパイラーが生成したコード内で行われません。
8. プリコンパイラーは、ホスト変数のタイプにキャストするために使用される構造体タグを生成します。
9. LOB ホスト変数へのポインターは、他のホスト変数タイプへのポインターの場合と同じ規則と制約事項を使用して宣言することができます。
10. LOB ホスト変数に対する CCSID 処理は、他の文字およびグラフィック・ホスト変数タイプに対する処理と同じです。
11. DBCLOB が初期設定されている場合は、文字ストリングの前に 'L' (ワイド文字ストリングを示す) を付けるのはユーザーの役割です。

CLOB の例

次のように宣言すると、

```
SQL TYPE IS CLOB(128K) var1, var2 = {10, "data2data2"};
```

プリコンパイラーは C 言語用に以下を生成します。

```
_Packed struct var1_t {  
    unsigned long length;  
    char data[131072];  
} var1,var2={10,"data2data2"};
```

DBCLOB の例

次のように宣言すると、

```
SQL TYPE IS DBCLOB(128K) my_dbclob;
```

プリコンパイラーは以下を生成します。

```
_Packed struct my_dbclob_t {  
    unsigned long length;  
    wchar_t data[131072]; } my_dbclob;
```

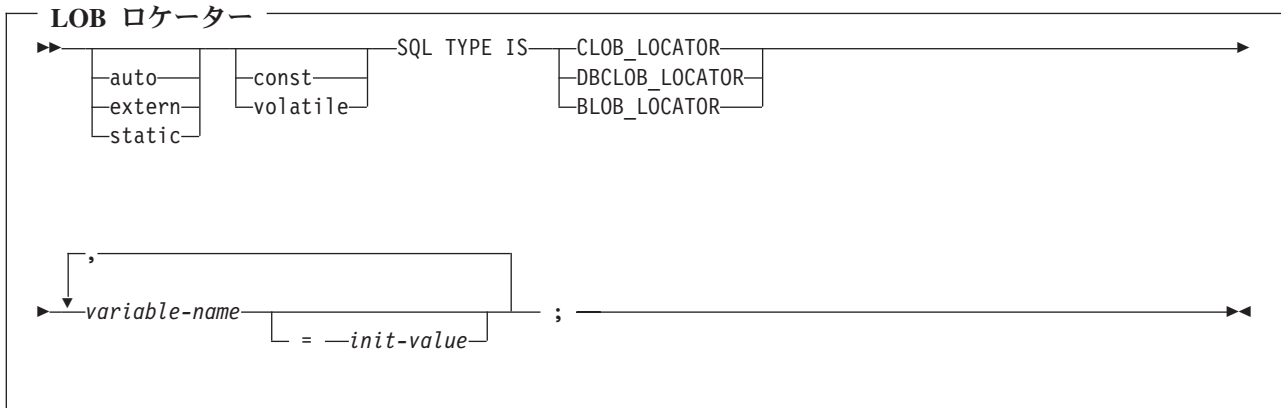
BLOB の例

次のように宣言すると、

```
static SQL TYPE IS BLOB(128K)  
    my_blob=SQL_BLOB_INIT("mydata");
```

以下の構造を生成します。

```
static struct my_blob_t {  
    unsigned long length;  
    char data[131072];  
} my_blob=SQL_BLOB_INIT("my_data");
```



注:

1. SQL TYPE IS、BLOB_LOCATOR、CLOB_LOCATOR、DBCLOB_LOCATOR は、大文字小文字混合にすることができます。
2. *init-value* により、ポインタ・ロケータ変数の初期設定が行えます。他のタイプの初期設定は意味がありません。
3. LOB ロケータへのポインタは、他のホスト変数タイプへのポインタの場合と同じ規則と制約事項を使用して宣言することができます。

CLOB ロケータの例

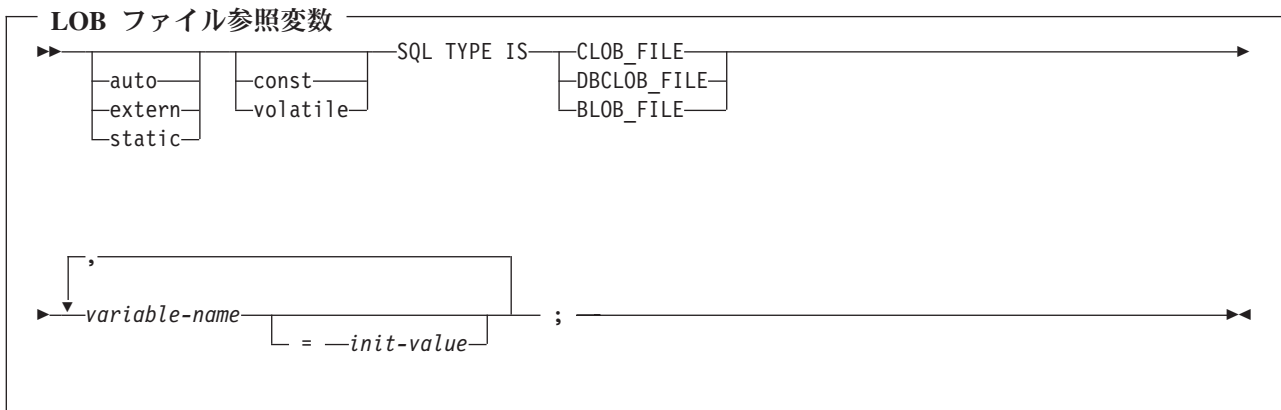
次のように宣言すると、

```
static SQL TYPE IS CLOB_LOCATOR my_locator;
```

以下の構造を生成します。

```
static long int unsigned my_locator;
```

BLOB ロケータおよび DBCLOB ロケータの構文は、似ています。



注:

1. SQL TYPE IS、BLOB_FILE、CLOB_FILE、DBCLOB_FILE は大文字小文字混合で構いません。

2. LOB ファイル参照変数へのポインターは、他のホスト変数タイプへのポインターの場合と同じ規則と制約事項を使用して宣言することができます。

CLOB ファイル参照の例

次のように宣言すると、

```
static SQL TYPE IS CLOB_FILE my_file;
```

以下の構造を生成します。

```
static _Packed struct {
    unsigned long    name_length;
    unsigned long    data_length;
    unsigned long    file_options;
    char             name[255];
} my_file;
```

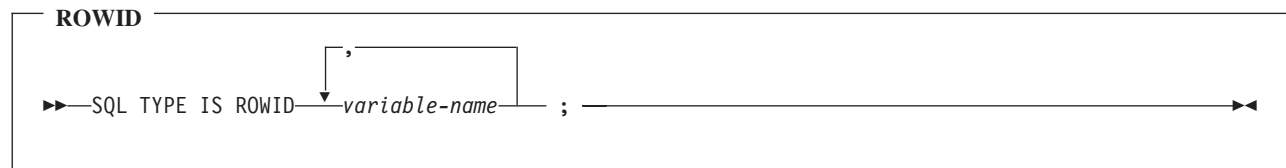
BLOB ファイル参照変数と DBCLOB ファイル参照変数は、類似の構文をもっています。

プリコンパイラーは、次のファイル・オプション定数に対する宣言を生成します。ファイル参照ホスト変数を使用する場合、これらの定数を使用して、file_options 変数を設定できます。これらの値の詳細については、「SQL プログラミング 概念」の『LOB ファイル参照変数』を参照してください。

- SQL_FILE_READ (2)
- SQL_FILE_CREATE (8)
- SQL_FILE_OVERWRITE (16)
- SQL_FILE_APPEND (32)

SQL を使用する C および C++ アプリケーションでの ROWID ホスト変数

C および C++ には、SQL データ・タイプ ROWID に対応する変数がありません。このデータ・タイプで使用するホスト変数を作成するには、SQL TYPE IS 文節を使用します。SQL プリコンパイラーは、この宣言を出力ソース・メンバー内で、C 言語構文に置き換えます。



注:

1. SQL TYPE IS ROWID は、大文字小文字混合にすることができます。

ROWID の例

次のように宣言すると、

```
SQL TYPE IS ROWID myrowid, myrowid2;
```

以下の構造を生成します。

```
_Packed struct { short len;
                  char data[40];}
myrowid1, myrowid2;
```

SQL を使用する C および C++ アプリケーションでのホスト構造の使用

C および C++ プログラムの中では、ホスト構造が定義できます。これは一連の基本 C または C++ 変数に名前を付けたものです。ホスト構造はそれ自体が複数レベルの構造の中に置かれることがあっても、その最大レベルは 2 レベルまでです。可変長の文字列を宣言するときは、別の構造が必要になるので、この場合だけは例外です。

ホスト構造名は、その従属レベルに基本 C または C++ 変数の名前が指定されているグループ名にすることができます。以下に、例を示します。

```
struct {
    struct {
        char c1;
        char c2;
    } b_st;
} a_st;
```

この例で、b_st は基本項目 c1 と c2 から成るホスト構造の名前です。

構造名は、スカラー・リストを簡略に表記するために使用できますが、これは 2 レベルの構造の場合に限られます。ホスト変数は構造名で修飾することができます (たとえば、structure.field)。ホスト構造は 2 レベルに限定されます。(たとえば、上記のホスト構造の例では、SQL の中で a_st を参照することはできません。) 構造に中間レベルの構造を含めることはできません。上記の例では、a_st はホスト変数として使用することも、SQL ステートメントの中で参照することもできません。SQL データのホスト構造は一連のホスト変数に名前を付けたものと考えられ、レベルは 2 つあります。ホスト構造を定義しておけば、SQL ステートメントの中でいくつかのホスト変数 (ホスト構造を構成するホスト変数の名前) を個別に参照せずに一括して参照することができます。

たとえば、次のようにコーディングすれば、テーブル CORPDATA.EMPLOYEE から選択した行のすべての列の値を検索することができます。

```
struct { char empno[7];
        struct { short int firstname_len;
                char firstname_text[12];
        } firstname;
        char midint;
        struct { short int lastname_len;
                char lastname_text[15];
        } lastname;
        char workdept[4];
    } pemp1;
.....
strcpy("000220",pemp1.empno);
.....
exec sql
  SELECT *
  INTO :pemp1
  FROM corpdata.employee
  WHERE empno=:pemp1.empno;
```

上の例に示すように、pemp1 の宣言には、2 つの可変長文字列要素、firstname と lastname が構造に含まれています。

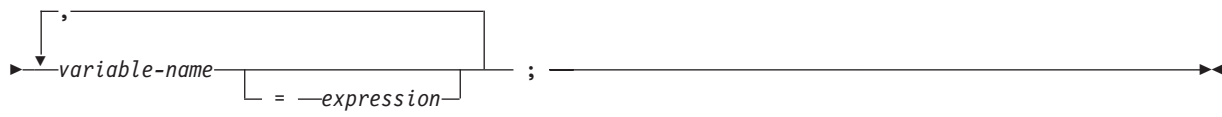
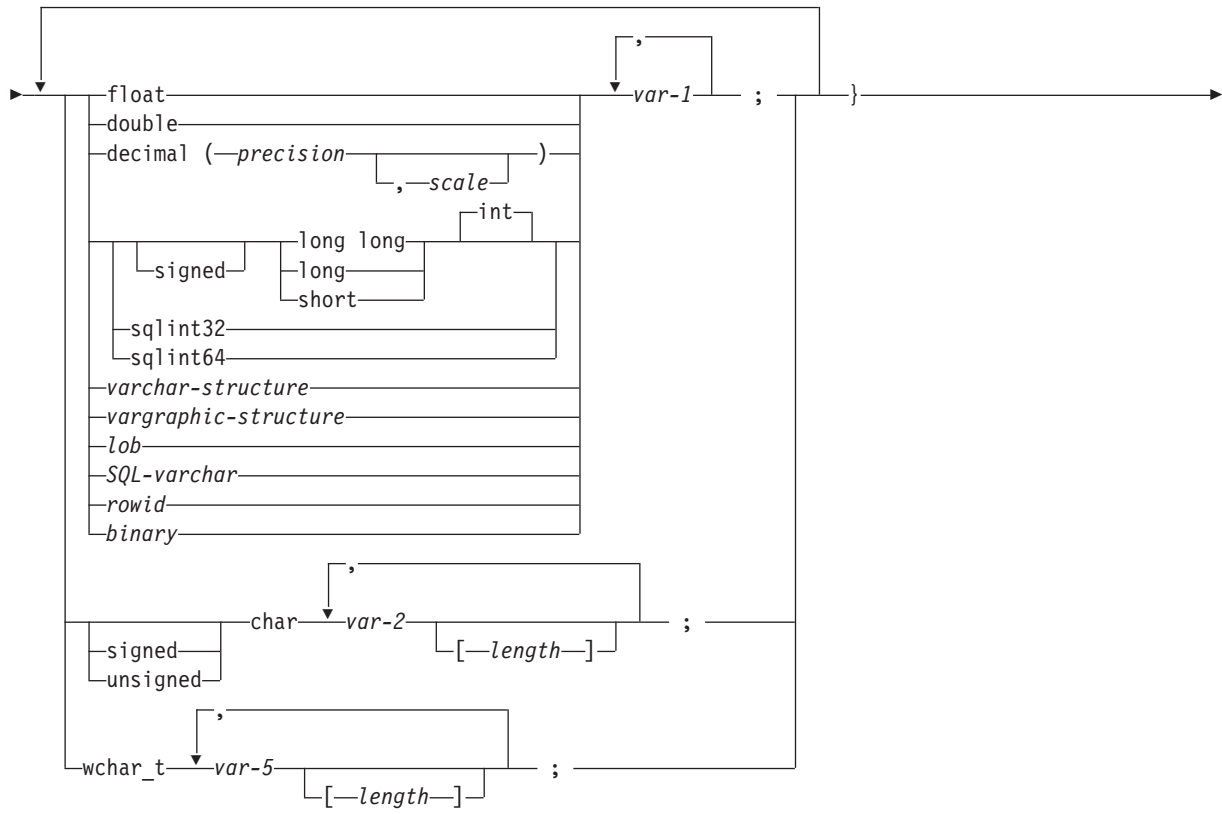
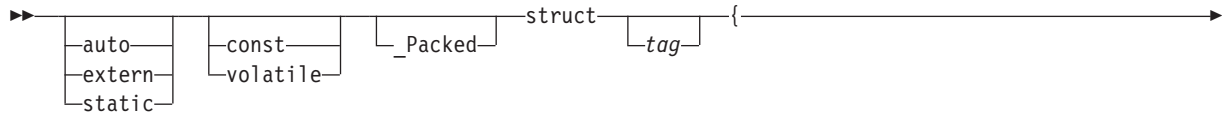
詳細については、以下のセクションを参照してください。

- 41 ページの『SQL を使用する C および C++ アプリケーションでのホスト構造宣言』
- 44 ページの『SQL を使用する C および C++ アプリケーションでのホスト構造標識配列』

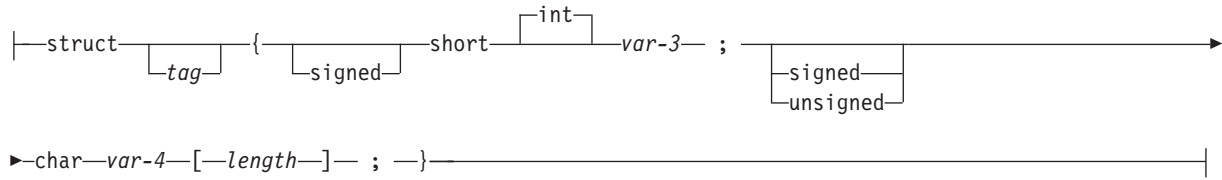
SQL を使用する C および C++ アプリケーションでのホスト構造宣言

次の図は、ホスト構造宣言に関する有効な構文を示しています。

ホスト構造

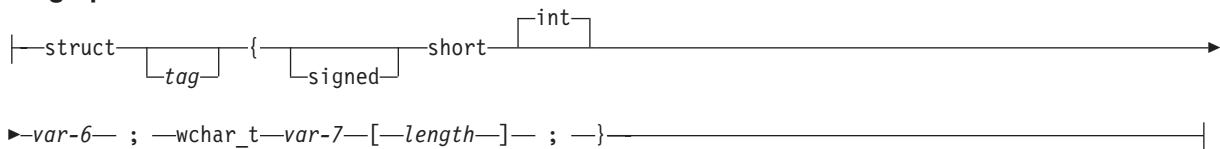


varchar-structure:

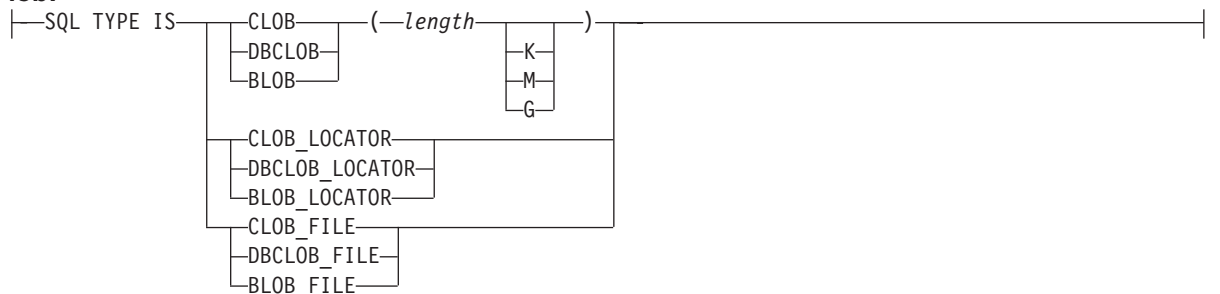


ホスト構造 (続き)

vargraphic-structure:



lob:



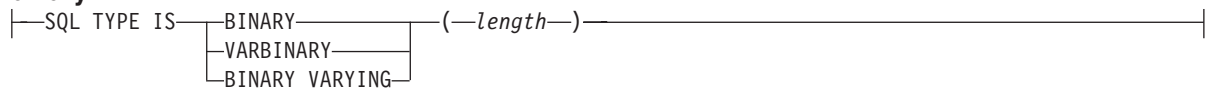
SQL-varchar:



rowid:



binary:



注:

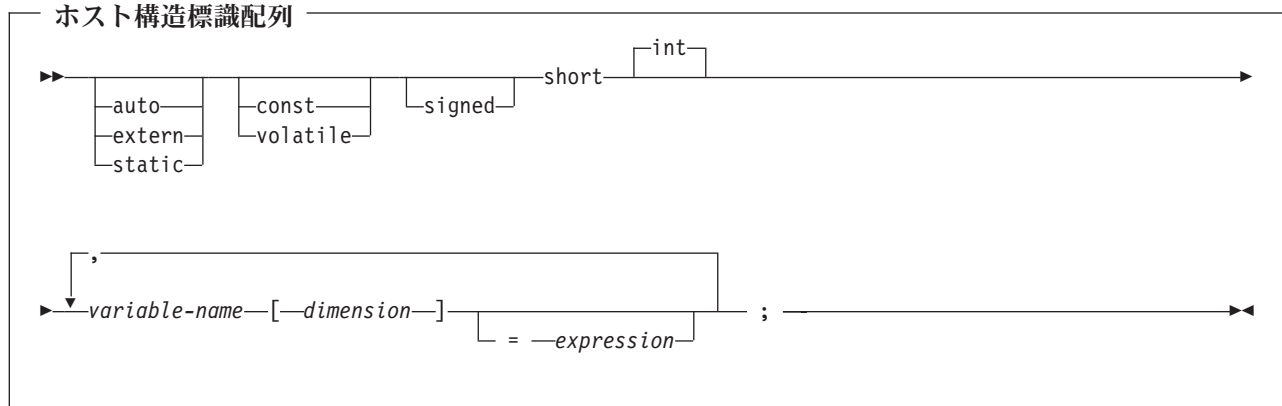
1. 数値ホスト変数、文字ホスト変数、グラフィック・ホスト変数、LOB ホスト変数、ROWID ホスト変数、およびバイナリー・ホスト変数の宣言についての詳細は、それぞれ数値、文字、グラフィック、LOB、ROWID、およびバイナリーの各ホスト変数に関する注意事項を参照してください。
2. char 配列または wchar_t 配列が続く short int の構造は、常に SQL C および C++ プリコンパイラーによって VARCHAR 構造または VARGRAPHIC 構造のいずれかとして解釈されます。
3. _Packed は、C++ では使用してはいけません。代わりに、宣言の前に #pragma pack(1) を、宣言の後に #pragma pack() を指定します。

```
#pragma pack(1)
struct {
    short myshort;
    long mylong;
    char mychar[5];
} a_st;
#pragma pack()
```

4. sqlint32 または sqlint64 を使用する場合は、ヘッダー・ファイルの sqlsystem.h が組み込まれている必要があります。

SQL を使用する C および C++ アプリケーションでのホスト構造標識配列

次の図は、ホスト構造標識配列の有効な構文を示しています。



注: dimension (次元) は 1 から 32767 までの整数定数でなければなりません。

SQL を使用する C および C++ アプリケーションでのホスト構造配列の使用

C および C++ プログラムの中では、ホスト構造配列が定義できます。これには次元属性が付いています。ホスト構造配列はそれ自体が複数レベルの構造の中に置かれることがあっても、その最大レベルは 2 レベルまでです。可変長文字ストリングや可変長漢字ストリングを使用しないのであれば、別の構造は必要ありません。

次の C の例、

```
struct {
    _Packed struct{
        char c1_var[20];
        short c2_var;
    } b_array[10];
} a_struct;
```

および次の C++ の例では、

```
#pragma pack(1)
struct {
    struct{
        char c1_var[20];
        short c2_var;
    } b_array[10];
} a_struct;
#pragma pack()
```

以下の条件が該当します。

- b_array 内のすべてのメンバーは、有効な変数宣言でなければなりません。
- _Packed 属性は struct タグに指定しなければなりません。
- b_array は、メンバー c1_var およびメンバー c2_var が入っているホスト構造の配列の名前です。

- `b_array` はブロック化形式の `FETCH` ステートメントおよび `INSERT` ステートメントでのみ使用できません。
- `c1_var` および `c2_var` は、`SQL` ステートメントでは有効なホスト変数ではありません。
- 構造に中間レベルの構造を含めることはできません。

たとえば、`C` では以下のカーソルから 10 行を取り出すことができます。

```

_Packed struct {char first_initial;
                char middle_initial;
                _Packed struct {short lastname_len;
                                char lastname_data[15];
                                } lastname;
                double total_salary;
                } employee_rec[10];
struct { short inds[4];
        } employee_inds[10];
...
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT SUBSTR(FIRSTNME,1,1), MIDINIT, LASTNAME,
         SALARY+BONUS+COMM
         FROM CORPDATA.EMPLOYEE;
EXEC SQL OPEN C1;
EXEC SQL FETCH C1 FOR 10 ROWS INTO :employee_rec:employee_inds;
...

```

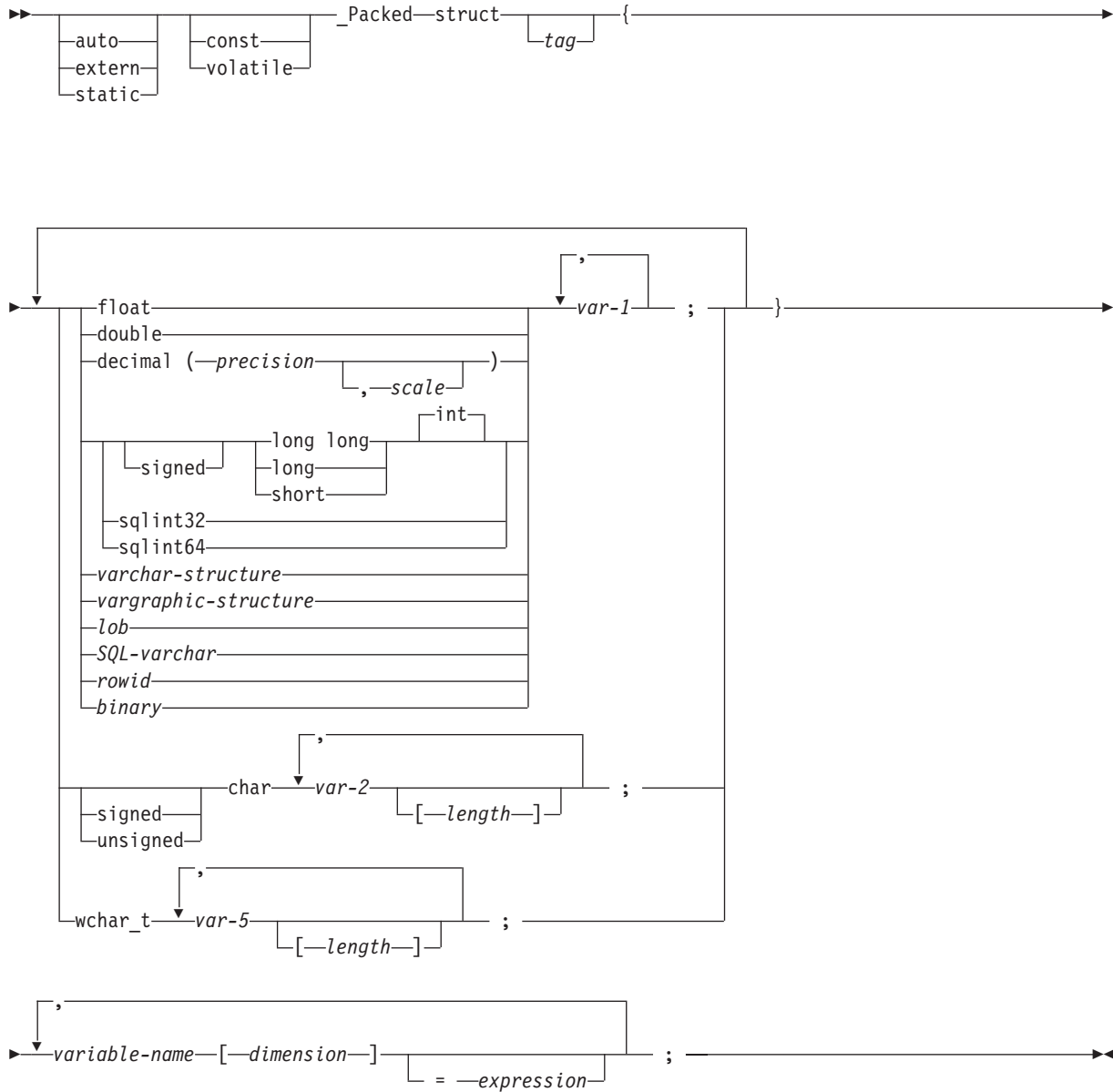
詳細については、以下のセクションを参照してください。

- 『SQL を使用する C および C++ アプリケーションでのホスト構造配列』
- 48 ページの『SQL を使用する C および C++ アプリケーションでのホスト構造配列標識構造』

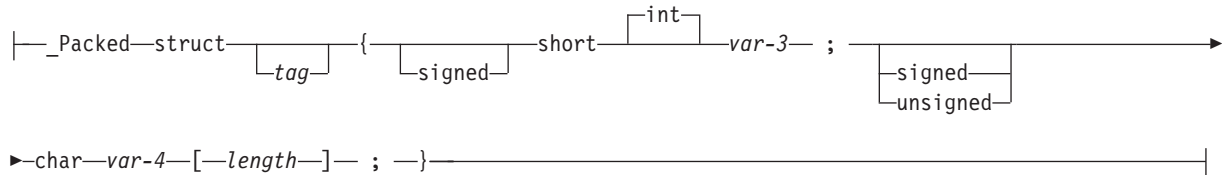
SQL を使用する C および C++ アプリケーションでのホスト構造配列

次の図は、ホスト構造配列宣言の有効な構文を示しています。

ホスト構造配列

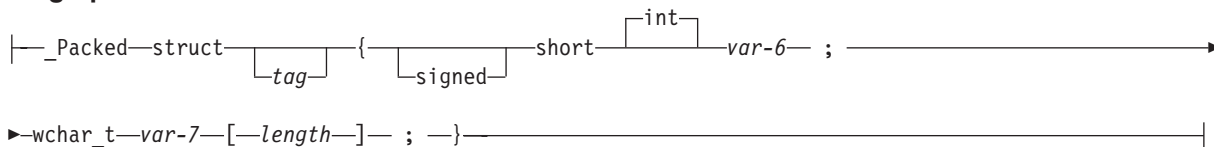


varchar-structure:

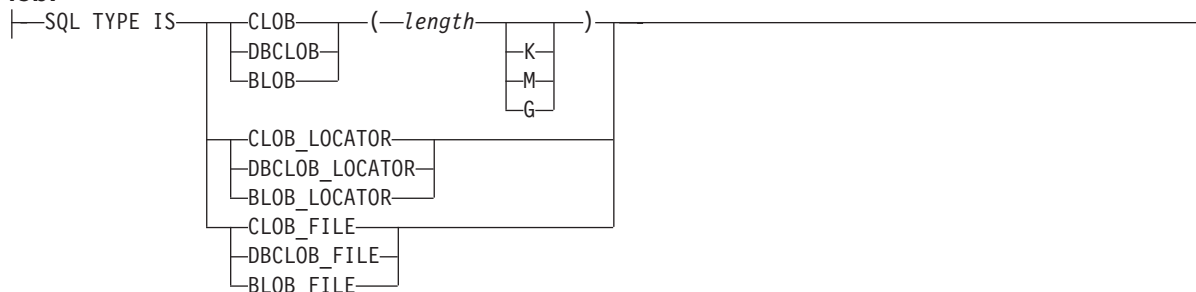


ホスト構造配列 (続き)

vargraphic-structure:



lob:



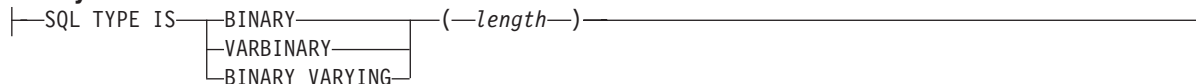
SQL-varchar:



rowid:



binary:

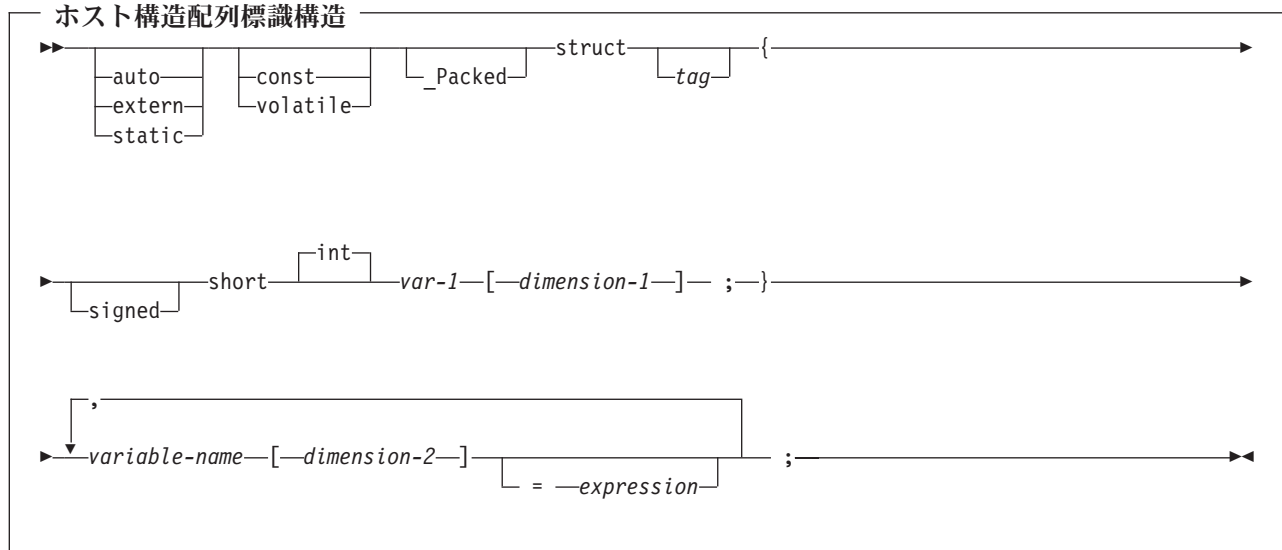


注:

1. 数値ホスト変数、文字ホスト変数、グラフィック・ホスト変数、LOB ホスト変数、ROWID ホスト変数、およびバイナリー・ホスト変数の宣言についての詳細は、それぞれ数値、文字、グラフィック、LOB、ROWID、およびバイナリーの各ホスト変数に関する注意事項を参照してください。
2. struct (構造) タグを使用すると、他のデータ域が定義できますが、これらをホスト変数として使用することはできません。
3. dimension (次元) は 1 から 32767 までの整数定数でなければなりません。
4. _Packed は、C++ では使用してはいけません。代わりに、宣言の前に #pragma pack(1) を、宣言の後に #pragma pack() を指定します。
5. sqlint32 または sqlint64 を使用する場合は、ヘッダー・ファイルの sqlsystem.h が組み込まれている必要があります。

SQL を使用する C および C++ アプリケーションでのホスト構造配列標識構造

次の図は、ホスト構造配列標識構造宣言の有効な構文を示しています。



注:

1. struct (構造) タグは他のデータ域を定義するために使用できますが、ホスト変数として使用することはできません。
2. dimension-1 (次元 1) および dimension-2 (次元 2) はともに、1 から 32767 までの整数定数でなければなりません。
3. _Packed は、C++ では使用してはいけません。代わりに、宣言の前に #pragma pack(1) を、宣言の後に #pragma pack() を指定します。

SQL を使用する C および C++ アプリケーションでのポインター・データ・タイプの使用

サポートされる C および C++ データ・タイプを指すポインターとなるホスト変数も宣言することができますが、次のような制約があります。

- あるホスト変数をポインターとして宣言するときは、そのホスト変数は、ホスト変数の頭にアスタリスクを付けて宣言しなければなりません。次の例はいずれも有効です。

```
short *mynum;           /* Ptr to an integer           */  
long **mynumptr;       /* Ptr to a ptr to a long integer */  
char *mychar;          /* Ptr to a single character    */  
char(*mychara)[20];    /* Ptr to a char array of 20 bytes */  
struct {               /* Ptr to a variable char array of 30 */  
    short mylen;        /* bytes.                        */  
    char mydata[30];  
} *myvchar;
```

注: 括弧は NUL 終了文字配列を指すポインターを宣言するときだけ許されますが、この場合、括弧は必須です。括弧を使用しないと、配列を指すために必要なポインターではなく、ポインターの配列を宣言することになります。以下に、例を示します。


```
char (*a)[10];          /* pointer to a null-terminated char array */
char *a[10];           /* pointer to an array of pointers */
```

- あるホスト変数をポインターとして宣言するときは、その変数を使って同じソース・ファイル内に他のホスト変数を宣言することはできません。たとえば、下に示す 2 番目の宣言は無効です。

```
char *mychar;          /* This declaration is valid */
char mychar;          /* But this one is invalid */
```

- あるホスト変数を SQL ステートメントの中で参照するときは、そのホスト変数は宣言したとおりに参照しなければなりません。ただし、NUL 終了文字配列を指すポインターだけは例外です。たとえば、次の宣言には括弧が必要です。

```
char (*mychara)[20];   /* ptr to char array of 20 bytes */
```

しかし、ホスト変数が SELECT のような SQL ステートメントの中で参照されるときは、括弧は許されません。

```
EXEC SQL SELECT name INTO :*mychara FROM mytable;
```

- アスタリスクだけがホスト変数名に対する演算子として使用できます。
- ホスト変数名の最大長は、アスタリスクも名前の一部として扱われるので、アスタリスクをいくつ指定したかによって影響されます。
- 構造を指すポインターは、可変長文字構造の場合を除き、ホスト変数として使用できません。また、構造の中のポインター・フィールドもホスト変数として使用できません。
- SQL は基底付きホスト変数のすべての指定記憶域を割り振るよう要求します。記憶域を割り振らないと、予期せぬ結果が生じる場合があります。

SQL を使用する C および C++ アプリケーションでの typedef の使用

typedef 宣言を使用して、short 型、float 型、double 型のような C 型指定子の代わりに使用される独自の識別コードを定義することもできます。ホスト変数の宣言に使用される typedef 識別コードは、typedef 宣言がそれぞれ別のブロックやプロシージャの中にある場合であっても、1 つのプログラム内では固有にならなければなりません。プログラムに BEGIN DECLARE SECTION ステートメントと END DECLARE SECTION ステートメントが含まれる場合、typedef 宣言が BEGIN DECLARE SECTION と END DECLARE SECTION と一緒に含まれる必要はありません。typedef 識別コードは BEGIN DECLARE SECTION 内で SQL プリコンパイラーによって認識されます。C および C++ プリコンパイラーは、typedef 宣言のサブセットだけを、ホスト変数宣言の場合と同様に認識します。

有効な typedef ステートメントの例:

- 長い typedef を宣言してから、typedef を参照するホスト変数を宣言します。

```
typedef long int LONG_T;
LONG_T I1, *I2;
```

- 文字配列の長さは、typedef またはホスト変数宣言のいずれかで指定されますが、両方では指定されません。

```
typedef char NAME_T[30];
typedef char CHAR_T;
CHAR_T name1[30]; /* Valid */
NAME_T name2;     /* Valid */
NAME_T name3[10]; /* Not valid for SQL use */
```

- SQL TYPE IS ステートメントは typedef で使用されることがあります。

```
typedef SQL TYPE IS CLOB(5K) CLOB_T;
CLOB_T clob_var1;
```

- ストレージ・クラス (auto、extern、static)、volatile、または const 修飾子をホスト変数宣言で指定することができます。

```
typedef short INT_T;
typedef short INT2_T;
static INT_T i1;
volatile INT2_T i2;
```

- 構造体の typedef がサポートされます。

```
typedef _Packed struct {char dept[3];
                        char deptname[30];
                        long Num_employees;} DEPT_T;

DEPT_T dept_rec;
DEPT_T dept_array[20]; /* use for blocked insert or fetch */
```

SQL を使用する C および C++ アプリケーションでの ILE C コンパイラ — 外部ファイル記述の使用

C または C++ の #pragma mapinc ディレクティブを #include ディレクティブと一緒に使用すると、プログラムの中に外部ファイル記述を含めることができます。SQL で使用するときには、#pragma mapinc ディレクティブは特定の形式だけが、SQL プリコンパイラーによって受け付けられます。必要とされる要素のすべてが指定されていないと、プリコンパイラーはディレクティブを無視し、ホスト変数構造を生成しません。必要とされる要素とは、次のものです。

- 組み込む名前
- 外部記述ファイル名
- 形式名または形式名のリスト
- オプション
- 変換オプション

ライブラリー名、結合名、変換オプション、および接頭部名は任意選択です。ユーザーがコーディングした typedef ステートメントはプリコンパイラーに認識されませんが、#pragma mapinc ディレクティブと、#include ディレクティブによって生成された typedef ステートメントは認識されます。SQL はオプション・パラメーターの入力値、出力値、およびキー値をサポートします。変換オプションについては、サポートされている値は D、p、z、_P、および 1BYTE_CHAR です。これらのオプションは D と p の両方が指定できない場合を除き、どのような順序でも指定することができます。#pragma mapinc ディレクティブと #include ディレクティブによって生成された typedef 結合を使用して宣言された結合は、SQL ステートメントの中でホスト変数として使用することはできません。しかし、結合のメンバーは使用できます。typedef 構造を含む構造は SQL ステートメントの中では使用できません。しかし、typedef を使用して宣言した構造は使用できます。

「DB2 UDB for iSeries SQL プログラミング 概念」の『DB2 UDB for iSeries サンプル・テーブル』に記載されているサンプル・テーブル DEPARTMENT の定義を検索するときは、次のようにコーディングすることができます。

```
#pragma mapinc ("dept","CORPDATA/DEPARTMENT(*ALL)","both")
#include "dept"
CORPDATA_DEPARTMENT_DEPARTMENT_both_t Dept_Structure;
```

Dept_Structure という名前のホスト構造は、次の要素を使用して定義されています。すなわち、DEPTNO、DEPTNAME、MGRNO、および ADMRDEPT です。これらのフィールド名は、SQL ステートメントの中でホスト変数として使用できます。

注: DATE、TIME、および TIMESTAMP の各列からは、文字ホスト変数定義が生成されます。これらは SQL により、DATE、TIME、および TIMESTAMP 列と同じ比較および割り当て規則を使用して扱われます。たとえば、日付ホスト変数は、DATE 列または日付の有効な表現である文字ストリングとだけ突き合わされて比較されます。

GRAPHIC 列または VARGRAPHIC 列が UCS-2 CCSID を持つ場合、生成されるホスト変数には UCS-2 CCSID が割り当てられます。GRAPHIC 列または VARGRAPHIC 列が UTF-16 CCSID を持つ場合、生成されるホスト変数には UTF-16 CCSID が割り当てられます。

ゾーン 10 進数、2 進数 (位取りフィールドがゼロでない)、および任意選択で 10 進数は、ILE C for iSeries では文字フィールドにマッピングされますが、SQL はこれらのフィールドを数値として扱います。拡張プログラム・モデル (EPM) ルーチンを使用してこれらのフィールドがゾーンおよびパック 10 進数データを変換するように操作できます。詳細については、「ILE C for iSeries Language

Reference 」を参照してください。

SQL データ・タイプおよび C または C++ データ・タイプの対応関係の判別

プリコンパイラーは、次の表に基づいて、ホスト変数のベース SQLTYPE とベース SQLLEN を判断します。ホスト変数が標識変数と一緒に記載されているときは、その SQLTYPE はベース SQLTYPE に 1 を加えたものです。

表 1. C または C++ 宣言と代表的 SQL データ・タイプとの対応関係

C または C++ データ・タイプ	ホスト変数の SQLTYPE	ホスト変数の SQLLEN	SQL データ・タイプ
short int	500	2	SMALLINT
long int	496	4	INTEGER
long long int	492	8	BIGINT
decimal(p,s)	484	バイト 1 には p、バイト 2 には s	DECIMAL (p,s)
float	480	4	FLOAT (単精度)
double	480	8	FLOAT (倍精度)
単一文字形式	452	1	CHAR(1)
NUL 終了文字形式	460	length	VARCHAR (長さ - 1)
VARCHAR 構造化形式	448	length	VARCHAR (長さ)
単一グラフィック形式	468	1	GRAPHIC (1)
NUL 終了単一グラフィック形式	400	length	VARGRAPHIC (長さ - 1)
VARGRAPHIC 構造化形式	464	length	VARGRAPHIC (長さ)

下表を参照すると、各 SQL データ・タイプに対応する C または C++ データ・タイプを判別することができます。

表 2. SQL データ・タイプと代表的な C または C++ 宣言との対応関係

SQL データ・タイプ	C または C++ データ・タイプ	注
SMALLINT	short int	

表 2. SQL データ・タイプと代表的な C または C++ 宣言との対応関係 (続き)

SQL データ・タイプ	C または C++ データ・タイプ	注
INTEGER	long int	
BIGINT	long long int	
DECIMAL(p,s)	decimal(p,s)	p は 1 から 63 までの正の整数。s は 0 から 63 までの正の整数。
NUMERIC(p,s) または非ゼロ位取り 2 進数	正確に対応するものなし	decimal(p,s) を使用。
FLOAT (単精度)	float	
FLOAT (倍精度)	double	
CHAR(1)	単一文字形式	
CHAR(n)	正確に対応するものなし	n > 1 なら、NUL 終了文字形式を使用。
VARCHAR(n)	NUL 終了文字形式	NUL 終了文字を受け入れるには少なくとも n +1 が必要。データに NUL (0) を含めることができる場合は、VARCHAR 構造化形式または SQL VARCHAR を使用。 n は正の整数です。n の最大値は 32740。
	VARCHAR 構造化形式	n の最大値は 32740。SQL VARCHAR 形式も使用可能。
CLOB	なし	C または C++ で CLOB を宣言するために SQL TYPE IS を使用。
GRAPHIC (1)	単一グラフィック形式	
GRAPHIC (n)	正確に対応するものなし	n > 1 なら、NUL 終了グラフィック形式を使用。
VARGRAPHIC(n)	NUL 終了グラフィック形式	データにグラフィック NUL 値 (/0/0) を含めることができる場合は、VARGRAPHIC 構造化形式を使用。NUL 終了文字を受け入れるには少なくとも n +1 が必要。 n は正の整数です。n の最大値は 16370。
	VARGRAPHIC 構造化形式	n は正の整数です。n の最大値は 16370。
DBCLOB	なし	C または C++ で DBCLOB を宣言するために SQL TYPE IS を使用。
BINARY	なし	C または C++ で BINARY を宣言するために SQL TYPE IS を使用。
VARBINARY	なし	C または C++ で VARBINARY を宣言するために SQL TYPE IS を使用。
BLOB	なし	C または C++ で BLOB を宣言するために SQL TYPE IS を使用。

表 2. SQL データ・タイプと代表的な C または C++ 宣言との対応関係 (続き)

SQL データ・タイプ	C または C++ データ・タイプ	注
DATE	NUL 終了文字形式	形式が *USA、*ISO、*JIS、または *EUR のときは、NUL 終了文字を受け入れるには少なくとも 11 文字が必要。形式が *MDY、*YMD、または *DMY のときは、NUL 終了文字を受け入れるには少なくとも 9 文字が必要。形式が *JUL のときは、NUL 終了文字を受け入れるには少なくとも 7 文字が必要。
	VARCHAR 構造化形式	形式が *USA、*ISO、*JIS、または *EUR のときは、少なくとも 10 文字が必要。形式が *MDY、*YMD、または *DMY のときは、少なくとも 8 文字が必要。形式が *JUL のときは、少なくとも 6 文字が必要。
TIME	NUL 終了文字形式	NUL 終了文字を受け入れるには少なくとも 7 文字 (秒を含む場合は、9 文字) が必要。
	VARCHAR 構造化形式	少なくとも 6 文字が必要。秒を含む場合は、8 文字が必要。
TIMESTAMP	NUL 終了文字形式	NUL ターミネーターを受け入れるには、少なくとも 20 文字 (マイクロ秒を全桁の精度で含める場合 27 文字) は必要。n が 27 未満のときは、マイクロ秒部分で切り捨てが起こる。
	VARCHAR 構造化形式	少なくとも 19 文字が必要。マイクロ秒を全桁の精度で含める場合は、26 文字必要。数値数が 26 未満の場合は、マイクロ秒部分で切り捨てが起こる。
DATALINK	サポートなし	
ROWID	なし	C または C++ で ROWID を宣言するために SQL TYPE IS を使用。

詳細については、『C および C++ 変数宣言と使用法についての注意』を参照してください。

C および C++ 変数宣言と使用法についての注意

アポストロフィと引用符は、C、C++ および SQL の間で意味が異なります。C および C++ では、引用符は文字列定数を区切るために、アポストロフィは文字定数を区切るために使用されます。SQL では、この区別がなく、引用符は識別名を区切るために、アポストロフィは文字ストリング定数を区切るために使用されます。SQL での文字データは、整数データと区別されています。

SQL を使用する C および C++ アプリケーションでの標識変数の使用

標識変数は 2 バイトの整数 (short int) です。ホスト構造をサポートするために標識構造 (ハーフワードの整数変数の配列として定義されているもの) を指定することもできます。検索される時、標識変数はその対応するホスト変数にヌル値が割り当てられているかどうかを示すために使用されます。列に割り当てるときには、ヌル値を割り当てるべきであることを示すために負の標識変数が使用されます。

詳細については、「SQL 解説書」の『標識変数』を参照してください。

標識変数の宣言の仕方は、ホスト変数の場合と同じです。これらの 2 つの変数の宣言は、適切な方法で組み合わせることができます。

例:

次のステートメントがあるとします。

```
EXEC SQL FETCH CLS_CURSOR INTO :ClsCd,  
                                :Day :DayInd,  
                                :Bgn :BgnInd,  
                                :End :EndInd;
```

変数は次のように宣言することができます。

```
EXEC SQL BEGIN DECLARE SECTION;  
char  ClsCd[8];  
char  Bgn[9];  
char  End[9];  
short Day, DayInd, BgnInd, EndInd;  
EXEC SQL END DECLARE SECTION;
```

第 6 章 COBOL アプリケーションでの SQL ステートメントのコーディング方法

iSeries では、複数の COBOL コンパイラーがサポートされます。ライセンス・プログラムである DB2 UDB SQL 開発キットがサポートする言語は、COBOL for iSeries と ILE COBOL for iSeries だけです。ここでは、SQL ステートメントを COBOL プログラムに組み込む場合に固有のアプリケーションおよびコーディング上の要件について説明します。ホスト構造およびホスト変数に関する要件についても説明します。

詳細については、以下のセクションを参照してください。

- 『SQL を使用する COBOL アプリケーションでの SQL 連絡域の定義』
- 56 ページの『SQL を使用する COBOL アプリケーションでの SQL 記述子域の定義』
- 58 ページの『SQL を使用する COBOL アプリケーションでの SQL ステートメントの組み込み』
- 60 ページの『SQL を使用する COBOL アプリケーションでのホスト変数の使用』
- 70 ページの『SQL を使用する COBOL アプリケーションでのホスト構造の使用』
- 80 ページの『SQL を使用する COBOL アプリケーションでの外部ファイル記述の使用』
- 82 ページの『SQL データ・タイプと COBOL データ・タイプの対応関係の判別』
- 85 ページの『SQL を使用する COBOL アプリケーションでの標識変数の使用』

SQL ステートメントの使い方を示した詳しいサンプル COBOL プログラムは、『第 12 章 DB2 UDB for iSeries ステートメントを使用したサンプル・プログラム』に記載されています。

注: 重要な法律的事柄に関する情報について、214 ページの『コードについての特記事項』をご覧ください。

SQL を使用する COBOL アプリケーションでの SQL 連絡域の定義

- | COBOL プログラムでは、SQLCA を使用して組み込み SQL ステートメントの戻される状況をチェック
- | したり、SQL 診断域を使用して戻される状況をチェックすることができます。SQLCA の代わりに SQL
- | 診断域を使用するには、SET OPTION SQL ステートメントでオプション SQLCA = *NO を指定しま
- | す。詳細については、15 ページの『SQL 診断域を使用する』を参照してください。

SQLCA を使用する場合、SQL ステートメントを組み込む COBOL プログラムは、次のいずれかまたは両方を含んでいなければなりません。

- PICTURE S9(9) BINARY、PICTURE S9(9) COMP-4、または PICTURE S9(9) COMP として宣言されている SQLCODE 変数。
- PICTURE X(5) として宣言されている SQLSTATE 変数

または、

- SQLCA (SQLCODE および SQLSTATE 変数が入っている)

SQLCODE 値および SQLSTATE 値は、各 SQL ステートメントが実行された後、データベース・マネージャーによって設定されます。アプリケーションは、SQLCODE 値または SQLSTATE 値を調べて、最後の SQL ステートメントが正しく実行されたかどうかを判定することができます。

SQLCA は、直後または、SQL の INCLUDE ステートメントの使用によって、COBOL プログラムの中にコーディングすることができます。SQL の INCLUDE ステートメントを使用するときは、次のような標準の宣言を含める必要があります。

```
EXEC SQL INCLUDE SQLCA END-EXEC.
```

SQLCODE、SQLSTATE、および SQLCA の各変数宣言は、ユーザーのプログラムの WORKING-STORAGE SECTION または LINKAGE SECTION に現れる必要があります。これらの変数宣言を置く場所は、これらのセクションにレコード記述項目を指定できる個所ならば、どこでも構いません。

INCLUDE ステートメントを使用すると、SQL の COBOL プリコンパイラーは、SQLCA 用の COBOL ソース・ステートメントを組み込みます。

```
01 SQLCA.  
 05 SQLCAID      PIC X(8).  
 05 SQLCABC      PIC S9(9) BINARY.  
 05 SQLCODE      PIC S9(9) BINARY.  
 05 SQLERRM.  
   49 SQLERRML   PIC S9(4) BINARY.  
   49 SQLERRMC   PIC X(70).  
 05 SQLERRP      PIC X(8).  
 05 SQLERRD      OCCURS 6 TIMES  
                  PIC S9(9) BINARY.  
  
 05 SQLWARN.  
   10 SQLWARN0   PIC X.  
   10 SQLWARN1   PIC X.  
   10 SQLWARN2   PIC X.  
   10 SQLWARN3   PIC X.  
   10 SQLWARN4   PIC X.  
   10 SQLWARN5   PIC X.  
   10 SQLWARN6   PIC X.  
   10 SQLWARN7   PIC X.  
   10 SQLWARN8   PIC X.  
   10 SQLWARN9   PIC X.  
   10 SQLWARNA   PIC X.  
 05 SQLSTATE     PIC X(5).
```

ILE COBOL for iSeries の場合、SQLCA は GLOBAL 文節を使用して宣言されます。SQLCODE の宣言がプログラムの中にあつて、SQLCA がプリコンパイラーによって与えられるとき、SQLCODE の個所は SQLCADE で置き換えられます。SQLSTATE の宣言がプログラムの中にあつて、SQLCA がプリコンパイラーによって与えられるとき、SQLSTATE の個所は SQLSTOTE で置き換えられます。

SQLCA の詳細については、「SQL 解説書」の『SQL 連絡域』を参照してください。

SQL を使用する COBOL アプリケーションでの SQL 記述子域の定義

SQLDA を必要とするステートメントには、次のものがあります。

```
EXECUTE...USING DESCRIPTOR 記述子名  
FETCH...USING DESCRIPTOR 記述子名  
OPEN...USING DESCRIPTOR 記述子名  
CALL...USING DESCRIPTOR 記述子名  
DESCRIBE ステートメント名 INTO 記述子名  
DESCRIBE TABLE ホスト変数 INTO 記述子名  
PREPARE ステートメント名 INTO 記述子名
```


SQLCA とは違って SQLDA はプログラムの中に 2 つ以上置くことができます。その SQLDA の名前は有効であればどの名前でも使えます。SQLDA は、COBOL プログラムの中で直接コーディングすることもできますが、INCLUDE ステートメントを使って追加することもできます。SQL の INCLUDE ステートメントを使用するときは、標準の SQLDA 宣言を組み込む必要があります。

```
EXEC SQL INCLUDE SQLDA END-EXEC.
```

SQLDA 用として組み込まれる COBOL 宣言は次のとおりです。

```
1 SQLDA.  
  05 SQLDAID      PIC X(8).  
  05 SQLDABC      PIC S9(9) BINARY.  
  05 SQLN         PIC S9(4) BINARY.  
  05 SQLD         PIC S9(4) BINARY.  
  05 SQLVAR OCCURS 0 TO 409 TIMES DEPENDING ON SQLD.  
    10 SQLTYPE    PIC S9(4) BINARY.  
    10 SQLLEN     PIC S9(4) BINARY.  
    10 FILLER     REDEFINES SQLLEN.  
      15 SQLPRECISION PIC X.  
      15 SQLSCALE    PIC X.  
    10 SQLRES     PIC X(12).  
    10 SQLDATA    POINTER.  
    10 SQLIND     POINTER.  
    10 SQLNAME.  
      49 SQLNAMEL PIC S9(4) BINARY.  
      49 SQLNAMEC PIC X(30).
```

図 1. COBOL 用の INCLUDE SQLDA 宣言

SQLDA の宣言は、プログラムの WORKING-STORAGE SECTION か LINKAGE SECTION に置かなければなりません。置く場所は、レコード記述項目をこれらのセクションに指定できるならば、どこでも構いません。ILE COBOL for iSeries の場合、SQLDA は GLOBAL 文節を使用して宣言されます。

動的 SQL は高度なプログラミング技法です。これについては、「SQL プログラミング」の『動的 SQL アプリケーション』で説明されています。動的 SQL を使用すると、ユーザーのプログラムはその実行と平行して SQL ステートメントを作成し、実行させることができます。動的に実行される変数 SELECT リスト (すなわち、照会の一部として返されるデータのリスト) を指定する SELECT ステートメントには、SQL 記述子域 (SQLDA) が必要です。これは、SELECT の結果を受け入れるために割り振るべき変数の数とタイプが事前に予測できないからです。

SQLDA の詳細については、「SQL 解説書」の『SQL 記述子域』を参照してください。

SQL を使用する COBOL アプリケーションでの SQL ステートメントの組み込み

SQL ステートメントは COBOL プログラム・セクションに次のようにコーディングすることができます。

SQL ステートメント	プログラム・セクション
BEGIN DECLARE SECTION	WORKING-STORAGE SECTION または LINKAGE SECTION
END DECLARE SECTION	
DECLARE VARIABLE	
DECLARE STATEMENT	
INCLUDE SQLCA	WORKING-STORAGE SECTION または LINKAGE SECTION
INCLUDE SQLDA	
INCLUDE メンバー名	DATA DIVISION または PROCEDURE DIVISION
その他	PROCEDURE DIVISION

COBOL プログラムの中の各 SQL ステートメントは、EXEC SQL で始まり、END-EXEC で終わっていません。SQL ステートメントが 2 つの COBOL ステートメントの間に置かれるときは、ピリオドは省略できますが、ピリオドがない方がよい場合があります。EXEC SQL キーワードはいずれも 1 行に置かなければなりません。ステートメントの残りの部分は次行とそれ以降の行に継続させることができます。

例:

COBOL プログラムの中にコーディングされる UPDATE ステートメントをコーディングすると、次のようになります。

```
EXEC SQL
  UPDATE DEPARTMENT
  SET MGRNO = :MGR-NUM
  WHERE DEPTNO = :INT-DEPT
END-EXEC.
```

詳細については、以下のセクションを参照してください。

- 59 ページの『SQL を使用する COBOL アプリケーションでの注記』
- 59 ページの『SQL を使用する COBOL アプリケーションでの SQL ステートメントの継続』
- 59 ページの『SQL を使用する COBOL アプリケーションでのコードの組み込み』
- 59 ページの『SQL を使用する COBOL アプリケーションでのマージン』
- 59 ページの『SQL を使用する COBOL アプリケーションでの順序番号』
- 59 ページの『SQL を使用する COBOL アプリケーションでの名前』
- 60 ページの『SQL を使用する COBOL アプリケーションでの COBOL コンパイル時オプション』
- 60 ページの『SQL を使用する COBOL アプリケーションでのステートメント・ラベル』
- 60 ページの『SQL を使用する COBOL アプリケーションでの WHENEVER ステートメント』
- 60 ページの『複数ソース COBOL プログラムおよび SQL COBOL プリコンパイラー』

SQL を使用する COBOL アプリケーションでの注記

SQL の注記 (--) の他に、組み込み SQL ステートメントの中に COBOL の注記行 (7 列目が * または / の行) を組み込むことができます。ただし、キーワードの EXEC と SQL の間には入れられません。COBOL のデバッグ行 (7 行目が D の行) は、プリコンパイラーにより注記行として扱われます。

SQL を使用する COBOL アプリケーションでの SQL ステートメントの継続

SQL ステートメントの場合の行継続の規則は、EXEC SQL を 1 行以内で指定する必要がある点を除けば、他の COBOL ステートメントの場合と同じです。

文字列定数のある行から次の行に継続させる場合は、2 番目の行の最初の非ブランク文字はアポストロフィか引用符でなければなりません。区切り文字付き識別コードのある行から次の行に継続させる場合は、2 番目の行の最初の非ブランク文字はアポストロフィか引用符でなければなりません。

DBCS データを含む定数は、継続される行の 72 桁目にシフトイン文字を入れ、継続行の最初の文字列区切り文字のあとにシフトアウト文字を入れることによって、複数行にわたって継続させることができます。

この SQL ステートメントの G'<AABBCCDDEEFFGGHHIIJJKK>' はグラフィック定数として有効です。重複しているシフトは除去されます。

```
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
EXEC SQL
SELECT * FROM GRAPHTAB          WHERE GRAPHCOL = G'<AABB>
-      '<CCDDEEFFGGHHIIJJKK>'
END-EXEC.
```

SQL を使用する COBOL アプリケーションでのコードの組み込み

SQL ステートメントまたは COBOL ホスト変数宣言ステートメントは、これらのステートメントが組み込まれるソース・コード内の個所に次の SQL ステートメントを組み込むことによって、挿入することができます。

```
EXEC SQL INCLUDE member-name END-EXEC.
```

COBOL の COPY ステートメントは、SQL ステートメントまたは SQL ステートメントの中で参照される COBOL ホスト変数の宣言を組み込むためには使用できません。

SQL を使用する COBOL アプリケーションでのマージン

SQL ステートメントは 12 桁目から 72 桁目までにコーディングします。EXEC SQL が指定したマージンの前 (すなわち、12 桁目の前) から始まっているときは、SQL プリコンパイラーはそのステートメントを認識しません。

SQL を使用する COBOL アプリケーションでの順序番号

SQL プリコンパイラーによって生成されるソース・ステートメントは、SQL ステートメントと同じ順序番号を使用して生成されます。

SQL を使用する COBOL アプリケーションでの名前

有効な COBOL 変数名ならば、どのような名前でもホスト変数に使用できますが、次のような制約を受けます。

'SQL'、'RDI'、または 'DSN' で始まるホスト変数名や外部入り口名は、使用してはなりません。これらの名前はデータベース・マネージャー用に予約されています。

FILLER が含まれる構造を使用すると、SQL ステートメントで、期待されるような動作をしないことがあります。COBOL 構造内のフィールドはすべて、予期しない結果を避けるように指定することをお勧めします。

SQL を使用する COBOL アプリケーションでの COBOL コンパイル時オプション

COBOL PROCESS ステートメントを使用すると、COBOL コンパイラーに対するコンパイル時オプションが指定できます。PROCESS ステートメントは、プログラムを作成するためにプリコンパイラーによって呼び出されるとき、COBOL コンパイラーによって認識されます。しかし、SQL プリコンパイラー自体は PROCESS ステートメントを認識しません。そのために、APOST や QUOTE のような COBOL ソース・プログラムの構文に影響を与えるオプションは、PROCESS 構文では指定してはなりません。その代わりに、*APOST と *QUOTE を CRTSQLCBL と CRTSQLCBLI の各コマンドの OPTION パラメーターで指定する必要があります。

SQL を使用する COBOL アプリケーションでのステートメント・ラベル

PROCEDURE DIVISION 中の実行可能な SQL ステートメントは、その前に段落名を置くことができます。

SQL を使用する COBOL アプリケーションでの WHENEVER ステートメント

SQL WHENEVER ステートメントの中で GOTO 文節の対象となるものは、PROCEDURE DIVISION 中のセクション名または非修飾の段落名でなければなりません。

複数ソース COBOL プログラムおよび SQL COBOL プリコンパイラー

SQL COBOL プリコンパイラーは、PROCESS ステートメントで区切った複数のソース・プログラムのプリコンパイルをサポートしません。

SQL を使用する COBOL アプリケーションでのホスト変数の使用

SQL ステートメントの中で使用するホスト変数はいずれも明示的に宣言しなければなりません。SQL ステートメントの中で使用するホスト変数は、SQL ステートメントの中でホスト変数を初めて使用する前に、宣言しておかなければなりません。

ホスト変数を定義するために使用される COBOL ステートメントは、その前に BEGIN DECLARE SECTION ステートメントを置き、その後、END DECLARE SECTION ステートメントを置く必要があります。BEGIN DECLARE SECTION と END DECLARE SECTION を指定する場合、SQL ステートメントで使用するすべてのホスト変数宣言は、BEGIN DECLARE SECTION ステートメントと END DECLARE SECTION ステートメントの間になければなりません。

SQL ステートメントの中のホスト変数はいずれも、その前にコロン (:) を付けなければなりません。

ホスト変数はレコードまたは要素にすることはできません。

COBOL ホスト変数名の中でダッシュを使えるようにするには、負符号の前後にブランクを置かなければなりません。

詳細については、『SQL を使用する COBOL アプリケーションでのホスト変数の宣言』を参照してください。

SQL を使用する COBOL アプリケーションでのホスト変数の宣言

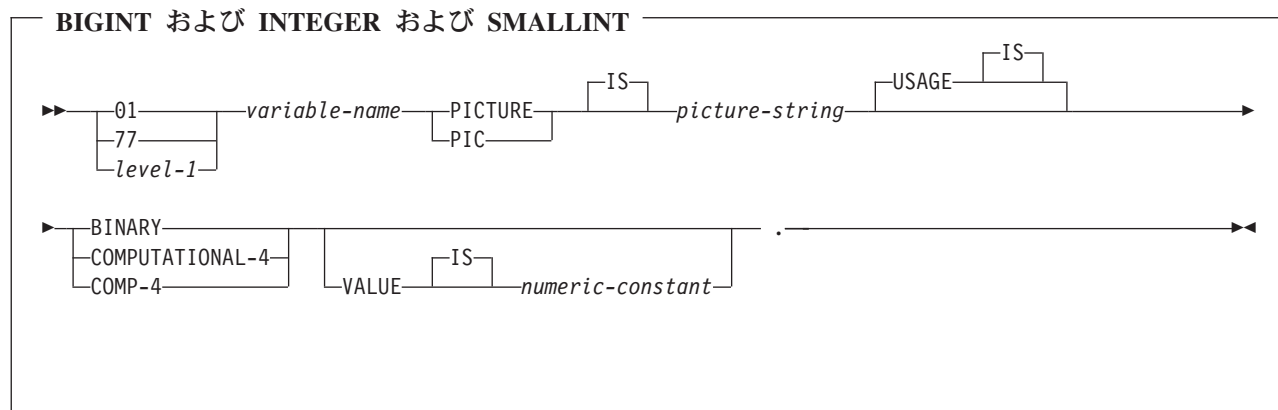
COBOL プリコンパイラーは、有効な COBOL 宣言のサブセットだけを有効なホスト変数宣言として認めます。

特定のタイプのホスト変数について、以下のトピックを参照してください。

- 『SQL を使用する COBOL アプリケーションでのホスト変数の数値』
- 63 ページの『SQL を使用する COBOL アプリケーションでの浮動小数点ホスト変数』
- 64 ページの『SQL を使用する COBOL アプリケーションでの文字ホスト変数』
- 66 ページの『SQL を使用する COBOL アプリケーションでのバイナリー・ホスト変数』
- 65 ページの『SQL を使用する COBOL アプリケーションでのグラフィック・ホスト変数』
- 67 ページの『SQL を使用する COBOL アプリケーションでの LOB ホスト変数』
- 69 ページの『SQL を使用する COBOL アプリケーションでの Datetime ホスト変数』
- 70 ページの『SQL を使用する COBOL アプリケーションでの ROWID ホスト変数』

SQL を使用する COBOL アプリケーションでのホスト変数の数値

次の図は、有効な整数ホスト変数宣言の構文を示しています。

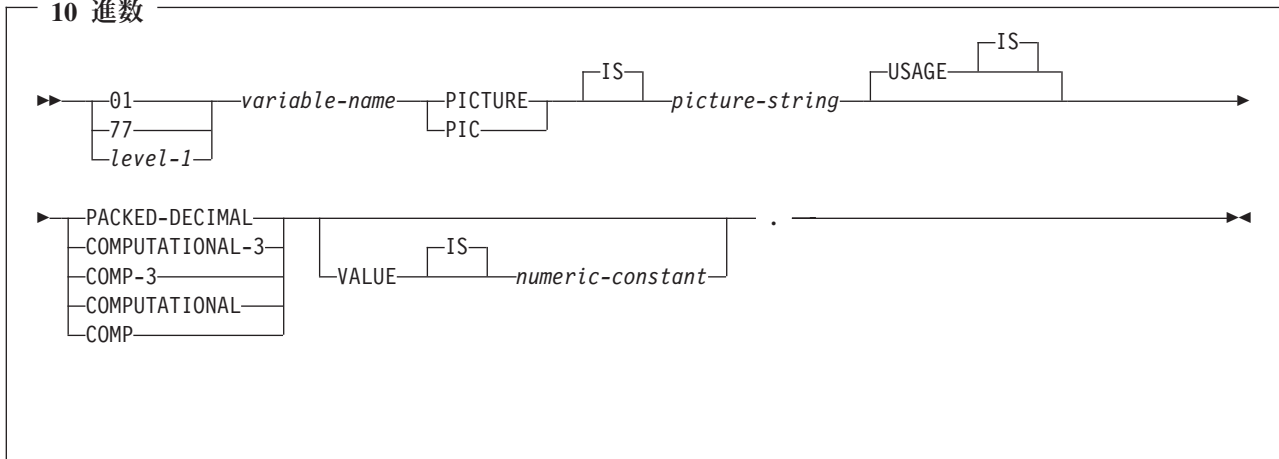


注:

- BINARY、COMPUTATIONAL-4、および COMP-4 は同じ働きをします。他のシステムでも実行されるようなアプリケーションの場合には BINARY をコーディングしておくべきです。COMPUTATIONAL-4 と COMP-4 は IBM の拡張機能であり、国際標準化機構 (ISO)/ANSI COBOL ではサポートされないからです。これらのタイプに関連する *picture-string* (ピクチャー・ストリング) は S9(i)V9(d) (または 9 のインスタンスが *i* 回および *d* 回現れる S9...9V9...9) の形式になっていなければなりません。 $i + d$ は 18 以下でなければなりません。
- level-1 (レベル 1) は、2 から 48 までの COBOL レベルを示します。

次の図は、有効な 10 進数ホスト変数宣言の構文を示しています。

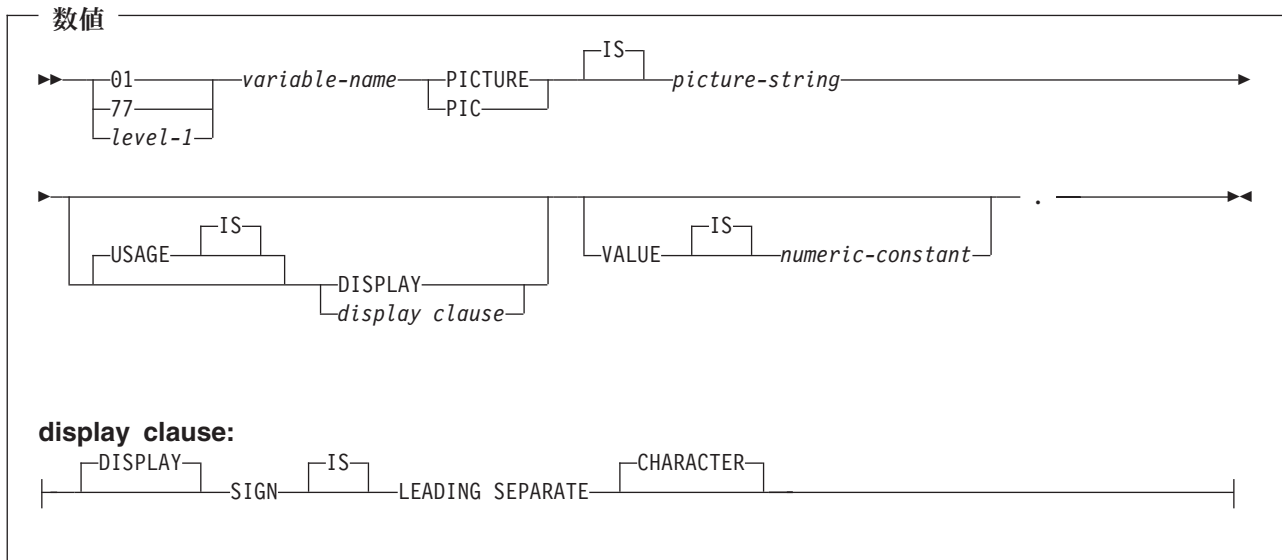
10 進数



注:

1. PACKED-DECIMAL、COMPUTATIONAL-3、および COMP-3 は同じ働きをします。他のシステムでも実行されるようなアプリケーションの場合には PACKED-DECIMAL をコーディングしておくべきです。COMPUTATIONAL-3 と COMP-3 は IBM の拡張機能であり、ISO/ANS COBOL ではサポートされないからです。これらのタイプに関連する *picture-string* (ピクチャー・ストリング) は S9(i)V9(d) (または 9 のインスタンスが *i* 回および *d* 回現れる S9...9V9...9) の形式になっていなければなりません。 $i + d$ は 63 以下でなければなりません。
2. COMPUTATIONAL と COMP は同じ働きをします。これらのデータ・タイプとそれが表すデータ・タイプに関連するピクチャー・ストリングはプロダクト固有になっています。したがって、COMP と COMPUTATIONAL は他のシステムでも実行されるようなアプリケーションの場合には使用してはなりません。COBOL for iSeries プログラムでは、これらのタイプに関連する *picture-string* (ピクチャー・ストリング) は S9(i)V9(d) (または 9 のインスタンスが *i* 回および *d* 回現れる S9...9V9...9) の形式になっていなければなりません。 $i + d$ は 63 以下でなければなりません。
3. level-1 (レベル 1) は、2 から 48 までの COBOL レベルを示します。

下図は、有効な数値ホスト変数宣言の構文を示しています。

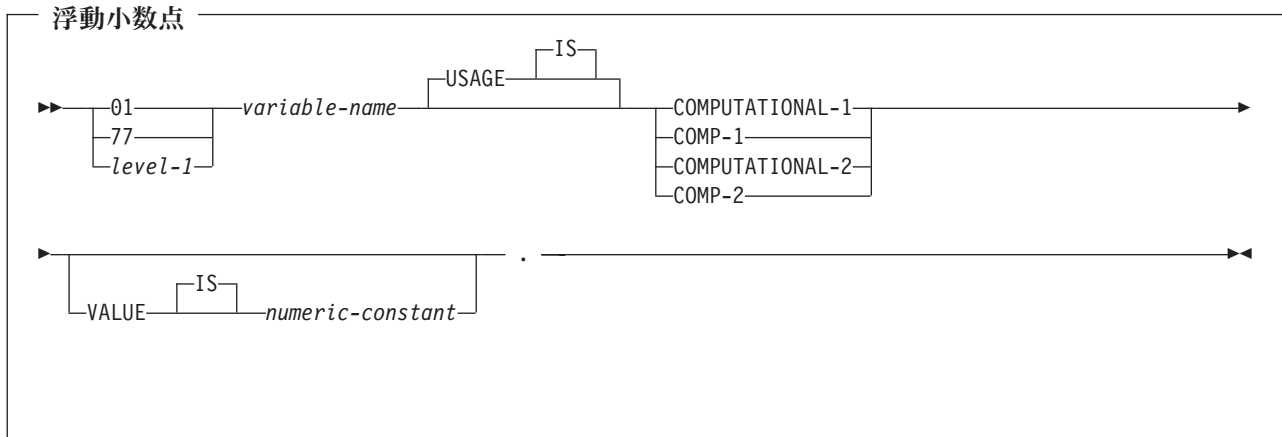


注:

1. SIGN LEADING SEPARATE および DISPLAY に関連する *picture-string* (ピクチャー・ストリング) は S9(i)V9(d) (または 9 のインスタンスが *i* 回および *d* 回現れる S9...9V9...9) の形式になっていなければなりません。i + d は 18 以下でなければなりません。
2. level-1 (レベル 1) は、2 から 48 までの COBOL レベルを示します。

SQL を使用する COBOL アプリケーションでの浮動小数点ホスト変数

次の図は、有効浮動小数点ホスト変数の宣言の構文を示しています。浮動小数点ホスト変数は、ILE COBOL for iSeries のみでサポートされます。



注:

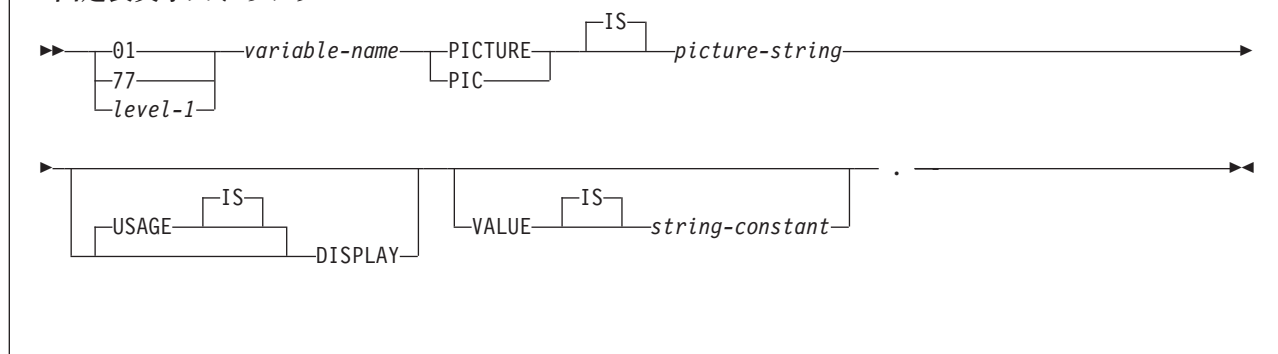
1. COMPUTATIONAL-1 と COMP-1 は同じ働きをします。COMPUTATIONAL-2 と COMP-2 は同じ働きをします。
2. level-1 (レベル 1) は、2 から 48 までの COBOL レベルを示します。

SQL を使用する COBOL アプリケーションでの文字ホスト変数

文字ホスト変数には、次の 2 つの形式があります。

- 固定長文字列
- 可変長文字列

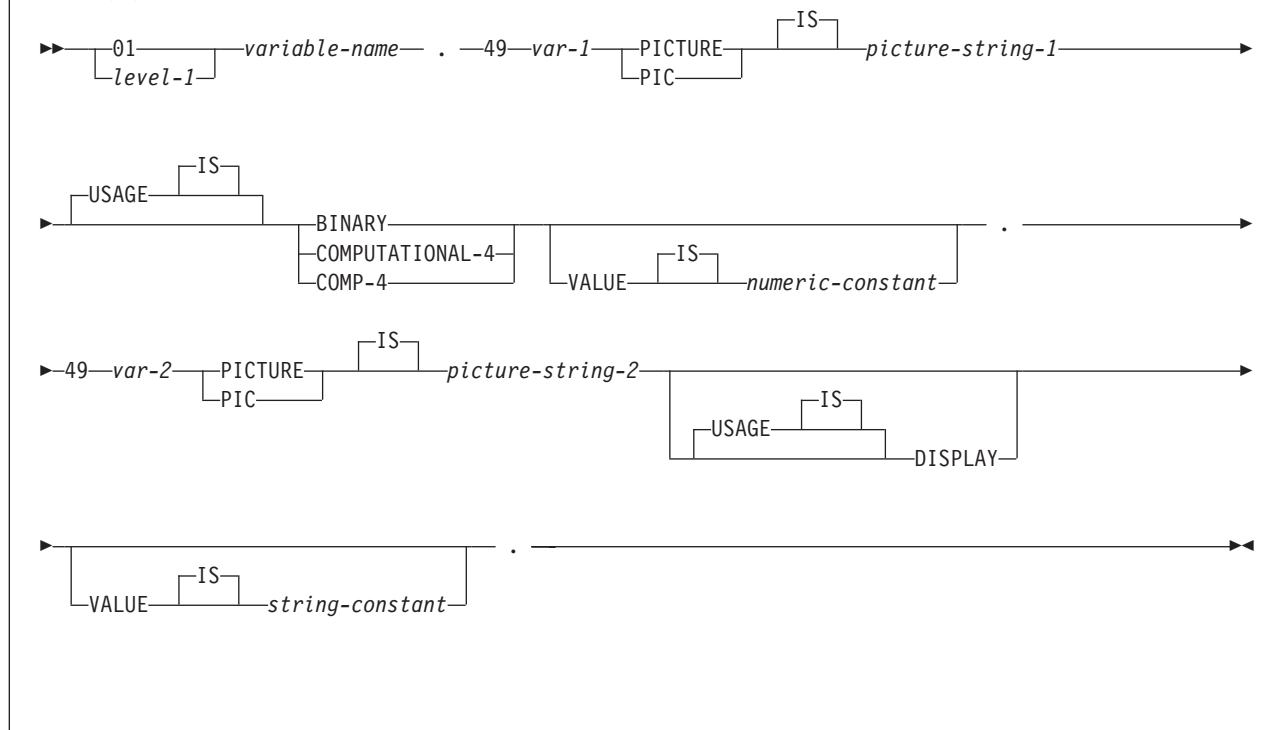
固定長文字列



注:

1. これらの形式に関連する *picture-string* (ピクチャー・STRING) は $X(m)$ (または X のインスタンスが m 回現れる $XXX...X$) で、 $1 \leq m \leq 32\,766$ になっていなければなりません。
2. level-1 (レベル 1) は、2 から 48 までの COBOL レベルを示します。

可変長文字列



注:

1. これらの形式に関連する *picture-string-1* (ピクチャー・STRING 1) は、 $S9(m)$ または 9 のインスタンスが m 回現れる $S9...9$ でなければなりません。 m は、1 から 4 まででなければなりません。

iSeries 上の COBOL で値が指定の精度までしか認識されない場合があっても、データベース・マネージャは S9(m) 変数の全桁を使用することに注意してください。この結果、COBOL ステートメントの実行中にデータ打ち切りエラーが生じることになるので、可変長文字列の最大長が事実上、指定の精度までに制限される場合があります。

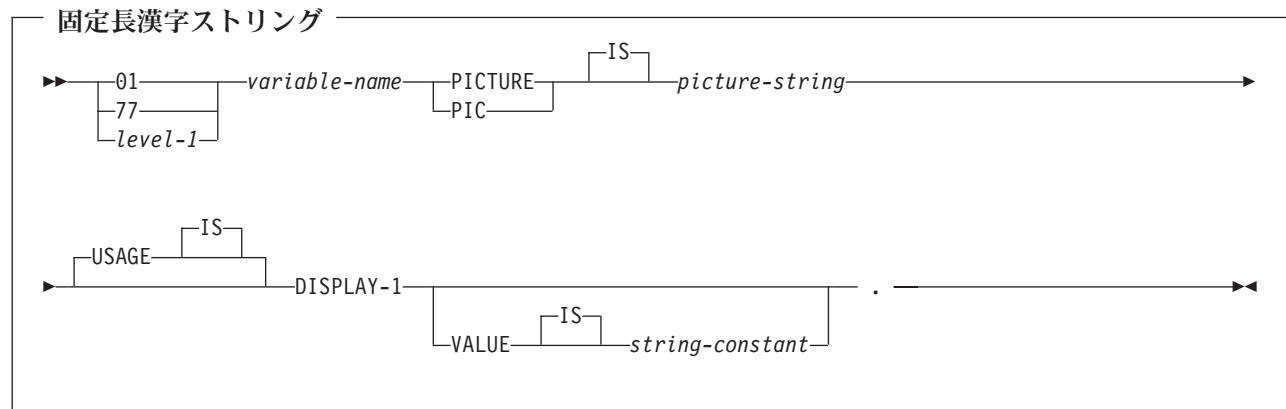
2. これらの形式に関連する *picture-string-2* (ピクチャー・ストリング 2) は X(m) または X のインスタンスが m 回現れる XX...X であって、 $1 \leq m \leq 32\,740$ でなければなりません。
3. *var-1* (変数 1) と *var-2* (変数 2) はホスト変数として使用できません。
4. *level-1* (レベル 1) は、2 から 48 までの COBOL レベルを示します。

SQL を使用する COBOL アプリケーションでのグラフィック・ホスト変数

グラフィック・ホスト変数は ILE COBOL for iSeries のみでサポートされます。

グラフィック・ホスト変数には、次の 2 つの形式があります。

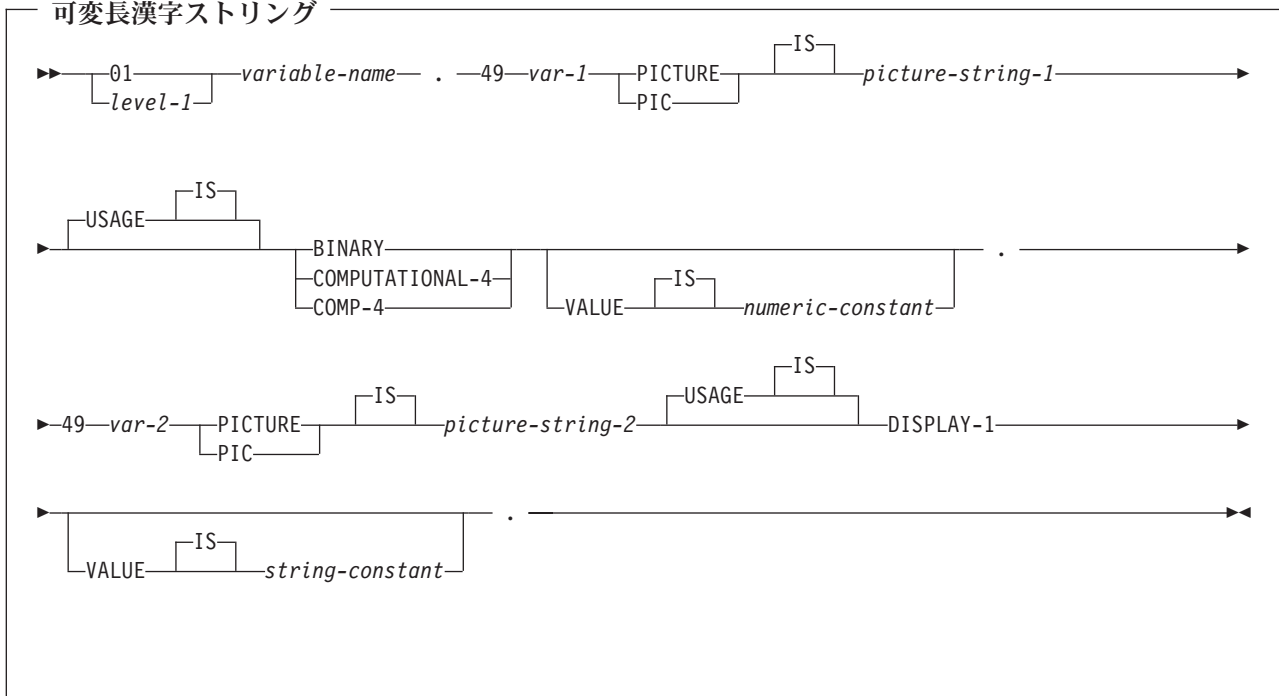
- 固定長漢字ストリング
- 可変長漢字ストリング



注:

1. これらの形式に関連する *picture-string* (ピクチャー・ストリング) は G(m) (または G のインスタンスが m 回現れる GGG...G) あるいは、N(m) (または N のインスタンスが m 回現れる NNN...N) で $1 \leq m \leq 16\,383$ になっていなければなりません。
2. *level-1* (レベル 1) は、2 から 48 までの COBOL レベルを示します。

可変長漢字ストリング



注:

1. これらの形式に関連する *picture-string-1* (ピクチャー・ストリング 1) は、S9(m) または 9 のインスタンスが m 回現れる S9...9 でなければなりません。m は、1 から 4 まででなければなりません。

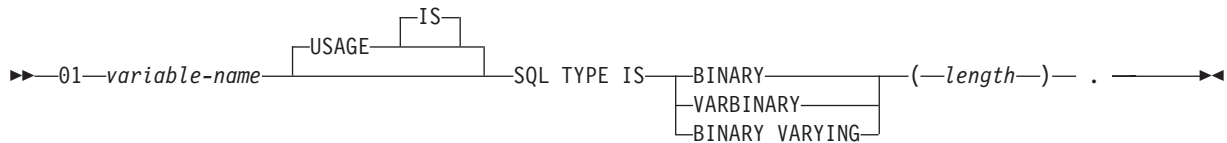
iSeries 上の COBOL で値が指定の精度までしか認識されない場合があっても、データベース・マネージャは S9(m) 変数の全桁を使用することに注意してください。この結果、COBOL ステートメントの実行中にデータ打ち切りエラーが生じることになるので、可変長漢字ストリングの最大長が事実上、指定の精度までに制限される場合があります。

2. これらの形式に関連する *picture-string-2* (ピクチャー・ストリング 2) は G(m)、G のインスタンスが m 回現れる GG...G、N(m)、または N のインスタンスが m 回現れる NN...N で、 $1 \leq m \leq 16\,370$ になっていなければなりません。
3. *var-1* (変数 1) と *var-2* (変数 2) はホスト変数として使用できません。
4. *level-1* (レベル 1) は、2 から 48 までの COBOL レベルを示します。

SQL を使用する COBOL アプリケーションでのバイナリー・ホスト変数

- | COBOL には、SQL バイナリー・データ・タイプに対応する変数がありません。これらのデータ・タイプ
- | で使用するホスト変数を作成するには、SQL TYPE IS 文節を使用します。SQL プリコンパイラーは、こ
- | の宣言を出カソース・メンバー内で、COBOL 言語構造に置き換えます。

BINARY および VARBINARY



注:

1. BINARY ホスト変数では、length (長さ) の範囲は 1 から 32766 まででなければなりません。
2. VARBINARY ホスト変数では、length (長さ) の範囲は 1 から 32740 まででなければなりません。
3. SQL TYPE IS、BINARY、VARBINARY、および BINARY VARYING は大文字小文字混合にすることができます。

BINARY の例

次のように宣言すると、

```
01 MY-BINARY SQL TYPE IS BINARY(200).
```

以下のような結果コードが生成されます。

```
01 MY-BINARY PIC X(200).
```

VARBINARY の例

次のように宣言すると、

```
01 MY-VARBINARY SQL TYPE IS VARBINARY(250).
```

以下の構造を生成します。

```
01 MY-VARBINARY.
```

```
49 MY-VARBINARY-LENGTH PIC 9(5) BINARY.
```

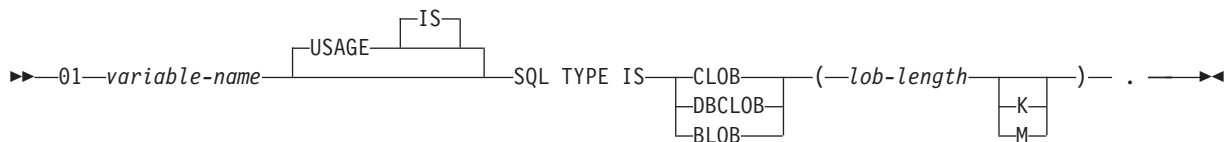
```
49 MY-VARBINARY-DATA PIC X(250).
```

SQL を使用する COBOL アプリケーションでの LOB ホスト変数

COBOL には、LOB (ラージ・オブジェクト) の SQL データ・タイプに対応する変数がありません。これらのデータ・タイプで使用するホスト変数を作成するには、SQL TYPE IS 文節を使用します。SQL プリコンパイラーは、この宣言を出カソース・メンバー内で、COBOL 言語構造に置き換えます。

LOB ホスト変数は、ILE COBOL for iSeries でのみサポートされています。

LOB ホスト変数



注:

1. BLOB および CLOB の場合、 $1 \leq \text{lob-length} \leq 15,728,640$ です。
2. DBCLOB の場合、 $1 \leq \text{lob-length} \leq 7,864,320$ です。
3. SQL TYPE IS、BLOB、CLOB、DBCLOB は大文字小文字にすることができます。

CLOB の例

次のように宣言すると、

```
01 MY-CLOB SQL TYPE IS CLOB(16384).
```

以下の構造を生成します。

```
01 MY-CLOB.  
  49 MY-CLOB-LENGTH PIC 9(9) BINARY.  
  49 MY-CLOB-DATA PIC X(16384).
```

DBCLOB の例

次のように宣言すると、

```
01 MY-DBCLOB SQL TYPE IS DBCLOB(8192).
```

以下の構造を生成します。

```
01 MY-DBCLOB.  
  49 MY-DBCLOB-LENGTH PIC 9(9) BINARY.  
  49 MY-DBCLOB-DATA PIC G(8192) DISPLAY-1.
```

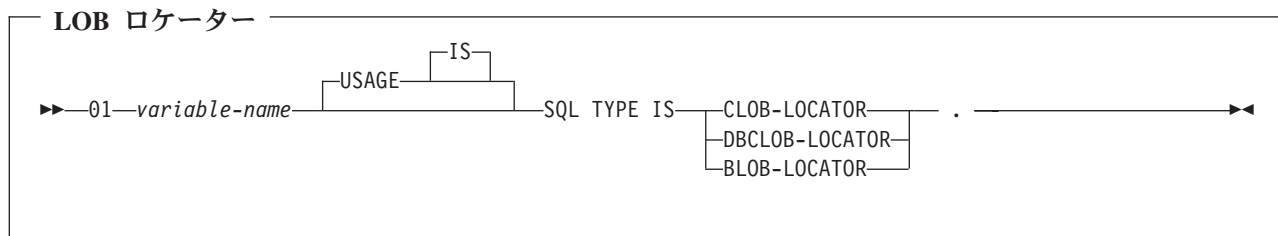
BLOB の例

次のように宣言すると、

```
01 MY-BLOB SQL TYPE IS BLOB(16384).
```

以下の構造を生成します。

```
01 MY-BLOB.  
  49 MY-BLOB-LENGTH PIC 9(9) BINARY.  
  49 MY-BLOB-DATA PIC X(16384).
```



注:

1. SQL TYPE IS、BLOB-LOCATOR、CLOB-LOCATOR、DBCLOB-LOCATOR は大文字小文字混合にすることができます。
2. LOB ロケーターは、SQL TYPE IS ステートメントの中で初期設定することはできません。

CLOB ロケーターおよび DBCLOB ロケーターの構文は、似ています。

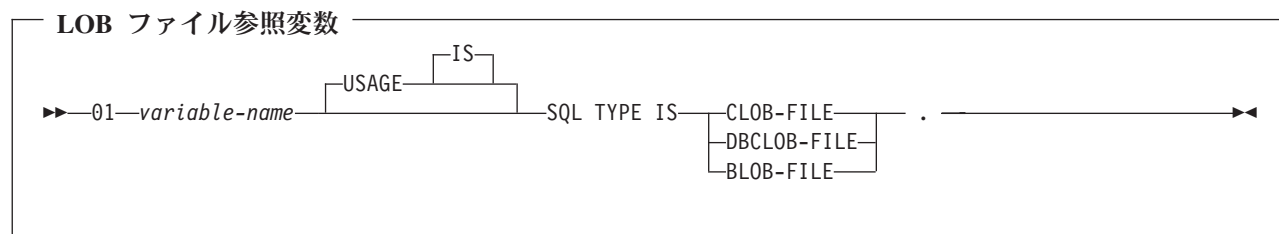
BLOB ロケーターの例

次のように宣言すると、

```
01 MY-LOCATOR SQL TYPE IS BLOB_LOCATOR.
```

以下の構造を生成します。

```
01 MY-LOCATOR PIC 9(9) BINARY.
```



注: SQL TYPE IS、BLOB-FILE、CLOB-FILE、DBCLOB-FILE は大文字小文字混合にすることができます。

BLOB ファイル参照の例

次のように宣言すると、

```
01 MY-FILE SQL TYPE IS BLOB-FILE.
```

以下の構造を生成します。

```
01 MY-FILE.  
  49 MY-FILE-NAME-LENGTH PIC S9(9) COMP-5.  
  49 MY-FILE-DATA-LENGTH PIC S9(9) COMP-5.  
  49 MY-FILE-FILE-OPTIONS PIC S9(9) COMP-5.  
  49 MY-FILE-NAME PIC X(255).
```

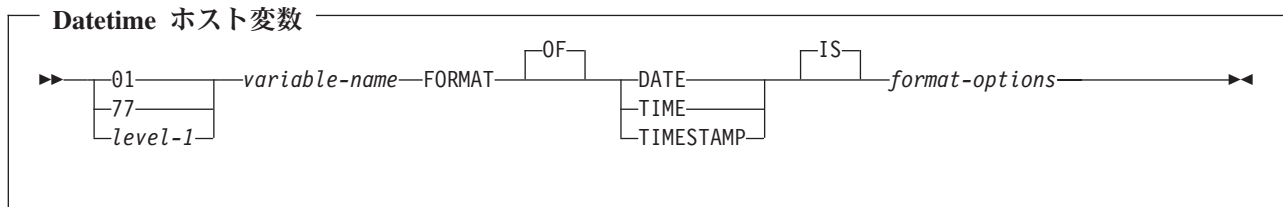
CLOB ファイル参照変数と DBCLOB ファイル参照変数は、類似の構文をもっています。

プリコンパイラーは、次のファイル・オプション定数に対する宣言を生成します。ファイル参照ホスト変数を使用する場合、これらの定数を使用して、xxx-FILE-OPTIONS 変数を設定できます。これらの値の詳細については、「SQL プログラミング 概念」の『LOB ファイル参照変数』を参照してください。

- SQL_FILE_READ (2)
- SQL_FILE_CREATE (8)
- SQL_FILE_OVERWRITE (16)
- SQL_FILE_APPEND (32)

SQL を使用する COBOL アプリケーションでの Datetime ホスト変数

次の図は、有効な日付、時刻、およびタイム・スタンプのホスト変数宣言の構文を示しています。Datetime ホスト変数は、ILE COBOL for iSeries でのみサポートされています。



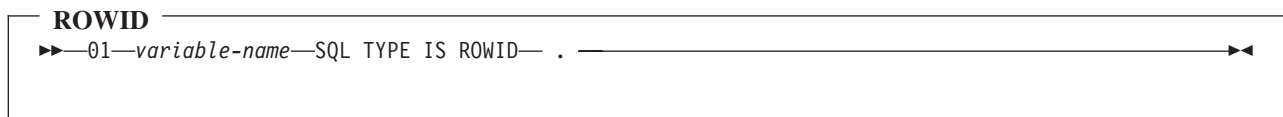
注:

1. *level-1* (レベル 1) は、2 から 48 までの COBOL レベルを示します。
2. *format-options* は、COBOL コンパイラーでサポートされる有効な datetime オプションを示します。詳

細は、V5R1 補足資料 Web サイトの「ILE COBOL 解説書 」を参照してください。

SQL を使用する COBOL アプリケーションでの ROWID ホスト変数

COBOL には、ROWID の SQL データ・タイプに対応する変数がありません。このデータ・タイプで使用するホスト変数を作成するには、SQL TYPE IS 文節を使用します。SQL プリコンパイラーは、この宣言を出力ソース・メンバー内で、COBOL 言語構造に置き換えます。



注: SQL TYPE IS ROWID は、大文字小文字混合にすることができます。

ROWID の例

次のように宣言すると、

```
01 MY-ROWID SQL TYPE IS ROWID.
```

以下の構造を生成します。

```
01 MY-ROWID.
   49 MY-ROWID-LENGTH PIC 9(2) BINARY.
   49 MY-ROWID-DATA PIC X(40).
```

SQL を使用する COBOL アプリケーションでのホスト構造の使用

ホスト構造は、ユーザーのプログラムの DATA DIVISION の中で定義されている一連のホスト変数に名前を付けたものです。ホスト構造はそれ自体が複数レベルの構造の中に置かれることがあっても、その最大レベルは 2 レベルまでです。可変長の文字ストリングの宣言だけは例外で、その場合は別のレベルが必要であり、これはレベル 49 でなければなりません。

ホスト構造名をグループ名とし、その従属レベルで基本データ項目の名前を指定することができます。以下に、例を示します。

```
01 A
   02 B
      03 C1 PICTURE ...
      03 C2 PICTURE ...
```

この例では、B は基本データ項目 C1 と C2 から成るホスト構造の名前です。

(たとえば、構造内のフィールドを識別するために) 修飾ホスト変数名を使用して SQL ステートメントを書くときには、構造の名前の後にピリオドとフィールドの名前を続けます。たとえば、C1 OF B または C1 IN B ではなく B.C1 を指定してください。ただし、この形式は、SQL ステートメントの中の修飾名だけに適用されます。この技法を用いて COBOL ステートメント内で修飾名を書くことはできません。

次のいずれかの項目を検出した場合は、ホスト構造は完全と見なされます。

- 領域 A で始まる必要のある COBOL 項目
- 任意の SQL ステートメント (ただし、SQL INCLUDE を除く)

ホスト構造を定義しておけば、いくつかのホスト変数 (ホスト構造を構成しているデータ項目の名前) を個別に指定しなくても、SQL ステートメントの中でそのホスト構造を参照することができます。

たとえば、次のようにコーディングすれば、テーブル CORPDATA.EMPLOYEE から選択した行のすべての列の値を検索することができます。

```
01 PEMPL.
   10 EMPNO                PIC X(6).
   10 FIRSTNME.
       49 FIRSTNME-LEN     PIC S9(4) USAGE BINARY.
       49 FIRSTNME-TEXT   PIC X(12).
   10 MIDINIT              PIC X(1).
   10 LASTNAME.
       49 LASTNAME-LEN     PIC S9(4) USAGE BINARY.
       49 LASTNAME-TEXT   PIC X(15).
   10 WORKDEPT             PIC X(3).
...
MOVE "000220" TO EMPNO.
...
EXEC SQL
  SELECT *
  INTO :PEMPL
  FROM CORPDATA.EMPLOYEE
  WHERE EMPNO = :EMPNO
END-EXEC.
```

上の例に示すように、PEMPL の宣言の中で、2 つの変長文字列要素、FIRSTNME と LASTNAME が構造に組み込まれています。

詳細については、以下のセクションを参照してください。

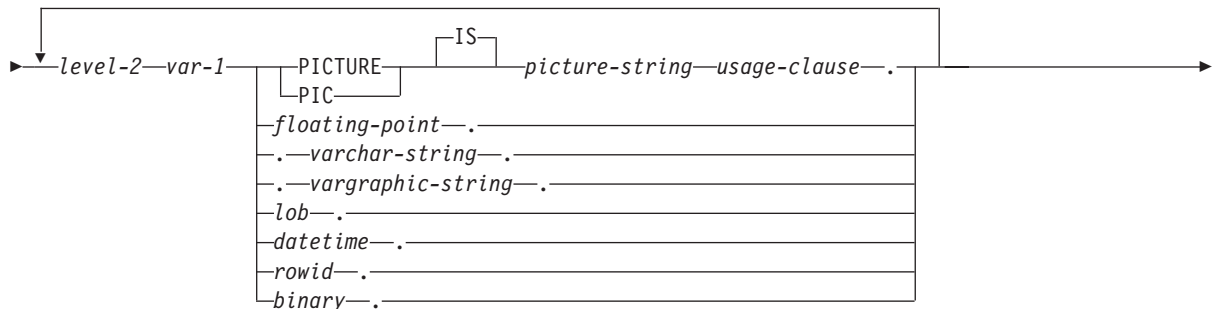
- 『SQL を使用する COBOL アプリケーションでのホスト構造』
- 74 ページの『SQL を使用する COBOL アプリケーションでのホスト構造標識配列』
- 75 ページの『SQL を使用する COBOL アプリケーションでのホスト構造配列の使用』
- 76 ページの『SQL を使用する COBOL アプリケーションでのホスト構造配列』
- 79 ページの『SQL を使用する COBOL アプリケーションでのホスト配列標識構造』

SQL を使用する COBOL アプリケーションでのホスト構造

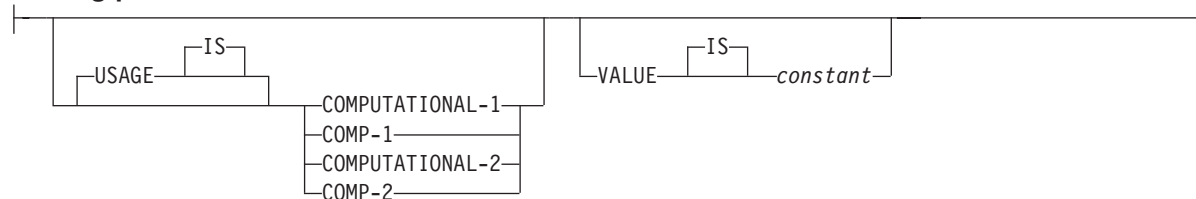
次の図は、有効なホスト構造の構文を示しています。

ホスト構造

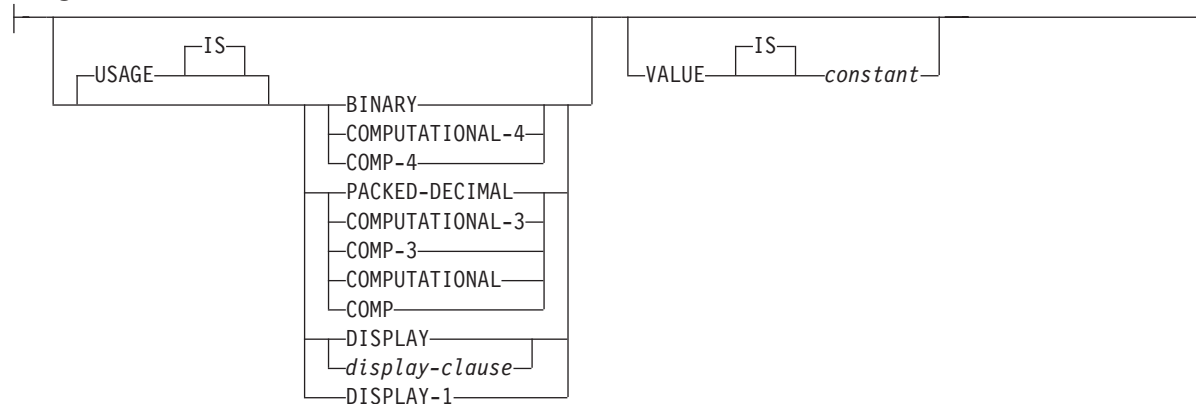
▶ *level-1* *variable-name* .



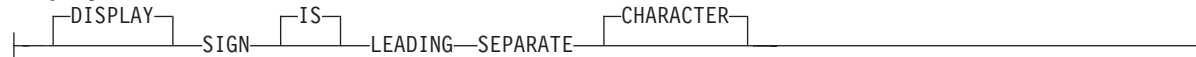
floating-point:



usage-clause:

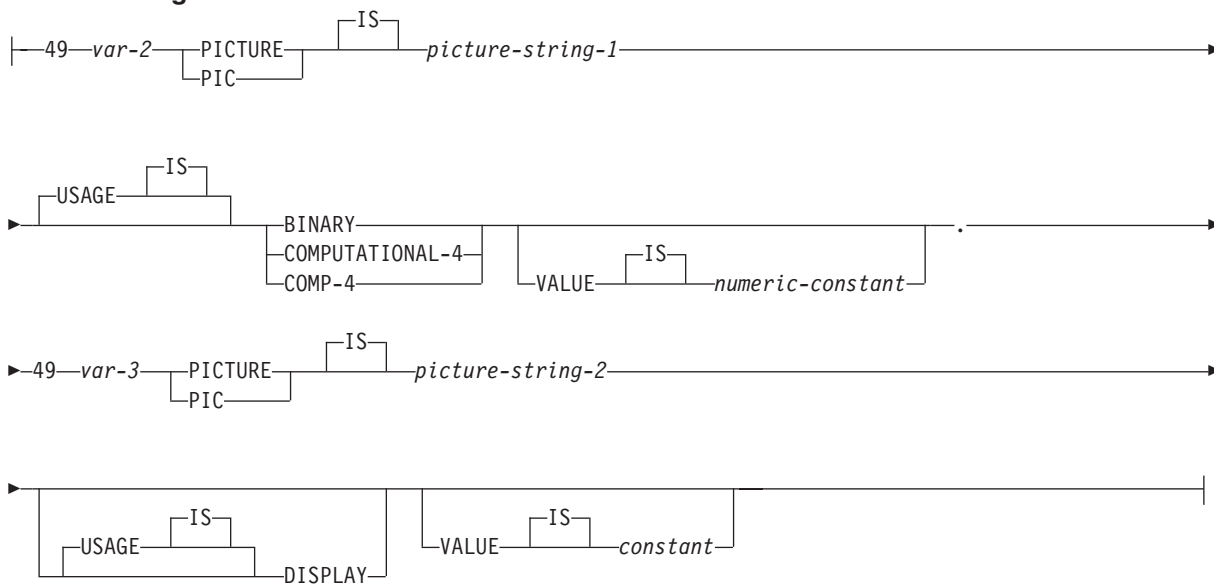


display-clause:

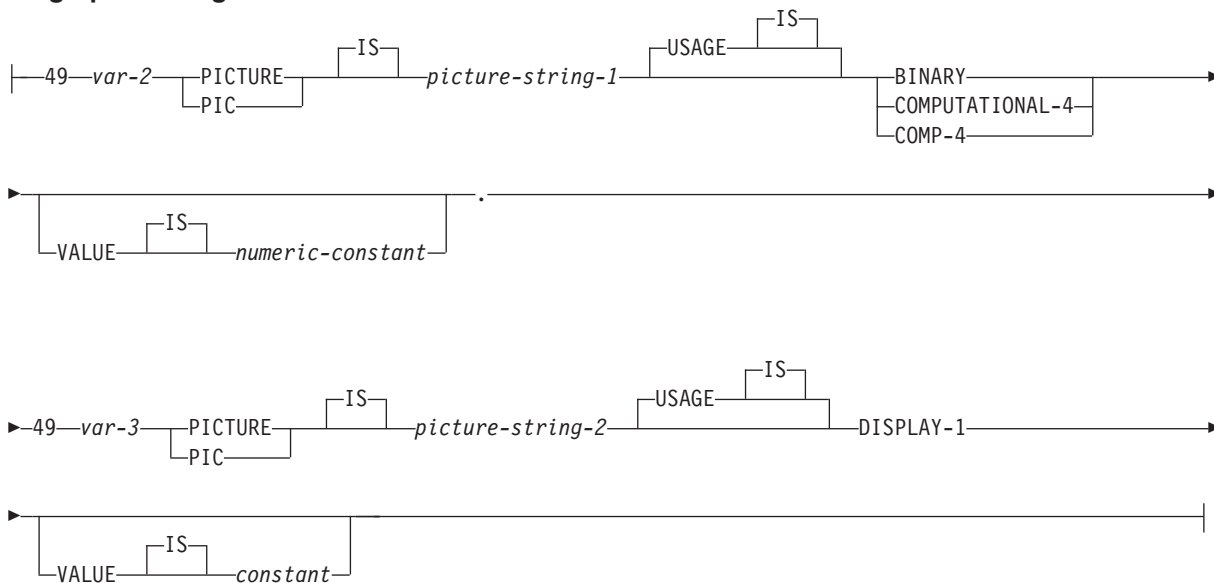


ホスト構造 (続き)

varchar-string:

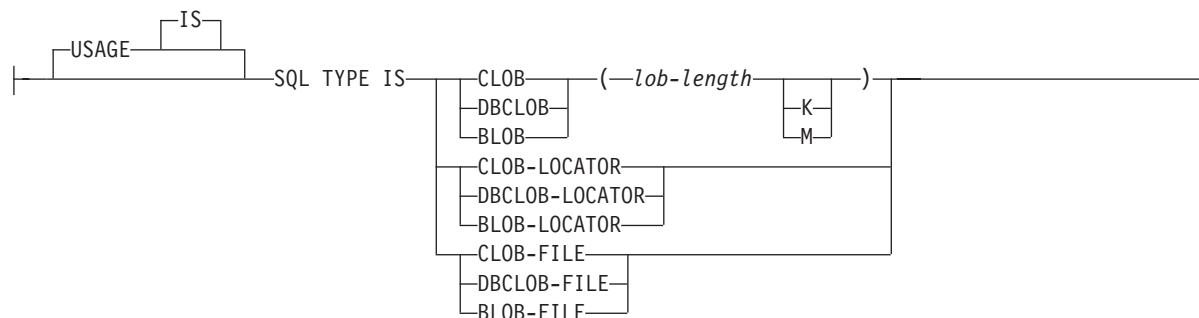


vargraphic-string:

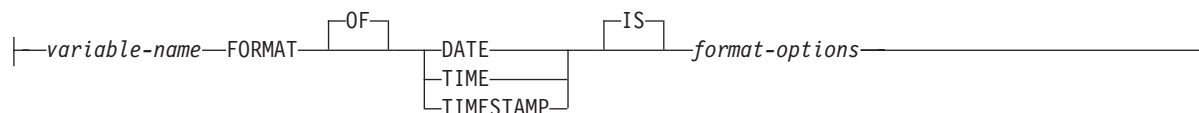


ホスト構造 (続き)

lob:



datetime:



rowid:



binary:



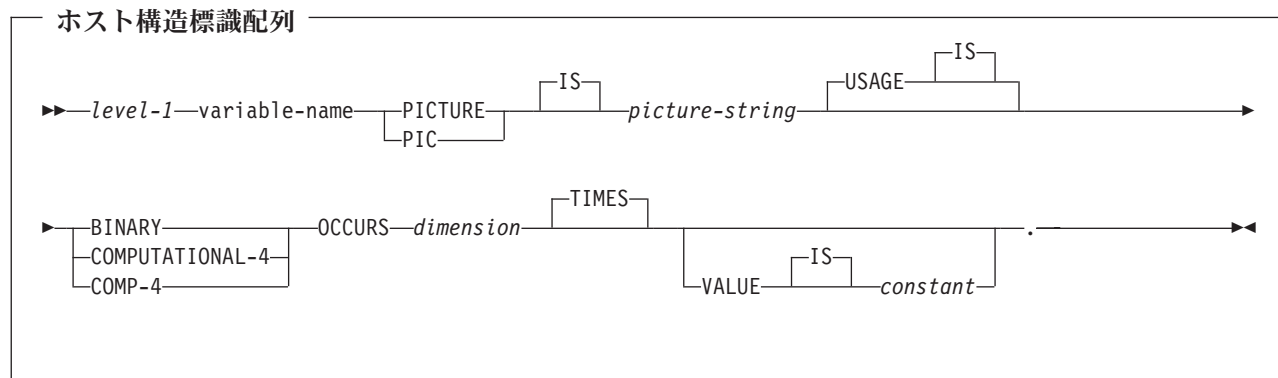
注:

1. level-1 (レベル 1) は 1 から 47 までの COBOL レベルを示します。
2. level-2 (レベル 2) は 2 から 48 までの COBOL レベルを示します。ただし、level-2 > level-1 でなければなりません。
3. グラフィック・ホスト変数、LOB ホスト変数、および浮動小数点ホスト変数は、ILE COBOL for iSeries のみでサポートされます。
4. 数値ホスト変数、文字ホスト変数、グラフィック・ホスト変数、LOB ホスト変数、ROWID ホスト変数、およびバイナリー・ホスト変数の宣言についての詳細は、それぞれ数値、文字、グラフィック、LOB、ROWID、およびバイナリーの各ホスト変数に関する注意事項を参照してください。
5. *format-options* は、COBOL コンパイラーでサポートされる有効な datetime オプションを示します。詳

細は、V5R1 補足資料 Web サイトの「ILE COBOL 解説書 」を参照してください。

SQL を使用する COBOL アプリケーションでのホスト構造標識配列

次の図は有効な標識配列宣言の構文を示しています。



注:

1. dimension (次元) は 1 から 32767 までの整数でなければなりません。
2. level-1 (レベル 1) は 2 から 48 までの整数でなければなりません。
3. BINARY、COMPUTATIONAL-4、および COMP-4 は同じ働きをします。他のシステムでも実行されるようなアプリケーションの場合には BINARY をコーディングしておくべきです。
COMPUTATIONAL-4 と COMP-4 は IBM の拡張機能であり、ISO/ANSI COBOL ではサポートされないからです。これらのタイプに関連する *picture-string* (ピクチャー・ストリング) は S9(i) (または 9 のインスタンスが i 回現れる S9...9) でなければなりません。i は 4 以下でなければなりません。

SQL を使用する COBOL アプリケーションでのホスト構造配列の使用

ホスト構造配列は、プログラムのデータ部の中で定義され、OCCURS 文節が付いている一連のホスト変数に名前を付けたものです。ホスト構造が複数レベルの構造に現れる場合があっても、ホスト構造配列のレベルは最高 2 レベルまでです。可変長文字列には、別のレベル、すなわちレベル 49 が必要です。ホスト構造配列名をグループ名とし、その従属レベルで基本データ項目の名前を指定することができます。

このような例では、以下の条件が該当します。

- B-ARRAY のすべてのメンバーは有効でなければなりません。
- B-ARRAY は修飾できません。
- B-ARRAY はブロック化形式の FETCH および INSERT ステートメントでのみ使用できます。
- B-ARRAY は、データ項目 C1-VAR および C2-VAR からなるホスト構造の配列名です。
- SYNCHRONIZED 属性は指定してはなりません。
- C1-VAR および C2-VAR は、SQL ステートメントでは有効なホスト変数ではありません。構造に中間レベルの構造を含めることはできません。

```
01 A-STRUCT.
  02 B-ARRAY OCCURS 10 TIMES.
    03 C1-VAR PIC X(20).
    03 C2-VAR PIC S9(4).
```

CORPDATA.DEPARTMENT テーブルから 10 行分検索する場合は、次の例を使用してください。

```
01 TABLE-1.
  02 DEPT OCCURS 10 TIMES.
    05 DEPTNO PIC X(3).
    05 DEPTNAME.
      49 DEPTNAME-LEN PIC S9(4) BINARY.
      49 DEPTNAME-TEXT PIC X(29).
    05 MGRNO PIC X(6).
```

```
      05 ADMRDEPT PIC X(3).
01  TABLE-2.
      02 IND-ARRAY OCCURS 10 TIMES.
          05 INDS PIC S9(4) BINARY OCCURS 4 TIMES.

.....
EXEC SQL
DECLARE C1 CURSOR FOR
      SELECT *
      FROM CORPDATA.DEPARTMENT
END-EXEC.

.....
EXEC SQL
      FETCH C1 FOR 10 ROWS INTO :DEPT :IND-ARRAY
END-EXEC.
```

SQL を使用する COBOL アプリケーションでのホスト構造配列

次の図は、有効なホスト構造配列宣言の構文を示しています。

ホスト構造配列

▶▶ *level-1* *variable-name* OCCURS *dimension* ^{TIMES} .

▶ *level-2* *var-1* ^{IS} *picture-string-1* *usage-clause* .

<i>floating-point</i> .
<i>varchar-string</i> .
<i>vgraphic-string</i> .
<i>lob</i> .
<i>datetime</i> .
<i>rowid</i> .
<i>binary</i> .

floating-point:

^{IS} *usage* ^{IS} *value* *constant*

COMPUTATIONAL-1
COMP-1
COMPUTATIONAL-2
COMP-2

usage-clause:

^{IS} *usage* ^{IS} *value* *constant*

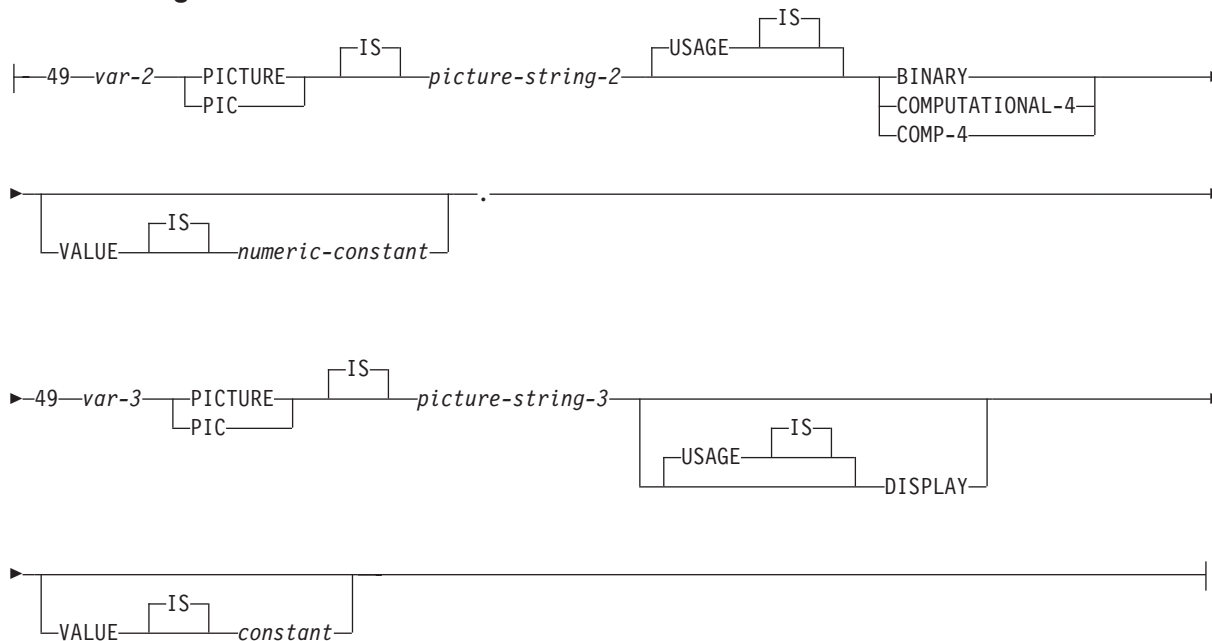
BINARY
COMPUTATIONAL-4
COMP-4
PACKED-DECIMAL
COMPUTATIONAL-3
COMP-3
COMPUTATIONAL
COMP
DISPLAY
<i>display-clause</i>
DISPLAY-1

display-clause:

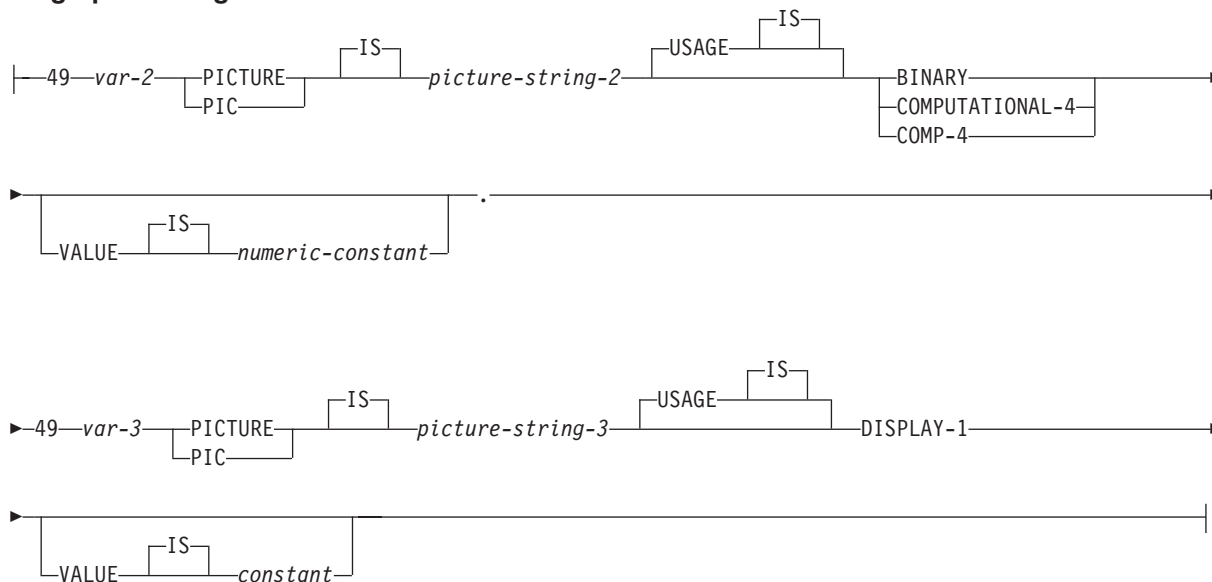
^{IS} *display* *sign* ^{IS} *leading* *separate* *character*

ホスト構造配列 (続き)

varchar-string:

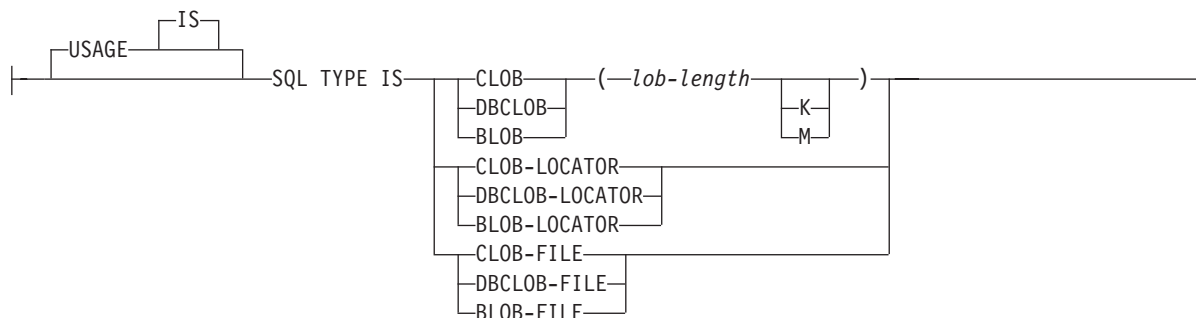


vargraphic-string:

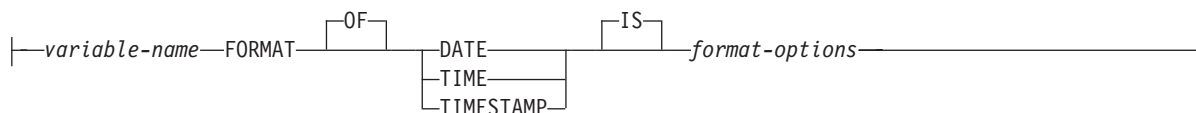


ホスト構造配列 (続き)

lob:



datetime:



rowid:



binary:



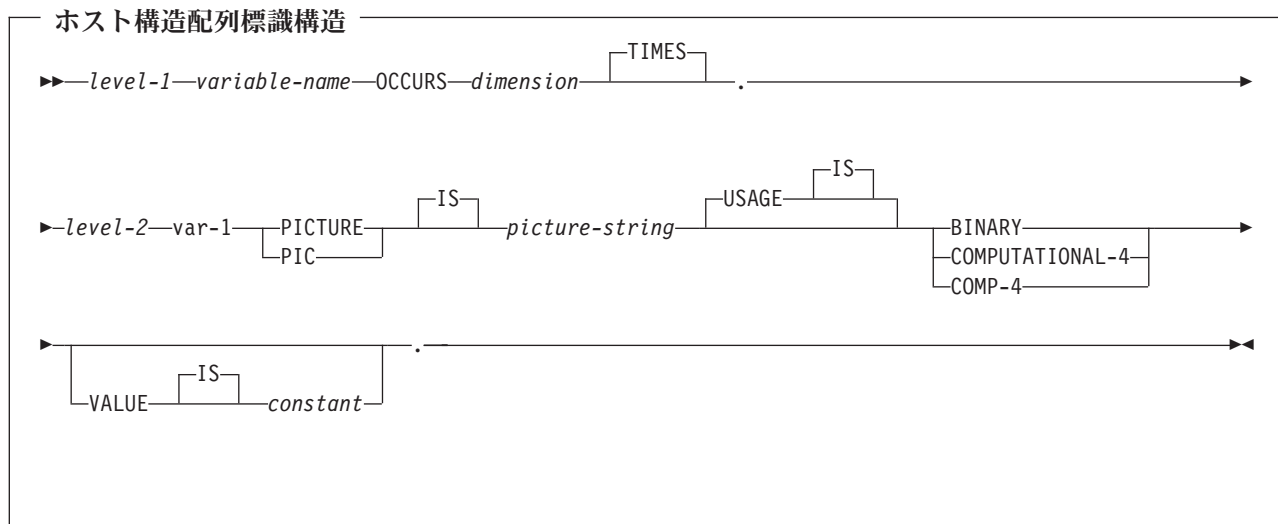
注:

1. level-1 (レベル 1) は 2 から 47 までの COBOL レベルを示します。
2. level-2 (レベル 2) は 3 から 48 までの COBOL レベルを示します。ただし、level-2 > level-1 でなければなりません。
3. グラフィック・ホスト変数、LOB ホスト変数、および浮動小数点ホスト変数は、ILE COBOL for iSeries のみでサポートされます。
4. 数値ホスト変数、文字ホスト変数、グラフィック・ホスト変数、LOB ホスト変数、ROWID ホスト変数、およびバイナリー・ホスト変数の宣言についての詳細は、それぞれ数値、文字、グラフィック、LOB、ROWID、およびバイナリーの各ホスト変数に関する注意事項を参照してください。
5. dimension (次元) は 1 から 32767 までの整数定数でなければなりません。
6. *format-options* は、COBOL コンパイラーでサポートされる有効な datetime オプションを示します。詳細は、V5R1 補足資料 Web サイトの「ILE COBOL 解説書」を参照してください。

細は、V5R1 補足資料 Web サイトの「ILE COBOL 解説書」を参照してください。

SQL を使用する COBOL アプリケーションでのホスト配列標識構造

次の図は、ホスト構造配列標識として有効な構文を示しています。



注:

1. level-1 (レベル 1) は、2 から 48 までの COBOL レベルを示します。
2. level-2 (レベル 2) は 3 から 48 までの COBOL レベルを示します。ただし、level-2 > level-1 でなければなりません。
3. dimension (次元) は 1 から 32767 までの整数定数でなければなりません。
4. BINARY、COMPUTATIONAL-4、および COMP-4 は同じ働きをします。他のシステムでも実行されるようなアプリケーションの場合には BINARY をコーディングしておくべきです。
COMPUTATIONAL-4 と COMP-4 は IBM の拡張機能であり、ISO/ANSI COBOL ではサポートされないからです。これらのタイプに関連する *picture-string* (ピクチャー・ストリング) は S9(i) (または 9 のインスタンスが i 回現れる S9...9) でなければなりません。i は 4 以下でなければなりません。

SQL を使用する COBOL アプリケーションでの外部ファイル記述の使用

SQL は、ファイル定義からホスト変数を検索するために、COPY DD 様式名、COPY DD-ALL-FORMATS、COPY DDS 様式名、COPY DDR 様式名、COPY DDR-ALL-FORMATS、COPY DDSR 様式名、COPY DDS-ALL-FORMATS、および COPY DDSR-ALL-FORMATS を使用します。REPLACING オプションの指定があるときは、完全名での書き換えだけが行われます。変数 1 は、様式名およびフィールド名と突き合わされて比較されます。両者が同じであれば、変数 2 が新しい名前として使用されます。

注: COBOL 予約語がフィールド名として使用されているファイル定義からホスト変数を検索することはできません。COBOL ホスト構造の中に、COPY DDx-format ステートメントを入れなければなりません。

「DB2 UDB for iSeries SQL プログラミング 概念」の『DB2 UDB for iSeries サンプル・テーブル』に記載されているサンプル・テーブル DEPARTMENT の定義を検索するときは、次のようにコーディングすることができます。

```
01 DEPARTMENT-STRUCTURE.
   COPY DDS-ALL-FORMATS OF DEPARTMENT.
```


DEPARTMENT-STRUCTURE という名前のホスト構造は、DEPARTMENT-RECORD という名前の 05 レベルのフィールドをもつものとして定義されており、そのフィールドには、さらに DEPTNO、DEPTNAME、MGRNO、および ADMRDEPT という名前の 4 つの 06 レベルのフィールドが含まれています。これらのフィールド名は、SQL ステートメントの中でホスト変数として使用できます。COBOL COPY verb の詳細については、V5R1 補足資料 Web サイトの「COBOL/400[®] User's Guide

(COBOL/400 使用者の手引き )、および「ILE COBOL 解説書 」を参照してください。

外部ファイル記述の詳細については、『SQL を使用する COBOL アプリケーションでのホスト構造配列の外部ファイル記述の使用』を参照してください。

SQL を使用する COBOL アプリケーションでのホスト構造配列の外部ファイル記述の使用

COBOL では外部記述データを組み入れるときに追加のレベルを作成するので、その前の 04 レベルに OCCURS 文節を置かなければなりません。05 レベルで追加の宣言を指定しても、その宣言は構造に入れることはできません。

FILLER として生成されたフィールドがファイルに含まれている場合は、構造をホスト構造配列として使用できません。

装置ファイルの場合は、INDARA の指定がなく、ファイルに標識が入っているときには、その宣言をホスト構造配列として使用できません。生成された構造には標識域が組み入れられ、その標識域があるためにレコード用の記憶域が連続しなくなります。

たとえば、次の例は、COPY-DDS を使用してホスト構造配列を生成し、10 行取り出してそのホスト構造配列に入れる方法を示しています。

```
01 DEPT.
   04 DEPT-ARRAY OCCURS 10 TIMES.
   COPY DDS-ALL-FORMATS OF DEPARTMENT.
   ...

EXEC SQL DECLARE C1 CURSOR FOR
      SELECT * FROM CORPDATA.DEPARTMENT
END EXEC.

EXEC SQL OPEN C1
END-EXEC.

EXEC SQL FETCH C1 FOR 10 ROWS INTO :DEPARTMENT
END-EXEC.
```

注: DATE、TIME、および TIMESTAMP の各列からは、文字ホスト変数定義が生成されます。SQL により、DATE、TIME、および TIMESTAMP 列と同じ比較規則と割り当て規則を使用して扱われます。たとえば、日付ホスト変数は、DATE 列または日付の有効な表現である文字ストリングとだけ突き合わされて比較されます。

| COBOL for iSeries では、GRAPHIC と VARGRAPHIC は文字変数にマップされますが、SQL はこれ
| らを GRAPHIC 変数および VARGRAPHIC 変数と見なします。GRAPHIC 列または VARGRAPHIC
| 列が UCS-2 CCSID を持つ場合、生成されるホスト変数には UCS-2 CCSID が割り当てられます。
| GRAPHIC 列または VARGRAPHIC 列が UTF-16 CCSID を持つ場合、生成されるホスト変数には
| UTF-16 CCSID が割り当てられます。

SQL データ・タイプと COBOL データ・タイプの対応関係の判別

プリコンパイラーは、次の表に基づいて、ホスト変数のベース SQLTYPE とベース SQLLEN を判断します。ホスト変数が標識変数と一緒に記載されているときは、その SQLTYPE はベース SQLTYPE に 1 を加えたものです。

表 3. COBOL 宣言と代表的 SQL データ・タイプとの対応関係

COBOL データ・タイプ	ホスト変数の SQLTYPE	ホスト変数の SQLLEN	SQL データ・タイプ
S9(i)V9(d) COMP-3 または S9(i)V9(d) COMP または S9(i)V9(d) PACKED-DECIMAL	484	バイト 1 には i+d, バイト 2 には d	DECIMAL(i+d,d)
S9(i)V9(d) DISPLAY SIGN LEADING SEPARATE	504	バイト 1 には i+d, バイト 2 には d	正確に対応するものなし。 DECIMAL(i+d,d) または NUMERIC(i+d,d)
S9(i)V9(d)DISPLAY	488	バイト 1 には i+d, バイト 2 には d	NUMERIC(i+d,d)
S9(i) BINARY または S9(i) COMP-4 (i は 1 から 4 まで)	500	2	SMALLINT
S9(i) BINARY または S9(i) COMP-4 (i は 5 から 9 まで)	496	4	INTEGER
S9(i) BINARY または S9(i) COMP-4 (i は 10 から 18 まで) COBOL for iSeries ではサポートなし。	492	8	BIGINT
S9(i)V9(d) BINARY または S9(i)V9(d) COMP-4 (i+d ≤ 4)	500	バイト 1 には i+d, バイト 2 には d	正確に対応するものなし。 DECIMAL(i+d,d) または NUMERIC(i+d,d)
S9(i)V9(d) BINARY または S9(i)V9(d) COMP-4 (4 < i+d ≤ 9)	496	バイト 1 には i+d, バイト 2 には d	正確に対応するものなし。 DECIMAL(i+d,d) または NUMERIC(i+d,d)
COMP-1 COBOL for iSeries ではサポートなし。	480	4	FLOAT (単精度)
COMP-2 COBOL for iSeries ではサポートなし。	480	8	FLOAT (倍精度)
固定長文字データ	452	m	CHAR(m)
I 可変長文字データ	448	m	VARCHAR(m)
固定長グラフィック・データ COBOL for iSeries ではサポートなし。	468	m	GRAPHIC(m)
可変長グラフィック・データ COBOL for iSeries ではサポートなし。	464	m	VARGRAPHIC(m)
DATE COBOL for iSeries ではサポートなし。	384		DATE
TIME COBOL for iSeries ではサポートなし。	388		TIME

表3. COBOL 宣言と代表的 SQL データ・タイプとの対応関係 (続き)

COBOL データ・タイプ	ホスト変数の SQLTYPE	ホスト変数の SQLLEN	SQL データ・タイプ
TIMESTAMP COBOL for iSeries ではサポートなし。	392	26	TIMESTAMP

下表を参照すると、各 SQL データ・タイプに対応する COBOL データ・タイプを判断することができます。

表4. SQL データ・タイプと代表的な COBOL 宣言との対応関係

SQL データ・タイプ	COBOL データ・タイプ	注
SMALLINT	S9(m) COMP-4	m は 1 から 4 まで。
INTEGER	S9(m) COMP-4	m は 5 から 9 まで。
BIGINT	ILE COBOL for iSeries の S9(m) COMP-4。 COBOL for iSeries ではサポートなし。	m は 10 から 18 まで。
DECIMAL(p,s)	p<64 の場合: S9(p-s)V9(s) PACKED-DECIMAL または S9(p-s)V9(s) COMP または S9(p-s)V9(s) COMP-3。 p>63 の場合: サポートなし。	p は精度、s は位取りです。 0<=s<=p<=63。 s=0 の場合は S9(p) または S9(p)V を使用します。 s=p の場合は、SV9(s) を使用します。
NUMERIC(p,s)	p<19 の場合: S9(p-s)V9(s) DISPLAY。 p>18 の場合: サポートなし。	p は精度、s は位取りです。 0<=s<=p<=18。 s=0 の場合は S9(p) ま たは S9(p)V を使用します。 s=p の 場合は、SV9(s) を使用します。
FLOAT (単精度)	ILE COBOL for iSeries の COMP-1。 COBOL for iSeries ではサポートなし。	
FLOAT (倍精度)	ILE COBOL for iSeries の COMP-2。 COBOL for iSeries ではサポートなし。	
CHAR(n)	固定長文字ストリング	32766≥n≥1
VARCHAR(n)	可変長文字ストリング	32740≥n≥1
CLOB	なし	SQL TYPE IS を使用して ILE COBOL for iSeries 用の CLOB を宣 言します。 COBOL for iSeries ではサポートなし。
GRAPHIC(n)	ILE COBOL for iSeries の固定長漢字 ストリング。 COBOL for iSeries ではサポートなし。	16383≥n≥1
VARGRAPHIC(n)	ILE COBOL for iSeries の可変長漢字 ストリング。 COBOL for iSeries ではサポートなし。	16370≥n≥1

表 4. SQL データ・タイプと代表的な COBOL 宣言との対応関係 (続き)

SQL データ・タイプ	COBOL データ・タイプ	注
DBCLOB	なし	SQL TYPE IS を使用して ILE COBOL for iSeries 用の DBCLOB を宣言します。
BINARY	なし	SQL TYPE IS を使用して BINARY を宣言します。
VARBINARY	なし	SQL TYPE IS を使用して VARBINARY を宣言します。
BLOB	なし	SQL TYPE IS を使用して BLOB を宣言します。
DATE	固定長文字ストリングまたは DATE (ILE COBOL for iSeries 用)。	形式が *USA、*JIS、*EUR、または *ISO のときは、少なくとも 10 文字が必要。形式が *YMD、*DMY、または *MDY のときは、少なくとも 8 文字が必要。形式が *JUL のときは、少なくとも 6 文字が必要。
TIME	固定長文字ストリングまたは TIME (ILE COBOL for iSeries 用)。	少なくとも 6 文字が必要。秒を含む場合は、8 文字が必要。
TIMESTAMP	固定長文字ストリングまたは TIMESTAMP (ILE COBOL for iSeries 用)。	n は少なくとも 19 が必要。マイクロ秒を全桁の精度で含める場合は、n は少なくとも 26 が必要。n が 26 未満のときは、マイクロ秒部分で切り捨てが起こる。
DATALINK	サポートなし	
ROWID	なし	SQL TYPE IS を使用して ROWID を宣言します。

詳細については、『COBOL 変数宣言と使用上の注意事項』を参照してください。

COBOL 変数宣言と使用上の注意事項

レベル 77 のデータ記述項目の後に 1 つ以上の REDEFINES 項目を続けることができます。しかし、これらの項目に入っている名前は、SQL ステートメントの中では使用できません。

構造体が FILLER 項目より下に定義されたレベルを含むとき、異常な結果を生じる場合があります。

SMALLINT、INTEGER、および BIGINT データ・タイプの COBOL 宣言は、数桁の 10 進数で表されません。データベース・マネージャーは整数の全桁を使用するので、COBOL 宣言の中の指定した桁数で許容されるよりも大きな値をホスト変数に入れることができます。しかし、これが行われると、COBOL ステートメントの実行時にデータ切り捨てやサイズのエラーが起こります。アプリケーションの中の数の大きさが宣言した桁数の範囲内にあることを確かめておくべきです。

SQL を使用する COBOL アプリケーションでの標識変数の使用

標識変数は 2 バイトの整数です (PIC S9(m) USAGE BINARY、ただし、m は 1 から 4 までです)。ホスト構造をサポートするために標識構造 (ハーフワードの整数変数の配列として定義されているもの) を指定することもできます。検索されるとき、標識変数はその対応するホスト変数にヌル値が割り当てられているかどうかを示すために使用されます。列に割り当てるときには、ヌル値を割り当てるべきであることを示すために負の標識変数が使用されます。

詳細については、「SQL 解説書」の『標識変数』を参照してください。

標識変数の宣言の仕方はホスト変数の場合と同じであり、これらの 2 つの変数の宣言をプログラマーに適切と思われる方法で組み合わせることができます。

例:

次のステートメントがあるとします。

```
EXEC SQL FETCH CLS_CURSOR INTO :CLS-CD,  
                                     :NUMDAY :NUMDAY-IND,  
                                     :BGN :BGN-IND,  
                                     :ENDCLS :ENDCLS-IND  
END-EXEC.
```

変数は次のように宣言することができます。

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.  
77 CLS-CD      PIC X(7).  
77 NUMDAY     PIC S9(4) BINARY.  
77 BGN        PIC X(8).  
77 ENDCLS     PIC X(8).  
77 NUMDAY-IND PIC S9(4) BINARY.  
77 BGN-IND    PIC S9(4) BINARY.  
77 ENDCLS-IND PIC S9(4) BINARY.  
EXEC SQL END DECLARE SECTION END-EXEC.
```

第 7 章 PL/I アプリケーションでの SQL ステートメントのコーディング方法

ここでは、SQL ステートメントを iSeries PL/I プログラムに組み込む場合に固有のアプリケーションおよびコーディング上の要件について説明します。ホスト構造およびホスト変数に関する要件についても説明します。

詳細については、以下のセクションを参照してください。

- 『SQL を使用する PL/I アプリケーションでの SQL 連絡域の定義』
- 88 ページの『SQL を使用する PL/I アプリケーションでの SQL 記述子域の使用』
- 89 ページの『SQL を使用する PL/I アプリケーションでの SQL ステートメントの組み込み』
- 91 ページの『SQL を使用する PL/I アプリケーションでのホスト変数の使用』
- 96 ページの『SQL を使用する PL/I アプリケーションでのホスト構造の使用』
- 98 ページの『SQL を使用する PL/I アプリケーションでのホスト構造配列の使用』
- 100 ページの『SQL を使用する PL/I アプリケーションでの外部ファイル記述の使用』
- 101 ページの『SQL データ・タイプと PL/I データ・タイプの対応関係の判別』
- 103 ページの『SQL を使用する PL/I アプリケーションでの標識変数の使用』
- 104 ページの『構造パラメーター受け渡し技法による PL/I での相違』

SQL ステートメントの使い方を示した詳しいサンプル PL/I プログラムは、『第 12 章 DB2 UDB for iSeries ステートメントを使用したサンプル・プログラム』に記載されています。

注: 重要な法律的事柄に関する情報について、214 ページの『コードについての特記事項』をご覧ください。

SQL を使用する PL/I アプリケーションでの SQL 連絡域の定義

SQL ステートメントを含んでいる PL/I プログラムには、次のいずれかまたは両方を持っている必要があります。

- FIXED BINARY(31) として宣言している SQLCODE 変数
- CHAR(5) として宣言している SQLSTATE 変数

または、

- SQLCA (SQLCODE および SQLSTATE 変数が入っている)

SQLCODE 値および SQLSTATE 値は、各 SQL ステートメントが実行された後、データベース・マネージャーによって設定されます。アプリケーションは、SQLCODE 値または SQLSTATE 値を調べて、最後の SQL ステートメントが正しく実行されたかどうかを判定することができます。

SQLCA は、直接または、SQL の INCLUDE ステートメントを使用して、PL/I プログラムの中にコーディングすることができます。SQL の INCLUDE ステートメントを使用するときは、標準の SQLCA 宣言を組み込む必要があります。

```
EXEC SQL INCLUDE SQLCA ;
```

SQLCODE、SQLSTATE、および SQLCA の各変数の有効範囲には、プログラムの中のすべての SQL ステートメントの有効範囲が含まれていなければなりません。

SQLCA を組み込んだ PL/I ソース・ステートメントは次のとおりです。

```
DCL 1 SQLCA,  
  2 SQLCAID      CHAR(8),  
  2 SQLCABC      FIXED(31) BINARY,  
  2 SQLCODE      FIXED(31) BINARY,  
  2 SQLERRM      CHAR(70) VAR,  
  2 SQLERRP      CHAR(8),  
  2 SQLERRD(6)   FIXED(31) BINARY,  
  2 SQLWARN,  
    3 SQLWARN0   CHAR(1),  
    3 SQLWARN1   CHAR(1),  
    3 SQLWARN2   CHAR(1),  
    3 SQLWARN3   CHAR(1),  
    3 SQLWARN4   CHAR(1),  
    3 SQLWARN5   CHAR(1),  
    3 SQLWARN6   CHAR(1),  
    3 SQLWARN7   CHAR(1),  
    3 SQLWARN8   CHAR(1),  
    3 SQLWARN9   CHAR(1),  
    3 SQLWARNA   CHAR(1),  
  2 SQLSTATE     CHAR(5);
```

SQLCODE の宣言がプログラムの中にあつて、SQLCA がプリコンパイラーによって与えられるとき、SQLCODE の個所は SQLCADE で置き換えられます。SQLSTATE の宣言がプログラムの中にあつて、SQLCA がプリコンパイラーによって与えられるとき、SQLSTATE の個所は SQLSTOTE で置き換えられます。

SQLCA の詳細については、「SQL 解説書」の『SQL 連絡域』を参照してください。

SQL を使用する PL/I アプリケーションでの SQL 記述子域の使用

SQLDA を必要とするステートメントには、次のものがあります。

```
EXECUTE...USING DESCRIPTOR 記述子名  
FETCH...USING DESCRIPTOR 記述子名  
OPEN...USING DESCRIPTOR 記述子名  
CALL...USING DESCRIPTOR 記述子名  
DESCRIBE ステートメント名 INTO 記述子名  
DESCRIBE TABLE ホスト変数 INTO 記述子名  
PREPARE ステートメント名 INTO 記述子名
```

SQLCA と異なり、SQLDA を 2 つ以上プログラムの中に置くことができ、また SQLDA の名前は有効であれば、どの名前でも使えます。SQLDA は、直接プログラムするか、または SQL INCLUDE ステートメントを使用して PL/I プログラムにコーディングすることができます。SQL の INCLUDE ステートメントを使用するときは、標準の SQLDA 宣言を組み込む必要があります。

```
EXEC SQL INCLUDE SQLDA ;
```

SQLDA 用に組み込まれる PL/I ソース・ステートメントは次のとおりです。

```
DCL 1 SQLDA BASED(SQLDAPTR),  
  2 SQLDAID      CHAR(8),  
  2 SQLDABC      FIXED(31) BINARY,  
  2 SQLN         FIXED(15) BINARY,  
  2 SQLD         FIXED(15) BINARY,
```



```

2 SQLVAR(99),
3 SQLTYPE      FIXED(15) BINARY,
3 SQLLEN       FIXED(15) BINARY,
3 SQLRES       CHAR(12),
3 SQLDATA      PTR,
3 SQLIND       PTR,
3 SQLNAME      CHAR(30) VAR;
DCL SQLDAPTR PTR;

```

動的 SQL は高度なプログラミング技法です。これについては、「SQL プログラミング」の『動的 SQL アプリケーション』で説明されています。動的 SQL を使用すると、ユーザーのプログラムはその実行と平行して SQL ステートメントを作成し、実行させることができます。動的に実行される変数 SELECT リスト (すなわち、照会の一部として返されるデータのリスト) を指定する SELECT ステートメントには、SQL 記述子域 (SQLDA) が必要です。これは、SELECT の結果を受け入れるために割り振るべき変数の数とタイプが事前に予測できないからです。

SQLDA の詳細については、「SQL 解説書」の『SQL 記述子域』を参照してください。

SQL を使用する PL/I アプリケーションでの SQL ステートメントの組み込み

PL/I プログラムの最初のステートメントは PROCEDURE ステートメントでなければなりません。

SQL ステートメントは、実行可能なステートメントを置くことができる個所ならば、PL/I プログラム内のどこにでもコーディングすることができます。

PL/I プログラムの各 SQL ステートメントは EXEC SQL で始まり、セミコロン (;) で終わらなければなりません。キーワード EXEC SQL は 1 行に置かなければなりません。ステートメントの残りの部分は次の行とそれ以降の行に置くことができます。

詳細については、以下のセクションを参照してください。

- 『例: SQL を使用する PL/I ステートメントでの SQL ステートメントの組み込み』
- 90 ページの『SQL を使用する PL/I アプリケーションでの注記』
- 90 ページの『SQL を使用する PL/I アプリケーションでの SQL ステートメントの継続』
- 90 ページの『SQL を使用する PL/I アプリケーションでのコードの組み込み』
- 90 ページの『SQL を使用する PL/I アプリケーションでのマージン』
- 90 ページの『SQL を使用する PL/I アプリケーションでの名前』
- 90 ページの『SQL を使用する PL/I アプリケーションでのステートメント・ラベル』
- 90 ページの『SQL を使用する PL/I アプリケーションでの WHENEVER ステートメント』

例: SQL を使用する PL/I ステートメントでの SQL ステートメントの組み込み

PL/I プログラムの中でコーディングする UPDATE ステートメントを書くと、次のようになります。

```

EXEC SQL  UPDATE DEPARTMENT
          SET MGRNO = :MGR_NUM
          WHERE DEPTNO = :INT_DEPT ;

```

SQL を使用する PL/I アプリケーションでの注記

SQL の注記 (--) の他に、PL/I の注記 (*...*) は、組み込み SQL ステートメント内のブランクが許されている場所のどこにでも置くことができます。ただし、キーワードの EXEC と SQL の間には入れられません。

SQL を使用する PL/I アプリケーションでの SQL ステートメントの継続

SQL ステートメントの場合の行継続に関する規則は、EXEC SQL を 1 行に指定する点を除けば、他の PL/I ステートメントの場合と同じです。

DBCS データを含む定数は、シフトイン文字とシフトアウト文字をマージンの外側に置くことによって、複数行にわたって継続させることができます。この例では、マージンは 2 桁目と 72 桁目に限定されているものとします。この SQL ステートメントの G'<AABBCCDDEEFFGGHHIIJJKK>' はグラフィック定数として有効です。

```
*(.+. . . .1. . . .+. . . .2. . . .+. . . .3. . . .+. . . .4. . . .+. . . .5. . . .+. . . .6. . . .+. . . .7.)..  
EXEC SQL SELECT * FROM GRAPHTAB WHERE GRAPHCOL = G'<AABBCCDD>  
<EEFFGGHHIIJJKK>';
```

SQL を使用する PL/I アプリケーションでのコードの組み込み

SQL ステートメントまたは PL/I ホスト変数宣言ステートメントは、ステートメントを組み込むソース・コード内の個所に次の SQL ステートメントを置くことによって挿入することができます。

```
EXEC SQL INCLUDE member-name ;
```

PL/I プリプロセッサ・ディレクティブは、SQL ステートメントの中では許されません。PL/I の %INCLUDE ステートメントは、SQL ステートメントまたは SQL ステートメントの中で参照される PL/I ホスト変数の宣言を組み入れるためには使用できません。

SQL を使用する PL/I アプリケーションでのマージン

SQL ステートメントは、CRTSQLPLI コマンドの MARGINS パラメーターで指定したマージンの範囲内でコーディングしてください。EXEC SQL が指定のマージン内で始まっていないときは、SQL プリコンパイラーはそれを SQL ステートメントと認識しません。CRTSQLPLI コマンドの詳しい説明については、『第 13 章 ホスト言語プリコンパイラー用の DB2 UDB for iSeries CL コマンドの説明』を参照してください。

SQL を使用する PL/I アプリケーションでの名前

有効な PL/I 変数名であれば、どの変数名でもホスト変数に使用できますが、次のような制約を受けます。

'SQL'、'RDI'、または 'DSN' で始まるホスト変数名や外部入り口名は、使用してはなりません。これらの名前はデータベース・マネージャー用に予約されています。

SQL を使用する PL/I アプリケーションでのステートメント・ラベル

実行可能なすべての SQL ステートメントは、PL/I ステートメントと同様、ラベル接頭部を持つことができます。

SQL を使用する PL/I アプリケーションでの WHENEVER ステートメント

SQL WHENEVER ステートメントの GOTO 文節の対象となるものは、PL/I ソース・コード内のラベルでなければならず、WHENEVER ステートメントの影響が及ぶ SQL ステートメントの有効範囲内になければなりません。

SQL を使用する PL/I アプリケーションでのホスト変数の使用

SQL ステートメントの中で使用するホスト変数はいずれも明示的に宣言しなければなりません。

ホスト変数を定義するために使用される PL/I ステートメントは、その前に BEGIN DECLARE SECTION ステートメントを置き、その後、END DECLARE SECTION ステートメントを置く必要があります。BEGIN DECLARE SECTION と END DECLARE SECTION を指定する場合、SQL ステートメントで使用するすべてのホスト変数宣言は、BEGIN DECLARE SECTION ステートメントと END DECLARE SECTION ステートメントの間になければなりません。

SQL ステートメントの中のホスト変数はいずれも、その前にコロン (:) を付けなければなりません。

ホスト変数の名前は、ホスト変数がそれぞれ別のブロックやプロシージャの中にある場合であっても、1 つのプログラム内では固有にならなければなりません。

ホスト変数を使用する SQL ステートメントは、その変数が宣言されたステートメントの有効範囲内にならなければなりません。

ホスト変数はスカラー変数でなければなりません。これらは配列の要素にすることはできません。

詳細については、『SQL を使用する PL/I アプリケーションでのホスト変数の宣言』を参照してください。

SQL を使用する PL/I アプリケーションでのホスト変数の宣言

PL/I プリコンパイラーは、有効な PL/I 宣言のサブセットだけを有効なホスト変数宣言として認識します。

変数の名前とデータ属性だけがプリコンパイラーによって使用されます。境界合わせ、有効範囲、および記憶域属性は無視されます。境界合わせ、有効範囲、および記憶域属性が無視される場合であっても、その使い方にいくつかの制約があり、もし無視されると、プリコンパイラーによって生成される PL/I ソース・コードをコンパイルするとき問題が起こることがあります。その制約とは、次のものです。

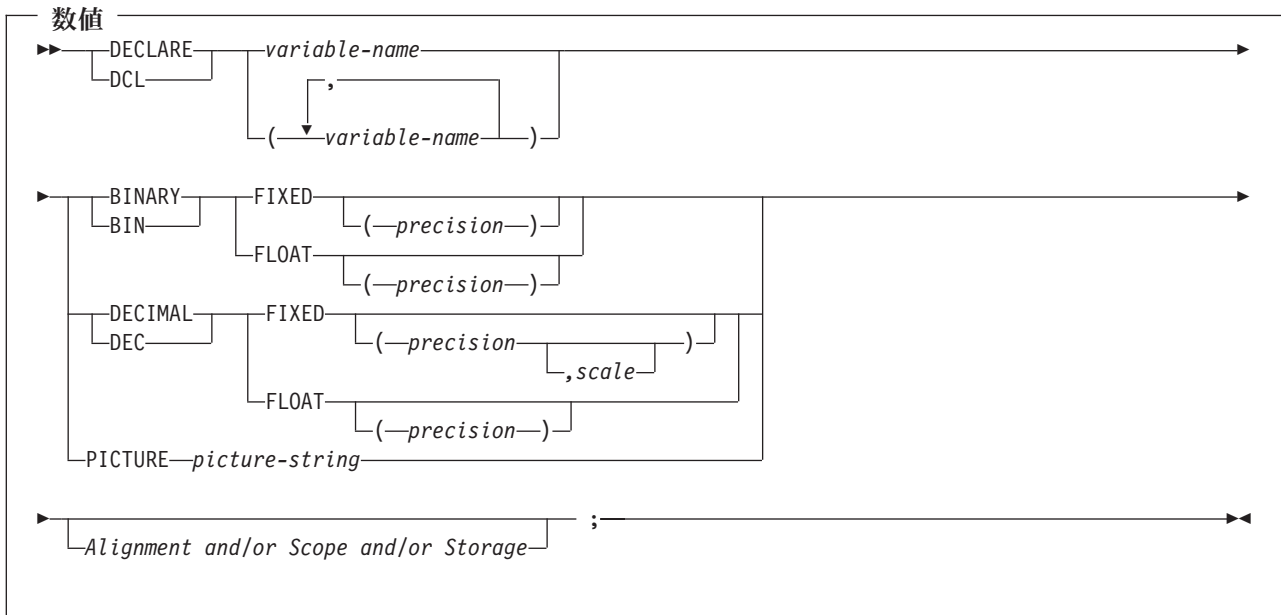
- EXTERNAL 有効範囲属性と STATIC 記憶属性をもつ宣言は INITIAL 記憶属性ももたなければなりません。
- BASED 記憶属性をコーディングする場合は、そのあとに PL/I 要素ロケター式を置かなければなりません。

特定のタイプのホスト変数について、以下のトピックを参照してください。

- 『SQL を使用する PL/I アプリケーションでの数値ホスト変数』
- 92 ページの『SQL を使用する PL/I アプリケーションでの文字ホスト変数』
- 93 ページの『SQL を使用する PL/I アプリケーションでのバイナリー・ホスト変数』
- 93 ページの『SQL を使用する PL/I アプリケーションでの LOB ホスト変数』
- 95 ページの『SQL を使用する PL/I アプリケーションでの ROWID ホスト変数』

SQL を使用する PL/I アプリケーションでの数値ホスト変数

次の図は、有効なスカラー数値ホスト変数宣言の構文を示しています。

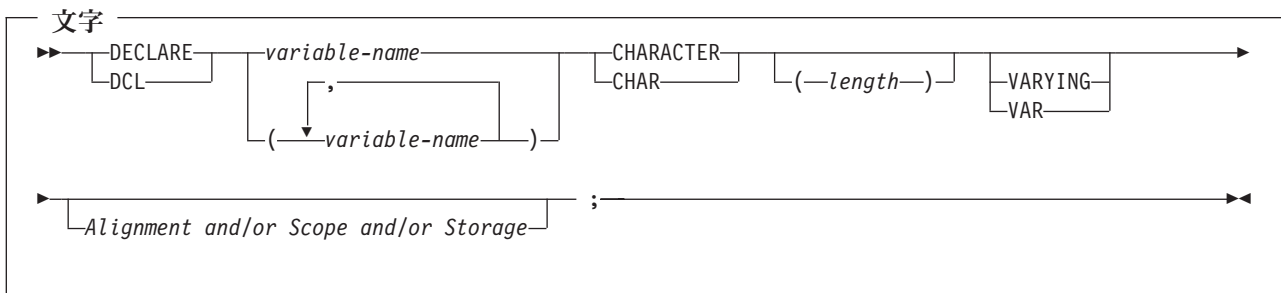


注:

1. (BINARY、BIN、DECIMAL、または DEC) および (FIXED または FLOAT) および (precision, scale) はどの順序でも指定できます。
2. '9...9V9...R' の形式の picture-string (ピクチャー・ストリング) は数値のホスト変数を示しています。R が必要です。任意選択の V は、暗黙の小数点を示しています。
3. 'S9...9V9...9' の形式の picture-string (ピクチャー・ストリング) は符号先行の分離ホスト変数を示しています。S が必要です。任意選択の V は、暗黙の小数点を示しています。

SQL を使用する PL/I アプリケーションでの文字ホスト変数

次の図は、有効なスカラー文字ホスト変数の構文を示しています。

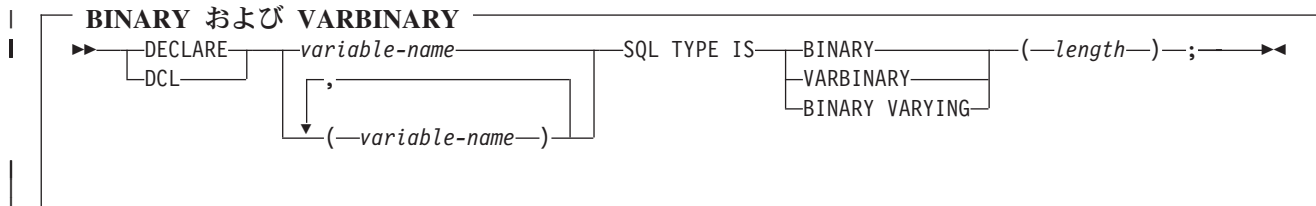


注:

1. VARYING または VAR を指定しない場合、length (長さ) は 32766 以下の整数定数でなければなりません。
2. VARYING または VAR を指定する場合、length (長さ) は 32740 以下の定数でなければなりません。

SQL を使用する PL/I アプリケーションでのバイナリー・ホスト変数

PL/I には、SQL バイナリー・データ・タイプに対応する変数がありません。これらのデータ・タイプで使用するホスト変数を作成するには、SQL TYPE IS 文節を使用します。SQL プリコンパイラーは、この宣言を出力ソース・メンバー中、PL/I 言語構造に置き換えます。



注:

1. BINARY ホスト変数では、length (長さ) の範囲は 1 から 32766 まででなければなりません。
2. VARBINARY および BINARY VARYING ホスト変数では、length (長さ) の範囲は 1 から 32740 まででなければなりません。
3. SQL TYPE IS、BINARY、VARBINARY、および BINARY VARYING は大文字小文字混合にすることができます。

BINARY の例

次のように宣言すると、

```
DCL MY_BINARY SQL TYPE IS BINARY(100);
```

以下のような結果コードが生成されます。

```
DCL MY_BINARY CHARACTER(100);
```

VARBINARY の例

次のように宣言すると、

```
DCL MY_VARBINARY SQL TYPE IS VARBINARY(250);
```

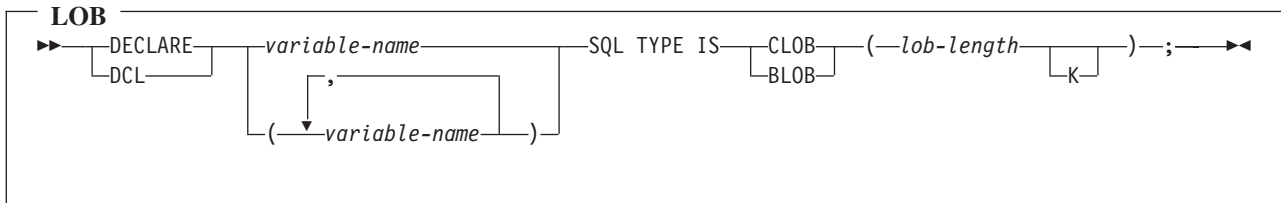
以下のような結果コードが生成されます。

```
DCL MY_VARBINARY CHARACTER(250) VARYING;
```

SQL を使用する PL/I アプリケーションでの LOB ホスト変数

PL/I には、LOB (ラージ・オブジェクト) の SQL データ・タイプに対応する変数がありません。これらのデータ・タイプで使用するホスト変数を作成するには、SQL TYPE IS 文節を使用します。SQL プリコンパイラーは、この宣言を出力ソース・メンバー中、PL/I 言語構造に置き換えます。

次の図は、有効な LOB ホスト変数の構文を示しています。



注:

1. BLOB および CLOB の場合、 $1 \leq \text{lob-length} \leq 32,766$
2. SQL TYPE IS、BLOB、CLOB は、大文字小文字混合にすることができます。

CLOB の例:

次のように宣言すると、

```
DCL MY_CLOB SQL TYPE IS CLOB(16384);
```

以下の構造を生成します。

```
DCL 1 MY_CLOB,
      3 MY_CLOB_LENGTH BINARY FIXED (31) UNALIGNED,
      3 MY_CLOB_DATA CHARACTER (16384);
```

BLOB の例:

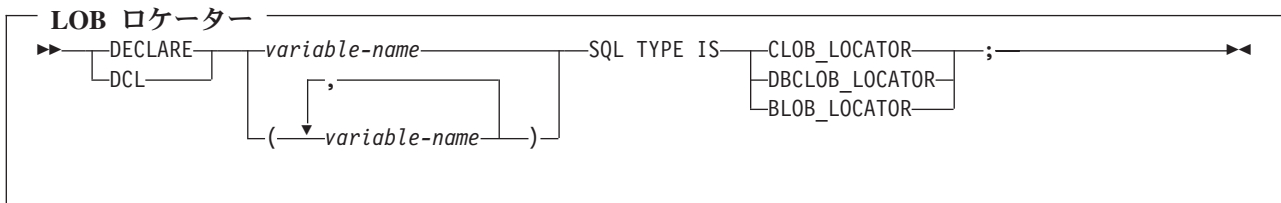
次のように宣言すると、

```
DCL MY_BLOB SQL TYPE IS BLOB(16384);
```

以下の構造を生成します。

```
DCL 1 MY_BLOB,
      3 MY_BLOB_LENGTH BINARY FIXED (31) UNALIGNED,
      3 MY_BLOB_DATA CHARACTER (16384);
```

次の図は、有効な LOB ロケーターの構文を示しています。



注: SQL TYPE IS、BLOB_LOCATOR、CLOB_LOCATOR、DBCLOB_LOCATOR は、大文字小文字混合にすることができます。

CLOB ロケーターの例:

次のように宣言すると、

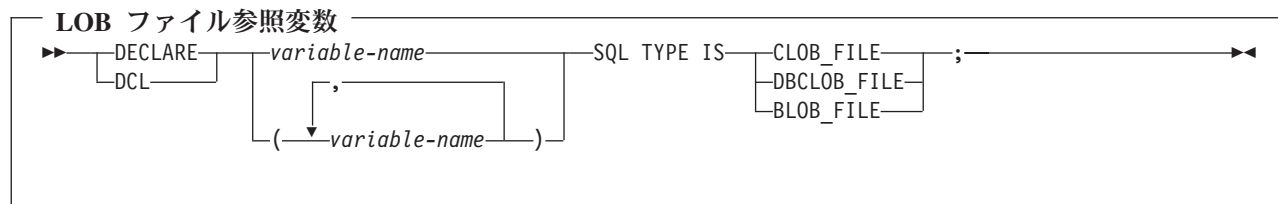
```
DCL MY_LOCATOR SQL TYPE IS CLOB_LOCATOR;
```

以下の構造を生成します。

```
DCL MY_LOCATOR BINARY FIXED(31) UNALIGNED;
```

BLOB ロケーターおよび DBCLOB ロケーターの構文は、似ています。

次の図は、有効な LOB ファイル参照変数の構文を示しています。



注: SQL TYPE IS、BLOB_FILE、CLOB_FILE、および DBCLOB_FILE は大文字小文字混合にすることができます。

CLOB ファイル参照の例:

次のように宣言すると、

```
DCL MY_FILE SQL TYPE IS CLOB_FILE;
```

以下の構造を生成します。

```
DCL 1 MY_FILE,  
  3 MY_FILE_NAME_LENGTH BINARY FIXED(31) UNALIGNED,  
  3 MY_FILE_DATA_LENGTH BINARY FIXED(31) UNALIGNED,  
  3 MY_FILE_FILE_OPTIONS BINARY FIXED(31) UNALIGNED,  
  3 MY_FILE_NAME_CHAR(255);
```

BLOB ロケーターおよび DBCLOB ロケーターの構文は、似ています。

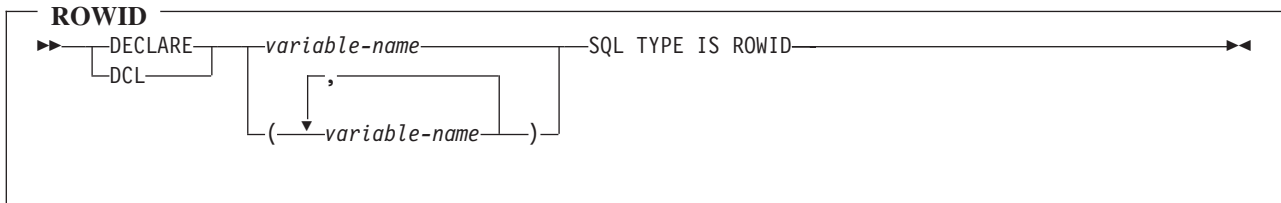
プリコンパイラーは、次のファイル・オプション定数に対する宣言を生成します。

- SQL_FILE_READ (2)
- SQL_FILE_CREATE (8)
- SQL_FILE_OVERWRITE (16)
- SQL_FILE_APPEND (32)

これらの値の詳細については、「SQL プログラミング 概念」の『LOB ファイル参照変数』を参照してください。

SQL を使用する PL/I アプリケーションでの ROWID ホスト変数

PL/I には、ROWID の SQL データ・タイプに対応する変数がありません。このデータ・タイプで使用するホスト変数を作成するには、SQL TYPE IS 文節を使用します。SQL プリコンパイラーは、この宣言を出力ソース・メンバー中、PL/I 言語構造に置き換えます。



注: SQL TYPE IS ROWID は、大文字小文字混合にすることができます。

ROWID の例

次のように宣言すると、

```
DCL MY_ROWID SQL TYPE IS ROWID;
```

以下の構造を生成します。

```
DCL MY_ROWID CHARACTER(40) VARYING;
```

SQL を使用する PL/I アプリケーションでのホスト構造の使用

PL/I プログラムの中では、**ホスト構造**が定義できます。これは一組の基本 PL/I 変数に名前を付けたものです。ホスト構造名は、その従属レベルに基本 PL/I 変数の名前が指定されているグループ名にすることができます。以下に、例を示します。

```
DCL 1 A,
    2 B,
    3 C1 CHAR(...),
    3 C2 CHAR(...);
```

この例では、B は基本項目 C1 と C2 から成るホスト構造の名前です。

構造名は、スカラーのリストの省略表現として使用することができます。ホスト変数は構造名で修飾することができます (たとえば、STRUCTURE.FIELD)。ホスト構造は 2 レベルに限定されます。(たとえば、上記のホスト構造の例では、SQL の中で A を参照することはできません。) 構造に中間レベルの構造を含めることはできません。上記の例では、A はホスト変数として使用することも、SQL ステートメントの中で参照することもできません。しかし、B は第 1 レベルの構造です。B は SQL ステートメントの中で参照することができます。SQL データのホスト構造は 2 レベルの深さであり、一連のホスト変数に名前を付けたものと考えすることができます。ホスト構造を定義しておけば、SQL ステートメントの中でいくつかのホスト変数 (ホスト構造を構成するホスト変数の名前) を個別に参照せずに一括して参照することができます。

たとえば、次のようにコーディングすれば、テーブル CORPDATA.EMPLOYEE から選択した行のすべての列の値を検索することができます。

```
DCL 1 PEMPL,
    5 EMPNO    CHAR(6),
    5 FIRSTNME CHAR(12) VAR,
    5 MIDINIT  CHAR(1),
    5 LASTNAME  CHAR(15) VAR,
    5 WORKDEPT CHAR(3);
...
EMPID = '000220';
...
EXEC SQL
```



```

SELECT *
INTO :PEMPL
FROM CORPDATA.EMPLOYEE
WHERE EMPNO = :EMPID;

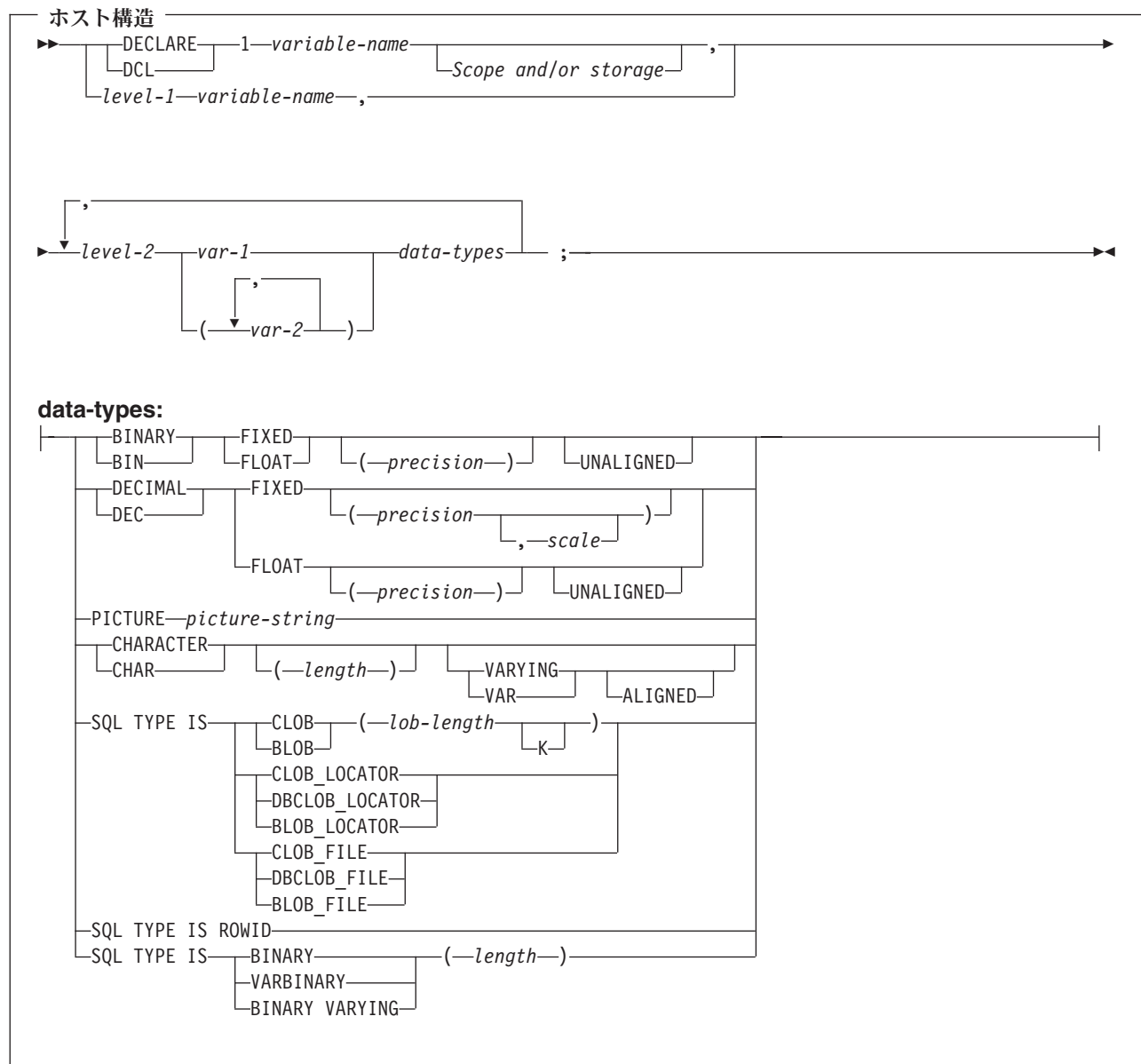
```

詳細については、以下のセクションを参照してください。

- 『SQL を使用する PL/I アプリケーションでのホスト構造』
- 98 ページの 『SQL を使用する PL/I アプリケーションでのホスト構造標識配列』

SQL を使用する PL/I アプリケーションでのホスト構造

次の図は、有効なホスト構造宣言の構文を示しています。

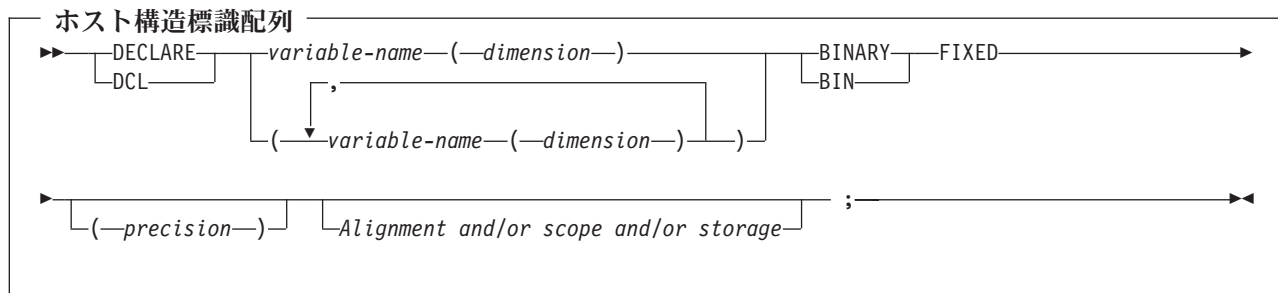


注:

1. level-1 (レベル-1) は、中間レベル構造があることを示しています。
2. level-1 (レベル-1) は、1 から 254 までの整数定数でなければなりません。
3. level-2 (レベル-2) は、2 から 255 までの整数定数でなければなりません。
4. 数値ホスト変数、文字ホスト変数、LOB ホスト変数、ROWID ホスト変数、およびバイナリー・ホスト変数の宣言についての詳細は、それぞれ数値、文字、LOB、ROWID、およびバイナリーの各ホスト変数に関する注意事項を参照してください。

SQL を使用する PL/I アプリケーションでのホスト構造標識配列

次の図は、有効な標識配列の構文を示しています。



注: 次元は 1 から 32766 までの整数定数でなければなりません。

SQL を使用する PL/I アプリケーションでのホスト構造配列の使用

PL/I プログラムでは、ホスト構造配列を定義することができます。このような例では、以下の条件が該当します。

- B_ARRAY は、項目 C1_VAR と C2_VAR が入っているホスト構造配列の名前です。
- B_ARRAY は修飾できません。
- B_ARRAY はブロック化形式の FETCH および INSERT ステートメントでのみ使用できます。
- B_ARRAY 内のすべての項目は有効なホスト変数でなければなりません。
- C1_VAR および C2_VAR は、SQL ステートメントでは有効なホスト変数ではありません。構造に中間レベルの構造を含めることはできません。A_STRUCT に次元属性を入れることはできません。

```
DCL 1 A_STRUCT,  
  2 B_ARRAY(10),  
  3 C1_VAR CHAR(20),  
  3 C2_FIXED BIN(15) UNALIGNED;
```

CORPDATA.DEPARTMENT テーブルから 10 行検索するには、次のようにコーディングします。

```
DCL 1 DEPT(10),  
  5 DEPTNO CHAR(3),  
  5 DEPTNAME CHAR(29) VAR,  
  5 MGRNO CHAR(6),  
  5 ADMRDEPT CHAR(3);  
DCL 1 IND_ARRAY(10),  
  5 INDS(4) FIXED BIN(15);  
EXEC SQL  
  DECLARE C1 CURSOR FOR  
  SELECT *
```

```

FROM CORPDATA.DEPARTMENT;

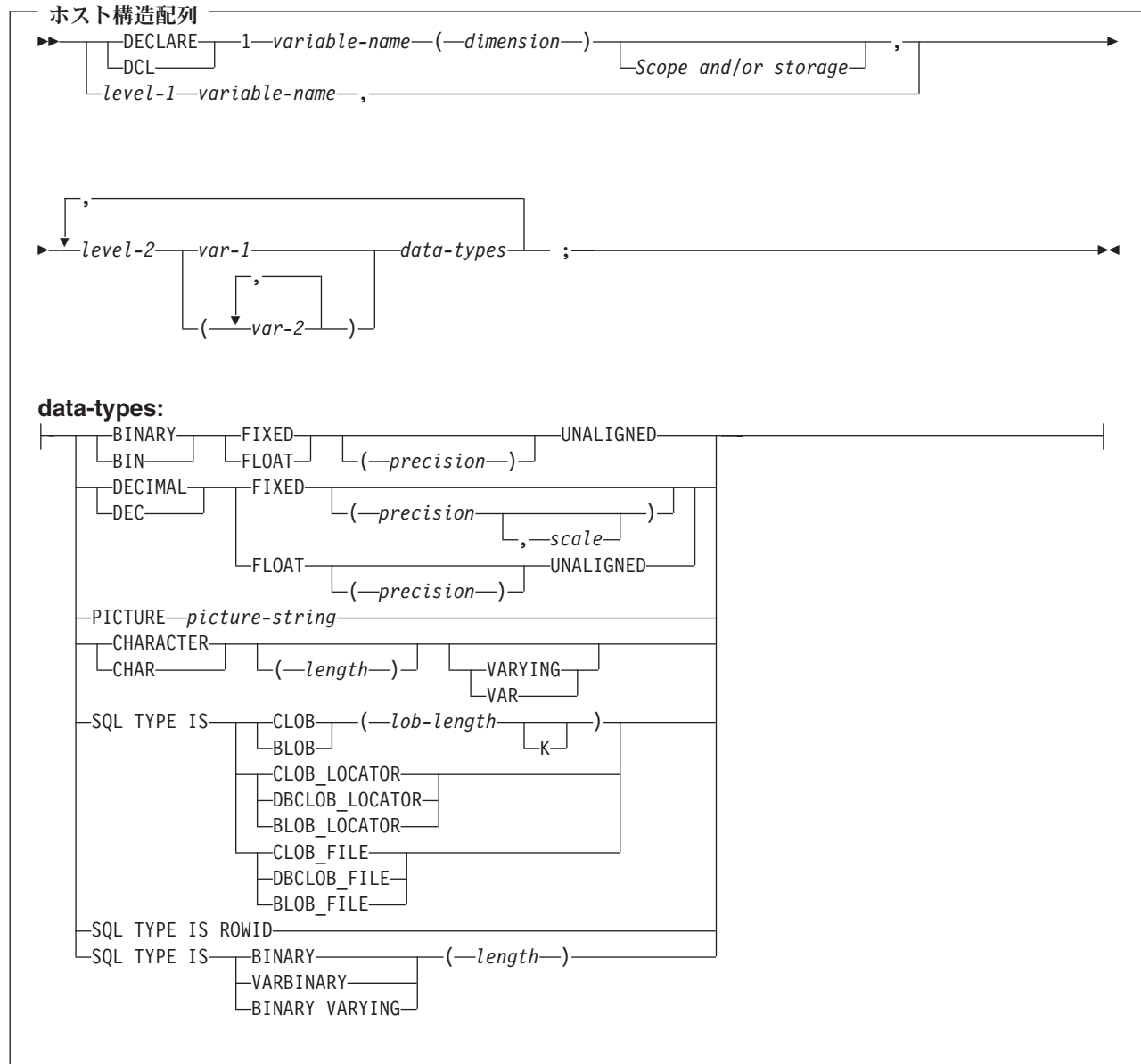
EXEC SQL
  FETCH C1 FOR 10 ROWS INTO :DEPT :IND_ARRAY;

```

詳細については、『SQL を使用する PL/I アプリケーションでのホスト構造配列』を参照してください。

SQL を使用する PL/I アプリケーションでのホスト構造配列

以下の構文図は、有効な構造配列宣言の構文を示しています。



注:

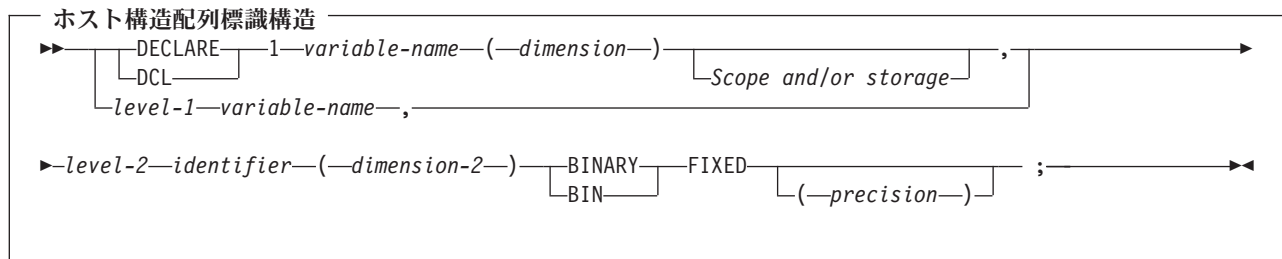
1. level-1 (レベル-1) は、中間レベル構造があることを示しています。
2. level-1 (レベル-1) は、1 から 254 までの整数定数でなければなりません。

- level-2 (レベル-2) は、2 から 255 までの整数定数でなければなりません。
- 数値ホスト変数、文字ホスト変数、LOB ホスト変数、ROWID ホスト変数、およびバイナリー・ホスト変数の宣言についての詳細は、それぞれ数値、文字、LOB、ROWID、およびバイナリーの各ホスト変数に関する注意事項を参照してください。
- dimension (次元) は 1 から 32767 までの整数定数でなければなりません。

詳しくは、『SQL を使用する PL/I アプリケーションでのホスト構造配列標識』を参照してください。

SQL を使用する PL/I アプリケーションでのホスト構造配列標識

次の図は、有効なホスト構造配列標識構造宣言の構文を示しています。



注:

- level-1 (レベル-1) は、中間レベル構造があることを示しています。
- level-1 (レベル-1) は、1 から 254 までの整数定数でなければなりません。
- level-2 (レベル-2) は、2 から 255 までの整数定数でなければなりません。
- dimension-1 (次元 1) および dimension-2 (次元 2) は、1 から 32767 までの整数定数でなければなりません。

SQL を使用する PL/I アプリケーションでの外部ファイル記述の使用

PL/I %INCLUDE ディレクティブを使用すると、外部記述ファイルの定義をソース・プログラムに組み入れることができます。SQL で使用するときには、特定形式の %INCLUDE ディレクティブだけが SQL プリコンパイラーによって認識されます。そのディレクティブの形式は、次の 3 つの要素またはパラメーター値を持っていなければなりません。そうでないと、プリコンパイラーはそのディレクティブを無視します。必要な 3 つの要素とは、ファイル名、様式名、および要素タイプです。このほかに、SQL プリコンパイラーによってサポートされる任意選択の要素が 2 つあります。それは接頭部名と COMMA です。

構造は、通常、レコードまたはキー構造の最後のデータ要素で終わります。しかし、%INCLUDE ディレクティブの中で COMMA 要素の指定があるときは、構造はそこで終わりません。

「DB2 UDB for iSeries SQL プログラミング 概念」の『DB2 UDB for iSeries サンプル・テーブル』に記載されているサンプル・テーブル DEPARTMENT の定義を組み込むには、次のようにコーディングすることができます。

```

DCL 1 TDEPT_STRUCTURE,
%INCLUDE DEPARTMENT(DEPARTMENT,RECORD);

```

上記の例で、TDEPT_STRUCTURE という名前のホスト構造は、4 つのフィールドをもつことが定義されます。これらのフィールドは DEPTNO、DEPTNAME、MGRNO、および ADMRDEPT となります。

装置ファイルの場合は、INDARA の指定がなく、ファイルに標識が入っているときには、その宣言をホスト構造配列として使用できません。生成された構造には標識域が組み入れられ、その標識域があるために記憶域が連続しなくなります。

```
DCL 1 DEPT_REC(10),
    %INCLUDE DEPARTMENT(DEPARTMENT,RECORD);

EXEC SQL DECLARE C1 CURSOR FOR
    SELECT * FROM CORPDATA.DEPARTMENT;

EXEC SQL OPEN C1;

EXEC SQL FETCH C1 FOR 10 ROWS INTO :DEPT_REC;
```

注: DATE、TIME、および TIMESTAMP の各列からはホスト変数定義が生成され、これらは DATE、TIME、および TIMESTAMP 列と同じ比較と割り当ての規則を使用して、SQL によって扱われます。たとえば、日付ホスト変数は、DATE 列または日付の有効な表現である文字ストリングとだけ比較することができます。

精度が 15 を超える 10 進数フィールドとゾーン・フィールド、および位取りフィールドがゼロでない 2 進数は、PL/I では文字フィールド変数に対応していますが、SQL では、これらのフィールドは数値であると見なされます。

GRAPHIC と VARGRAPHIC は PL/I で文字変数にマップされますが、SQL はこれらを GRAPHIC ホスト変数および VARGRAPHIC ホスト変数と見なします。GRAPHIC 列または VARGRAPHIC 列が UCS-2 CCSID を持つ場合、生成されるホスト変数には UCS-2 CCSID が割り当てられます。GRAPHIC 列または VARGRAPHIC 列が UTF-16 CCSID を持つ場合、生成されるホスト変数には UTF-16 CCSID が割り当てられます。

SQL データ・タイプと PL/I データ・タイプの対応関係の判別

プリコンパイラーは、次の表に基づいて、ホスト変数のベース SQLTYPE とベース SQLLEN を判断します。ホスト変数が標識変数と一緒に記載されているときは、その SQLTYPE はベース SQLTYPE に 1 を加えたものです。

表 5. PL/I 宣言と代表的 SQL データ・タイプとの対応関係

PL/I データ・タイプ	ホスト変数の SQLTYPE	ホスト変数の SQLLEN	SQL データ・タイプ
BIN FIXED(p) (p は 1 から 15 までの範囲)	500	2	SMALLINT
BIN FIXED(p) (p は 16 から 31 までの範囲)	496	4	INTEGER
DEC FIXED(p,s)	484	バイト 1 には p、バイト 2 には s	DECIMAL(p,s)
BIN FLOAT(p) (p は 1 から 24 までの範囲)	480	4	FLOAT (単精度)
BIN FLOAT(p) (p は 25 から 53 までの範囲)	480	8	FLOAT (倍精度)
DEC FLOAT(m) (m は 1 から 7 までの範囲)	480	4	FLOAT (単精度)

表 5. PL/I 宣言と代表的 SQL データ・タイプとの対応関係 (続き)

PL/I データ・タイプ	ホスト変数の SQLTYPE	ホスト変数の SQLLEN	SQL データ・タイプ
DEC FLOAT(m) (m は 8 から 16 までの範囲)	480	8	FLOAT (倍精度)
PICTURE ピクチャー・ストリング (数値)	488	バイト 1 には p、バイト 2 には s	NUMERIC(p,s)
PICTURE ピクチャー・ストリング (符号先行の分離)	504	バイト 1 には p、バイト 2 には s	正確に対応するものなし。NUMERIC(p,s) を使用。
CHAR(n)	452	n	CHAR(n)
CHAR(n) VARYING	448	n	VARCHAR(n)

下表を参照すると、各 SQL データ・タイプに対応する PL/I データ・タイプを判別することができます。

表 6. SQL データ・タイプと代表的な PL/I 宣言との対応関係

SQL データ・タイプ	対応する PL/I 宣言	説明と注
SMALLINT	BIN FIXED(p)	p は 1 から 15 までの正の整数です。
INTEGER	BIN FIXED(p)	p は 16 から 31 までの正の整数です。
BIGINT	正確に対応するものなし	DEC FIXED(18) を使用。
DECIMAL(p,s) または NUMERIC(p,s)	DEC FIXED(p) または DEC FIXED(p,s) または PICTURE ピクチャー・ストリング	s (位取り係数) および p (精度) は正の整数です。p は 1 から 31 までの正の整数です。s は 0 から p までの正の整数です。
FLOAT (単精度)	BIN FLOAT(p) または DEC FLOAT(m)	p は 1 から 24 までの正の整数です。 m は 1 から 7 までの正の整数です。
FLOAT (倍精度)	BIN FLOAT(p) または DEC FLOAT(m)	p は 25 から 53 までの正の整数です。 m は 8 から 16 までの正の整数です。
CHAR(n)	CHAR(n)	n は 1 から 32766 までの正の整数です。
VARCHAR(n)	CHAR(n) VARYING	n は 1 から 32740 までの正の整数です。
CLOB	なし	SQL TYPE IS を使用して CLOB を宣言します。
GRAPHIC(n)	サポートなし	サポートなし。
VARGRAPHIC(n)	サポートなし	サポートなし。
DBCLOB	サポートなし	サポートなし
BINARY	なし	SQL TYPE IS を使用して BINARY を宣言します。

表 6. SQL データ・タイプと代表的な PL/I 宣言との対応関係 (続き)

SQL データ・タイプ	対応する PL/I 宣言	説明と注
VARBINARY	なし	SQL TYPE IS を使用して VARBINARY を宣言します。
BLOB	なし	SQL TYPE IS を使用して BLOB を 宣言します。
DATE	CHAR(n)	形式が *USA、*JIS、*EUR、または *ISO のときは、n は少なくとも 10 文字にする必要があります。形式が *YMD、*DMY、または *MDY のと きは、n は少なくとも 8 文字にする 必要があります。形式が *JUL のとき は、n は少なくとも 6 文字にする必 要があります。
TIME	CHAR(n)	n は少なくとも 6 文字に、秒を含め る場合は、n は少なくとも 8 文字に する必要があります。
TIMESTAMP	CHAR(n)	n は少なくとも 19 が必要。マイクロ 秒を全桁の精度で含める場合には、n は少なくとも 26 にする必要があります。 n が 26 未満のときは、マイク ロ秒部分で切り捨てが起こります。
DATALINK	サポートなし	サポートなし
ROWID	なし	SQL TYPE IS を使用して ROWID を 宣言します。

SQL を使用する PL/I アプリケーションでの標識変数の使用

標識変数は 2 バイトの整数です (BIN FIXED(p)、ただし、p は 1 から 15 までです)。ホスト構造をサポートするために標識構造 (ハーフワードの整数変数の配列として定義されているもの) を指定することもできます。検索されるとき、標識変数はその対応するホスト変数にヌル値が割り当てられているかどうかを示すために使用されます。列に割り当てるときには、ヌル値を割り当てるべきであることを示すために負の標識変数が使用されます。

詳細については、「SQL 解説書」の『標識変数』を参照してください。

標識変数の宣言の仕方はホスト変数の場合と同じであり、これらの 2 つの変数の宣言をプログラマーに適切と思われる方法で組み合わせることができます。

例:

次のステートメントがあるとします。

```
EXEC SQL FETCH CLS_CURSOR INTO :CLS_CD,
                                :DAY :DAY_IND,
                                :BGN :BGN_IND,
                                :END :END_IND;
```

変数は次のように宣言することができます。

```
EXEC SQL BEGIN DECLARE SECTION;
DCL CLS_CD CHAR(7);
DCL DAY BIN FIXED(15);
DCL BGN CHAR(8);
DCL END CHAR(8);
DCL (DAY_IND, BGN_IND, END_IND) BIN FIXED(15);
EXEC SQL END DECLARE SECTION;
```

構造パラメーター受け渡し技法による PL/I での相違

PL/I プリコンパイラーは、可能ならば、構造パラメーター受け渡し技法の使用を試みます。この構造パラメーター受け渡し技法を使用すると、SQL を使用するほとんどの PL/I プログラムのパフォーマンスが向上します。プリコンパイラーは、以下の条件に該当する場合、各ホスト変数が分離パラメーターになっているコードを生成します。

- 外部テキストをソース・プログラムにコピーする PL/I %INCLUDE コンパイラー指示が見つかった場合。
- ステートメントで参照されているホスト変数のデータ長が 32703 を超えている場合。SQL は構造のうち 64 バイトを使用するため、データ構造の最大長は $32703 + 64 = 32767$ となります。
- PL/I プリコンパイラーがユーザー定義名の PL/I 限度を超えると推定した場合。
- 符号先行分離ホスト変数が SQL ステートメントのホスト変数リストに見つかった場合。

構造パラメーター受け渡し技法の詳細については、DB2 UDB for iSeries データベース・パフォーマンスと最適化に関するトピックの『データベース・アプリケーション設計のヒント: 構造パラメーター受け渡し方法の使用』を参照してください。

第 8 章 RPG for iSeries アプリケーションでの SQL ステートメントのコーディング方法

RPG for iSeries ライセンス・プログラムは RPG II プログラムと RPG III プログラムを共にサポートしません。SQL ステートメントは、RPG III プログラムの中でしか使用できません。RPG II と報告書簡易作成機能はサポートされません。本書で RPG という場合は、RPG III または ILE RPG のいずれかを指しています。



ここでは、SQL ステートメントを RPG for iSeries プログラムに組み込む場合に固有のアプリケーションおよびコーディング上の要件について説明します。ホスト変数に必要な要件についても説明します。

詳細については、以下のセクションを参照してください。

- 『SQL を使用する RPG for iSeries アプリケーションでの SQL 連絡域の定義』
- 106 ページの『SQL を使用する RPG for iSeries アプリケーションでの SQL 記述子域の定義』
- 107 ページの『SQL を使用する RPG for iSeries アプリケーションでの SQL ステートメントの組み込み』
- 109 ページの『SQL を使用する RPG for iSeries アプリケーションでのホスト変数の使用』
- 110 ページの『SQL を使用する RPG for iSeries アプリケーションでのホスト構造の使用』
- 110 ページの『SQL を使用する RPG for iSeries アプリケーションでのホスト構造配列の使用』
- 111 ページの『SQL を使用する RPG for iSeries アプリケーションでの外部ファイル記述の使用』
- 113 ページの『SQL データ・タイプと RPG for iSeries データ・タイプの対応関係の判別』
- 115 ページの『SQL を使用する RPG for iSeries アプリケーションでの標識変数の使用』
- 116 ページの『構造パラメーター受け渡し技法による RPG for iSeries での相違』
- 116 ページの『呼び出された SQL を使用する RPG for iSeries プログラムの正しい終了方法』

SQL ステートメントの使い方を示した詳しいサンプル RPG for iSeries プログラムは、『第 12 章 DB2 UDB for iSeries ステートメントを使用したサンプル・プログラム』に記載されています。

注: 重要な法律的事柄に関する情報について、214 ページの『コードについての特記事項』をご覧ください。

RPG を使用するプログラミングの詳細については、V5R1 補足資料 Web サイトの「RPG/400® User's Guide」 および「RPG/400 Reference」 を参照してください。

SQL を使用する RPG for iSeries アプリケーションでの SQL 連絡域の定義

SQL プリコンパイラーは、RPG iSeries 用プログラムの最初の演算仕様の前の入力仕様に SQLCA を自動的に入れます。INCLUDE SQLCA をソース・プログラムにコーディングする必要はありません。ソース・プログラムに INCLUDE SQLCA を指定した場合、そのステートメントは受け入れられますが、余分なものとして扱われます。SQLCA を RPG for iSeries 用に定義すると、次のようになります。

ISQLCA	DS		SQL
I*	SQL Communications area		SQL
I		1 8 SQLAID	SQL
I		B 9 120SQLABC	SQL
I		B 13 160SQLCOD	SQL
I		B 17 180SQLERL	SQL
I		19 88 SQLERM	SQL
I		89 96 SQLERP	SQL
I		97 120 SQLERR	SQL
I		B 97 1000SQLER1	SQL
I		B 101 1040SQLER2	SQL
I		B 105 1080SQLER3	SQL
I		B 109 1120SQLER4	SQL
I		B 113 1160SQLER5	SQL
I		B 117 1200SQLER6	SQL
I		121 131 SQLWRN	SQL
I		121 121 SQLWN0	SQL
I		122 122 SQLWN1	SQL
I		123 123 SQLWN2	SQL
I		124 124 SQLWN3	SQL
I		125 125 SQLWN4	SQL
I		126 126 SQLWN5	SQL
I		127 127 SQLWN6	SQL
I		128 128 SQLWN7	SQL
I		129 129 SQLWN8	SQL
I		130 130 SQLWN9	SQL
I		131 131 SQLWNA	SQL
I		132 136 SQLSTT	SQL
I*	End of SQLCA		SQL

注: RPG for iSeries では変数名は 6 文字までに制限されています。したがって、標準 SQLCA 名は 6 文字の長さに変更されています。RPG for iSeries では、配列を拡張仕様にも定義しておかないと、配列をデータ構造に定義することはできません。SQLERR は、要素の名前に使用される SQLER1 から SQLER6 までの文字として定義されます。

詳細については、「SQL 解説書」の『SQL 連絡域』を参照してください。

SQL を使用する RPG for iSeries アプリケーションでの SQL 記述子域の定義

SQLDA を必要とするステートメントには、次のものがあります。

```
EXECUTE...USING DESCRIPTOR 記述子名
FETCH...USING DESCRIPTOR 記述子名
OPEN...USING DESCRIPTOR 記述子名
CALL...USING DESCRIPTOR 記述子名
DESCRIBE ステートメント名 INTO 記述子名
DESCRIBE TABLE ホスト変数 INTO 記述子名
PREPARE ステートメント名 INTO 記述子名
```

SQLCA と異なり、SQLDA を 2 つ以上プログラムの中に置くことができ、また SQLDA の名前は有効であれば、どの名前でも使えます。

動的 SQL は高度なプログラミング技法です。これについては、「SQL プログラミング」の『動的 SQL アプリケーション』で説明されています。動的 SQL を使用すると、ユーザーのプログラムはその実行と平行して SQL ステートメントを作成し、実行させることができます。動的に実行される変数 SELECT リ

スト (すなわち、照会の一部として返されるデータのリスト) を指定する SELECT ステートメントには、SQL 記述子域 (SQLDA) が必要です。これは、SELECT の結果を受け入れるために割り振るべき変数の数とタイプが事前に予測できないからです。

SQLDA ではポインター変数を使用しますが、これは RPG for iSeries ではサポートされないため、INCLUDE SQLDA ステートメントは RPG for iSeries プログラムでは指定できません。SQLDA を使用するためには、C、COBOL、PL/I、または ILE RPG の各プログラムでセットアップして RPG プログラムに渡さなければなりません。

SQLDA の詳細については、「SQL 解説書」の『SQL 記述子域』を参照してください。

SQL を使用する RPG for iSeries アプリケーションでの SQL ステートメントの組み込み

RPG iSeries プログラムの中にコーディングする SQL ステートメントは、演算部分に置かなければなりません。このためには C を 6 桁目に入れる必要があります。SQL ステートメントは、明細演算にも、合計計算にも、RPG for iSeries サブルーチンにも置くことができます。SQL ステートメントは、RPG for iSeries ステートメントのロジックに基づいて実行されます。

キーワード EXEC は SQL ステートメントの始まりを示します。EXEC はソース・ステートメントの 8 桁目から 16 桁目までに置き、7 桁目にスラッシュ (/) を置かなければなりません。SQL ステートメントを 17 桁目から始めて、74 桁目まで続けることができます。

キーワード END-EXEC は SQL ステートメントの終わりを示します。END-EXEC はソース・ステートメントの 8 桁目から 16 桁目までに置き、7 桁目にスラッシュ (/) を置かなければなりません。17 桁目から 74 桁目まではブランクにしておきます。

SQL ステートメントでは大文字と小文字の両方が使用できます。

詳細については、以下のセクションを参照してください。

- 『例: SQL を使用する RPG for iSeries アプリケーションでの SQL ステートメントの組み込み』
- 108 ページの『SQL を使用する RPG for iSeries アプリケーションでの注記』
- 108 ページの『SQL を使用する RPG for iSeries アプリケーションでの SQL ステートメントの継続』
- 108 ページの『SQL を使用する RPG for iSeries アプリケーションでのコードの組み込み』
- 108 ページの『SQL を使用する RPG for iSeries アプリケーションでの順序番号』
- 108 ページの『SQL を使用する RPG for iSeries アプリケーションでの名前』
- 108 ページの『SQL を使用する RPG for iSeries アプリケーションでのステートメント・ラベル』
- 109 ページの『SQL を使用する RPG for iSeries アプリケーションでの WHENEVER ステートメント』

例: SQL を使用する RPG for iSeries アプリケーションでの SQL ステートメントの組み込み

RPG for iSeries プログラムの中にコーディングされる UPDATE ステートメントをコーディングすると、次のようになります。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....*  
C/EXEC SQL UPDATE DEPARTMENT  
C+          SET MANAGER = :MGRNUM  
C+          WHERE DEPTNO = :INTDEP  
C/END-EXEC
```

SQL を使用する RPG for iSeries アプリケーションでの注記

SQL の注記 (--) の他に、RPG for iSeries の注記は、SQL ステートメント内の空白が許される場所にはどこにでも入れることができます。ただし、キーワードの EXEC と SQL の間には入れられません。RPG for iSeries の注記を SQL ステートメントに組み込むときは、7 桁目にアスタリスク (*) を置きます。

SQL を使用する RPG for iSeries アプリケーションでの SQL ステートメントの継続

SQL ステートメントを収めるために追加レコードが必要なときは、9 桁目から 74 桁目が使用できます。7 桁目は + (正符号) にし、8 桁目は空白にしなければなりません。

DBCS データを含む定数は、継続される行の 75 桁目にシフトイン文字を入れ、継続行の 8 桁目にシフトアウト文字を入れることによって、複数行にわたって継続させることができます。この SQL ステートメントの G'<AABBCDDDEEFFGGHHIIJJKK>' はグラフィック定数として有効です。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
C/EXEC SQL SELECT * FROM GRAPHTAB          WHERE GRAPHCOL = G'<AABB>
C+<CCDDEEFFGGHHIIJJKK>'
C/END-EXEC
```

SQL を使用する RPG for iSeries アプリケーションでのコードの組み込み

SQL ステートメントと RPG for iSeries 演算仕様は、SQL ステートメントを組み込むことによって取り込むことができます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
C/EXEC SQL INCLUDE member-name
C/END-EXEC
```

/COPY ステートメントを使用すると、SQL ステートメントまたは RPG for iSeries 仕様を取り込むことができます。

SQL を使用する RPG for iSeries アプリケーションでの順序番号

SQL プリコンパイラーによって生成されるソース・ステートメントの順序番号は、CRTSQLRPG コマンドの OPTION パラメーターの *NOSEQSRC/*SEQSRC キーワードに基づきます。*NOSEQSRC が指定されると、入力ソース・メンバーからの順序番号が使用されます。*SEQSRC を指定すると、順序番号は 000001 から始まり、1 ずつ増えます。

SQL を使用する RPG for iSeries アプリケーションでの名前

有効な RPG 変数名であれば、どの名前でもホスト変数に使えますが、次のような制約を受けます。

'SQ'、'SQL'、'RDI'、または 'DSN' で始まるホスト変数名や外部入り口名は、使用してはなりません。これらの名前はデータベース・マネージャー用に予約されています。

SQL を使用する RPG for iSeries アプリケーションでのステートメント・ラベル

どの SQL ステートメントの場合も、その前に TAG ステートメントを置くことができます。TAG ステートメントは EXEC SQL の前の行にコーディングします。

SQL を使用する RPG for iSeries アプリケーションでの WHENEVER ステートメント

GOTO 文節の対象となるものは、TAG ステートメントのラベルでなければなりません。GOTO/TAG の有効範囲に従わなければなりません。

SQL を使用する RPG for iSeries アプリケーションでのホスト変数の使用

SQL ステートメントの中で使用するホスト変数はいずれも明示的に宣言しなければなりません。LOB ホスト変数、ROWID ホスト変数、およびバイナリー・ホスト変数は、RPG for iSeries ではサポートされません。

RPG for iSeries に組み込まれた SQL は、ホスト変数を識別するために SQL の BEGIN DECLARE SECTION および END DECLARE SECTION ステートメントを使用しません。これらのステートメントをソース・プログラムに入れてはなりません。

SQL ステートメントの中のホスト変数はいずれも、その前にコロンの (;) を付けなければなりません。

ホスト変数の名前は、プログラムの中で固有でなければなりません。

詳細については、『SQL を使用する RPG for iSeries アプリケーションでのホスト変数の宣言』を参照してください。

SQL を使用する RPG for iSeries アプリケーションでのホスト変数の宣言

SQL RPG for iSeries プリコンパイラーは、有効なホスト変数宣言として RPG for iSeries 宣言のサブセットしか認識しません。

RPG for iSeries で定義されているほとんどの変数は、SQL ステートメントの中で使用できます。サポートされていない変数の一部を以下にリストします。

標識フィールド名 (*INxx)

テーブル

UPDATE

UDAY

UMONTH

UYEAR

先読みフィールド

名前付き定数

ホスト変数として使用されるフィールドは、RPG for iSeries の CALL/PARM 機能を使用して SQL に渡されます。PARM の結果フィールドで使用できないフィールドは、ホスト変数として使用できません。

SQL を使用する RPG for iSeries アプリケーションでのホスト構造の使用

RPG for iSeries データ構造名は、データ構造にサブフィールドがあれば、ホスト構造名として使用できます。SQL ステートメントの中でデータ構造名を使用したときは、そのデータ構造を構成するサブフィールド名のリストを暗黙に指定したことになります。

データ構造にサブフィールドがないときは、そのデータ構造名は文字タイプのホスト変数です。データ構造は最大 9999 までにできるので、これにより、文字変数を 256 より大きくすることができます。

次の例では、BIGCHR はサブフィールドのない RPG for iSeries データ構造です。BIGCHR が参照された場合、SQL はそれを長さが 642 の文字ストリングとして扱います。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
IBIGCHR      DS                                642
```

次の例では、PEMPL は EMPNO、FIRSTN、MIDINT、LASTNAME、および DEPTNO のサブフィールドから成るホスト構造の名前です。PEMPL が参照されると、これらのサブフィールドが使用されます。たとえば、EMPLOYEE の最初の列は EMPNO に入れられ、2 番目の列は FIRSTN に入れられます (以下同様です)。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7. ...*
IPEMPL      DS
I                                01 06 EMPNO
I                                07 18 FIRSTN
I                                19 19 MIDINT
I                                20 34 LASTNA
I                                35 37 DEPTNO
...
C                                MOVE '000220' EMPNO
...
C/EXEC SQL
C+ SELECT * INTO :PEMPL
C+ FROM CORPDATA.EMPLOYEE
C+ WHERE EMPNO = :EMPNO
C/END-EXEC
```

SQL ステートメントを書くとき、サブフィールドに対する参照を修飾することができます。それには、データ構造の名前の後にピリオドとサブフィールドの名前を付けます。たとえば、PEMPL.MIDINT は MIDINT だけを指定したのと同じです。

SQL を使用する RPG for iSeries アプリケーションでのホスト構造配列の使用

ホスト構造は、オカレンス・データ構造として定義されます。オカレンス・データ構造は、複数行を取り出すときに SQL の FETCH ステートメントで使用することができます。このような例では、以下の条件が該当します。

- BARRAY 内のすべての項目は有効なホスト変数でなければなりません。
- BARRAY 内のすべての項目は連続していなければなりません。最初の FROM の位置は 1 行でなければならず、TO と FROM の位置はオーバーラップできません。
- 複数行用 FETCH およびブロック化 INSERT 以外のすべてのステートメントの場合、オカレンス・データ構造を使用すると、現行オカレンスが使用されます。複数行用 FETCH および INSERT の場合、オカレンスは 1 にセットします。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7. ...*
IBARRAY      DS                      10
I              01 20 C1VAR
I              B 21 220C2VAR
```

以下の例では、DEPT というホスト構造配列および複数行用 FETCH ステートメントを使用して、DEPARTMENT テーブルから 10 行を取り出しています。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....*
E              INDS          4 4 0
IDEPT         DS              10
I              01 03 DEPTNO
I              04 32 DEPTNM
I              33 38 MGRNO
I              39 41 ADMRD
IINDARR       DS              10
I              B 1 80INDS
...
C/EXEC SQL
C+ DECLARE C1 CURSOR FOR
C+   SELECT *
C+   FROM CORPDATA.DEPARTMENT
C/END-EXEC
C/EXEC SQL
C+ OPEN C1
C/END-EXEC
C/EXEC SQL
C+ FETCH C1 FOR 10 ROWS INTO :DEPT:INDARR
C/END-EXEC
```

SQL を使用する RPG for iSeries アプリケーションでの外部ファイル記述の使用

SQL プリコンパイラーは、ILE RPG for iSeries コンパイラーと全く同じ方法で RPG for iSeries ソース・コードを処理します。このことは、プリコンパイラーがホスト変数の定義のために /COPY ステートメントを処理することを意味します。異なる名前が定義されているときは、外部記述ファイルのフィールド定義が取り出されて、名前が変更されます。データ構造の外部定義形式を使用すると、列名のコピーをとって、それをホスト変数として使用することができます。

次の例では、サンプル・テーブル DEPARTMENT が RPG for iSeries プログラムの中でファイルとして使用されています。SQL プリコンパイラーは DEPARTMENT のフィールド (列) 定義を取り出して、ホスト変数として使用します。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....*
FTDEPT IP E          DISK
F      TDEPT          KRENAMEDPTREC
IDEPTREC
I      DEPTNAME      DEPTN
I      ADMRDEPT      ADMRD
```

注: RPG for iSeries のステートメントを使用してファイルに入出力操作を行う場合にだけ、RPG プログラムの中のファイルに F 仕様をコーディングしてください。SQL ステートメントだけを使用してファイルに入出力操作を行うときは、外部データ構造を使用すると、外部定義を組み込むことができます。

次の例では、サンプル・テーブルは外部データ構造として指定されています。SQL プリコンパイラーはフィールド (列) 定義をデータ構造のサブフィールドとして検索します。サブフィールド名はホスト変数名として、データ構造名 TDEPT はホスト構造名として使用できます。フィールド名は、長さが 6 文字を超えているので変更しなければなりません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....*
ITDEPT      E DSDEPARTMENT
I           DEPTNAME           DEPTN
I           ADMRDEPT           ADMRD
```

注: DATE、TIME、および TIMESTAMP の各列からはホスト変数定義が生成され、その定義は DATE、TIME、および TIMESTAMP 列と同じ比較と割り当ての規則を使用して、SQL によって扱われます。たとえば、日付ホスト変数は、DATE 列または日付の有効な表現である文字ストリングとだけ突き合わされて比較されます。

可変長の列からは固定長の文字ホスト変数定義が生成されますが、SQL はそれを可変長文字変数と見なします。

- | GRAPHIC 列と VARGRAPHIC 列は RPG for iSeries で文字変数にマップされますが、SQL はこれらを GRAPHIC 変数および VARGRAPHIC 変数と見なします。GRAPHIC 列または VARGRAPHIC 列が UCS-2 CCSID を持つ場合、生成されるホスト変数には UCS-2 CCSID が割り当てられます。GRAPHIC 列または VARGRAPHIC 列が UTF-16 CCSID を持つ場合、生成されるホスト変数には UTF-16 CCSID が割り当てられます。

別の例については、『SQL を使用する RPG for iSeries アプリケーションでのホスト構造配列の外部ファイル記述に関する考慮事項』を参照してください。

SQL を使用する RPG for iSeries アプリケーションでのホスト構造配列の外部ファイル記述に関する考慮事項

外部記述ファイルのフィールド定義 (フィールドの名前変更を含む) は、SQL プリコンパイラーによって認識されます。データ構造の外部定義形式を使用すると、列名のコピーをとって、それをホスト変数として使用することができます。

以下の例では、DEPARTMENT テーブルは RPG for iSeries プログラムに取り込まれて、ホスト構造配列を宣言するために使用されます。その後、複数行用 FETCH ステートメントが使用されて、10 行をホスト構造配列に取り出します。

```
*...1....+....2....+....3....+....4....+....5....+....6....*
ITDEPT      E DSDEPARTMENT           10
I           DEPARTMENT           DEPTN
I           ADMRDEPT           ADMRD
```

...

```
C/EXEC SQL
C+  DECLARE C1 CURSOR FOR
C+    SELECT *
C+    FROM CORPDATA.DEPARTMENT
C/END-EXEC
```

...

```
C/EXEC SQL
C+  FETCH C1 FOR 10 ROWS INTO :TDEPT
C/END-EXEC
```


SQL データ・タイプと RPG for iSeries データ・タイプの対応関係の判別

プリコンパイラーは、次の表に基づいて、ホスト変数のベース SQLTYPE とベース SQLLEN を判断します。ホスト変数が標識変数と一緒に記載されているときは、その SQLTYPE はベース SQLTYPE に 1 を加えたものです。

表 7. RPG for iSeries 宣言と代表的 SQL データ・タイプとの対応関係

RPG for iSeries データ・タイプ	43 桁目	52 桁目	その他の RPG for iSeries コーディング	ホスト変数の SQLTYPE	ホスト変数の SQLLEN	SQL データ・ タイプ
データ構造サブフィールド	ブランク	ブランク	長さ = n (n ≤ 256)	452	n	CHAR(n)
データ構造 (サブフィールドなし)	適用外	適用外	長さ = n (n ≤ 9999)	452	n	CHAR(n)
入力フィールド	ブランク	ブランク	長さ = n (n ≤ 256)	452	n	CHAR(n)
計算結果フィールド	適用外	ブランク	長さ = n (n ≤ 256)	452	n	CHAR(n)
データ構造サブフィールド	B	0	長さ = 2	500	2	SMALLINT
データ構造サブフィールド	B	0	長さ = 4	496	4	INTEGER
データ構造サブフィールド	B	1~4	長さ = 2	500	2	DECIMAL(4,s) (s = 52 桁目)
データ構造サブフィールド	B	1~9	長さ = 4	496	4	DECIMAL(9,s) (s = 52 桁目)
データ構造サブフィールド	P	0 から 9 まで	長さ = n (n は 1 から 16 まで)	484	バイト 1 には p、バイト 2 には s	DECIMAL(p,s) (p = n*2-1、s = 52 桁目)
入力フィールド	P	0 から 9 まで	長さ = n (n は 1 から 16 まで)	484	バイト 1 には p、バイト 2 には s	DECIMAL(p,s) (p = n*2-1、s = 52 桁目)
入力フィールド	ブランク	0 から 9 まで	長さ = n (n は 1 から 30 まで)	484	バイト 1 には p、バイト 2 には s	DECIMAL(p,s) (p = n、s = 52 桁目)
入力フィールド	B	n = 2 であれば 0 から 4 まで、n = 4 であれば 0 から 9 まで	長さ = 2 または 4	484	バイト 1 には p、バイト 2 には s	DECIMAL(p,s) (n = 2 であれば p = 4、n = 4 かつ s = 52 桁目であれば 9)
計算結果フィールド	適用外	0 から 9 まで	長さ = n (n は 1 から 30 まで)	484	バイト 1 には p、バイト 2 には s	DECIMAL(p,s) (p = n、s = 52 桁目)
データ構造サブフィールド	ブランク	0 から 9 まで	長さ = n (n は 1 から 30 まで)	488	バイト 1 には p、バイト 2 には s	NUMERIC(p,s) (p = n、s = 52 桁目)

下表を参照すると、各 SQL データ・タイプに対応する RPG for iSeries データ・タイプを判別することができます。

表 8. SQL データ・タイプと代表的な RPG for iSeries 宣言との対応関係

SQL データ・タイプ	RPG for iSeries データ・タイプ	注
SMALLINT	データ構造のサブフィールド。サブフィールド仕様の 43 桁目が B、長さが 2、52 桁目が 0。	
INTEGER	データ構造のサブフィールド。サブフィールド仕様の 43 桁目が B、長さが 4、52 桁目が 0。	
BIGINT	正確に対応するものなし	サブフィールド仕様の 43 桁目は P、52 桁目は 0 とする。
DECIMAL	データ構造のサブフィールド。サブフィールド仕様の 43 桁目は P、52 桁目は 0 から 9 とする。 または 数値として定義され、データ構造のサブフィールドではないもの。	最大長は 16 (精度 30)、最大位取りは 9。
NUMERIC	データ構造のサブフィールド。サブフィールドの 43 桁目がブランク、52 桁目が 0 から 9 まで。	最大長は 30 (精度 30)、最大位取りは 9。
FLOAT (単精度)	正確に対応するものなし	上述した代替数値データ・タイプの 1 つを使用してください。
FLOAT (倍精度)	正確に対応するものなし	上述した代替数値データ・タイプの 1 つを使用してください。
CHAR(n)	データ構造のサブフィールドまたは入力フィールド。仕様の 43 桁目と 52 桁目がブランク。 または 小数部の桁なしで定義された計算結果フィールド。	n は 1 から 256 まで可能。
CHAR(n)	データ構造にサブフィールドがないデータ構造名。	n は 1 から 9999 まで可能。
VARCHAR(n)	正確に対応するものなし	予想される最大の VARCHAR 値が収まるだけの大きさの文字ホスト変数を使用してください。
CLOB	サポートなし	サポートなし
GRAPHIC(n)	サポートなし	サポートなし
VARGRAPHIC(n)	サポートなし	サポートなし
DBCLOB	サポートなし	サポートなし
BINARY	サポートなし	サポートなし
VARBINARY	サポートなし	サポートなし
BLOB	サポートなし	サポートなし

表 8. SQL データ・タイプと代表的な RPG for iSeries 宣言との対応関係 (続き)

SQL データ・タイプ	RPG for iSeries データ・タイプ	注
DATE	データ構造のサブフィールド。サブフィールド仕様の 52 桁目がブランク。 または 小数部の桁なしで定義されたフィールド。	形式が *USA、*JIS、*EUR、または *ISO のときは、長さは少なくとも 10 が必要。形式が *YMD、*DMY、または *MDY のときは、長さは少なくとも 8 が必要。形式が *JUL のときは、長さは少なくとも 6 が必要。
TIME	データ構造のサブフィールド。サブフィールド仕様の 52 桁目がブランク。 または 小数部の桁なしで定義されたフィールド。	長さは少なくとも 6 が必要。秒を含めるときは、長さは少なくとも 8 が必要。
TIMESTAMP	データ構造のサブフィールド。サブフィールド仕様の 52 桁目がブランク。 または 小数部の桁なしで定義されたフィールド。	長さは少なくとも 19 が必要。マイクロ秒を全桁の精度で含める場合は、長さは少なくとも 26 が必要。長さが 26 未満のときは、マイクロ秒部分で切り捨てが起きます。
DATALINK	サポートなし	サポートなし
ROWID	サポートなし	サポートなし

詳細については、『SQL を使用する RPG for iSeries アプリケーションでの割り当て規則』を参照してください。

SQL を使用する RPG for iSeries アプリケーションでの割り当て規則

RPG for iSeries は、精度と位取りをすべての数値タイプと関連付けます。RPG for iSeries は、データがパック形式であるものとして、数値演算を定義します。すなわち、2 進数の変数が関係する演算は暗黙的にパック形式に変換されてから、演算が行われます (必要ならば、2 進数に逆変換されます)。データは暗黙の小数点位置に合わされて、SQL 演算が行われます。

SQL を使用する RPG for iSeries アプリケーションでの標識変数の使用

標識変数は 2 バイトの整数です (113 ページの表 7 の SMALLINT SQL データ・タイプの項目を参照)。

標識構造は、要素の長さが 4,0 の配列として変数を宣言し、43 桁目が B のデータ構造のサブフィールドとして配列名を宣言することによって定義することができます。検索されるとき、標識変数はその対応するホスト変数にヌル値が割り当てられているかどうかを示すために使用されます。列に割り当てるときには、ヌル値を割り当てべきであることを示すために負の標識変数が使用されます。

詳細については、「SQL 解説書」の『標識変数』を参照してください。

標識変数の宣言の仕方はホスト変数の場合と同じであり、これらの 2 つの変数の宣言をプログラマーに適切と思われる方法で組み合わせることができます。

標識変数の使用例については、116 ページの『例: SQL を使用する RPG for iSeries アプリケーションでの標識変数の使用』を参照してください。

例: SQL を使用する RPG for iSeries アプリケーションでの標識変数の使用

次のステートメントがあるとします。

```
*...1....+...2....+...3....+...4....+...5....+...6....+...7...*
C/EXEC SQL FETCH CLS_CURSOR INTO :CLSCD,
C+           :DAY :DAYIND,
C+           :BGN :BGNIND,
C+           :END :ENDIND
C/END-EXEC
```

変数は次のように宣言することができます。

```
*...1....+...2....+...3....+...4....+...5....+...6....+...7...*
I           DS
I           1   7 CLSCD
I           B   8   90DAY
I           B  10 110DAYIND
I           12  19 BGN
I           B  20 210BGNIND
I           22  29 END
I           B  30 310ENDIND
```

構造パラメーター受け渡し技法による RPG for iSeries での相違

SQL RPG for iSeries プリコンパイラーは、可能ならば、構造パラメーター受け渡し技法の使用を試みます。プリコンパイラーは、以下の条件に該当する場合、各ホスト変数が分離パラメーターになっているコードを生成します。

- ステートメントで参照されているホスト変数のデータ長が 9935 を超えている場合。SQL は構造のうち 64 バイトを使用するため、データ構造の最大長は $9935 + 64 = 9999$ となります。
- ステートメントで標識が指定されており、指標付き標識名の長さに必要な標識値を加えたものが 6 文字を超えている場合。プリコンパイラーは結果フィールドの標識名を用いて標識に関する割り当てステートメントを生成しなければなりません、この長さは 6 文字に制限されています ("INDIC,1" には 7 文字が必要)。
- ホスト変数名の長さが 256 を超える場合。これは、サブフィールドがないデータ構造がホスト変数として使用され、その長さが 256 を超えていると、起こります。サブフィールドは、256 を超える長さで定義することはできません。

構造パラメーター受け渡し技法の詳細については、データベース・パフォーマンスと最適化に関するトピックの『データベース・アプリケーション設計のヒント: 構造パラメーター受け渡し方法の使用』を参照してください。

呼び出された SQL を使用する RPG for iSeries プログラムの正しい終了方法

SQL は実行時に、ホスト変数を含む各 SQL ステートメントについて、データ域 (内部 SQLDA) を作成し、保持します。これらの内部 SQLDA は、ステートメントが最初に実行されるときに作成され、その後のステートメント実行時に再使用され、パフォーマンスの向上が図られます。内部 SQLDA は、少なくとも 1 つの SQL プログラムが活動している限り、再使用されます。SQL プリコンパイラーは、SQL 実行時に使用される静的記憶域を割り振り、内部 SQLDA を正しく管理します。

SQL を含む RPG for iSeries プログラムが、やはり SQL を含む他のプログラムから呼び出された場合、RPG for iSeries プログラムは最終レコード (LR) 標識をオンに設定してはなりません。LR 標識をオンに

設定すると、静的記憶域は RPG for iSeries プログラムが次に実行されるときに再初期設定されます。静的記憶域の再初期設定は内部 SQLDA の再作成をもたらす、したがってパフォーマンスが低下します。

SQL ステートメントを含んでいる RPG for iSeries プログラムが、やはり SQL ステートメントを含んでいるプログラムから呼び出されたとき、その RPG for iSeries プログラムは以下の 2 つの方法のいずれかによって終了しなければなりません。

- RETRN ステートメントによって
- RT 標識をオンにセットすることによって

このような方法によれば、内部 SQLDA が再使用可能になり、合計実行時間を減らすことができます。

第 9 章 ILE RPG for iSeries アプリケーションでの SQL ステートメントのコーディング方法



ここでは、SQL ステートメントを ILE RPG for iSeries プログラムに組み込む場合に固有のアプリケーションおよびコーディング上の要件について説明します。ホスト変数に関するコーディング要件についても説明します。

詳細については、以下のセクションを参照してください。

- 『SQL を使用する ILE RPG for iSeries アプリケーションでの SQL 連絡域の定義』
- 120 ページの『SQL を使用する ILE RPG for iSeries アプリケーションでの SQL 記述子域の定義』
- 122 ページの『SQL を使用する ILE RPG for iSeries アプリケーションでの SQL ステートメントの組み込み』
- 124 ページの『SQL を使用する ILE RPG for iSeries アプリケーションでのホスト変数の使用』
- 129 ページの『SQL を使用する ILE RPG for iSeries アプリケーションでのホスト構造の使用』
- 132 ページの『SQL を使用する ILE RPG for iSeries アプリケーションでのホスト構造配列の使用』
- 127 ページの『SQL を使用する ILE RPG iSeries アプリケーションでの LOB ホスト変数の宣言』
- 129 ページの『SQL を使用する ILE RPG for iSeries アプリケーションでの ROWID 変数の宣言』
- 133 ページの『SQL を使用する ILE RPG iSeries アプリケーションでの外部ファイル記述の使用』
- 134 ページの『SQL データ・タイプと ILE RPG データ・タイプの対応関係の判別』
- 141 ページの『SQL を使用する ILE RPG for iSeries アプリケーションでの標識変数の使用』
- 142 ページの『SQL を使用する ILE RPG for iSeries アプリケーションでの複数行領域取り出し用 SQLDA の例』
- 143 ページの『SQL を使用する ILE RPG for iSeries アプリケーションでの動的 SQL の例』

SQL ステートメントの使い方を示した詳しい ILE RPG プログラムは、197 ページの『例: ILE RPG for iSeries プログラム内の SQL ステートメント』に記載されています。

注: 重要な法律的事柄に関する情報について、214 ページの『コードについての特記事項』をご覧ください。

ILE RPG を使用するプログラミングの詳細については、「ILE RPG プログラマーの手引き 」および「ILE RPG 言語解説書 」を参照してください。

SQL を使用する ILE RPG for iSeries アプリケーションでの SQL 連絡域の定義

ILE SQL プリコンパイラーは、SET OPTION SQLCA = *NO ステートメントが指定されていない限り、ILE RPG for iSeries プログラムの最初の演算仕様の前の定義仕様に SQLCA を自動的に入れます。INCLUDE SQLCA をソース・プログラムにコーディングする必要はありません。ソース・プログラムに INCLUDE SQLCA を指定した場合、そのステートメントは受け入れられますが、余分なものとして扱われます。ILE RPG for iSeries 用の SQLCA ソース・ステートメントは、次のようになります。

```

| D*      SQL Communications area
| D SQLCA      DS
| D SQLCAID      8A  INZ(X'0000000000000000')
| D SQLAID      8A  OVERLAY(SQLCAID)
| D SQLCABC      10I 0
| D SQLABC      9B 0 OVERLAY(SQLCABC)
| D SQLCODE      10I 0
| D SQLCOD      9B 0 OVERLAY(SQLCODE)
| D SQLERRML     5I 0
| D SQLERL      4B 0 OVERLAY(SQLERRML)
| D SQLERRMC     70A
| D SQLERM      70A OVERLAY(SQLERRMC)
| D SQLERRP      8A
| D SQLERP      8A  OVERLAY(SQLERRP)
| D SQLERR       24A
| D SQLER1      9B 0 OVERLAY(SQLERR:*NEXT)
| D SQLER2      9B 0 OVERLAY(SQLERR:*NEXT)
| D SQLER3      9B 0 OVERLAY(SQLERR:*NEXT)
| D SQLER4      9B 0 OVERLAY(SQLERR:*NEXT)
| D SQLER5      9B 0 OVERLAY(SQLERR:*NEXT)
| D SQLER6      9B 0 OVERLAY(SQLERR:*NEXT)
| D SQLERRR     10I 0 DIM(6) OVERLAY(SQLERR)
| D SQLWRN      11A
| D SQLWN0      1A  OVERLAY(SQLWRN:*NEXT)
| D SQLWN1      1A  OVERLAY(SQLWRN:*NEXT)
| D SQLWN2      1A  OVERLAY(SQLWRN:*NEXT)
| D SQLWN3      1A  OVERLAY(SQLWRN:*NEXT)
| D SQLWN4      1A  OVERLAY(SQLWRN:*NEXT)
| D SQLWN5      1A  OVERLAY(SQLWRN:*NEXT)
| D SQLWN6      1A  OVERLAY(SQLWRN:*NEXT)
| D SQLWN7      1A  OVERLAY(SQLWRN:*NEXT)
| D SQLWN8      1A  OVERLAY(SQLWRN:*NEXT)
| D SQLWN9      1A  OVERLAY(SQLWRN:*NEXT)
| D SQLWNA      1A  OVERLAY(SQLWRN:*NEXT)
| D SQLWARN     1A  DIM(11) OVERLAY(SQLWRN)
| D SQLSTATE     5A
| D SQLSTT      5A  OVERLAY(SQLSTATE)
| D* End of SQLCA

```

| SQLCA の詳細については、「SQL 解説書」の『SQL 連絡域』を参照してください。

| SET OPTION SQLCA = *NO ステートメントが見つかった場合、SQL プリコンパイラーは、SQLCODE
| 変数および SQLSTATE 変数を定義仕様に自動的に入れます。これらの変数は、SQLCA が組み込まれてい
| ない場合、次のように定義されます。

```

| D SQLCODE      S      10I 0
| D SQLSTATE     S      5A

```

SQL を使用する ILE RPG for iSeries アプリケーションでの SQL 記述子域の定義

SQLDA を必要とするステートメントには、次のものがあります。

```

EXECUTE...USING DESCRIPTOR 記述子名
FETCH...USING DESCRIPTOR 記述子名
OPEN...USING DESCRIPTOR 記述子名
CALL...USING DESCRIPTOR 記述子名
DESCRIBE ステートメント名 INTO 記述子名
DESCRIBE TABLE ホスト変数 INTO 記述子名
PREPARE ステートメント名 INTO 記述子名

```


SQLCA と異なり、SQLDA を 2 つ以上プログラムの中に置くことができ、また SQLDA の名前は有効であれば、どの名前でも使えます。

動的 SQL はプログラミング技法の 1 つであり、「SQL プログラミング 概念」についてのトピックで説明されています。動的 SQL を使用すると、ユーザーのプログラムはその実行と平行して SQL ステートメントを作成し、実行させることができます。動的に実行される変数 SELECT リスト (すなわち、照会の一部として返される列のリスト) を指定する SELECT ステートメントには、SQL 記述子域 (SQLDA) が必要です。これは、SELECT の結果を受け入れるために割り振るべき変数の数とタイプが事前に予測できないからです。

INCLUDE SQLDA ステートメントは、ILE RPG for iSeries プログラムで指定することができます。このステートメントの形式は次のとおりです。

```
C/EXEC SQL INCLUDE SQLDA
C/END-EXEC
```

INCLUDE SQLDA は、次のデータ構造を生成します。

```
D*      SQL Descriptor area
D SQLDA      DS
D  SQLDAID      1      8A
D  SQLDABC      9     12B 0
D  SQLN        13     14B 0
D  SQLD        15     16B 0
D  SQL_VAR      80A   DIM(SQL_NUM)
D
D              17     18B 0
D              19     20B 0
D              21     32A
D              33     48*
D              49     64*
D              65     66B 0
D              67     96A
D*
D SQLVAR      DS
D  SQLTYPE      1      2B 0
D  SQLLEN       3      4B 0
D  SQLRES       5      16A
D  SQLDATA      17     32*
D  SQLIND       33     48*
D  SQLNAMELEN   49     50B 0
D  SQLNAME      51     80A
D* End of SQLDA
```

ユーザーは、SQL_NUM の定義を行わなければなりません。SQL_NUM は、SQL_VAR に必要な次元を持つ数値定数として定義する必要があります。

INCLUDE SQLDA は、2 つのデータ構造を生成します。2 番目のデータ構造を使用して、フィールド記述の入っている SQLDA の部分をセットアップおよび参照できます。

SQLDA のフィールド記述をセットするために、プログラムは SQLVAR のサブフィールドにフィールド記述をセットアップした後、SQLVAR を SQL_VAR(n) に割り当てます (n は SQLDA 内のフィールド数)。この動作は、すべてのフィールド記述がセットされるまで繰り返されます。

SQLDA フィールド記述を参照するとき、ユーザーは SQLVAR(n) を SQL_VAR に割り当てます (n は処理されるフィールド記述の数)。

SQLDA の詳細については、「SQL 解説書」の『SQL 記述子域』を参照してください。

SQL を使用する ILE RPG for iSeries アプリケーションでの SQL ステートメントの組み込み

ILE RPG プログラムの中にコーディングする SQL ステートメントは、演算部分に置かなければなりません。このためには C を 6 桁目に入れる必要があります。SQL ステートメントは、明細演算にも、合計演算にも、RPG サブルーチンにも置くことができます。SQL ステートメントは、RPG ステートメントのロジックに基づいて実行されます。

キーワード EXEC は SQL ステートメントの始まりを示します。EXEC SQL はソース・ステートメントの 8 桁目から 16 桁目までに置き、7 桁目にスラッシュ (/) を置かなければなりません。SQL ステートメントを 17 桁目から始めて、80 桁目まで続けることができます。

キーワード END-EXEC は SQL ステートメントの終わりを示します。END-EXEC はソース・ステートメントの 8 桁目から 16 桁目までに置き、7 桁目にスラッシュ (/) を置かなければなりません。17 桁目から 80 桁目までは空白にしておきます。

SQL ステートメントでは大文字と小文字の両方が使用できます。

ILE RPG for iSeries プログラムの中にコーディングされる UPDATE ステートメントは、次のようにコーディングされます。

```
C/EXEC SQL UPDATE DEPARTMENT
C+          SET MANAGER = :MGRNUM
C+          WHERE DEPTNO = :INTDEP
C/END-EXEC
```

詳細については、以下のセクションを参照してください。

- 『SQL を使用する ILE RPG for iSeries アプリケーションでの注記』
- 『SQL を使用する ILE RPG for iSeries アプリケーションでの SQL ステートメントの継続』
- 123 ページの『SQL を使用する ILE RPG for iSeries アプリケーションでのコードの組み込み』
- 123 ページの『SQL を使用する ILE RPG for iSeries アプリケーションでのディレクティブの使用』
- 124 ページの『SQL を使用する ILE RPG for iSeries アプリケーションでの順序番号』
- 124 ページの『SQL を使用する ILE RPG for iSeries アプリケーションでの名前』
- 124 ページの『SQL を使用する ILE RPG for iSeries アプリケーションでのステートメント・ラベル』
- 124 ページの『SQL を使用する ILE RPG for iSeries アプリケーションでの WHENEVER ステートメント』

SQL を使用する ILE RPG for iSeries アプリケーションでの注記

SQL 注記 (--) の他に、ILE RPG for iSeries の注記は、SQL ステートメント内の空白文字が許されている場所のどこにでも置くことができます。ILE RPG for iSeries の注記を SQL ステートメントに組み込むときは、7 桁目にアスタリスク (*) を置きます。

SQL を使用する ILE RPG for iSeries アプリケーションでの SQL ステートメントの継続

SQL ステートメントを収めるために追加レコードが必要なときは、9 桁目から 80 桁目が使用できます。7 桁目は + (正符号) にし、8 桁目は空白にしなければなりません。継続される行の 80 桁目は、継続行の 9 桁目と連結します。

DBCS データが入っている定数は、継続される行の 80 桁目にシフトイン文字を入れ、継続行の 8 桁目にシフトアウト文字を入れることによって、複数行にわたって継続させることができます。

この例では、SQL ステートメントの G'<AABBCCDDEEFFGGHHIIJJKK>' はグラフィック定数として有効です。

```
C/EXEC SQL SELECT * FROM GRAPHTAB WHERE GRAPHCOL = G'<AABBCCDDEE>
C+<FFGGHHIIJJKK>'
C/END-EXEC
```

SQL を使用する ILE RPG for iSeries アプリケーションでのコードの組み込み

以下の SQL ステートメントを使用して、SQL ステートメントと RPG 仕様を含めることができます。

```
C/EXEC SQL INCLUDE member-name
C/END-EXEC
```

RPG ディレクティブは、RPG プリプロセッサ・オプション・パラメーター (RPGPPOPT) の値に応じて SQL プリコンパイラによって処理されます。

コードを組み込むための RPG ディレクティブの使用については、『SQL を使用する ILE RPG for iSeries アプリケーションでのディレクティブの使用』を参照してください。

SQL を使用する ILE RPG for iSeries アプリケーションでのディレクティブの使用

RPG ディレクティブは、RPG プリプロセッサ・オプション・パラメーター (RPGPPOPT) の値に応じて SQL プリコンパイラによって処理されます。RPG プリプロセッサを使用する場合、展開されたプリプロセス・ソースを使って SQL プリコンパイルが実行されます。

- 値が *NONE の場合、RPG ソースをプリプロセスするために RPG プリプロセッサが呼び出されることはありません。SQL プリコンパイラによって処理されるディレクティブは /COPY だけです。ネストされた /COPY ステートメントは処理されません。その他のすべてのディレクティブは、RPG コンパイラが呼び出されるまで、無視されます。つまり、条件付き論理ブロック内にある RPG ステートメントおよび SQL ステートメントはすべて、SQL プリコンパイラによって無条件に処理されます。
- 値が *LVL1 の場合、RPG ソースをプリプロセスするために RPG プリプロセッサが呼び出されません。ネストされた /COPY ステートメントを含むすべての /COPY ステートメントが展開されて、条件付きコンパイル・ディレクティブが処理されます。
- 値が *LVL2 の場合、RPG ソースをプリプロセスするために RPG プリプロセッサが呼び出されません。すべての /COPY および /INCLUDE ステートメントが展開されて、条件付きコンパイル・ディレクティブが処理されます。
- *LVL1 または *LVL2 を使用する場合、/COPY および /INCLUDE ステートメントの展開が原因で、RPG プリプロセッサによって生成される展開されたソースが非常に大きくなり、リソースの限界に達する可能性があります。これが発生する場合、ソースを細かく分割する必要があります。または、RPG プリプロセッサを使用しないでください。

SQL を使用する ILE RPG for iSeries アプリケーションでの順序番号

SQL プリコンパイラーによって生成されるソース・ステートメントの順序番号は、CRTSQLRPGI コマンドの OPTION パラメーターの有効な *NOSEQSRC/*SEQSRC キーワードに基づきます。*NOSEQSRC が指定されると、入力ソース・メンバーからの順序番号が使用されます。*SEQSRC を指定すると、順序番号は 000001 から始まり、1 ずつ増えます。

SQL を使用する ILE RPG for iSeries アプリケーションでの名前

有効な ILE RPG for iSeries 変数名であれば、どんな名前でもホスト変数に使用できますが、次のような制約があります。

- 'SQ'、'SQL'、'RDI'、または 'DSN' の文字で始まるホスト変数や外部入り口名は、使用してはなりません。これらの名前はデータベース・マネージャー用に予約されています。
- ホスト変数名の長さは 64 までです。
- ホスト変数の名前は、プログラムの中で固有にしてください。複数の変数の名前が同じで、データ・タイプが異なっている場合には、ホスト変数のデータ・タイプは予測できません。

SQL を使用する ILE RPG for iSeries アプリケーションでのステートメント・ラベル

どの SQL ステートメントの場合も、その前に TAG ステートメントを置くことができます。TAG ステートメントは EXEC SQL の前の行にコーディングします。

SQL を使用する ILE RPG for iSeries アプリケーションでの WHENEVER ステートメント

GOTO 文節の対象となるものは、TAG ステートメントのラベルでなければなりません。GOTO/TAG の有効範囲に従わなければなりません。

SQL を使用する ILE RPG for iSeries アプリケーションでのホスト変数の使用

SQL ステートメントの中で使用するホスト変数はいずれも明示的に宣言しなければなりません。

ILE RPG for iSeries に組み込まれた SQL は、ホスト変数を識別するために SQL の BEGIN DECLARE SECTION および END DECLARE SECTION ステートメントを使用しません。これらのステートメントをソース・プログラムに入れてはなりません。

SQL ステートメントの中のホスト変数はいずれも、その前にコロン (:) を付けなければなりません。

- ホスト変数の名前は、ホスト変数がそれぞれ別のプロシージャの中にある場合であっても、1 つのプログラム内では固有でなければなりません。ただし、データ構造に QUALIFIED キーワードが含まれる場合には、そのデータ構造のサブフィールドの名前が、別のデータ構造のサブフィールドまたは独立型の変数と同じ名前になる可能性があります。サブフィールド名を修飾するために、データ構造名を使用することによって、QUALIFIED キーワードを含むデータ構造のサブフィールドを参照する必要があります。

ホスト変数を使用する SQL ステートメントは、その変数が宣言されたステートメントの有効範囲内になければなりません。

ホスト変数が未定義または使用できないというエラーが発生した場合には、プリコンパイラー・リスト内の相互参照を調べて、プリコンパイラーがどのように変数を定義したかを確認してください。リスト内に相互参照を生成するには、OPTIONS パラメーターに *XREF を指定してプリコンパイル・コマンドを実行します。

詳細については、『SQL を使用する ILE RPG for iSeries アプリケーションでのホスト変数の宣言』を参照してください。

SQL を使用する ILE RPG for iSeries アプリケーションでのホスト変数の宣言

SQL ILE RPG for iSeries プリコンパイラーは、有効なホスト変数として有効な ILE RPG for iSeries 宣言のサブセットしか認識しません。

ILE RPG for iSeries で定義した変数はほとんど、SQL ステートメントの中で使用できます。サポートされていない変数の一部を以下にリストします。

符号なし整数

ポインター

テーブル

UPDATE

UDAY

UMONTH

UYEAR

先読みフィールド

名前付き定数

複数次元の配列

%SIZE または %ELEM の解決を必要とする定義

定数を OCCURS または DIM で使用していない限り、定数の解決が必要な定義

ホスト変数として使用されるフィールドは、ILE RPG for iSeries の CALL/PARM 機能を使用して SQL に渡されます。PARM の結果フィールドで使用できないフィールドは、ホスト変数として使用できません。

日付および時刻ホスト変数は、SQL プリコンパイラーが生成する構造の中の対応する日付および時刻サブフィールドに常に割り当てられます。生成される日時サブフィールドは、CRTSQLRPGI コマンド (または SET OPTION ステートメント) の DATFMT、DATSEP、TIMFMT、TIMSEP パラメーターで指定された形式および分離文字によって宣言されます。ユーザーが宣言したホスト変数形式からプリコンパイルを指定した形式への変換は、SQL 生成構造への割り当て時および SQL 生成構造からの割り当て時に起こります。DATFMT パラメーター値がシステム形式 (*MDY、*YMD、*DMY、あるいは *JUL) である場合は、すべての入出力ホスト変数に 1940 ~ 2039 の範囲内の日付値が入っていなければなりません。日付値がこの範囲外である場合は、プリコンパイル時に DATFMT を *ISO、*USA、*EUR、または *JIS の IBM SQL 形式の 1 つとして指定する必要があります。

グラフィック・ホスト変数は、RPG CCSID 値を使用します (指定されている場合)。RPG によって CCSID を定義されたホスト変数の CCSID を変更するために、あるいは UCS-2 または UTF-16 として定義されたホスト変数の CCSID を変更するために、SQL DECLARE VARIABLE ステートメントを使用することはありません。

プリコンパイラーは、RPG 論理 (標識) 変数を長さ 1 の文字として生成します。SQL が文字ホスト変数を許容する場合には常に、このタイプを使用できます。これを SQL 標識変数として使用することはできません。ユーザーは、必ず 1 または 0 の値だけがこれに割り当てられるようにする必要があります。

プリコンパイラーは EXTNAME(filename : fmtname) をサポートしますが、EXTNAME(filename : fmtname : fieldtype) はサポートしません (fieldtype は *ALL、*INPUT、*OUTPUT、または *KEY)。

プリコンパイラーは LIKEREC(intrecname) をサポートしますが、オプションの 2 次パラメーターをサポートしません。

名前のないサブフィールドが存在する場合、プリコンパイラーは、ブロック化したフェッチまたは挿入 (INSERT) ステートメント内でサブフィールドを含むデータ構造が使用されないようにします。サブフィールドを含むデータ構造を使用する他のすべての SQL ステートメントでは、名前の付いたサブフィールドだけが使用されます。

PREFIX キーワードの接頭部にピリオドが含まれる場合、プリコンパイラーは外部記述ファイルを認識しません。

SQL を使用する ILE RPG アプリケーションでのバイナリー・ホスト変数の宣言

ILE RPG for iSeries には、SQL バイナリー・データ・タイプに対応する変数がありません。これらのデータ・タイプで使用するホスト変数を作成するには、SQLTYPE キーワードを使用します。SQL プリコンパイラーは、この宣言を出力ソース・メンバー中、ILE RPG for iSeries 言語宣言に置き換えます。バイナリー宣言は、独立して、またはデータ構造内に置くことができます。

BINARY の例

次のように宣言すると、

```
D MYBINARY S SQLTYPE(BINARY:50)
```

以下のような結果コードが生成されます。

```
D MYBINARY S 50A
```

VARBINARY の例

次のように宣言すると、

```
D MYVARBINARY S SQLTYPE(VARBINARY:100)
```

以下のような結果コードが生成されます。

```
D MYVARBINARY S 100A VARYING
```

注:

1. BINARY ホスト変数では、length (長さ) の範囲は 1 から 32766 まででなければなりません。
2. VARBINARY ホスト変数では、length (長さ) の範囲は 1 から 32740 まででなければなりません。
3. BINARY および VARBINARY ホスト変数はホスト構造内で宣言できます。
4. SQLTYPE、BINARY、および VARBINARY は大文字小文字混合にすることができます。
5. SQLTYPE は、44 から 80 桁の間にする必要があります。
6. BINARY または VARBINARY が独立型ホスト変数として宣言される場合、24 桁目は文字 S、25 桁目はブランクにする必要があります。

7. BINARY または VARBINARY ホスト変数をホスト構造で宣言する場合は、24 桁目にある独立型のフィールド標識 S を省略します。

SQL を使用する ILE RPG iSeries アプリケーションでの LOB ホスト変数の宣言

ILE RPG for iSeries には、LOB (ラージ・オブジェクト) の SQL データ・タイプに対応する変数がありません。これらのデータ・タイプで使用するホスト変数を作成するには、SQLTYPE キーワードを使用します。SQL プリコンパイラーは、この宣言を出力ソース・メンバー中、ILE RPG for iSeries 言語構造に置き換えます。LOB 宣言は、独立して、またはデータ構造内に置くことができます。

詳細については、以下のセクションを参照してください。

- 『SQL を使用する ILE RPG for iSeries アプリケーションでの LOB ホスト変数』
- 128 ページの『SQL を使用する ILE RPG for iSeries アプリケーションでの LOB ロケーター』
- 128 ページの『SQL を使用する ILE RPG for iSeries アプリケーションでの LOB ファイル参照変数』

SQL を使用する ILE RPG for iSeries アプリケーションでの LOB ホスト変数: CLOB の例

次のように宣言すると、

```
D MYCLOB          S          SQLTYPE(CLOB:1000)
```

以下の構造を生成します。

```
D MYCLOB          DS
D MYCLOB_LEN      10U
D MYCLOB_DATA     1000A
```

DBCLOB の例

次のように宣言すると、

```
D MYDBCLOB        S          SQLTYPE(DBCLOB:400)
```

以下の構造を生成します。

```
D MYDBCLOB        DS
D MYDBCLOB_LEN    10U
D MYDBCLOB_DATA   400G
```

BLOB の例

次のように宣言すると、

```
D MYBLOB          S          SQLTYPE(BLOB:500)
```

以下の構造を生成します。

```
D MYBLOB          DS
D MYBLOB_LEN      10U
D MYBLOB_DATA     500A
```

注:

- BLOB、CLOB の場合、 $1 \leq \text{lob-length} \leq 32,766$
- DBCLOB の場合 $1 \leq \text{lob-length} \leq 16,383$
- LOB ホスト変数はホスト構造内で宣言することができます。
- LOB ホスト変数は、ホスト構造配列内では使用できません。代わりに LOB ロケーターを使用します。
- 構造配列内に宣言された LOB ホスト変数は、独立型ホスト変数として使用することはできません。

6. SQLTYPE、BLOB、CLOB、DBCLOB は大文字小文字混合にすることができます。
7. SQLTYPE は、44 から 80 桁の間にする必要があります。
8. LOB が独立型ホスト変数として宣言された場合、24 桁目は文字 'S' とし、25 桁目はブランクにする必要があります。
9. 24 桁目にある独立型のフィールド標識 'S' は、LOB がホスト構造で宣言されている場合は、省略します。
10. LOB ホスト変数は、初期化できません。

SQL を使用する ILE RPG for iSeries アプリケーションでの LOB ロケーター: BLOB ロケーターの例

次のように宣言すると、

```
D MYBLOB          S          SQLTYPE(BLOB_LOCATOR)
```

以下の構造を生成します。

```
D MYBLOB          S          10U
```

CLOB ロケーターおよび DBCLOB ロケーターの構文は、似ています。

注:

1. LOB ロケーターは、ホスト構造内で宣言することができます。
2. SQLTYPE、BLOB_LOCATOR、CLOB_LOCATOR、DBCLOB_LOCATOR は大文字小文字混合にすることができます。
3. SQLTYPE は、44 から 80 桁の間にする必要があります。
4. LOB ロケーターが独立型ホスト変数として宣言された場合、24 桁目は文字 'S' とし、25 桁目はブランクにする必要があります。
5. 24 桁目にある独立型のフィールド標識 'S' は、LOB ロケーターがホスト構造で宣言されている場合は、省略します。
6. LOB ロケーターは、初期化できません。

SQL を使用する ILE RPG for iSeries アプリケーションでの LOB ファイル参照変数: CLOB ファイル参照の例

次のように宣言すると、

```
D MY_FILE         S          SQLTYPE(CLOB_FILE)
```

以下の構造を生成します。

```
D MY_FILE         DS
D MY_FILE_NL      10U
D MY_FILE_DL      10U
D MY_FILE_FO      10U
D MY_FILE_NAME    255A
```

BLOB ロケーターおよび DBCLOB ロケーターの構文は、似ています。

注:

1. LOB ファイル参照変数は、ホスト構造内で宣言することができます。
2. SQLTYPE、BLOB_FILE、CLOB_FILE、DBCLOB_FILE は大文字小文字混合にすることができます。
3. SQLTYPE は、44 から 80 桁の間にする必要があります。

4. LOB ファイル参照が独立型ホスト変数として宣言された場合、24 桁目は文字 'S' とし、25 桁目は空白にする必要があります。
5. 24 桁目にある独立型のフィールド標識 'S' は、LOB ファイル参照変数がホスト構造で宣言されている場合は、省略します。
6. LOB ファイル参照変数は、初期化できません。

プリコンパイラーは、次のファイル・オプション定数に対する宣言を生成します。ファイル参照ホスト変数を使用する場合、これらの定数を使用して、xxx_FO 変数を設定できます。これらの値の詳細については、「SQL プログラミング 概念」の『LOB ファイル参照変数』を参照してください。

- SQFRD (2)
- SQFCRT (8)
- SQFOVR (16)
- SQFAPP (32)

SQL を使用する ILE RPG for iSeries アプリケーションでの ROWID 変数の宣言

ILE RPG for iSeries には、ROWID の SQL データ・タイプに対応する変数がありません。このデータ・タイプで使用するホスト変数を作成するには、SQLTYPE キーワードを使用します。SQL プリコンパイラーは、この宣言を出力ソース・メンバー中、ILE RPG for iSeries 言語宣言に置き換えます。ROWID 宣言は、独立して、またはデータ構造内に置くことができます。

ROWID の例

次のように宣言すると、

```
D MY_ROWID      S          SQLTYPE(ROWID)
```

以下の構造を生成します。

```
D MYROWID      S          40A VARYING
```

注:

1. SQLTYPE、ROWID は大文字小文字混合にすることができます。
2. ROWID ホスト変数はホスト構造内で宣言することができます。
3. SQLTYPE は、44 から 80 桁の間にする必要があります。
4. ROWID が独立型ホスト変数として宣言された場合、24 桁目は文字 'S' とし、25 桁目は空白にする必要があります。
5. 24 桁目にある独立型のフィールド標識 'S' は、ROWID がホスト構造で宣言されている場合は、省略します。
6. ROWID ホスト変数は、初期化できません。

SQL を使用する ILE RPG for iSeries アプリケーションでのホスト構造の使用

ILE RPG for iSeries データ構造名は、データ構造にサブフィールドがあれば、ホスト構造名として使用できます。SQL ステートメントの中でデータ構造名を使用したときは、そのデータ構造を構成するサブフィールド名のリストを暗黙に指定したことになります。

名前のない 1 つまたは複数のサブフィールドがデータ構造に含まれる場合、そのデータ構造名は SQL ステートメント内のホスト構造として使用できません。名前の付いたサブフィールドは、ホスト変数として使用できます。

次の例では、BIGCHR はサブフィールドのない ILE RPG for iSeries データ構造となっています。BIGCHR が参照される場合、SQL はそれを長さ 642 の文字ストリングとして扱います。

```
DBIGCHR          DS          642
```

次の例では、PEMPL は EMPNO、FIRSTN、MIDINT、LASTNAME、および DEPTNO のサブフィールドから成るホスト構造の名前です。PEMPL が参照されると、これらのサブフィールドが使用されます。たとえば、CORPDATA.EMPLOYEE の最初の例は EMPNO に入れられ、2 番目の列は FIRSTN に入れられます (以下同様です)。

```
DPEMPL          DS
D EMPNO                01      06A
D FIRSTN               07      18A
D MIDINT               19      19A
D LASTNA               20      34A
D DEPTNO               35      37A

...
C                    MOVE      '000220'      EMPNO

...
C/EXEC SQL
C+ SELECT * INTO :PEMPL
C+ FROM CORPDATA.EMPLOYEE
C+ WHERE EMPNO = :EMPNO
C/END-EXEC
```

SQL ステートメントを書くとき、QUALIFIED データ構造内に存在しないサブフィールドへの参照を修飾することができます。それには、データ構造の名前の後にピリオドとサブフィールドの名前を付けます。たとえば、PEMPL.MIDINT は MIDINT だけを指定したのと同じです。データ構造に QUALIFIED キーワードに含まれる場合、サブフィールド名を修飾するために、データ構造名を使用してサブフィールドを参照する必要があります。

この例では 2 つのデータ構造があり (1 つは QUALIFIED、もう 1 つは非 QUALIFIED)、2 つには同じサブフィールド名が含まれます。

```
Dfststruct      DS
D sub1          4B 0
D sub2          9B 0
D sub3          20I 0
D sub4          9B 0
|
Dsecstruct      DS          QUALIFIED
D sub1          4A
D sub2          12A
D sub3          20I 0
D myvar         5A
D sub5          20A
|
D myvar         S          10I 0
```

ホスト変数としての *secstruct.sub1* の参照は、長さ 4 の文字変数になります。

ホスト変数としての *sub2* は、SQL データ・タイプが小整数になります。その属性は、QUALIFIED ではないデータ構造から取得されます。

- | *myvar* へのホスト変数参照では、独立型の宣言を使用して整数のデータ・タイプを取得します。
- | *secstruct.myvar* を使用する場合、QUALIFIED 構造内の文字変数が使用されます。

sub5 は QUALIFIED データ構造にあるため、*secstruct* を使って修飾しない限り、これを参照することはできません。

- | プリコンパイラーは、LIKEDS キーワードを使って定義されたホスト構造を認識します。ただし、ホスト変数の SQL 構文では、SQL ステートメントで 1 レベルの修飾だけを使用できます。つまり、データ構造 DS のサブフィールド S1 がサブフィールド S2 のデータ構造のように定義されている場合、SQL ステートメントでは、完全修飾ホスト変数名 DS.S1.S2 を使って S2 を参照することができません。S1.S2 をホスト変数参照として使用すると、プリコンパイラーはそれを DS.S1.S2 として認識します。このほかに、以下の制限も適用されます。
- | • 最上位レベルの構造 DS は、配列であってはならない。
- | • S1.S2 は固有でなければならない。つまり、S1.S2 で終わる他の有効な名前がプログラム内に存在してはなりません (たとえば、サブフィールド S1.S2 を含む構造 S1 や、サブフィールド DS3.S0.S1.S2 を含む構造 DS3)。

| 例:

```
| D CustomerInfo      DS              QUALIFIED
| D   Name            20A
| D   Address         50A
|
| D ProductInfo      DS              QUALIFIED
| D   Number          5A
| D   Description     20A
| D   Cost             9P 2
|
| D SalesTransaction...
| D                   DS              QUALIFIED
| D   Buyer           LIKEDS(CustomerInfo)
| D   Seller          LIKEDS(CustomerInfo)
| D   NumProducts     10I 0
| D   Product         LIKEDS(ProductInfo)
| D                   DIM(10)
|
| C/EXEC SQL
| C+ SELECT * INTO :CustomerInfo.Name, :Buyer.Name FROM MYTABLE
| C/END-EXEC
```

| *CustomerInfo.Name* は、QUALIFIED 構造の変数への参照として認識されます。*Buyer.Name* は、*SalesTransaction.Buyer.Name* として定義されます。

| *SalesTransaction.Buyer.Name* は SQL ステートメントで使用できません。SQL 構文では 1 レベルの修飾だけが許容されるためです。COST はディメンションのある配列ですから、*Product.Cost* を SQL ステートメントの中で使用することはできません。

| *SalesTransaction* と同様に定義された *SalesTransaction2* が存在する場合、構造であるサブフィールドを SQL ステートメントの中で使用することはできません。SQL では 1 レベルの修飾だけがサポートされるため、*Buyer.Name* への参照は未確定になります。

SQL を使用する ILE RPG for iSeries アプリケーションでのホスト構造配列の使用

- ホスト構造は、オカレンス・データ構造として、またはキーワード DIM がコーディングされたデータ構造として定義されます。この 2 つのデータ構造はどちらも、複数行を処理するときに、SQL の FETCH または INSERT ステートメントで使用できます。次の項目リストは、複数の行ブロック化サポートによりデータ構造を使用する場合に考慮する必要があります。
- すべてのサブフィールドは有効なホスト変数でなければなりません。
- すべてのサブフィールドは連続していなければなりません。最初の FROM の位置は 1 行でなければならず、TO と FROM の位置はオーバーラップできません。
- ホスト構造内の日付と時刻の形式、および日付と時刻サブフィールドの分離文字が、CRTSQLRPGI コマンド (または SET OPTION ステートメント) の DATFMT、DATSEP、TIMFMT、および TIMSEP の各パラメーターと同じでない場合には、ホスト構造配列は使用できません。

ブロック化した FETCH およびブロック化した INSERT 以外のすべてのステートメントについて、オカレンス・データ構造を使用する場合、現行のオカレンスが使用されます。ブロック化した FETCH およびブロック化した INSERT の場合、オカレンスは 1 にセットされます。

以下の例では、DEPARTMENT というホスト構造配列およびブロック化した FETCH ステートメントを使用して、DEPARTMENT テーブルから 10 行を取り出します。

```
DDEPARTMENT          DS          OCCURS(10)
D DEPTNO              01          03A
D DEPTNM              04          32A
D MGRNO               33          38A
D ADMRD               39          41A
```

```
DIND_ARRAY           DS          OCCURS(10)
D INDS                4B 0 DIM(4)
```

```
...
C/EXEC SQL
C+ DECLARE C1 CURSOR FOR
C+   SELECT *
C+   FROM CORPDATA.DEPARTMENT
C/END-EXEC
...
```

```
C/EXEC SQL
C+   FETCH C1 FOR 10 ROWS
C+   INTO :DEPARTMENT:IND_ARRAY
C/END-EXEC
```

- DIM キーワードを含むデータ構造を許容する SQL ステートメントは、ブロック化した FETCH およびブロック化した INSERT のみです。 *MyStructure(index).MySubfield* のような添え字を含むホスト変数参照は、SQL ではサポートされません。

例:

```
Dfststruct          DS          DIM(10)  QUALIFIED
D sub1              4B 0
D sub2              9B 0
D sub3              20I 0
D sub4              9B 0
|
| C/EXEC SQL
| C+   FETCH C1 FOR 10 ROWS INTO :fststruct
| C/END-EXEC
```

SQL を使用する ILE RPG iSeries アプリケーションでの外部ファイル記述の使用

外部記述ファイルのフィールド定義 (フィールドの名前変更を含む) は、SQL プリコンパイラーによって認識されます。データ構造の外部定義形式を使用すると、列名のコピーをとって、それをホスト変数として使用することができます。

SQL プリコンパイラーにより日付および時刻フィールド定義を検索および処理する方法は、*NOCVTDT または *CVTDT のいずれかを CRTSQLRPGI コマンドの OPTION パラメーターに指定するかによって異なります。*NOCVTDT を指定した場合は、日付および時刻フィールド定義は形式および分離文字と共に取り出されます。*CVTDT を指定した場合は、日付および時刻フィールド定義を取り出したときに形式と分離文字は無視され、プリコンパイラーは、変数宣言が文字形式の日付 / 時刻ホスト変数であると想定します。*CVTDT は ILE RPG for iSeries プリコンパイラー用の互換性オプションです。

- GRAPHIC 列または VARGRAPHIC 列が UCS-2 CCSID を持つ場合、生成されるホスト変数には UCS-2 CCSID が割り当てられます。GRAPHIC 列または VARGRAPHIC 列が UTF-16 CCSID を持つ場合、生成されるホスト変数には UTF-16 CCSID が割り当てられます。

次の例では、サンプル・テーブル DEPARTMENT が ILE RPG for iSeries プログラムの中でファイルとして使用されています。SQL プリコンパイラーは DEPARTMENT のフィールド (列) 定義を取り出して、ホスト変数として使用します。

```
FDEPARTMENTIP E DISK RENAME(ORIGREC:DEPTREC)
```

注: ILE RPG for iSeries ステートメントを使用してファイルに入出力操作を行う場合にだけ、ILE RPG for iSeries プログラムの中のファイルに F 仕様書をコーディングしてください。SQL ステートメントだけを使用してファイルに入出力操作を行うときは、外部データ構造を使用して、ファイル (テーブル) の外部定義を組み込むことができます。

次の例では、サンプル・テーブルは外部データ構造として指定されています。SQL プリコンパイラーはフィールド (列) 定義をデータ構造のサブフィールドとして検索します。サブフィールド名はホスト変数名として、データ構造名 TDEPT はホスト構造名として使用できます。次の例は、プログラムが要求する場合にはフィールド名を変更できることを示します。

```
DTDEPT E DS EXTNAME(DEPARTMENT)
D DEPTN E EXTFLD(DEPTNAME)
D ADMRD E EXTFLD(ADMRDEPT)
```

詳細については、『SQL を使用する ILE RPG for iSeries アプリケーションでのホスト構造配列の外部ファイル記述に関する考慮事項』を参照してください。

SQL を使用する ILE RPG for iSeries アプリケーションでのホスト構造配列の外部ファイル記述に関する考慮事項

装置ファイルの場合は、INDARA の指定がなく、ファイルに標識が入っているときには、その宣言をホスト構造配列として使用できません。生成された構造には標識域が組み入れられ、その標識域があるために記憶域が分離されます。

OPTION(*NOCVTDT) を指定して、ファイル内の日付および時刻形式と日付および時刻フィールド定義の分離文字が CRTSQLRPGI コマンド上の DATFMT、DATSEP、TIMFMT、および TIMSEP の各パラメーターと同じでない場合は、ホスト構造配列は使用できません。

以下の例では、DEPARTMENT テーブルは ILE RPG for iSeries プログラムに取り込まれて、ホスト構造配列を宣言するために使用されます。次にブロック化した FETCH ステートメントを使用して、10 行をホスト構造配列に取り出します。

```
DDEPARTMENT      E DS                      OCCURS(10)
```

```
C/EXEC SQL
C+  DECLARE C1 CURSOR FOR
C+  SELECT *
C+  FROM CORPDATA.DEPARTMENT
C/END-EXEC
```

...

```
C/EXEC SQL
C+  FETCH C1 FOR 10 ROWS
C+  INTO :DEPARTMENT
C/END-EXEC
```

SQL データ・タイプと ILE RPG データ・タイプの対応関係の判別

プリコンパイラーは、下表に基づいて、ホスト変数のベース SQLTYPE とベース SQLLEN を判断します。ホスト変数が標識変数と一緒に記載されているときは、その SQLTYPE はベース SQLTYPE に 1 を加えたものです。

表 9. ILE RPG for iSeries 宣言と代表的 SQL データ・タイプとの対応関係

RPG データ・タイプ	RPG コーディング	ホスト変数の SQLTYPE	ホスト変数の SQLLEN	SQL データ・タイプ
データ構造 (サブフィールドなし)	長さ = n (n ≤ 32766)	452	n	CHAR(n)
ゾーン・データ	<ul style="list-style-type: none"> データ・タイプ S またはブランクのサブフィールドとして定義仕様に定義される。 データ・タイプ S として定義仕様に定義される。 データ・タイプ S またはブランクとして入力仕様に定義される。 	488	バイト 1 には p、バイト 2 には s	NUMERIC(p, s)。ここで p は桁数、s は小数点以下の桁数。

表9. ILE RPG for iSeries 宣言と代表的 SQL データ・タイプとの対応関係 (続き)

RPG データ・タイプ	RPG コーディング	ホスト変数の SQLTYPE	ホスト変数の SQLLEN	SQL データ・タイプ
パック・データ	<ul style="list-style-type: none"> • 小数点以下の桁数 (pos 69-70) を含む、非ブランクとして定義仕様に定義される。 • データ・タイプ P として定義仕様サブフィールドに定義される。 • データ・タイプ P またはブランクとして定義仕様に定義される。 • データ・タイプ P として入力仕様に定義される。 	484	バイト 1 には p、バイト 2 には s	DECIMAL(p, s)。ここで p は桁数、s は小数点以下の桁数。
2 バイト・バイナリー、小数点以下の桁数ゼロ。	<ul style="list-style-type: none"> • TO と FROM の位置を含む、データ・タイプ B、バイト長 2 のサブフィールドとして定義仕様に定義される。 • データ・タイプ B、桁数が 1 から 4 として定義仕様に定義される。 • データ・タイプ B、バイト長 2 として入力仕様に定義される。 	500	2	SMALLINT
4 バイト・バイナリー、小数点以下の桁数ゼロ。	<ul style="list-style-type: none"> • TO と FROM の位置を含む、データ・タイプ B、バイト長 4 のサブフィールドとして定義仕様に定義される。 • データ・タイプ B、桁数が 5 から 9 として定義仕様に定義される。 • データ・タイプ B、バイト長 4 として入力仕様に定義される。 	496	4	INTEGER

表9. ILE RPG for iSeries 宣言と代表的 SQL データ・タイプとの対応関係 (続き)

RPG データ・タイプ	RPG コーディング	ホスト変数の SQLTYPE	ホスト変数の SQLLEN	SQL データ・タイプ
2 バイト整数	<ul style="list-style-type: none"> TO と FROM の位置を含む、データ・タイプ I、バイト長 2 のサブフィールドとして定義仕様に定義される。 データ・タイプ I、桁数が 5 として定義仕様に定義される。 データ・タイプ I、バイト長 2 として入力仕様に定義される。 	500	2	SMALLINT
4 バイト整数	<ul style="list-style-type: none"> TO と FROM の位置を含む、データ・タイプ I、バイト長 4 のサブフィールドとして定義仕様に定義される。 データ・タイプ I、桁数が 10 として定義仕様に定義される。 データ・タイプ I、バイト長 4 として入力仕様に定義される。 	496	4	INTEGER
8 バイト整数	<ul style="list-style-type: none"> TO と FROM の位置を含む、データ・タイプ I、バイト長 8 のサブフィールドとして定義仕様に定義される。 データ・タイプ I、桁数が 20 として定義仕様に定義される。 データ・タイプ I、バイト長 8 として入力仕様に定義される。 	492	8	BIGINT
short float	データ・タイプ = F、長さ = 4	480	4	FLOAT (単精度)
long float	データ・タイプ = F、長さ = 8	480	8	FLOAT (倍精度)
文字	データ・タイプ = A またはブランク、小数点以下の桁数ブランク、長さ 1 から 32766。	452	n	CHAR (n)、ここで n は長さ

表9. ILE RPG for iSeries 宣言と代表的 SQL データ・タイプとの対応関係 (続き)

RPG データ・タイプ	RPG コーディング	ホスト変数の SQLTYPE	ホスト変数の SQLLEN	SQL データ・タイプ
文字、長さは 254 より大の可変	データ・タイプ = A またはブランク、小数点以下の桁数ブランク、定義仕様に VARYING キーワードまたは入力仕様に *VAR 形式。	448	n	VARCHAR (n)、ここで n は長さ
文字、長さは 1 から 254 までの可変	データ・タイプ = A またはブランク、小数点以下の桁数ブランク、定義仕様に VARYING キーワードまたは入力仕様に *VAR 形式。	456	n	VARCHAR (n)、ここで n は長さ
グラフィック	<ul style="list-style-type: none"> TO と FROM の位置を含む、データ・タイプ G バイト長 b のサブフィールドとして定義仕様に定義される。 データ・タイプ G、長さが n として定義仕様に定義される。 データ・タイプ G、バイト長 b として入力仕様に定義される。 	468	m	GRAPHIC(m)、ここで m = n または m = b/2
可変グラフィック	<ul style="list-style-type: none"> TO と FROM の位置を含む、データ・タイプ G、バイト長 b、VARYING キーワードを含むサブフィールドとして定義仕様に定義される。 データ・タイプ G、長さが n、VARYING キーワード付きとして定義仕様に定義される。 データ・タイプ G、バイト長 b、形式 *VAR 付きとして入力仕様に定義される。 	464	m	VARGRAPHIC(m)、ここで m = n または m = (b-2)/2

表9. ILE RPG for iSeries 宣言と代表的 SQL データ・タイプとの対応関係 (続き)

RPG データ・タイプ	RPG コーディング	ホスト変数の SQLTYPE	ホスト変数の SQLLEN	SQL データ・タイプ
UCS-2	<ul style="list-style-type: none"> TO と FROM の位置を含む、データ・タイプ C、バイト長 b のサブフィールドとして定義仕様に定義される。 データ・タイプ C、長さが n として定義仕様に定義される。 データ・タイプ C、バイト長 b として入力仕様に定義される。 	468	m	CCSID 13488 を含む GRAPHIC(m)、ここで m = n または m = b/2
可変 UCS-2	<ul style="list-style-type: none"> TO と FROM の位置を含む、データ・タイプ C、バイト長 b、VARYING キーワードを含むサブフィールドとして定義仕様に定義される。 データ・タイプ C、長さが n、VARYING キーワード付きとして定義仕様に定義される。 データ・タイプ C、バイト長 b、形式 *VAR 付きとして入力仕様に定義される。 	464	m	CCSID 13488 を含む VARGRAPHIC(m)、ここで m = n または m = b/2
日付	<ul style="list-style-type: none"> データ・タイプ D、形式 f、分離文字 s (DATFMT キーワードから取得) として定義仕様に定義される。 データ・タイプ D、pos 31-34 の形式、pos 35 の分離文字として入力仕様に定義される。 	384	n	DATE DATFMT(f) DATSEP(s)

表 9. ILE RPG for iSeries 宣言と代表的 SQL データ・タイプとの対応関係 (続き)

RPG データ・タイプ	RPG コーディング	ホスト変数の SQLTYPE	ホスト変数の SQLLEN	SQL データ・タイプ
時刻	<ul style="list-style-type: none"> データ・タイプ T、形式 f、分離文字 s (TIMFMT キーワードから取得) として定義仕様に定義される。 データ・タイプ T、pos 31-34 の形式、pos 35 の分離文字として入力仕様に定義される。 	388	n	TIME TIMFMT(f) TIMSEP(s)
タイム・スタンプ	データ・タイプ Z	392	n	TIMESTAMP

注:

- SQL は CRTSQLRPGI コマンドで指定した DATE/TIME 形式を使用して日付 / 時刻サブフィールドを作成します。ホスト変数 DATE/TIME 形式への変換は、ホスト変数と SQL が生成したサブフィールド間でマッピングを行う際に行われます。

下表を参照すると、各 SQL データ・タイプに対応する RPG データ・タイプを判別することができます。

表 10. SQL データ・タイプと代表的な RPG 宣言との対応関係

SQL データ・タイプ	RPG データ・タイプ	注
SMALLINT	定義仕様。40 桁目が I、長さ 5、42 桁目が 0。 または 定義仕様。40 桁目が B、長さ ≤ 4、42 桁目が 0。	
INTEGER	定義仕様。40 桁目が I、長さ 10、42 桁目が 0。 または 定義仕様。40 桁目が B、長さ ≤ 9 かつ ≥ 5、42 桁目が 0。	
BIGINT	定義仕様。40 桁目が I、長さ 20、42 桁目が 0。	
DECIMAL	定義仕様。サブフィールドでない場合、40 桁目が P またはブランク、41、42 桁目が 0 なら 30 まで。 または 非定義仕様で数値として定義される。	最大長は 16 (精度 30)、最大位取りは 30。
NUMERIC	定義仕様。サブフィールドの 40 桁目が S またはブランク、41、42 桁目が 0 なら 30 まで。	最大長は 30 (精度 30)、最大位取りは 30。

表 10. SQL データ・タイプと代表的な RPG 宣言との対応関係 (続き)

SQL データ・タイプ	RPG データ・タイプ	注
FLOAT (単精度)	定義仕様。40 桁目が F、長さ 4。	
FLOAT (倍精度)	定義仕様。40 桁目が F、長さ 8。	
CHAR(n)	定義仕様。40 桁目が A またはブランク、41、42 桁目がブランク。 または 小数部の桁なしで定義してある入力フィールド。 または 小数部の桁なしで定義された計算結果フィールド。	n は 1 から 32766 まで可能。
CHAR(n)	データ構造にサブフィールドがないデータ構造名。	n は 1 から 32766 まで可能。
VARCHAR(n)	定義仕様。40 桁目が A またはブランク、44 ~ 80 桁目が VARYING。	n は 1 から 32740 まで可能。
CLOB	サポートなし	SQLTYPE キーワードを使用して CLOB を宣言。
GRAPHIC(n)	定義仕様。40 桁目が G。 または 36 桁目に G を定義してある入力フィールド。	n は 1 から 16383 まで可能。
VARGRAPHIC(n)	定義仕様。40 桁目が G、44 ~ 80 桁目が VARYING。	n は 1 から 16370 まで可能。
DBCLOB	サポートなし	SQLTYPE キーワードを使用して DBCLOB を宣言。
BINARY	サポートなし	SQLTYPE キーワードを使用して BINARY を宣言。
VARBINARY	サポートなし	SQLTYPE キーワードを使用して VARBINARY を宣言。
BLOB	サポートなし	SQLTYPE キーワードを使用して BLOB を宣言。
DATE	文字フィールド または 40 桁目に D を指定した定義仕様。 または 36 桁目に D を定義してある入力フィールド。	形式が *USA、*JIS、*EUR、または *ISO のときは、長さは少なくとも 10 が必要。形式が *YMD、*DMY、または *MDY のときは、長さは少なくとも 8 が必要。形式が *JUL のときは、長さは少なくとも 6 が必要。

表 10. SQL データ・タイプと代表的な RPG 宣言との対応関係 (続き)

SQL データ・タイプ	RPG データ・タイプ	注
TIME	文字フィールド または 40 桁目に T を指定した定義仕様。 または 36 桁目に T を定義してある入力フィールド。	長さは少なくとも 6 が必要。秒を含めるときは、長さは少なくとも 8 が必要。
TIMESTAMP	文字フィールド または 40 桁目に Z を指定した定義仕様。 または 36 桁目に Z を定義してある入力フィールド。	長さは少なくとも 19 が必要。マイクロ秒を含めるときは、長さは少なくとも 26 が必要。長さが 26 未満のときは、マイクロ秒部分で切り捨てが起ります。
DATALINK	サポートなし	
ROWID	サポートなし	SQLTYPE キーワードを使用して ROWID を宣言。

詳細については、『ILE RPG for iSeries 変数宣言と使用法に関する注意事項』を参照してください。

ILE RPG for iSeries 変数宣言と使用法に関する注意事項

SQL を使用する ILE RPG for iSeries アプリケーションでの割り当て規則

ILE RPG for iSeries は、精度と位取りをすべての数値タイプと関連付けます。ILE RPG for iSeries は、データがパック形式であるものとして、数値演算を定義します。すなわち、2 進数の変数が関係する演算は暗黙的にパック形式に変換されてから、演算が行われます (必要ならば、2 進数に逆変換されます)。データは暗黙の小数点位置に合わされて、SQL 演算が行われます。

SQL を使用する ILE RPG for iSeries アプリケーションでの標識変数の使用

標識変数は長さ 5 未満の (2 バイト) の 2 進数フィールドです。

標識配列は、要素の長さが 4,0 の変数として宣言し、定義仕様に DIM を指定することによって定義できます。

検索されるとき、標識変数はその対応するホスト変数にヌル値が割り当てられているかどうかを示すために使用されます。列に割り当てるときには、ヌル値を割り当てるべきであることを示すために負の標識変数が使用されます。

詳細については、「SQL 解説書」の『標識変数』を参照してください。

標識変数の宣言の仕方はホスト変数の場合と同じであり、これらの 2 つの変数の宣言をプログラマーに適切と思われる方法で組み合わせることができます。

ILE RPG での標識変数の使用例については、『例: SQL を使用する ILE RPG for iSeries アプリケーションでの標識変数の使用』を参照してください。

例: SQL を使用する ILE RPG for iSeries アプリケーションでの標識変数の使用

次のステートメントがあるとします。

```
C/EXEC SQL FETCH CLS_CURSOR INTO :CLSCD,  
C+                :DAY :DAYIND,  
C+                :BGN :BGNIND,  
C+                :END :ENDIND  
C/END-EXEC
```

変数は次のように宣言することができます。

```
D CLSCD          S           7  
D DAY            S          2B 0  
D DAYIND         S          2B 0  
D BGN            S           8A  
D BGNIND         S          2B 0  
D END            S           8  
D ENDIND         S          2B 0
```

SQL を使用する ILE RPG for iSeries アプリケーションでの複数行領域 取り出し用 SQLDA の例

```
C/EXEC SQL INCLUDE SQLDA  
C/END-EXEC  
DDEPARTMENT      DS                OCCURS(10)  
D DEPTNO          01           03A  
D DEPTNM          04           32A  
D MGRNO           33           38A  
D ADMRD           39           41A  
...  
  
DIND_ARRAY       DS                OCCURS(10)  
D INDS            4B 0 DIM(4)  
...  
C* setup number of sqlda entries and length of the sqlda  
C                eval          sqld = 4  
C                eval          sqln = 4  
C                eval          sqldabc = 336  
C*  
C* setup the first entry in the sqlda  
C*  
C                eval          sqltype = 453  
C                eval          sqlllen = 3  
C                eval          sql_var(1) = sqlvar  
C*  
C* setup the second entry in the sqlda  
C*  
C                eval          sqltype = 453  
C                eval          sqlllen = 29  
C                eval          sql_var(2) = sqlvar  
...  
C*  
C* setup the forth entry in the sqlda  
C*  
C                eval          sqltype = 453
```

```

C          eval      sqllen = 3
C          eval      sql_var(4) = sqlvar

...
C/EXEC SQL
C+ DECLARE C1 FOR
C+   SELECT *
C+   FROM CORPDATA.DEPARTMENT
C/END-EXEC

...

C/EXEC SQL
C+   FETCH C1 FOR 10 ROWS
C+   USING DESCRIPTOR :SQLDA
C+   INTO :DEPARTMENT:IND_ARRAY
C/END-EXEC

```

SQL を使用する ILE RPG for iSeries アプリケーションでの動的 SQL の例

```

D*****
D* Declare program variables.                *
D* STMT initialized to the                   *
D* listed SQL statement.                     *
D*****
D EMPNUM      S          6A
D NAME        S          15A
D STMT        S          500A  INZ('SELECT LASTNAME      -
D                                     FROM CORPDATA.EMPLOYEE WHERE -
D                                     EMPNO = ?')

...

C*****
C* Prepare STMT as initialized in declare section *
C*****
C/EXEC SQL
C+ PREPARE S1 FROM :STMT
C/END-EXEC
C*
C*****
C* Declare Cursor for STMT                    *
C*****
C/EXEC SQL
C+ DECLARE C1 CURSOR FOR S1
C/END-EXEC
C*
C*****
C* Assign employee number to use in select statement *
C*****
C          eval      EMPNUM = '000110'

C*****
C* Open Cursor                                *
C*****
C/EXEC SQL
C+ OPEN C1 USING :EMPNUM
C/END-EXEC
C*
C*****
C* Fetch record and put value of              *
C* LASTNAME into NAME                        *
C*****
C/EXEC SQL
C+   FETCH C1 INTO :NAME

```

C/END-EXEC

...

C*****

C* Program processes NAME here *

C*****

...

C*****

C* Close cursor *

C*****

C/EXEC SQL

C+ CLOSE C1

C/END-EXEC

第 10 章 REXX アプリケーションでの SQL ステートメントのコーディング方法


REXX プロシージャは、プリプロセスを行う必要はありません。実行時に REXX インタープリターは、理解できないステートメントを現行活動コマンド環境の処理のために渡します。このコマンド環境を *EXECSQL に変更し、全未知ステートメントをデータベース・マネージャーに渡すことができますが、その方法は 2 つあります。

1. STRREXPRC CL コマンドの CMDENV パラメーター
2. ADDRESS REXX コマンドのアドレス定位置パラメーター

詳細については、以下のセクションを参照してください。

- 『REXX アプリケーションでの SQL 連絡域の定義』
- 146 ページの『REXX アプリケーションでの SQL 記述子域の使用』
- 147 ページの『REXX アプリケーションでの SQL ステートメントの組み込み』
- 150 ページの『SQL を使用する REXX アプリケーションでのホスト変数の使用』
- 153 ページの『SQL を使用する REXX アプリケーションでの標識変数の使用』

STRREXPRC CL コマンドまたは ADDRESS REXX コマンドの詳細については、「REXX/400

Programmer's Guide 」および「REXX/400 Reference 」を参照してください。

SQL ステートメントの使い方を示した詳しいサンプル REXX プログラムは、203 ページの『例: REXX プログラム内の SQL ステートメント』に記載されています。

注: 重要な法律的事柄に関する情報について、214 ページの『コードについての特記事項』をご覧ください。

REXX アプリケーションでの SQL 連絡域の定義

SQL 連絡域 (SQLCA) を構成するフィールドは、SQL/REXX インターフェースにより自動的に組み込まれます。INCLUDE SQLCA ステートメントは不要でしかも使用できません。SQLCA の SQLCODE フィールドおよび SQLSTATE フィールドには、SQL 戻りコードが入っています。これらの値は、各 SQL ステートメントが実行された後、データベース・マネージャーによって設定されます。アプリケーションは、SQLCODE 値または SQLSTATE 値を調べて、最後の SQL ステートメントが正しく実行されたかどうかを判定することができます。

SQL/REXX インターフェースは、SQLCA を典型的な SQL の使用法に従って使用します。ただし、SQL/REXX インターフェースは SQLCA のフィールドを連続したデータ域としてではなく個別の変数として維持します。SQL/REXX インターフェースが SQLCA 用に維持する変数の定義は次のとおりです。

SQLCODE	1 次 SQL 戻りコード。
SQLERRMC	エラーおよび警告メッセージ・トークン。
SQLERRP	製品コード。エラーがあった場合は、エラーを返したモジュールの名前。
SQLERRD,<i>n</i>	診断情報が入っている 6 個の変数 (<i>n</i> は、1 から 6 までの数字)。

SQLWARN.n 警告フラグが入っている 11 個の変数 (*n* は、0 から 10 までの数字)。
SQLSTATE 代替 SQL 戻りコード。

SQLCA の詳細については、「SQL 解説書」の『SQL 連絡域』を参照してください。

REXX アプリケーションでの SQL 記述子域の使用

SQLDA を必要とするステートメントには、次のものがあります。

EXECUTE...USING DESCRIPTOR 記述子名
FETCH...USING DESCRIPTOR 記述子名
OPEN...USING DESCRIPTOR 記述子名
CALL...USING DESCRIPTOR 記述子名
DESCRIBE ステートメント名 INTO 記述子名
DESCRIBE TABLE ホスト変数 INTO 記述子名

SQLCA とは違い、プロシージャに複数の SQLDA を入れることができ、しかも有効な名前であれば SQLDA にどのような名前を付けても構いません。各 SQLDA は、共通ステムを持つ 1 組の REXX 変数からなります。ステムの名前は、該当する SQL ステートメントからの記述子名 です。これは単純なステムでなければなりません。すなわち、ステム自体にピリオドが含まれていてはなりません。SQL/REXX インターフェースは、各固有の記述子名について SQLDA のフィールドを自動的に用意します。INCLUDE SQLDA ステートメントは不要でしかも使用できません。

SQL/REXX インターフェースは、SQLDA を典型的な SQL の使用法に従って使用します。ただし、SQL/REXX インターフェースは SQLDA のフィールドを連続したデータ域としてではなく個別の変数として維持します。

SQLDA の詳細については、「SQL 解説書」の『SQL 記述子域』を参照してください。

次の変数は、DESCRIBE、DESCRIBE TABLE、または PREPARE INTO の各ステートメントの後にアプリケーションに返されます。

stem.n.SQLNAME

結果テーブルの中の *n* 番目の列の名前。

次の変数は、EXECUTE...USING DESCRIPTOR、OPEN...USING DESCRIPTOR、CALL...USING DESCRIPTOR、または FETCH...USING DESCRIPTOR の各ステートメントの前にアプリケーションにより指定される必要があります。これらの変数は、DESCRIBE、DESCRIBE TABLE、または PREPARE INTO ステートメントの後にアプリケーションに返されます。

stem.SQLD

SQLDA に実際に入っている変数要素の数。

stem.n.SQLTYPE

n 番目の要素のデータ・タイプを表す整数 (たとえば、最初の要素は stem.1.SQLTYPE にあります)。

次のデータ・タイプは使用できません。

400/401 NUL 終了漢字ストリング
404/405 BLOB ホスト変数
408/409 CLOB ホスト変数

412/413	DBCLOB ホスト変数
460/461	NUL 終了文字ストリング
476/477	PASCAL L 文字列
496/497	長精度整数 (位取りが 0 より大きい)
500/501	短精度整数 (位取りが 0 より大きい)
504/505	DISPLAY SIGN LEADING SEPARATE
904/905	ROWID
908/909	VARBINARY ホスト変数
912/913	BINARY ホスト変数
916/917	BLOB ファイル参照変数
920/921	CLOB ファイル参照変数
924/925	DBCLOB ファイル参照変数
960/961	BLOB ロケータ
964/965	CLOB ロケータ
968/969	DBCLOB ロケータ

stem.n.SQLLEN

SQLTYPE が DECIMAL または NUMERIC データ・タイプを指示していない場合、データの最大長は stem.n.SQLDATA に入っています。

stem.n.SQLLEN.SQLPRECISION

データ・タイプが DECIMAL または NUMERIC の場合、これには数の精度が入ります。

stem.n.SQLLEN.SQLSCALE

データ・タイプが DECIMAL または NUMERIC の場合、これには数の位取りが入ります。

stem.n.SQLCCSID

データの n 列目の CCSID。

次の変数は、EXECUTE...USING DESCRIPTOR または OPEN...USING DESCRIPTOR の各ステートメントの前にアプリケーションによって必ず提供される必要があります。これらの FETCH...USING DESCRIPTOR ステートメントの後でアプリケーションに返されます。これらの変数は、DESCRIBE、DESCRIBE TABLE、または PREPARE INTO の各ステートメントの後では使用されません。

stem.n.SQLDATA

これには、アプリケーションから提供された入力値、または SQL が取り出した出力値が含まれます。

この値は、SQLTYPE、SQLLEN、SQLPRECISION、および SQLSCALE で指定した属性に変換されます。

stem.n.SQLIND

入力値または出力値が NULL の場合、この値は負の数になります。

REXX アプリケーションでの SQL ステートメントの組み込み

SQL ステートメントは REXX コマンドに入れられるところならどこにでも入れることができます。

REXX プロシーチャー内の各 SQL ステートメントは、必ず EXECSQL で始まり (大文字と小文字はどのように組み合わせても構いません)、次に以下のいずれかが続かなければなりません

- 一重引用符または二重引用符で囲まれている SQL ステートメント。または
- ステートメントが入っている REXX 変数。REXX 変数に SQL ステートメントが入っている場合は、REXX 変数の前にコロンを入れてはなりません。

以下に、例を示します。

```
EXECSQL "COMMIT"
```

上記の例は次に相当します。

```
rexxvar = "COMMIT"  
EXECSQL rexxvar
```

このコマンドは通常の REXX 規則に従います。たとえば、任意選択でコマンドの後にセミコロン (;) を入れて、1 行に複数の REXX ステートメントを入れることができます。また、REXX を使用すると、一重引用符の中にコマンド名を入れることができます。たとえば、次のとおりです。

```
'EXECSQL COMMIT'
```

SQL/REXX インターフェースは、次の SQL ステートメントをサポートします。

ALTER SEQUENCE	EXECUTE IMMEDIATE
ALTER TABLE	FETCH ²
CALL ³	GRANT
CLOSE	INSERT ^{2, 3}
COMMENT ON	LABEL ON
COMMIT	LOCK TABLE
CREATE ALIAS	OPEN
CREATE DISTINCT TYPE	PREPARE
CREATE FUNCTION	REFRESH
CREATE INDEX	RELEASE SAVEPOINT
CREATE PROCEDURE	RENAME
CREATE SCHEMA	REVOKE
CREATE SEQUENCE	ROLLBACK
CREATE TABLE	SAVEPOINT
CREATE TRIGGER	SET ENCRYPTION PASSWORD
CREATE VIEW	SET OPTION ⁴
DECLARE CURSOR ³	SET PATH
DECLARE GLOBAL TEMPORARY TABLE	SET SCHEMA
DELETE ³	SET TRANSACTION
DESCRIBE	SET variable (変数) ³
DESCRIBE TABLE	UPDATE ³
DROP	VALUES INTO ³
EXECUTE	

次の SQL ステートメントは、SQL/REXX インターフェースではサポートされません。

2. このステートメントのブロック化形式はサポートされていません。

3. これらのステートメントがホスト変数を含む場合には、直接実行することはできません。これらのステートメントは、PREPARE のオブジェクト、次に EXECUTE のオブジェクトでなければなりません。

BEGIN DECLARE SECTION	GET DIAGNOSTICS
CONNECT	HOLD LOCATOR
DECLARE PROCEDURE	INCLUDE
DECLARE STATEMENT	RELEASE
DECLARE VARIABLE	SELECT INTO
DISCONNECT	SET CONNECTION
END DECLARE SECTION	SET RESULT SETS
FREE LOCATOR	SIGNAL
	WHENEVER ⁵

詳細については、以下のセクションを参照してください。

- 『SQL を使用する REXX アプリケーションでの注記』
- 『SQL を使用する REXX アプリケーションでの SQL ステートメントの継続』
- 『SQL を使用する REXX アプリケーションでのコードの組み込み』
- 『SQL を使用する REXX アプリケーションでのマージン』
- 『SQL を使用する REXX アプリケーションの名前』
- 150 ページの『SQL を使用する REXX アプリケーションでのヌル』
- 150 ページの『SQL を使用する REXX アプリケーションでのステートメント・ラベル』
- 150 ページの『SQL を使用する REXX アプリケーションでのエラーおよび警告の処理』

SQL を使用する REXX アプリケーションでの注記

SQL の注記 (--) も REXX の注記も SQL ステートメントを表している文字列に入れることはできません。

SQL を使用する REXX アプリケーションでの SQL ステートメントの継続

SQL ステートメントの入っている文字列は、標準 REXX の使用方法に従ってコンマや連結記号で分離することによって、複数行の複数の文字列に分割することができます。

SQL を使用する REXX アプリケーションでのコードの組み込み

他のホスト言語と違い、外部で定義されたステートメントの組み込みについてのサポートはありません。

SQL を使用する REXX アプリケーションでのマージン

SQL/REXX インターフェースの場合、特殊なマージンの規則はありません。

SQL を使用する REXX アプリケーションの名前

ホスト変数には、ピリオド (.) で終わらない任意の有効な REXX 名を使用できます。名前は必ず 64 文字以下でなければなりません。

変数名は、'SQL'、'RDI'、'DSN'、'RXSQL'、または 'QRW' の文字で始まってはなりません。

4. SET OPTION ステートメントを REXX プロシージャで使用することにより、SQL ステートメント実行用の処理オプションの一部を変更できます。これらのオプションにはコミットメント制御レベルや日付形式が含まれます。SET OPTION ステートメントの詳細については、「SQL 解説書」を参照してください。

5. 詳しくは、150 ページの『SQL を使用する REXX アプリケーションでのエラーおよび警告の処理』を参照してください。

SQL を使用する REXX アプリケーションでのヌル

ヌル という用語は、REXX でも SQL でも使用していますが、この用語はこれら 2 つの言語では異なる意味を持っています。REXX には、ヌル・ストリング (ゼロ長の文字列) とヌル文節 (ブランクと注記のみから成る文節) があります。SQL のヌル値は、すべての非ヌル値とは異なる特殊な値で、(ヌルでない) 値がないことを表しています。

SQL を使用する REXX アプリケーションでのステートメント・ラベル

REXX コマンド・ステートメントは、通常どおりにラベル付けすることができます。

SQL を使用する REXX アプリケーションでのエラーおよび警告の処理

WHENEVER ステートメントは、SQL/REXX インターフェースではサポートしていません。その代わりに、次のどれを使用しても構いません。

- データベース・マネージャーが出したエラーおよび警告条件 (ただし、SQL/REXX インターフェースが出したものでない) を検出するための各 SQL ステートメントの後の REXX SQLCODE 変数または SQLSTATE 変数のテスト。
- エラーおよび警告条件を検出するための各 SQL ステートメントの後の REXX RC 変数のテスト。EXECSQL コマンドを使用するたびに RC 変数は次のようにセットされます。

- 0** ステートメントが正常に完了。
- +10** SQL 警告が発生。
- 10** SQL エラーが発生。
- 100** SQL/REXX インターフェース・エラーが発生。

これを使用すると、データベース・マネージャーまたは SQL/REXX インターフェースのいずれかによって出されたエラーおよび警告を検出することができます。

- SIGNAL ON ERROR 機能および SIGNAL ON FAILURE 機能を使用すると、エラー (負の RC 値) を検出することができますが、警告を検出することはできません。

SQL を使用する REXX アプリケーションでのホスト変数の使用

REXX には変数宣言がありません。LOB ホスト変数、ROWID ホスト変数、およびバイナリー・ホスト変数は、REXX ではサポートされません。新規の変数は、それが割り当てステートメントに現れることによって認識されます。したがって、宣言部分はなく、BEGIN DECLARE SECTION ステートメントおよび END DECLARE SECTION ステートメントをサポートしていません。

SQL ステートメントの中のホスト変数はいずれも、その前にコロン (:) を付けなければなりません。

SQL/REXX インターフェースは、ステートメントをデータベース・マネージャーに渡す前に複合変数で置換を行います。以下に、例を示します。

```
a = 1
b = 2
EXECSQL 'OPEN c1 USING :x.a.b'
```

上記により、x.1.2 の内容が SQL に渡されます。

詳細については、以下のセクションを参照してください。

- 151 ページの『SQL を使用する REXX アプリケーションでの入力ホスト変数のデータ・タイプの判別』

- 152 ページの『SQL を使用する REXX アプリケーションでの出力ホスト変数のフォーマット』
- 153 ページの『SQL を使用する REXX アプリケーションでの REXX 変換の回避』

SQL を使用する REXX アプリケーションでの入力ホスト変数のデータ・タイプの判別

REXX におけるすべてのデータは、文字列形式になっています。入力ホスト変数のデータ・タイプ (すなわち、EXECUTE ステートメントまたは OPEN ステートメント内の 'USING ホスト変数' 文節で使用しているホスト変数) は、表 11 に従って、実行時にデータ・タイプ管理プログラムによって変数の内容から推論されます。

これらの規則は、数値、文字値、またはグラフィック値のいずれかを定義します。数値は、任意のタイプの数値列への入力として使用することができます。文字値は、任意のタイプの文字列への入力、または日付、時刻、またはタイム・スタンプの各列への入力として使用することができます。グラフィック値は、任意のタイプのグラフィック列への入力として使用することができます。

表 11. REXX でのホスト変数のデータ・タイプの判別

ホスト変数の内容	想定データ・タイプ	SQL タイプ・コード	SQL タイプ記述
小数点も指数も入っていない数字。先頭に正符号または負符号が付いても構わない。	符号付き整数	496/497	INTEGER
小数点を含む数字。ただし、指数は含まれない。	パック 10 進数	484/485	DECIMAL(m,n)
または、小数点または指数を含まない数で、2147483647 より大きいか、-2147483647 より小さい数。			
先頭に正符号または負符号が付いても構わない。 m は、数の総桁数。 n は、小数点の左にある桁数 (存在する場合)。			
科学または技術的表記での数字 (すなわち、直後に 'E' または 'e'、任意選択で正符号または負符号、および一連の数字が続く)。先頭に正符号または負符号が付いても構わない。	浮動小数点	480/481	DOUBLE PRECISION
先行と後書きアポストロフィ (') または引用符 (") をもつ文字列で、2 つの区切り文字を除く長さが n である。	可変長文字ストリング	448/449	VARCHAR(n)
または、先行 X または先行 x の後にアポストロフィ (') または引用符 (")、および後書きアポストロフィ (') または引用符 (") をもつ文字列。この文字列の長さは、X または x と 2 つの区切り文字を取り除くと $2n$ になる。残りの文字の各対は、単一文字の 16 進表示。			
または、このテーブルの他の規則により文字、数値またはグラフィックとして認識できない長さ n の文字列。			

表 11. REXX でのホスト変数のデータ・タイプの判別 (続き)

ホスト変数の内容	想定データ・タイプ	SQL タイプ・	
		コード	SQL タイプ記述
前にコロンの付いている先行および後書きアポストロフィ (') または引用符 (") をもつ文字列。 ⁶	可変長漢字ストリング	464/465	VARGRAPHIC(n)
<ul style="list-style-type: none"> • G、g、N または n で始まる文字列。その後にはアポストロフィまたは引用符とシフトアウト (x'OE') が続く。その後には、n 個のグラフィック文字が続く。それぞれの長さは 2 文字。この文字列は、必ずシフト・イン (X'OF') とアポストロフィまたは引用符 (文字列の最初にあるいずれか一方) で終わること。 • 文字列の先行は GX、Gx、gX、または gx とし、次にアポストロフィまたは引用符とシフトアウト (x'OE') が続く。その後には、n 個のグラフィック文字が続く。それぞれの長さは 2 文字。この文字列は、必ずシフト・イン (X'OF') とアポストロフィまたは引用符 (文字列の最初にあるいずれか一方) で終わること。GX または 区切り文字を取り除くと、この文字列の長さは 4n になる。残りの 4 文字の各グループは、単一グラフィック文字の 16 進表示となる。 			
未定義の変数	値を割り当てていない変数	なし	無効なデータを検出した。

SQL を使用する REXX アプリケーションでの出力ホスト変数のフォーマット

出力ホスト変数 (すなわち、FETCH ステートメントの 'INTO ホスト変数' 文節で使用されるホスト変数) のデータ・タイプを判別する必要はありません。出力値は、次のようにホスト変数に割り当てられます。

- 文字値は、前後のアポストロフィなしで割り当てられます。
- グラフィック値は、先行 G またはアポストロフィ、後書きアポストロフィ、シフトアウトとシフトインの文字のいずれも伴うことなく割り当てられます。
- 数値は、文字列に変換されます。
- 整数値は先行ゼロを保持しません。負の値には、先頭に負符号が付いています。
- 10 進数値は、それらの精度と位取りに従って先行ゼロと後続ゼロを持っています。負の値には、先頭に負符号が付いています。正の値には、先頭に正符号は付いていません。
- 浮動小数点値は浮動小数になっていて、小数点の左に 1 桁入っています。'E' は、大文字になっています。

6. 先行アポストロフィの直後のバイトは X'OE' シフトアウトで、後書きアポストロフィの直前のバイトは X'OF' シフトインです。

SQL を使用する REXX アプリケーションでの REXX 変換の回避

文字列を確実に数に変換しないようにするか、またはグラフィック・タイプと見なす場合は、文字列を "" の引用符で囲む必要があります。ただアポストロフィで囲んでも認識されません。以下に、例を示します。

```
stringvar = '100'
```

上記のように指定すると、REXX は、変数 *stringvar* を文字の 100 (アポストロフィなし) の文字列にセットします。これは、SQL/REXX インターフェースで数字 100 として評価され、そのまま SQL に渡されます。

一方、次の例を見てください。

```
stringvar = ""100""
```

上記のように指定すると、REXX は、変数 *stringvar* を文字 '100' (アポストロフィを含む) の文字列にセットします。これは、SQL/REXX インターフェースにより文字列 100 として評価され、そのまま SQL に渡されます。

SQL を使用する REXX アプリケーションでの標識変数の使用

標識変数は整数です。検索時に、標識変数が使用され、その関連ホスト変数がヌル値に割り当てられたかどうかを示します。列に割り当てるときには、ヌル値を割り当てべきであることを示すために負の標識変数が使用されます。

その他の言語と違い、たとえその関連標識変数に負の値が入っていても、必ずホスト変数に有効な値を指定する必要があります。

詳細については、「SQL 解説書」の『標識変数』を参照してください。

第 11 章 SQL ステートメントを含むプログラムの準備と実行

ここでは、アプリケーション・プログラムの準備と実行に必要な作業をいくつか説明します。詳細については、以下のセクションを参照してください。

- 『SQL プリコンパイラーの基本処理』
- 163 ページの『非 ILE SQL プリコンパイラー・コマンド』
- 164 ページの『ILE SQL プリコンパイラー・コマンド』
- 166 ページの『SQL を使用するアプリケーションでのコンパイル・エラーの解釈』
- 167 ページの『SQL を使用するアプリケーションのバインド』
- 168 ページの『SQL プリコンパイラー・オプションの表示』
- 168 ページの『組み込み SQL を使用したプログラムの実行』

注: 重要な法律的事柄に関する情報について、214 ページの『コードについての特記事項』をご覧ください。

SQL プリコンパイラーの基本処理

SQL ステートメントが組み込まれているアプリケーション・プログラムを実行させるには、その前にプログラムをプリコンパイルし、コンパイルしなければなりません。

注: REXX プロシージャ内の SQL ステートメントは、プリコンパイルもコンパイルも行われません。このようなプログラムのプリコンパイルは SQL プリコンパイラーによって行われます。SQL プリコンパイラーは、アプリケーション・プログラムのソース仕様の各ステートメントを走査して、次のことを行います。

- **SQL ステートメントおよびホスト変数名の定義を見つける。**変数名とその定義は、SQL ステートメントの妥当性を検査するために使用されます。ユーザーは、SQL プリコンパイラーが処理を完了した後、リストを調べて、エラーの有無を確かめることができます。
- **各ステートメントが有効で構文エラーがないことを確かめる。**この妥当性検査のプロシージャは、出力リストにエラー・メッセージを出すので、エラーがあればそれを訂正するのに役立ちます。
- **データベースの記述に基づいて SQL ステートメントの妥当性を検査する。**プリコンパイルの過程で、テーブル、ビュー、および列名が有効であることを確かめるために SQL ステートメントが検査されます。指定したテーブルまたはビューが存在しない場合や、プリコンパイルまたはコンパイル時にテーブルまたはビューを使用する権限がない場合は、妥当性検査は実行時に行われます。実行時にテーブルやビューが存在しない時は、エラーが起きます。

注:

1. 一時変更の処理は外部定義の検査時に行われます。詳細については、「データベース・プログラミング」および「ファイル管理」を参照してください。
2. SQL ステートメントが有効であることを確かめるためには、SQL ステートメントで参照されているテーブルやビューに対する何らかの権限 (少なくとも *OBJOPR) が必要です。SQL ステートメントの処理に必要な実際の権限は、実行時に検査されます。SQL ステートメントの詳細については、「SQL 解説書」を参照してください。
3. CRTSQLxxx コマンドで RDB パラメーターを指定すると、プリコンパイラーは、指定したリレーショナル・データベースにアクセスして、テーブル記述とビュー記述を入手します。

- ホスト言語でのコンパイルに備えて各 SQL ステートメントの準備を行う。ほとんどの SQL ステートメントの場合、SQL プリコンパイラーは、注記と CALL ステートメントを SQL インターフェース・モジュールの 1 つに挿入します。一部の SQL ステートメント (たとえば、DECLARE ステートメント) では、SQL プリコンパイラーは注記だけを生成し、ホスト言語ステートメントを生成しません。
- プリコンパイルされた SQL ステートメントに関する情報を生成する。この情報は一時ソース・ファイル・メンバーに内部的に保管され、バインド処理の際に使用されます。

プリコンパイルするときに完全な診断情報を入手するには、次のいずれかを指定してください。

- CRTSQLxxx の場合 (ただし、xxx=CBL、PLI、または RPG)、OPTION(*SOURCE *XREF)
- CRTSQLxxx の場合 (ただし、xxx=CI、CPPI、CBLI、または RPGI) OPTION(*XREF) OUTPUT(*PRINT)

詳細については、以下のセクションを参照してください。

- 『SQL プリコンパイラーへの入力』
- 157 ページの 『SQL プリコンパイラーのソース・ファイル CCSID』
- 157 ページの 『SQL プリコンパイラーの出力』

SQL プリコンパイラーへの入力

SQL プリコンパイラーへの主要な入力は、アプリケーション・プログラミング・ステートメントと組み込み SQL ステートメントです。PL/I、C、および C++ プログラムでは、SQL ステートメントは、CRTSQLPLI、CRTSQLCI、および CRTSQLCPPI コマンドの MARGINS パラメーターに指定されたマージンを使用しなければなりません。

SQL プリコンパイラーは、ホスト言語ステートメントが構文的に正しいと想定します。ホスト言語ステートメントが構文的に正しくない、プリコンパイラーは SQL ステートメントとホスト変数宣言を正しく識別できないことがあります。プリコンパイラーを経由して渡すことができるソース・ステートメントの形式には制限があります。アプリケーション言語コンパイラーに受け付けられないリテラルと注記は、プリコンパイラーによるソース・ステートメントの走査処理を阻害し、エラーの原因となるおそれがあります。

SQL INCLUDE ステートメントを使用すると、CRTSQLxxx⁷ の INCFILE パラメーターで指定されたファイルから 2 次入力を得ることができます。SQL INCLUDE ステートメントは、指定したメンバーの終わりに達するまでそのメンバーから入力の読み取りを行います。このようにして組み込まれるメンバーには、プリコンパイラーの他の INCLUDE ステートメントが含まれてはなりません。アプリケーション・プログラム・ステートメントと SQL ステートメントの両方を含むことができます。

アプリケーション・プログラムのソース・ステートメントで混合 DBCS 定数が指定されている場合には、ソース・ファイルは混合 CCSID でなければなりません。

SQL SET OPTION ステートメントを使用することにより、入力ソース・メンバーに多くのプリコンパイラー・オプションを指定することができます。SET OPTION 構文については、「SQL 解説書」を参照してください。

- 1 CRTSQLRPGI コマンドの RPG プリプロセッサ・オプション・パラメーター (RPGPPORT) には、RPG
- 1 プリプロセッサを呼び出す 2 つのオプションがあります。*LVL1 または *LVL2 のいずれかを指定す
- 1 ると、SQL プリコンパイルの実行前に RPG コンパイラーが呼び出されて、ソース・メンバーをプリプロ

7. このコマンドの xxx はホスト言語標識を表しています。COBOL for iSeries 言語の場合は CBL、ILE COBOL for iSeries 言語の場合は CBLI、iSeries PL/I 言語の場合は PLI、ILE C for iSeries 言語の場合は CI、RPG for iSeries 言語の場合は RPG、ILE RPG for iSeries 言語の場合は RPGI、ILE C++/400 言語の場合は CPPI です。

1 | セスします。SQL ソース・メンバーのプリプロセスによって、SQL プリコンパイルの前に多数のコンパ
1 | イラー指示が処理されます。プリプロセスされたソースは、QTEMP 内のファイル QSQLPRE に格納され
1 | ます。このソースは、SQL プリコンパイル時の入力として使用されます。SQL プリコンパイルで使用さ
1 | れる CCSID は、QSQLPRE の CCSID です。

SQL プリコンパイラーのソース・ファイル CCSID

SQL プリコンパイラーは、ソース・ファイルの CCSID を使用してソース・レコードを読み取ります。SQL INCLUDE ステートメントを処理するとき、組み込まれるソース・ファイルは、必要ならば、元のソース・ファイルの CCSID に変換されます。組み込まれるソース・ファイルが元のソース・ファイルの CCSID に変換できないときは、エラーが起こります。

SQL プリコンパイラーは、ソース・ファイルの CCSID を使用して SQL ステートメントを処理します。この影響を最も受けるのは、変換文字です。たとえば、NOT 記号 (¬) は CCSID 500 では 'BA'X に置かれています。これは、ソース・ファイルの CCSID が 500 である場合に、SQL が NOT 記号 (¬) が 'BA'X に置かれるよう要求したことを意味します。

ソース・ファイルの CCSID が 65535 の場合、SQL は、CCSID が 37 であるものとして可変文字を処理します。すなわち、SQL は NOT 記号 (¬) が '5F'X にあるものとして探します。

SQL プリコンパイラーの出力

以下のセクションでは、プリコンパイラーから得られる種々の出力について説明します。

- 『リスト』
- 158 ページの『SQL プリコンパイラーにより作成される一時ソース・ファイル・メンバー』
- 158 ページの『SQL プリコンパイラー出力のサンプル』

リスト

出力リストは、CRTSQLxxx コマンドの PRTFILE パラメーターで指定した印刷ファイルに送られます。印刷ファイルに書き込まれる項目には、次のものがあります。

- プリコンパイラー・オプション

CRTSQLxxx コマンドで指定されているオプション。

- プリコンパイラー・ソース・ステートメント

リスト・オプションが有効な場合、この出力は、プリコンパイラー・ソース・ステートメントと、プリコンパイラーによって割り当てられたレコード番号を提供します。

- プリコンパイラー相互参照

OPTION パラメーターに *XREF を指定すると、この出力は相互参照リストを提供します。このリストは、参照されるホスト名と列名が入っている SQL ステートメントのプリコンパイラー・レコード番号を示します。

- プリコンパイラー診断情報

この出力からは、エラーのあるステートメントのプリコンパイラー・レコード番号を示す診断メッセージが得られます。

印刷ファイルに送られる出力には、CCSID 値の 65535 が使用されます。印刷ファイルに書き込まれるとき、データは変換されません。

SQL プリコンパイラーにより作成される一時ソース・ファイル・メンバー

プリコンパイラーによって処理されたソース・ステートメントは、出力ソース・ファイルに書き出されま
す。プリコンパイラーによって変更されたソース・コードでは、SQL ステートメントは、注記と SQL 実
行時への呼び出しに変換されています。SQL によって処理される組み込みは拡張されます。

出力ソース・ファイルは、CRTSQLxxx コマンドの TOSRCFILE パラメーターで指定されます。C および
C++ 以外の言語について、省略時ファイルは、QTEMP ライブラリーの QSQLTEMP (ILE RPG for iSeries
の場合は QSQLTEMP1) です。*CALC が出力ソース・ファイルとして指定されるときは C および C++
については、ソース・ファイルのレコード長が 92 以下である場合に QSQLTEMP が使用されます。レコ
ード長が 92 より大きい C または C++ ソース・ファイルについては、出力ソース・ファイル名が
QSQLTxxxxx として生成されます。ここで xxxxx はレコード長です。出力ソース・ファイル・メンバーの
名前は、CRTSQLxxx コマンドの PGM パラメーターまたは OBJ パラメーターに指定した名前と同じにな
ります。このメンバーは、コンパイラーへの入力として使用する前に変更することはできません。SQL
が出力ソース・ファイルを作成するとき、ソース・ファイルの CCSID 値を新しいファイルの CCSID 値と
して使用します。

プリコンパイルで QTEMP のソース・ファイルに出力を生成する場合で、あとでコンパイルを行いたい場
合は、プリコンパイル後にそのファイルを永続ライブラリーに移しておくことができます。ソース・メンバ
ーのレコードを変更することはできません。変更すると、コンパイルは失敗します。

プリコンパイルの結果として SQL によって生成されるソース・メンバーは、別のプリコンパイル・ステッ
プへの入力メンバーとして編集および再利用してはなりません。最初のプリコンパイル中にソース・メンバ
ーとともに保管される追加の SQL 情報によって、2 番目のプリコンパイルが正しく行われなくなります。
この情報は一度ソース・メンバーに付加されると、そのメンバーが削除されるまでこのメンバーとともに存
在し続けます。

SQL プリコンパイラーは、CRTSRCPF コマンドを使用して、出力ソース・ファイルを作成します。このコ
マンドの省略時値が変更されていると、結果が予測できないものになる可能性があります。SQL プリコン
パイラーではなくユーザーがソース・ファイルを作成した場合、ファイル属性も異なる場合があります。ユ
ーザーが SQL に出力ソース・ファイルを作成させるようお勧めします。このファイルは、いったん SQL
によって作成されたら、後のプリコンパイルで再利用することができます。

COBOL プリコンパイラーの出力例については、『SQL プリコンパイラー出力のサンプル』を参照してく
ださい。

SQL プリコンパイラー出力のサンプル

プリコンパイラーの出力は、プログラムのソース仕様に関する情報を提供することができます。リストを作
成するには、次のように行ってください。

- 非 ILE プリコンパイラーの場合は、*SOURCE(*SRC) および *XREF の各オプションを CRTSQLxxx
コマンドの OPTION パラメーターに指定してください。
- ILE プリコンパイラーの場合は、OPTION(*XREF) および OUTPUT(*PRINT) を CRTSQLxxx コマンド
に指定してください。

プリコンパイラーの出力の形式は次のとおりです。

```

5722ST1 V5R3M0 040528          SQL COBOL プログラムの作成          CBLTEST1          03/11/27 01:34:28          ページ 1
ソース仕様タイプ . . . . .COBOL
プログラム名 . . . . .CORPDATA/CBLTEST1
ソース・ファイル . . . . .CORPDATA/SRC
メンバー . . . . .CBLTEST1
TO ソース・ファイル . . . . .QTEMP/QSQLTEMP
1 オプション . . . . .*SRC          *XREF          *SQL
ターゲット・リリース . . . V5R3M0
INCLUDE ファイル . . . . .*SRCFILE
コミット . . . . .*CHG
データのコピー可能 . . . *YES
SQL カーソルのクローズ . . *ENDPGM
ブロック化可能 . . . . .*READ
PREPARE 遅延 . . . . .*NO
生成レベル . . . . .10
印刷装置ファイル . . . . .*LIBL/QSYSPT
日付の形式 . . . . .*JOB
日付区切り記号 . . . . .*JOB
時刻の形式 . . . . .*HMS
時刻区切り記号 . . . . .*JOB
置き換え . . . . .*YES
リレーショナル・データベース*LOCAL
ユーザー . . . . .*CURRENT
RDB 接続方式 . . . . .*DUW
省略時のコレクション . . . *NONE
動的省略時の
  コレクション . . . . .*NO
パッケージ名 . . . . .*PGMLIB/*PGM
パス . . . . .*NAMING
SQL 規則 . . . . .*DB2
ユーザー・プロファイル . . *NAMING
動的ユーザー・プロファイル*USER
ソート順序 . . . . .*JOB
言語 ID . . . . .*JOB
IBM SQL フラグづけ . . . . .*NOFLAG
ANS フラグ付け . . . . .*NONE
テキスト . . . . .*SRCMBRTXT
ソース・ファイルの CCSID . . 65535
ジョブの CCSID . . . . .65535
10 進数結果オプション
  最大精度 . . . . .31
  最大位取り . . . . .31
  除算の最小位取り . . . . .0
コンパイラー・オプション . *NONE
2 03/11/27 01:32:58 にソース・メンバーが変更された。

```

- 1 SQL プリコンパイラーの呼び出し時に指定したオプションのリスト。
- 2 ソース・メンバーが最後に変更された日付。

図2. サンプル COBOL プリコンパイラー出力の形式 (1/4)

1	IDENTIFICATION DIVISION.	100
2	PROGRAM-ID. CBLTEST1.	200
3	ENVIRONMENT DIVISION.	300
4	CONFIGURATION SECTION.	400
5	SOURCE-COMPUTER. IBM-AS400.	500
6	OBJECT-COMPUTER. IBM-AS400.	600
7	INPUT-OUTPUT SECTION.	700
8	FILE-CONTROL.	800
9	SELECT OUTFILE, ASSIGN TO PRINTER-QPRINT,	900
10	FILE STATUS IS FSTAT.	1000
11	DATA DIVISION.	1100
12	FILE SECTION.	1200
13	FD OUTFILE	1300
14	DATA RECORD IS REC-1,	1400
15	LABEL RECORDS ARE OMITTED.	1500
16	01 REC-1.	1600
17	05 CC PIC X.	1700
18	05 DEPT-NO PIC X(3).	1800
19	05 FILLER PIC X(5).	1900
20	05 AVERAGE-EDUCATION-LEVEL PIC ZZZ.	2000
21	05 FILLER PIC X(5).	2100
22	05 AVERAGE-SALARY PIC ZZZZ9.99.	2200
23	01 ERROR-RECORD.	2300
24	05 CC PIC X.	2400
25	05 ERROR-CODE PIC S9(5).	2500
26	05 ERROR-MESSAGE PIC X(70).	2600
27	WORKING-STORAGE SECTION.	2700
28	EXEC SQL	2800
29	INCLUDE SQLCA	2900
30	END-EXEC.	3000
31	77 FSTAT PIC XX.	3100
32	01 AVG-RECORD.	3200
33	05 WORKDEPT PIC X(3).	3300
34	05 AVG-EDUC PIC S9(4) USAGE COMP-4.	3400
35	05 AVG-SALARY PIC S9(6)V99 COMP-3.	3500
36	PROCEDURE DIVISION.	3600
37	*****	3700
38	* This program will get the average education level and the *	3800
39	* average salary by department. *	3900
40	*****	4000
41	A000-MAIN-PROCEDURE.	4100
42	OPEN OUTPUT OUTFILE.	4200
43	*****	4300
44	* Set-up WHENEVER statement to handle SQL errors. *	4400
45	*****	4500
46	EXEC SQL	4600
47	WHENEVER SQLERROR GO TO B000-SQL-ERROR	4700
48	END-EXEC.	4800
49	*****	4900
50	* Declare cursor *	5000
51	*****	5100
52	EXEC SQL	5200
53	DECLARE CURS CURSOR FOR	5300
54	SELECT WORKDEPT, AVG(EDLEVEL), AVG(SALARY)	5400
55	FROM CORPDATA.EMPLOYEE	5500
56	GROUP BY WORKDEPT	5600
57	END-EXEC.	5700
58	*****	5800
59	* Open cursor *	5900
60	*****	6000
61	EXEC SQL	6100
62	OPEN CURS	6200
63	END-EXEC.	6300

- 1** プリコンパイラーがソース・レコードを読み取った時に割り当てるレコード番号。レコード番号は、エラー・メッセージや SQL 実行時処理でソース・レコードを識別するのに使用されます。
- 2** ソース・レコードから取られた順序番号。この順序番号は、原始ステートメント入力ユーティリティ (SEU) を使ってソース・メンバーを編集するときに表示される番号です。
- 3** ソース・レコードが最後に変更された日付。Last Change がブランクの場合は、そのレコードが作成されてから変更されていないことを示します。

図2. サンプル COBOL プリコンパイラー出力の形式 (2/4)


```

5722ST1 V5R3M0 040528          SQL COBOL プログラムの作成          CBLTEST1          03/11/27 01:34:28   ページ 3
レコード *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8  SEQNBR 最終変更
64 *****
65 * Fetch all result rows *
66 *****
67     PERFORM A010-FETCH-PROCEDURE THROUGH A010-FETCH-EXIT
68     UNTIL SQLCODE IS = 100.
69 *****
70 * Close cursor *
71 *****
72     EXEC SQL
73     CLOSE CURS
74     END-EXEC.
75     CLOSE OUTFILE.
76     STOP RUN.
77 *****
78 * Fetch a row and move the information to the output record. *
79 *****
80     A010-FETCH-PROCEDURE.
81     MOVE SPACES TO REC-1.
82     EXEC SQL
83     FETCH CURS INTO :AVG-RECORD
84     END-EXEC.
85     IF SQLCODE IS = 0
86     MOVE WORKDEPT TO DEPT-NO
87     MOVE AVG-SALARY TO AVERAGE-SALARY
88     MOVE AVG-EDUC TO AVERAGE-EDUCATION-LEVEL
89     WRITE REC-1 AFTER ADVANCING 1 LINE.
90     A010-FETCH-EXIT.
91     EXIT.
92 *****
93 * An SQL error occurred. Move the error number to the error *
94 * record and stop running. *
95 *****
96     B000-SQL-ERROR.
97     MOVE SPACES TO ERROR-RECORD.
98     MOVE SQLCODE TO ERROR-CODE.
99     MOVE "AN SQL ERROR HAS OCCURRED" TO ERROR-MESSAGE.
100    WRITE ERROR-RECORD AFTER ADVANCING 1 LINE.
101    CLOSE OUTFILE.
102    STOP RUN.
***** ソースの終わり *****

```

図2. サンプル COBOL プリコンパイラー出力の形式 (3/4)

1 データ名	2 定義	3 参照
AVERAGE-EDUCATION-LEVEL	20	IN REC-1
AVERAGE-SALARY	22	IN REC-1
AVG-EDUC	34	SMALL INTEGER PRECISION(4,0) IN AVG-RECORD
AVG-RECORD	32	STRUCTURE 83
AVG-SALARY	35	DECIMAL(8,2) IN AVG-RECORD
BIRTHDATE	55	DATE(10) COLUMN IN CORPDATA.EMPLOYEE
BONUS	55	DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
B000-SQL-ERROR	****	LABEL 47
CC	17	CHARACTER(1) IN REC-1
CC	24	CHARACTER(1) IN ERROR-RECORD
COMM	55	DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
CORPDATA	****	4 COLLECTION 5 55
CURS	53	5 CURSOR 62 73 83
DEPT-NO	18	CHARACTER(3) IN REC-1
EDLEVEL	****	COLUMN 54
EDLEVEL	55	6 SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMPLOYEE	****	TABLE IN CORPDATA 7 55
EMPNO	55	CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
ERROR-CODE	25	NUMERIC(5,0) IN ERROR-RECORD
ERROR-MESSAGE	26	CHARACTER(70) IN ERROR-RECORD
ERROR-RECORD	23	STRUCTURE
FIRSTNME	55	VARCHAR(12) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
FSTAT	31	CHARACTER(2)
HIREDATE	55	DATE(10) COLUMN IN CORPDATA.EMPLOYEE
JOB	55	CHARACTER(8) COLUMN IN CORPDATA.EMPLOYEE
LASTNAME	55	VARCHAR(15) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
MIDINIT	55	CHARACTER(1) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
PHONENO	55	CHARACTER(4) COLUMN IN CORPDATA.EMPLOYEE
REC-1	16	
SALARY	****	COLUMN 54
SALARY	55	DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
SEX	55	CHARACTER(1) COLUMN IN CORPDATA.EMPLOYEE
WORKDEPT	33	CHARACTER(3) IN AVG-RECORD
WORKDEPT	****	COLUMN 54 56
WORKDEPT	55	CHARACTER(3) COLUMN IN CORPDATA.EMPLOYEE

ソースにエラーは見つからなかった。
102 ソース・レコードが処理された。
***** リストの終わり *****

1 Data Names (データ名) は、ソース・ステートメントで使用されている記号名です。

2 Define (定義) の列には、その名前が定義されている行番号が指定されます。行番号は、SQL プリコンパイラにより生成されます。**** は、そのオブジェクトが定義されていないか、あるいはプリコンパイラが宣言を認識することができなかったことを意味します。

3 Reference (参照) の列には次の 2 種類の情報が示されます。

- 記号名がどのように定義されているか **4**
- 記号名が現れる行の番号 **5**

記号名が有効なホスト変数を参照している場合には、データ・タイプ **6** またはデータ構造 **7** も注記されます。

図 2. サンプル COBOL プリコンパイラ出力の形式 (4/4)

非 ILE SQL プリコンパイラー・コマンド

DB2 UDB SQL 開発キットには、以下のホスト言語のための非 ILE プリコンパイラー・コマンドがあります。CRTSQLCBL (COBOL for iSeries)、CRTSQLPLI (iSeries PL/I)、および CRTSQLRPG (RPG III。これは RPG for iSeries の一部) です。オプションによっては特定の言語だけに適用されます。たとえば、*APOST と *QUOTE は COBOL に固有のオプションであり、他の言語のコマンドには用意されていません。詳しくは、209 ページの『第 13 章 ホスト言語プリコンパイラー用の DB2 UDB for iSeries CL コマンドの説明』を参照してください。

詳細については、『SQL を使用する非 ILE アプリケーション・プログラムのコンパイル』を参照してください。

SQL を使用する非 ILE アプリケーション・プログラムのコンパイル

プリコンパイルが正常に完了すると、*NOGEN が指定されている場合を除き、SQL プリコンパイラーは自動的にホスト言語コンパイラーを呼び出します。そして、プログラム名、ソース・ファイル名、プリコンパイラーにより作成されたソース・メンバー名、テキスト、および USRPRF を指定して CRTxxxPGM コマンドが実行されます。

これらの言語では、次のパラメーターが渡されます。

- COBOL では、*QUOTE または *APOST は CRTCLPGM コマンドに渡されます。
- RPG と COBOL では、SAAFLAG (*FLAG) は CRTxxxPGM コマンドに渡されます。
- RPG および COBOL では、CRTSQLxxx コマンドからの SRTSEQ および LANGID パラメーターは CRTxxxPGM コマンドで指定されます。
- RPG と COBOL では、CVTOPT (*DATETIME *VARCHAR) は常に CRTxxxPGM コマンドで指定されます。
- COBOL および RPG では、CRTSQLxxx コマンドからの TGTRLS パラメーターは、CRTxxxPGM コマンドで指定されます。TGTRLS は CRTPLIPGM コマンドでは指定されません。プログラムは、CRTSQLPLI コマンドの TGTRLS パラメーターで指定されたレベルで保管したり、または復元することができます。
- PL/I では、MARGINS は一時ソース・ファイルの中にセットされます。
- どの言語でも、CRTSQLxxx コマンドからの REPLACE パラメーターは CRTxxxPGM コマンドで指定されます。

パッケージをプリコンパイル処理の一部として作成した場合、CRTSQLxxx コマンドからの REPLACE パラメーター値は CRTSQLPKG コマンドで指定されます。

- どの言語の場合も、USRPRF(*USER) または USRPRF(*NAMING) によるシステム命名が指定される場合、USRPRF(*USER) は CRTxxxPGM コマンドで指定されます。USRPRF(*OWNER) または USRPRF(*NAMING) によって SQL 命名 (*SQL) が指定される場合、USRPRF (*OWNER) が CRTxxxPGM コマンドで指定されます。

CRTxxxPGM コマンドの他のすべてのパラメーターには、省略時値が使用されます。

プリコンパイラー・コマンドの OPTION パラメーターに *NOGEN を指定することにより、ホスト言語コンパイラーの呼び出しを中断することができます。*NOGEN では、ホスト言語コンパイラーを呼び出さないことを指定します。CRTSQLxxx コマンドでメンバー名としてオブジェクト名を使用すると、プリコンパイラーはソース・メンバーを出力ソース・ファイル (CRTSQLxxx コマンドの TOSRCFILE パラメーターとして指定) の中に作成します。この後、ホスト言語コンパイラーを明示的に呼び出し、出力ソース・ファイ

ルの中のソース・メンバーを指定し、省略時値を変更することができます。プリコンパイルとコンパイルを別々のステップとして行ったときは、CRTSQLPKG コマンドを使用して分散プログラム用の SQL パッケージを作成することができます。

注: CRTxxxPGM コマンドを出す前に QTEMP/QSQLTEMP の中のソース・メンバーを変更してはなりません。変更すると、コンパイルは正常に実行されません。

ILE SQL プリコンパイラー・コマンド

DB2 UDB SQL 開発キットには、ILE プリコンパイラー・コマンド CRTSQLCI、CRTSQLCPPI、CRTSQLCBLI、および CRTSQLRPGI が存在します。また、ILE C for iSeries、ILE C++ for iSeries、ILE COBOL for iSeries、および ILE RPG for iSeries の各ホスト言語ごとに、プリコンパイラー・コマンドがあります。言語ごとの個別コマンドにより、必須パラメーターを指定して、残りのパラメーターについては省略時値をとるようにすることができます。省略時値は、現在使用中の言語だけに適用できます。たとえば、*APOST と *QUOTE は COBOL に固有のオプションであり、他の言語のコマンドには用意されていません。詳しくは、209 ページの『第 13 章 ホスト言語プリコンパイラー用の DB2 UDB for iSeries CL コマンドの説明』を参照してください。

詳細については、以下のセクションを参照してください。

- 『SQL を使用する ILE アプリケーション・プログラムのコンパイル』

SQL を使用する ILE アプリケーション・プログラムのコンパイル

CRTSQLxxx コマンドのプリコンパイルが正常に完了すると、*NOGEN が指定されている場合を除き、SQL プリコンパイラーは自動的にホスト言語コンパイラーを呼び出します。*MODULE オプションを指定すると、SQL プリコンパイラーが CRTxxxMOD コマンドを出してモジュールを作成します。*PGM オプションを指定すると、SQL プリコンパイラーは CRTBNDxxx コマンドを出してプログラムを作成します。*SRVPGM オプションを指定すると、SQL プリコンパイラーは CRTxxxMOD コマンドを出してモジュールを作成してから、サービス・プログラム作成 (CRTSRVPGM) コマンドを出してサービス・プログラムを作成します。CRTSQLCPPI コマンドは *MODULE オブジェクトだけを作成します。

これらの言語では、次のパラメーターが渡されます。

- CRTSQLxxx コマンドで DBGVIEW(*SOURCE) を指定すると、DBGVIEW(*ALL) が CRTxxxMOD コマンドおよび CRTBNDxxx コマンドの両方で指定されます。
- CRTSQLxxx コマンドで OUTPUT(*PRINT) を指定すると、これは CRTxxxMOD コマンドおよび CRTBNDxxx コマンドの両方に渡されます。

CRTSQLxxx コマンドで OUTPUT(*NONE) を指定すると、これは CRTxxxMOD コマンドでも CRTBNDxxx コマンドでも指定されません。

- CRTSQLxxx コマンドからの TGTRLS パラメーター値は、CRTxxxMOD、CRTBNDxxx、およびサービス・プログラム作成 (CRTSRVPGM) コマンドで指定されます。
- CRTSQLxxx コマンドからの REPLACE パラメーター値は、CRTxxxMOD、CRTBNDxxx、および CRTSRVPGM の各コマンドで指定されます。

パッケージをプリコンパイル処理の一部として作成した場合、CRTSQLxxx コマンドからの REPLACE パラメーター値は CRTSQLPKG コマンドで指定されます。

- OBJTYPE が *PGM または *SRVPGM のいずれかで、USRPRF(*USER) または USRPRF(NAMING) を伴うシステム (*SYS) 命名を使用している場合、USRPRF(*USER) が CRTBNDxxx コマンドまたは CRTSRVPGM コマンドで指定されます。

OBJTYPE が *PGM または *SRVPGM のいずれかで、USRPRF(*OWNER) または USRPRF(*NAMING) を伴う SQL(*SQL) 命名を使用している場合、USRPRF(*OWNER) が CRTBNDxxx コマンドまたは CRTSRVPGM コマンドで指定されます。

- C および C++ では、MARGINS は一時ソース・ファイルの中にセットされます。

LOB ホスト変数の全長が 15M に近いとプリコンパイラーが計算する場合は、TERASPACE(*YES *TSIFC) オプションが CRTCMOD、CRTBNDc、または CRTCPPMOD コマンドで指定されます。

- COBOL では、*QUOTE または *APOST が CRTBNDcBL コマンドまたは CRTcBLMOD コマンドに渡されます。
- RPG および COBOL では、CRTSQLxxx コマンドからの SRTSEQ および LANGID パラメーターを CRTxxxMOD コマンドおよび CRTBNDxxx コマンドで指定します。
- COBOL では、CVTOPT(*VARCHAR *DATETIME *PICGGRAPHIC *FLOAT) は、常に CRTcBLMOD コマンドおよび CRTBNDcBL コマンドで指定されます。OPTION(*NOCVTDT) を指定すると (出荷時のコマンドの省略時値)、CVTOPT について追加のオプション *DATE *TIME *TIMESTAMP も指定されます。
- RPG では、OPTION(*CVTDT) を指定すると、CVTOPT(*DATETIME) は CRTRPGMOD コマンドおよび CRTBNDRPG コマンドで指定されます。

プリコンパイラー・コマンドの OPTION パラメーターに *NOGEN を指定することにより、ホスト言語コンパイラーの呼び出しを中断することができます。*NOGEN では、ホスト言語コンパイラーを呼び出さないことを指定します。CRTSQLxxx コマンドでメンバー名としてオブジェクト名を使用すると、プリコンパイラーはソース・メンバーを出力ソース・ファイル (TOSRCFILE パラメーター) の中に作成します。この後、ホスト言語コンパイラーを明示的に呼び出し、出力ソース・ファイルの中のソース・メンバーを指定し、省略時値を変更することができます。プリコンパイルとコンパイルを別々のステップとして行ったときは、CRTSQLPKG コマンドを使用して分散プログラム用の SQL パッケージを作成することができます。

プログラムまたはサービス・プログラムを後から作成する場合、USRPRF パラメーターが、CRTBNDxxx コマンド、プログラム作成 (CRTPGM) コマンド、またはサービス・プログラム作成 (CRTSRVPGM) コマンドで正しくセットされていない可能性があります。SQL プログラムは、USRPRF パラメーターを訂正してからでないと、予測どおりに実行されません。システム命名規則を使用している場合、USRPRF パラメーターを *USER にセットしなければなりません。SQL 命名規則を使用している場合、USRPRF パラメーターは *OWNER にセットしなければなりません。

1 プリコンパイラー・コマンドを使用したコンパイラー・オプションの設定

1 コンパイラー・コマンドで追加のパラメーターを使用するために、プリコンパイラー・コマンドおよび
1 SET OPTION ステートメントにおいて COMPILEOPT スtringを使用できます。COMPILEOPT ストリ
1 ングは、プリコンパイラーによって構築されるコンパイラー・コマンドに追加されます。これによって、プ
1 リコンパイルとコンパイルの 2 つの処理を実行しなくても、コンパイラー・パラメーターを指定すること
1 ことができます。COMPILEOPT スtring内には、SQL プリコンパイラーによって渡されるパラメーター
1 を指定しないでください。そのようにした場合、パラメーター重複エラーのためにコンパイラー・コマンド
1 が失敗します。将来的に、SQL プリコンパイラーがさらに追加のパラメーターをコンパイラーに渡すよう
1 になる可能性があります。そうなった場合、パラメーターの重複エラーが発生し、COMPILEOPT ストリ
1 ングの変更が必要になるかもしれません。

1 COMPILEOPT スtringのいずれかの場所に "INCDIR(" が存在する場合、プリコンパイラーは
1 SRCSTMF パラメーターを使用してコンパイラーを呼び出します。

```
| EXEC SQL SET OPTION COMPILEOPT ='OPTION(*SHOWINC *EXPMAC)
| INCDIR(''/QSYS.LIB/MYLIB.LIB/MYFILE.MBR ''');
```

SQL を使用するアプリケーションでのコンパイル・エラーの解釈

重要: プリコンパイルとコンパイルのステップを分けていて、ソース・プログラムが外部記述ファイルを参照している場合、参照されるファイルをプリコンパイルとコンパイルの間で変更してはなりません。そうしないと、フィールド定義への変更が一時ソース・メンバー内で変更されないため、予期しない結果が生じることがあります。

外部記述ファイルの例を以下に示します。

- COBOL の COPY DDS
- PL/I の %INCLUDE
- C または C++ の #pragma mapinc および #include
- RPG の外部記述ファイルおよび外部記述データ構造

SQL プリコンパイラーがホスト変数を認識しない時は、ソース・プログラムのコンパイルを行ってください。コンパイラーは EXEC SQL ステートメントを認識せず、これらのエラーを無視します。コンパイラーがその言語の SQL プリコンパイラーで定義したとおりにホスト変数宣言を解釈していることを確かめてください。

詳細については、『SQL を使用するアプリケーション・プログラムのコンパイル時のエラーおよび警告メッセージ』を参照してください。

SQL を使用するアプリケーション・プログラムのコンパイル時のエラーおよび警告メッセージ

以下のトピックのような条件が発生すると、コンパイル処理時にエラー・メッセージまたは警告メッセージが出されることがあります。

- 『SQL を使用するアプリケーション・プログラムのコンパイル時のエラーおよび警告メッセージ』
- 『PL/I、C、または C++ コンパイル時のエラーおよび警告メッセージ』
- 『COBOL コンパイル時のエラーおよび警告メッセージ』
- 『RPG コンパイル時のエラーおよび警告メッセージ』

PL/I、C、または C++ コンパイル時のエラーおよび警告メッセージ

左マージン (MARGINS パラメーターの指定値、または省略時値) より前で EXEC SQL が始まっている場合には、SQL プリコンパイラーはそのステートメントを SQL ステートメントとして認識しません。その結果、そのステートメントはそのままの形でコンパイラーに渡されます。

COBOL コンパイル時のエラーおよび警告メッセージ

EXEC SQL が 12 桁目より前で始まっている場合には、SQL プリコンパイラーはそのステートメントを SQL ステートメントとして認識しません。その結果、そのステートメントはそのままの形でコンパイラーに渡されます。

RPG コンパイル時のエラーおよび警告メッセージ

EXEC SQL が 8 桁目から 16 桁目までにコーディングされていないで、7 桁目に ' ' 文字が置かれている場合には、SQL プリコンパイラーはそのステートメントを SQL ステートメントとして認識しません。その結果、そのステートメントはそのままの形でコンパイラーに渡されます。

詳しい説明については、『第 5 章 C および C++ アプリケーションでの SQL ステートメントのコーディング方法』から『第 10 章 REXX アプリケーションでの SQL ステートメントのコーディング方法』までのそれぞれのプログラミング例を参照してください。

SQL を使用するアプリケーションのバインド

アプリケーション・プログラムを実行するためには、その前に、プログラムと、プログラムで指定したテーブルおよびビューとを関係づけておかなければなりません。このプロセスを**バインド処理**と呼びます。そしてバインド処理の結果を**アクセス・プラン**と呼びます。

アクセス・プランは、各 SQL 要求を満たすのに必要な処置を記述した制御構造です。アクセス・プランには、プログラムに関する情報とそのプログラムが使用しようとするデータに関する情報が収められています。

非分散 SQL プログラムの場合、アクセス・プランはそのプログラム内に保管されます。分散 SQL プログラム (RDB パラメーターが CRTSQLxxx コマンドで指定されている) の場合は、アクセス・プランは、指定したリレーショナル・データベースに置かれている SQL パッケージに保管されます。

SQL は、プログラム・オブジェクトが作成されると、自動的にアクセス・プランをバインドして作成しようとします。非 ILE コンパイルの場合、これは CRTxxxPGM が正常に完了した結果として起こります。ILE コンパイルの場合、これは CRTBNDxxx、CRTPGM、または CRTSRVPGM の各コマンドが正常に完了した結果として起こります。実行時に、アクセス・プランが無効であること (たとえば、参照テーブルが異なるライブラリーにある)、あるいはパフォーマンスを向上させるような変更 (たとえば、索引の追加) などがデータベースに行われたことを、DB2 UDB for iSeries が見つけると、新しいアクセス・プランが自動的に作成されます。バインド処理では、次の 3 つのことが行われます。

1. **データベースの記述を用いて SQL ステートメントの妥当性検査を再度行う。** バインド処理の時点で、テーブル、ビュー、および列の名前が有効かどうかについて、SQL ステートメントが検査されます。指定したテーブルまたはビューがプリコンパイルまたはコンパイル時に存在していない時は、妥当性検査は実行時に行われます。実行時にもテーブルまたはビューが存在しない場合には、負の SQLCODE が返されます。
2. **プログラムで処理したいデータにアクセスするのに必要な索引を選択する。** 索引を選択する際には、テーブル・サイズ、その他の要因が考慮されて、アクセス・プランが作成されます。データをアクセスするのに使用できるすべての索引が考慮され、データに至るパスを選択するとき、(もしあれば) どの索引を使用するかが決定されます。
3. **アクセス・プランの作成を試みる。** すべての SQL ステートメントが有効であれば、バインド処理はアクセス・プランを作成して、それをプログラムの中に保管します。

プログラムがアクセスするテーブルまたはビューの特性が変わると、それまでのアクセス・プランは有効でなくなる場合があります。有効でないアクセス・プランを含むプログラムを実行しようとする時、システムはアクセス・プランを再作成することを自動的に試みます。アクセス・プランが再作成できない時は、負の SQLCODE が返されます。この場合は、ユーザーはプログラムの SQL ステートメントを変更して、CRTSQLxxx コマンドを再度出して状況を訂正しなければならないことがあります。

たとえば、あるプログラムに、TABLEA の COLUMNNA を参照する SQL ステートメントが含まれている場合に、ユーザーが TABLEA を削除し再作成した結果、COLUMNNA が存在しなくなったとすると、そのプログラムを呼び出したとき、COLUMNNA が存在していないので、自動再バインドは正しく行われません。この場合は、ユーザーはプログラムのソース仕様を変更して、CRTSQLxxx コマンドを出し直さなければなりません。

詳細については、『SQL を使用するアプリケーションでのプログラム参照』を参照してください。

SQL を使用するアプリケーションでのプログラム参照

SQL プログラムの中の SQL ステートメントで参照されるスキーマ、テーブル、ビュー、SQL パッケージ、および索引は、いずれも、プログラムの作成時にライブラリーのオブジェクト情報リポジトリ (OIR) に保管されます。

CL コマンドのプログラム参照表示 (DSPPGMREF) を使用すると、プログラムの中のすべてのオブジェクト参照を表示することができます。SQL の命名規則が使用されている場合は、ライブラリー名は次のいずれかの形で OIR に保管されます。

1. SQL 名が完全修飾されている場合には、コレクション名が名前の修飾子として保管されます。
2. SQL 名が完全修飾されていないで、DFTRDBCOL パラメーターの指定がない場合は、ステートメントの権限 ID が名前の修飾子として保管されます。
3. SQL 名が完全修飾されず、DFTRDBCOL パラメーターが指定されている場合には、DFTRDBCOL パラメーターで指定されたスキーマ名が名前の修飾子として保管されます。

システムの命名規則が使用されている場合には、ライブラリー名は次のいずれかの形で OIR に保管されず。

1. オブジェクト名が完全修飾されている場合には、ライブラリー名が名前の修飾子として保管されます。
2. オブジェクトが完全修飾されていないで、DFTRDBCOL パラメーターの指定がない場合は、*LIBL が保管されます。
3. SQL 名が完全修飾されず、DFTRDBCOL パラメーターが指定されている場合には、DFTRDBCOL パラメーターで指定されたスキーマ名が名前の修飾子として保管されます。

SQL プリコンパイラー・オプションの表示


SQL アプリケーション・プログラムのコンパイルが正常に完了したら、モジュール表示 (DSPMOD)、プログラム表示 (DSPPGM)、またはサービス・プログラム表示 (DPSRVPGM) の各コマンドを使用して、SQL プリコンパイルで指定済みのいくつかのオプションを判別することができます。この情報が必要になるのは、プログラムのソース仕様を変更する必要がある時です。プログラムを再度コンパイルするときに、これらと同じ SQL プリコンパイラー・オプションを CRTSQLxxx コマンドで指定することができます。

SQL 情報印刷 (PRTSQLINF) コマンドを使用しても、SQL プリコンパイルで指定したいいくつかのオプションを判別することができます。

組み込み SQL を使用したプログラムの実行

プリコンパイルとコンパイルが正常に完了した後で、SQL ステートメントが組み込まれているホスト言語プログラムを実行する方法は、他のホスト・プログラムを実行する場合と同じです。次のように、

```
CALL pgm-name
```

システム・コマンド行から入力してください。プログラムの実行方法の詳細については、「CL プログラミング 」を参照してください。

注: 新しいリリースを導入した後、ユーザーがプログラムに対する *CHANGE 権限を持っていない場合、構造化照会言語 (SQL) プログラムを使用する QHST で、メッセージ CPF2218 を受け取ることがあり

ます。*CHANGE 権限を持つユーザーがいったんプログラムを呼び出した後は、アクセス・プランが更新され、その旨を知らせるメッセージが出されます。

詳細については、以下のセクションを参照してください。

- 『組み込み SQL を使用したプログラムの実行: OS/400 DDM の考慮事項』
- 『組み込み SQL を使用したプログラム実行: 一時変更に関する考慮事項』
- 『組み込み SQL を使用したプログラムの実行: SQL 戻りコード』

組み込み SQL を使用したプログラムの実行: OS/400 DDM の考慮事項

SQL は、DDM (分散データ管理) ファイルを経由するリモート・ファイルのアクセスをサポートしていません。SQL は、DRDA[®] (分散リレーショナル・データベース体系 (Distributed Relational Database Architecture[™])) を経由するリモート・アクセスをサポートしています。

組み込み SQL を使用したプログラム実行: 一時変更に関する考慮事項

一時変更 (OVRDBF コマンドによって指定します) を使用すれば、別のテーブルまたはビューを参照したり、あるいは、プログラムまたは SQL パッケージの操作特性の一部を変更することができます。一時変更を指定した場合には、下記のパラメーターが処理されます。一時変更を指定した場合には、下記のパラメーターが処理されます。

TOFILE
MBR
SEQONLY
INHWRT
WAITRCD

上記以外の一時変更パラメーターはすべて無視されます。SQL パッケージ内のステートメントの一時変更は、以下の両方の操作によって行われます。

1. OVRDBF コマンドでの OVRSCOPE(*JOB) パラメーターの指定
2. リモート・コマンド投入 (SBMRMTCMD) コマンドの使用による、アプリケーション・サーバーへのコマンドの送信

長名で作成されたテーブルおよびビューを一時変更するには、そのテーブルまたはビューと関連付けられたシステム名を使用して一時変更を作成することができます。SQL ステートメントで長名が指定されている場合、一時変更は対応するシステム名を用いて検出されます。

別名は、実際には DDM ファイルとして作成されます。別名 (DDM ファイル) を参照する一時変更を作成することもできます。この場合、その一時変更を含むファイルを参照する SQL ステートメントは、実際には別名が参照するファイルを使用していることとなります。

一時変更の詳細については、「データベース・プログラミング」および「ファイル管理」を参照してください。

組み込み SQL を使用したプログラムの実行: SQL 戻りコード

SQL 戻りコードのリストは、iSeries Information Center の『SQL メッセージおよびコード』を参照してください。

第 12 章 DB2 UDB for iSeries ステートメントを使用したサンプル・プログラム

このトピックでは、DB2 UDB for iSeries システムがサポートする各言語で SQL ステートメントをコーディングする方法を示したサンプル・アプリケーションが記載されています。

注: 重要な法律的事柄に関する情報について、214 ページの『コードについての特記事項』をご覧ください。

SQL ステートメントを使用するプログラム例

以下のプログラム言語について、組み込み SQL ステートメントのコーディング例を示したプログラムが提供されています。

- ILE C および C++
- COBOL および ILE COBOL
- PL/I
- RPG for iSeries
- ILE RPG for iSeries
- REXX

サンプル・アプリケーションは、年俸に基づく昇給を行うものです。

どのサンプル・プログラムでも同じ報告書が作成されますが、その報告書はこのトピックの最後に示されています。報告書の最初の部分には、プロジェクトに参加し、昇給を受けたすべての社員がプロジェクト別に示されます。報告書の第 2 の部分には、新しい給与支出額がプロジェクト別に示されます。

サンプル・プログラムについての注:

次の注記は、すべてのサンプル・プログラムに適用されます。

SQL ステートメントは大文字でも小文字でも入力できます。

- 1** このホスト言語ステートメントは、SQL テーブル PROJECT に関する外部定義を検索するためのものです。これらの外部定義は、ホスト変数としてもホスト構造としても使用できます。

注:

1. RPG for iSeries では、外部で記述された構造の中のフィールド名のうち、6 文字より長い名前は変更しなければなりません。
2. REXX では、外部定義の検索はサポートされません。

- 2** SQL INCLUDE SQLCA ステートメントは、PL/I、C、および COBOL プログラム用の SQLCA を挿入するために使用されます。RPG プログラムの場合は、SQL プリコンパイラーが入力仕様セクションの最後にあるソース・プログラムに SQLCA データ構造を自動的に置きます。REXX の場合は、SQLCA によってマッピングされる連続データ域ではなく、個別の変数の形で SQLCA フィールドが保管されます。

- 3** この SQL WHENEVER ステートメントは、SQL ステートメントに SQLERROR (SQLCODE < 0) が現れたとき、制御権が渡されるホスト言語ラベルを定義します。この WHENEVER SQLERROR ステートメントは、次の WHENEVER SQLERROR ステートメントが現れるまですべての SQL ス

ステートメントに適用されます。REXX では、WHENEVER ステートメントはサポートされません。その代わりに、REXX では SIGNAL ON ERROR 機能を使用します。

4 この SQL UPDATE ステートメントは、SALARY 列を更新するためのものです。この列には、ホスト変数 PERCENTAGE (RPG では PERCNT) に入っている比率として社員の給与が入ります。更新される行は、社員の年俸が 2000 を超えている行です。REXX の場合、ホスト変数があると UPDATE を直接実行できないため、このステートメントは PREPARE と EXECUTE になります。

5 この SQL COMMIT ステートメントは、SQL UPDATE ステートメントによって行われた変更をコミットするためのものです。変更されたすべての行のレコード・ロックが解除されます。

注: プログラムは、COMMIT(*CHG) を使用してプリコンパイルされています。(REXX の場合は、*CHG が省略時値になります。)

6 この SQL DECLARE CURSOR ステートメントはカーソル C1 を定義しています。カーソル C1 は、2 つのテーブル EMPLOYEE と EMPPROJECT を結合し、昇給を受けた社員 (commission > 2000) の行を返します。行は、プロジェクト番号と社員番号 (PROJNO 列と EMPNO 列) の昇順に返されます。REXX の場合は、ホスト変数を含む場合にステートメント文字列を指定して DECLARE CURSOR ステートメントを直接指定することができないため、このステートメントは PREPARE と DECLARE CURSOR になります。

7 この SQL OPEN ステートメントはカーソル C1 をオープンして、行の取り出し (FETCH) ができるようにします。

8 この SQL WHENEVER ステートメントは、すべての行が取り出されたとき (SQLCODE = 100) 制御権が渡されるホスト言語ラベルを定義します。REXX の場合は、SQLCODE を明示的に検査しなければなりません。

9 この SQL FETCH ステートメントはカーソル C1 に関するすべての列を返し、返した値をホスト構造の対応する要素に入れます。

10 すべての行が取り出されると、制御権がこのラベルに返されます。SQL CLOSE ステートメントはカーソル C1 をクローズします。

11 この SQL DECLARE CURSOR ステートメントはカーソル C2 を定義しています。カーソル C2 は 3 つのテーブル EMPPROJECT、PROJECT、および EMPLOYEE を結合します。その結果は、PROJNO 列と PROJNAME 列によってグループ化されます。COUNT 関数は各グループの行数を返します。SUM 関数は新しい給与支給額を各プロジェクト別に計算します。ORDER BY 1 文節は、最終結果の列 (EMPPROJECT.PROJNO) の内容に基づいて行を検索することを指定しています。REXX の場合は、ホスト変数を含む場合にステートメント文字列を指定して DECLARE CURSOR ステートメントを直接指定することができないため、このステートメントは PREPARE と DECLARE CURSOR になります。

12 この SQL FETCH ステートメントはカーソル C2 に関する結果の列を返し、返した値をプログラムにより記述されているホスト構造の対応する要素に入れます。

13 この SQL WHENEVER ステートメントには CONTINUE オプションが指定されているので、SQL ROLLBACK ステートメントでエラーが起こったかどうかに関係なく、次のステートメントに移って処理が続行されます。SQL ROLLBACK ステートメントにはエラーがないことが前提とされます。仮にエラーが生じて、このオプションによりプログラムがループに入ることが防止されます。SQLERROR ステートメントは、次の WHENEVER SQLERROR ステートメントが現れるまですべての SQL ステートメントに適用されます。REXX では、WHENEVER ステートメントはサポートされません。その代わりに、REXX では SIGNAL OFF ERROR 機能を使用します。

- 14** この SQL ROLLBACK ステートメントは、更新中にエラーが起こった場合にテーブルを元の状態に復元します。

IBM は、お客様に、このプログラムをサンプルとして使用することができる非独占的な使用権を許諾します。お客様は、このサンプル・コードから、お客様独自の特別のニーズに合わせた類似のプログラムを作成することができます。

このサンプル・コードは、例として示す目的でのみ、IBM により提供されます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

ここに含まれるすべてのプログラムは、現存するままの状態を提供され、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

例: ILE C および C++ プログラム内の SQL ステートメント

このサンプル・プログラムは、C プログラミング言語で作成されています。次の条件が満たされれば、同じプログラムは C++ でも動作します。

- 18 行目の前に SQL BEGIN DECLARE SECTION ステートメントを追加する
- 42 行目の後に SQL END DECLARE SECTION ステートメントを追加する

注: 重要な法律的事柄に関する情報について、214 ページの『コードについての特記事項』をご覧ください。

```

ソース仕様タイプ . . . . .C
オブジェクト名 . . . . .CORPDATA/CEX
ソース・ファイル . . . . .CORPDATA/SRC
メンバー . . . . .CEX
TO ソース・ファイル. . . . .QTEMP/QSQLTEMP
オプション . . . . .*XREF
プログラムの印刷 . . . . .*PRINT
ターゲット・リリース . . . V5R3M0
INCLUDE ファイル . . . . .*SRCFILE
コミット . . . . .*CHG
データのコピー可能 . . . . *YES
SQL カーソルのクローズ . . *ENDACTGRP
ブロック化可能 . . . . .*READ
PREPARE 遅延 . . . . .*NO
生成レベル . . . . .10
マージン . . . . .*SRCFILE
印刷装置ファイル . . . . .*LIBL/QSYSPRT
日付の形式 . . . . .*JOB
日付区切り記号 . . . . .*JOB
時刻の形式 . . . . .*HMS
時刻区切り記号 . . . . .*JOB
置き換え . . . . .*YES
リレーショナル・データベース*LOCAL
ユーザー . . . . .*CURRENT
RDB 接続方式. . . . .*DUW
省略時のコレクション . . . *NONE
動的省略時の
コレクション . . . . .*NO
パッケージ名 . . . . .*OBJLIB/*OBJ
パス . . . . .*NAMING
SQL 規則 . . . . .*DB2
作成オブジェクト・タイプ . *PGM
デバッグ・ビュー . . . . .*NONE
ユーザー・プロファイル . . *NAMING
動的ユーザー・プロファイル*USER
ソート順序 . . . . .*JOB
言語 ID. . . . .*JOB
IBM SQL フラグづけ. . . . .*NOFLAG
ANS フラグ付け. . . . .*NONE
テキスト . . . . .*SRCMBRTXT
ソース・ファイルの CCSID. . 65535
ジョブの CCSID. . . . .65535
10 進数結果オプション
最大精度. . . . .31
最大位取り. . . . .31
除算の最小位取り. . . . .0
コンパイラ・オプション . *NONE
03/11/27 02:12:33 にソース・メンバーが変更された。

```

レコード*	1	2	3	4	5	6	7	8	9	0	SEQNBR	最終変更
1	#include "string.h"										100	
2	#include "stdlib.h"										200	
3	#include "stdio.h"										300	
4											400	
5	main()										500	
6	{										600	
7	/* A sample program which updates the salaries for those employees	*/									700	
8	/* whose current commission total is greater than or equal to the	*/									800	
9	/* value of 'commission'. The salaries of those who qualify are	*/									900	
10	/* increased by the value of 'percentage' retroactive to 'raise_date'*/										1000	
11	/* A report is generated showing the projects which these employees	*/									1100	
12	/* have contributed to ordered by project number and employee ID.	*/									1200	
13	/* A second report shows each project having an end date occurring	*/									1300	
14	/* after 'raise_date' (is potentially affected by the retroactive	*/									1400	
15	/* raises) with its total salary expenses and a count of employees	*/									1500	
16	/* who contributed to the project.	*/									1600	
17											1700	

図3. SQL ステートメントを使用したサンプル C プログラム (1/5)

5722ST1	V5R3M0	040528	SQL ILE C オブジェクトの作成	CEX	03/11/27 02:13:36	ページ 2				
レコード	*...+... 1	...+... 2	...+... 3	...+... 4	...+... 5	...+... 6	...+... 7	...+... 8	SEQNBR	最終変更

```

18 short work_days = 253; /* work days during in one year */ 1800
19 float commission = 2000.00; /* cutoff to qualify for raise */ 1900
20 float percentage = 1.04; /* raised salary as percentage */ 2000
21 char raise_date?(12??) = "1982-06-01"; /* effective raise date */ 2100
22 2200
23 /* File declaration for qprint */ 2300
24 FILE *qprint; 2400
25 2500
26 /* Structure for report 1 */ 2600
27 1 #pragma mapinc ("project","CORPDATA/PROJECT(PROJECT)","both","p z") 2700
28 #include "project" 2800
29 struct { 2900
30 CORPDATA_PROJECT_PROJECT_both_t Proj_struct; 3000
31 char empno?(7??); 3100
32 char name?(30??); 3200
33 float salary; 3300
34 } rpt1; 3400
35 3500
36 /* Structure for report 2 */ 3600
37 struct { 3700
38 char projno?(7??); 3800
39 char project_name?(37??); 3900
40 short employee_count; 4000
41 double total_proj_cost; 4100
42 } rpt2; 4200
43 4300
44 2 exec sql include SQLCA; 4400
45 4500
46 qprint=fopen("QPRINT","w"); 4600
47 4700
48 /* Update the selected projects by the new percentage. If an error */ 4800
49 /* occurs during the update, ROLLBACK the changes. */ 4900
50 3 EXEC SQL WHENEVER SQLERROR GO TO update_error; 5000
51 4 EXEC SQL 5100
52 UPDATE CORPDATA/EMPLOYEE 5200
53 SET SALARY = SALARY * :percentage 5300
54 WHERE COMM >= :commission ; 5400
55 5500
56 /* Commit changes */ 5600
57 5 EXEC SQL 5700
58 COMMIT; 5800
59 EXEC SQL WHENEVER SQLERROR GO TO report_error; 5900
60 6000
61 /* Report the updated statistics for each employee assigned to the */ 6100
62 /* selected projects. */ 6200
63 6300
64 /* Write out the header for Report 1 */ 6400
65 fprintf(qprint," REPORT OF PROJECTS AFFECTED ¥ 6500
66 BY RAISES"); 6600
67 fprintf(qprint,"%n%nPROJECT EMPID EMPLOYEE NAME "); 6700
68 fprintf(qprint, " SALARY%n"); 6800
69 6900
70 6 exec sql 7000
71 declare c1 cursor for 7100
72 select distinct projno, empproject.empno, 7200
73 lastname||', '||firstme, salary 7300
74 from corpdata/empproject, corpdata/employee 7400
75 where empproject.empno = employee.empno and comm >= :commission 7500
76 order by projno, empno; 7600
77 7 EXEC SQL 7700
78 OPEN C1; 7800
79 7900
80 /* Fetch and write the rows to QPRINT */ 8000
81 8 EXEC SQL WHENEVER NOT FOUND GO TO done1; 8100
82 8200
83 do { 8300
84 10 EXEC SQL 8400
85 FETCH C1 INTO :Proj_struct.PROJNO, :rpt1.empno, 8500
86 :rpt1.name,:rpt1.salary; 8600
87 fprintf(qprint,"%n%6s %6s %-30s %8.2f", 8700
88 rpt1.Proj_struct.PROJNO,rpt1.empno, 8800
89 rpt1.name,rpt1.salary); 8900
90 } 9000
91 while (SQLCODE==0); 9100
92 9200
93 done1: 9300
94 EXEC SQL 9400
95 CLOSE C1; 9500

```

図 3. SQL ステートメントを使用したサンプル C プログラム (2/5)

```

5722ST1 V5R3M0 040528          SQL ILE C オブジェクトの作成          CEX          03/11/27 02:13:36   ページ 3
レコード *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8  SEQNBR 最終変更
 96                                     9600
 97 /* For all projects ending at a date later than the 'raise_date' */
 98 /* (i.e. those projects potentially affected by the salary raises) */
 99 /* generate a report containing the project number, project name */
100 /* the count of employees participating in the project and the */
101 /* total salary cost of the project. */
102                                     10200
103 /* Write out the header for Report 2 */
104 fprintf(qprint,"%n%n%n          ACCUMULATED STATISTICS¥
105 BY PROJECT");
106 fprintf(qprint, " ¥n¥nPROJECT          ¥
107 NUMBER OF          TOTAL");
108 fprintf(qprint, " ¥nNUMBER          PROJECT NAME          ¥
109 EMPLOYEES          COST¥n");
110                                     11000
111 11 EXEC SQL
112     DECLARE C2 CURSOR FOR
113         SELECT EMPPROJECT.PROJNO, PROJNAME, COUNT(*),
114             SUM ( ( DAYS(EMENDATE) - DAYS(EMSTDATE) ) * EMPTIME *
115                 (DECIMAL( SALARY / :work_days ,8,2)))
116         FROM CORPDATA/EMPPROJECT, CORPDATA/PROJECT, CORPDATA/EMPLOYEE
117         WHERE EMPPROJECT.PROJNO=PROJECT.PROJNO AND
118             EMPPROJECT.EMPNO =EMPLOYEE.EMPNO AND
119             PRENDATE > :raise_date
120         GROUP BY EMPPROJECT.PROJNO, PROJNAME
121         ORDER BY 1;
122 EXEC SQL
123     OPEN C2;
124                                     12400
125 /* Fetch and write the rows to QPRINT */
126 EXEC SQL WHENEVER NOT FOUND GO TO done2;
127                                     12700
128 do {
129     12 EXEC SQL
130         FETCH C2 INTO :rpt2;
131         fprintf(qprint,"%n%6s %-36s %6d %9.2f",
132             rpt2.projno,rpt2.project_name,rpt2.employee_count,
133             rpt2.total_proj_cost);
134     }
135     while (SQLCODE==0);
136                                     13600
137 done2:
138     EXEC SQL
139         CLOSE C2;
140     goto finished;
141                                     14100
142 /* Error occured while updating table. Inform user and rollback */
143 /* changes. */
144 update_error:
145     13 EXEC SQL WHENEVER SQLERROR CONTINUE;
146     fprintf(qprint,"*** ERROR Occurred while updating table. SQLCODE="
147         "%5d¥n",SQLCODE);
148     14 EXEC SQL
149         ROLLBACK;
150     goto finished;
151                                     15100
152 /* Error occured while generating reports. Inform user and exit. */
153 report_error:
154     fprintf(qprint,"*** ERROR Occurred while generating reports. "
155         "SQLCODE=%5d¥n",SQLCODE);
156     goto finished;
157                                     15700
158 /* All done */
159 finished:
160     fclose(qprint);
161     exit(0);
162                                     16200
163 }
*** ソースの終わり ***

```

図3. SQL ステートメントを使用したサンプル C プログラム (3/5)

相互参照 データ名	定義	参照
commission	19	FLOAT(24) 54 75
done1	****	LABEL 81
done2	****	LABEL 126
employee_count	40	SMALL INTEGER PRECISION(4,0) IN rpt2
empno	31	VARCHAR(7) IN rpt1 85
name	32	VARCHAR(30) IN rpt1 86
percentage	20	FLOAT(24) 53
project_name	39	VARCHAR(37) IN rpt2
projno	38	VARCHAR(7) IN rpt2
raise_date	21	VARCHAR(12) 119
report_error	****	LABEL 59
rpt1	34	
rpt2	42	STRUCTURE 130
salary	33	FLOAT(24) IN rpt1 86
total_proj_cost	41	FLOAT(53) IN rpt2
update_error	****	LABEL 50
work_days	18	SMALL INTEGER PRECISION(4,0) 115
ACTNO	74	SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPPROJECT
BIRTHDATE	74	DATE(10) COLUMN IN CORPDATA.EMPLOYEE
BONUS	74	DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
COMM	****	COLUMN 54 75
COMM	74	DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
CORPDATA	****	COLLECTION 52 74 74 116 116 116
C1	71	CURSOR 78 85 95
C2	112	CURSOR 123 130 139
DEPTNO	27	VARCHAR(3) IN Proj_struct
DEPTNO	116	CHARACTER(3) COLUMN (NOT NULL) IN CORPDATA.PROJECT
EDLEVEL	74	SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMENDATE	74	DATE(10) COLUMN IN CORPDATA.EMPPROJECT
EMENDATE	****	COLUMN 114
EMPLOYEE	****	TABLE IN CORPDATA 52 74 116
EMPLOYEE	****	TABLE 75 118
EMPNO	****	COLUMN IN EMPPROJECT 72 75 76 118
EMPNO	****	COLUMN IN EMPLOYEE 75 118
EMPNO	74	CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJECT
EMPNO	74	CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMPPROJECT	****	TABLE 72 75 113 117 118 120
EMPPROJECT	****	TABLE IN CORPDATA 74 116
EMPTIME	74	DECIMAL(5,2) COLUMN IN CORPDATA.EMPPROJECT
EMPTIME	****	COLUMN 114
EMSTDATE	74	DATE(10) COLUMN IN CORPDATA.EMPPROJECT
EMSTDATE	****	COLUMN 114
FIRSTNME	****	COLUMN 73
FIRSTNME	74	VARCHAR(12) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
HIREDATE	74	DATE(10) COLUMN IN CORPDATA.EMPLOYEE
JOB	74	CHARACTER(8) COLUMN IN CORPDATA.EMPLOYEE
LASTNAME	****	COLUMN 73
LASTNAME	74	VARCHAR(15) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE

図 3. SQL ステートメントを使用したサンプル C プログラム (4/5)

```
MAJPROJ          27      VARCHAR(6) IN Proj_struct
MAJPROJ          116     CHARACTER(6) COLUMN IN CORPDATA.PROJECT
MIDINIT          74      CHARACTER(1) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
Proj_struct      30      STRUCTURE IN rpt1
PHONENO          74      CHARACTER(4) COLUMN IN CORPDATA.EMPLOYEE
PRENDATE         27      DATE(10) IN Proj_struct
PRENDATE         ****     COLUMN
                119
PRENDATE         116     DATE(10) COLUMN IN CORPDATA.PROJECT
PROJECT          ****     TABLE IN CORPDATA
                116
PROJECT          ****     TABLE
                117
PROJNAME         27      VARCHAR(24) IN Proj_struct
PROJNAME         ****     COLUMN
                113 120
PROJNAME         116     VARCHAR(24) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PROJNO           27      VARCHAR(6) IN Proj_struct
                85
PROJNO           ****     COLUMN
                72 76
PROJNO           74      CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJECT
PROJNO           ****     COLUMN IN EMPPROJECT
                113 117 120
PROJNO           ****     COLUMN IN PROJECT
                117
PROJNO           116     CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PRSTAFF          27      DECIMAL(5,2) IN Proj_struct
PRSTAFF          116     DECIMAL(5,2) COLUMN IN CORPDATA.PROJECT
PRSTDATE         27      DATE(10) IN Proj_struct
PRSTDATE         116     DATE(10) COLUMN IN CORPDATA.PROJECT
RESPEMP          27      VARCHAR(6) IN Proj_struct
RESPEMP          116     CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
SALARY           ****     COLUMN
                53 53 73 115
SALARY           74      DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
SEX              74      CHARACTER(1) COLUMN IN CORPDATA.EMPLOYEE
WORKDEPT         74      CHARACTER(3) COLUMN IN CORPDATA.EMPLOYEE
ソースにエラーは見つからなかった。
163 ソース・レコードが処理された。
***** リストの終わり *****
```

1 図3. SQL ステートメントを使用したサンプル C プログラム (5/5)

例: COBOL および ILE COBOL プログラム内の SQL ステートメント

注: 重要な法律的事柄に関する情報について、214 ページの『コードについての特記事項』をご覧ください。

```

5722ST1 V5R3M0 040528          SQL COBOL プログラムの作成          CBLEX          03/11/27 01:34:28   ページ 1
ソース仕様タイプ . . . . .COBOL
プログラム名 . . . . .CORPDATA/CBLEX
ソース・ファイル . . . . .CORPDATA/SRC
メンバー . . . . .CBLEX
TO ソース・ファイル.....QTEMP/QSQLTEMP
オプション . . . . .*SRC          *XREF
ターゲット・リリース . . . V5R3M0
INCLUDE ファイル . . . . .*SRCFILE
コミット . . . . .*CHG
データのコピー可能 . . . *YES
SQL カーソルのクローズ . . *ENDPGM
ブロック化可能 . . . . . *READ
PREPARE 遅延 . . . . . *NO
生成レベル . . . . . 10
印刷装置ファイル . . . . . *LIBL/QSYSVRT
日付の形式 . . . . . *JOB
日付区切り記号 . . . . . *JOB
時刻の形式 . . . . . *HMS
時刻区切り記号 . . . . . *JOB
置き換え . . . . . *YES
リレーショナル・データベース *LOCAL
ユーザー . . . . . *CURRENT
RDB 接続方式..... *DUW
省略時のコレクション . . . *NONE
動的省略時の
  コレクション . . . . . *NO
パッケージ名 . . . . . *PGLIB/*PGM
パス . . . . . *NAMING
作成オブジェクト・タイプ *PGM
SQL 規則 . . . . . *DB2
ユーザー・プロファイル . . *NAMING
動的ユーザー・プロファイル *USER
ソート順序 . . . . . *JOB
言語 ID..... *JOB
IBM SQL フラグづけ..... *NOFLAG
ANS フラグ付け..... *NONE
テキスト . . . . . *SRCMBRTXT
ソース・ファイルの CCSID. .65535
ジョブの CCSID.....65535
10 進数結果オプション
  最大精度.....31
  最大位取り.....31
  除算の最小位取り.....0
コンパイラ・オプション . *NONE
03/11/27 01:32:58 にソース・メンバーが変更された。
1
2
3 *****
4 * A sample program which updates the salaries for those *
5 * employees whose current commission total is greater than or *
6 * equal to the value of COMMISSION. The salaries of those who *
7 * qualify are increased by the value of PERCENTAGE retroactive *
8 * to RAISE-DATE. A report is generated showing the projects *
9 * which these employees have contributed to ordered by the *
10 * project number and employee ID. A second report shows each *
11 * project having an end date occurring after RAISE-DATE *
12 * (i.e. potentially affected by the retroactive raises ) with *
13 * its total salary expenses and a count of employees who *
14 * contributed to the project. *
15 *****
16
17 IDENTIFICATION DIVISION.
18
19 PROGRAM-ID. CBLEX.
20 ENVIRONMENT DIVISION.
21 CONFIGURATION SECTION.
22 SOURCE-COMPUTER. IBM-AS400.
23 OBJECT-COMPUTER. IBM-AS400.
24 INPUT-OUTPUT SECTION.
25
26 FILE-CONTROL.
27 SELECT PRINTFILE ASSIGN TO PRINTER-QPRINT
28 ORGANIZATION IS SEQUENTIAL.
29
30 DATA DIVISION.
31

```

図 4. SQL ステートメントを使用したサンプル COBOL プログラム (1/7)

```

32 FILE SECTION.
33
34 FD PRINTFILE
35 BLOCK CONTAINS 1 RECORDS
36 LABEL RECORDS ARE OMITTED.
37 01 PRINT-RECORD PIC X(132).
38
39 WORKING-STORAGE SECTION.
40 77 WORK-DAYS PIC S9(4) BINARY VALUE 253.
41 77 RAISE-DATE PIC X(11) VALUE "1982-06-01".
42 77 PERCENTAGE PIC S999V99 PACKED-DECIMAL.
43 77 COMMISSION PIC S99999V99 PACKED-DECIMAL VALUE 2000.00.
44
45 *****
46 * Structure for report 1. *
47 *****
48
49 1 01 RPT1.
50 COPY DDS-PROJECT OF CORPDATA-PROJECT.
51 05 EMPNO PIC X(6).
52 05 NAME PIC X(30).
53 05 SALARY PIC S9(6)V99 PACKED-DECIMAL.
54
55 *****
56 * Structure for report 2. *
57 *****
58
59 01 RPT2.
60 15 PROJNO PIC X(6).
61 15 PROJECT-NAME PIC X(36).
62 15 EMPLOYEE-COUNT PIC S9(4) BINARY.
63 15 TOTAL-PROJ-COST PIC S9(10)V99 PACKED-DECIMAL.
64
65
66 2 EXEC SQL
67 INCLUDE SQLCA
68 END-EXEC.
69 77 CODE-EDIT PIC ---99.
70
71 *****
72 * Headers for reports. *
73 *****
74
75 01 RPT1-HEADERS.
76 05 RPT1-HEADER1.
77 10 FILLER PIC X(21) VALUE SPACES.
78 10 FILLER PIC X(111)
79 VALUE "REPORT OF PROJECTS AFFECTED BY RAISES".
80 05 RPT1-HEADER2.
81 10 FILLER PIC X(9) VALUE "PROJECT".
82 10 FILLER PIC X(10) VALUE "EMPID".
83 10 FILLER PIC X(35) VALUE "EMPLOYEE NAME".
84 10 FILLER PIC X(40) VALUE "SALARY".
85 01 RPT2-HEADERS.
86 05 RPT2-HEADER1.
87 10 FILLER PIC X(21) VALUE SPACES.
88 10 FILLER PIC X(111)
89 VALUE "ACCUMULATED STATISTICS BY PROJECT".
90 05 RPT2-HEADER2.
91 10 FILLER PIC X(9) VALUE "PROJECT".
92 10 FILLER PIC X(38) VALUE SPACES.
93 10 FILLER PIC X(16) VALUE "NUMBER OF".
94 10 FILLER PIC X(10) VALUE "TOTAL".
95 05 RPT2-HEADER3.
96 10 FILLER PIC X(9) VALUE "NUMBER".
97 10 FILLER PIC X(38) VALUE "PROJECT NAME".
98 10 FILLER PIC X(16) VALUE "EMPLOYEES".
99 10 FILLER PIC X(65) VALUE "COST".
100 01 RPT1-DATA.
101 05 PROJNO PIC X(6).
102 05 FILLER PIC XXX VALUE SPACES.
103 05 EMPNO PIC X(6).
104 05 FILLER PIC X(4) VALUE SPACES.
105 05 NAME PIC X(30).
106 05 FILLER PIC X(3) VALUE SPACES.
107 05 SALARY PIC ZZZZ9.99.
108 05 FILLER PIC X(96) VALUE SPACES.

```

図 4. SQL ステートメントを使用したサンプル COBOL プログラム (2/7)

```

109      01 RPT2-DATA.
110         05 PROJNO PIC X(6).
111         05 FILLER PIC XXX VALUE SPACES.
112         05 PROJECT-NAME PIC X(36).
113         05 FILLER PIC X(4) VALUE SPACES.
114         05 EMPLOYEE-COUNT PIC ZZ9.
115         05 FILLER PIC X(5) VALUE SPACES.
116         05 TOTAL-PROJ-COST PIC ZZZZZZZ9.99.
117         05 FILLER PIC X(56) VALUE SPACES.
118
119      PROCEDURE DIVISION.
120
121      A000-MAIN.
122         MOVE 1.04 TO PERCENTAGE.
123         OPEN OUTPUT PRINTFILE.
124
125      *****
126      * Update the selected employees by the new percentage. If an *
127      * error occurs during the update, ROLLBACK the changes, *
128      *****
129
130      3 EXEC SQL
131         WHENEVER SQLERROR GO TO E010-UPDATE-ERROR
132      END-EXEC.
133      4 EXEC SQL
134         UPDATE CORPDATA/EMPLOYEE
135            SET SALARY = SALARY * :PERCENTAGE
136            WHERE COMM >= :COMMISSION
137      END-EXEC.
138
139      *****
140      * Commit changes. *
141      *****
142
143      5 EXEC SQL
144         COMMIT
145      END-EXEC.
146
147      EXEC SQL
148         WHENEVER SQLERROR GO TO E020-REPORT-ERROR
149      END-EXEC.
150
151      *****
152      * Report the updated statistics for each employee receiving *
153      * a raise and the projects that s/he participates in *
154      *****
155
156      *****
157      * Write out the header for Report 1. *
158      *****
159
160      write print-record from rpt1-header1
161         before advancing 2 lines.
162      write print-record from rpt1-header2
163         before advancing 1 line.
164      6 exec sql
165         declare c1 cursor for
166            SELECT DISTINCT projno, empproject.empno,
167               lastname||", "||firstnme ,salary
168            from corpdata/empproject, corpdata/employee
169            where empproject.empno =employee.empno and
170               comm >= :commission
171            order by projno, empno
172      end-exec.
173      7 EXEC SQL
174         OPEN C1
175      END-EXEC.
176
177      PERFORM B000-GENERATE-REPORT1 THRU B010-GENERATE-REPORT1-EXIT
178         UNTIL SQLCODE NOT EQUAL TO ZERO.
179

```

注: 8 および 9 はこの図の第 5 部にあります。

図 4. SQL ステートメントを使用したサンプル COBOL プログラム (3/7)

```

5722ST1 V5R3M0 040528          SQL COBOL プログラムの作成          CBLEX          03/11/27 01:34:28   ページ 4
レコード *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR 最終変更
180      10 A100-DONE1.
181          EXEC SQL
182              CLOSE C1
183          END-EXEC.
184
185          *****
186          * For all projects ending at a date later than the RAISE- *
187          * DATE ( i.e. those projects potentially affected by the *
188          * salary raises generate a report containing the project *
189          * project number, project name, the count of employees *
190          * participating in the project and the total salary cost *
191          * for the project *
192          *****
193
194
195          *****
196          * Write out the header for Report 2. *
197          *****
198
199          MOVE SPACES TO PRINT-RECORD.
200          WRITE PRINT-RECORD BEFORE ADVANCING 2 LINES.
201          WRITE PRINT-RECORD FROM RPT2-HEADER1
202              BEFORE ADVANCING 2 LINES.
203          WRITE PRINT-RECORD FROM RPT2-HEADER2
204              BEFORE ADVANCING 1 LINE.
205          WRITE PRINT-RECORD FROM RPT2-HEADER3
206              BEFORE ADVANCING 2 LINES.
207
208          EXEC SQL
209              11 DECLARE C2 CURSOR FOR
210                  SELECT EMPPROJECT.PROJNO, PROJNAME, COUNT(*),
211                      SUM ( (DAYS(EMENDATE)-DAYS(EMSTDATE)) *
212                          EMPTIME * DECIMAL((SALARY / :WORK-DAYS),8,2))
213                  FROM CORPDATA/EMPPROJECT, CORPDATA/PROJECT,
214                      CORPDATA/EMPLOYEE
215                  WHERE EMPPROJECT.PROJNO=PROJECT.PROJNO AND
216                      EMPPROJECT.EMPNO =EMPLOYEE.EMPNO AND
217                      PRENDATE > :RAISE-DATE
218                  GROUP BY EMPPROJECT.PROJNO, PROJNAME
219                  ORDER BY 1
220          END-EXEC.
221          EXEC SQL
222              OPEN C2
223          END-EXEC.
224
225          PERFORM C000-GENERATE-REPORT2 THRU C010-GENERATE-REPORT2-EXIT
226              UNTIL SQLCODE NOT EQUAL TO ZERO.
227
228          A200-DONE2.
229          EXEC SQL
230              CLOSE C2
231          END-EXEC
232
233          *****
234          * All done. *
235          *****
236
237          A900-MAIN-EXIT.
238          CLOSE PRINTFILE.
239          STOP RUN.
240

```

図4. SQL ステートメントを使用したサンプル COBOL プログラム (4/7)

```

5722ST1 V5R3M0 040528          SQL COBOL プログラムの作成          CBLEX          03/11/27 01:34:28   ページ 5
レコード *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR 最終変更
241 *****
242 * Fetch and write the rows to PRINTFILE. *
243 *****
244
245 B000-GENERATE-REPORT1.
246   8 EXEC SQL
247     WHENEVER NOT FOUND GO TO A100-DONE1
248   END-EXEC.
249   9 EXEC SQL
250     FETCH C1 INTO :PROJECT.PROJNO, :RPT1.EMPNO,
251     :RPT1.NAME, :RPT1.SALARY
252   END-EXEC.
253   MOVE CORRESPONDING RPT1 TO RPT1-DATA.
254   MOVE PROJNO OF RPT1 TO PROJNO OF RPT1-DATA.
255   WRITE PRINT-RECORD FROM RPT1-DATA
256     BEFORE ADVANCING 1 LINE.
257
258 B010-GENERATE-REPORT1-EXIT.
259   EXIT.
260
261 *****
262 * Fetch and write the rows to PRINTFILE. *
263 *****
264
265 C000-GENERATE-REPORT2.
266   EXEC SQL
267     WHENEVER NOT FOUND GO TO A200-DONE2
268   END-EXEC.
269   12 EXEC SQL
270     FETCH C2 INTO :RPT2
271   END-EXEC.
272   MOVE CORRESPONDING RPT2 TO RPT2-DATA.
273   WRITE PRINT-RECORD FROM RPT2-DATA
274     BEFORE ADVANCING 1 LINE.
275
276 C010-GENERATE-REPORT2-EXIT.
277   EXIT.
278
279 *****
280 * Error occured while updating table. Inform user and *
281 * rollback changes. *
282 *****
283
284 E010-UPDATE-ERROR.
285   13 EXEC SQL
286     WHENEVER SQLERROR CONTINUE
287   END-EXEC.
288   MOVE SQLCODE TO CODE-EDIT.
289   STRING "*** ERROR Occurred while updating table. SQLCODE="
290     CODE-EDIT DELIMITED BY SIZE INTO PRINT-RECORD.
291   WRITE PRINT-RECORD.
292   14 EXEC SQL
293     ROLLBACK
294   END-EXEC.
295   STOP RUN.
296
297 *****
298 * Error occured while generating reports. Inform user and *
299 * exit. *
300 *****
301
302 E020-REPORT-ERROR.
303   MOVE SQLCODE TO CODE-EDIT.
304   STRING "*** ERROR Occurred while generating reports. SQLCODE
305 -   =" CODE-EDIT DELIMITED BY SIZE INTO PRINT-RECORD.
306   WRITE PRINT-RECORD.
307   STOP RUN.
          * * * * * ソースの終わり * * * * *

```

図4. SQL ステートメントを使用したサンプル COBOL プログラム (5/7)

相互参照
データ名

	定義	参照
ACTNO	168	SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPPROJECT
A100-DONE1	****	LABEL 247
A200-DONE2	****	LABEL 267
BIRTHDATE	134	DATE(10) COLUMN IN CORPDATA.EMPLOYEE
BONUS	134	DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
CODE-EDIT	69	
COMM	****	COLUMN 136 170
COMM	134	DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
COMMISSION	43	DECIMAL(7,2) 136 170
CORPDATA	****	COLLECTION 134 168 168 213 213 214
C1	165	CURSOR 174 182 250
C2	209	CURSOR 222 230 270
DEPTNO	50	CHARACTER(3) IN PROJECT
DEPTNO	213	CHARACTER(3) COLUMN (NOT NULL) IN CORPDATA.PROJECT
EDLEVEL	134	SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMENDATE	168	DATE(10) COLUMN IN CORPDATA.EMPPROJECT
EMENDATE	****	COLUMN 211
EMPLOYEE	****	TABLE IN CORPDATA 134 168 214
EMPLOYEE	****	TABLE 169 216
EMPLOYEE-COUNT	63	SMALL INTEGER PRECISION(4,0) IN RPT2
EMPLOYEE-COUNT	114	IN RPT2-DATA
EMPNO	51	CHARACTER(6) IN RPT1 250
EMPNO	103	CHARACTER(6) IN RPT1-DATA
EMPNO	134	CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMPNO	****	COLUMN IN EMPPROJECT 166 169 171 216
EMPNO	****	COLUMN IN EMPLOYEE 169 216
EMPNO	168	CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJECT
EMPPROJECT	****	TABLE 166 169 210 215 216 218
EMPPROJECT	****	TABLE IN CORPDATA 168 213
EMPTIME	168	DECIMAL(5,2) COLUMN IN CORPDATA.EMPPROJECT
EMPTIME	****	COLUMN 212
EMSTDATE	168	DATE(10) COLUMN IN CORPDATA.EMPPROJECT
EMSTDATE	****	COLUMN 211
E010-UPDATE-ERROR	****	LABEL 131
E020-REPORT-ERROR	****	LABEL 148
FIRSTNME	134	VARCHAR(12) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
FIRSTNME	****	COLUMN 167
HIREDATE	134	DATE(10) COLUMN IN CORPDATA.EMPLOYEE
JOB	134	CHARACTER(8) COLUMN IN CORPDATA.EMPLOYEE
LASTNAME	134	VARCHAR(15) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
LASTNAME	****	COLUMN 167
MAJPROJ	50	CHARACTER(6) IN PROJECT
MAJPROJ	213	CHARACTER(6) COLUMN IN CORPDATA.PROJECT
MIDINIT	134	CHARACTER(1) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
NAME	52	CHARACTER(30) IN RPT1 251
NAME	105	CHARACTER(30) IN RPT1-DATA

図 4. SQL ステートメントを使用したサンプル COBOL プログラム (6/7)


```

相互参照
PERCENTAGE          42      DECIMAL(5,2)
                      135
PHONENO             134      CHARACTER(4) COLUMN IN CORPDATA.EMPLOYEE
PRENDATE           50       DATE(10) IN PROJECT
PRENDATE           ****    COLUMN
                      217
PRENDATE           213      DATE(10) COLUMN IN CORPDATA.PROJECT
PRINT-RECORD       37      CHARACTER(132)
PROJECT            50       STRUCTURE IN RPT1
PROJECT            ****    TABLE IN CORPDATA
                      213
PROJECT            ****    TABLE
                      215
PROJECT-NAME        62      CHARACTER(36) IN RPT2
PROJECT-NAME       112      CHARACTER(36) IN RPT2-DATA
PROJNAME           50       VARCHAR(24) IN PROJECT
PROJNAME           ****    COLUMN
                      210 218
PROJNAME           213      VARCHAR(24) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PROJNO             50       CHARACTER(6) IN PROJECT
                      250
PROJNO             61      CHARACTER(6) IN RPT2
PROJNO            101      CHARACTER(6) IN RPT1-DATA
PROJNO            110      CHARACTER(6) IN RPT2-DATA
PROJNO            ****    COLUMN
                      166 171
PROJNO            168      CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJECT
PROJNO            ****    COLUMN IN EMPPROJECT
                      210 215 218
PROJNO            ****    COLUMN IN PROJECT
                      215
PROJNO            213      CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PRSTAFF           50       DECIMAL(5,2) IN PROJECT
PRSTAFF           213      DECIMAL(5,2) COLUMN IN CORPDATA.PROJECT
PRSTDATE          50       DATE(10) IN PROJECT
PRSTDATE          213      DATE(10) COLUMN IN CORPDATA.PROJECT
RAISE-DATE        41      CHARACTER(11)
                      217
RESPEMP           50       CHARACTER(6) IN PROJECT
RESPEMP           213      CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
RPT1              49
RPT1-DATA         100
RPT1-HEADERS      75
RPT1-HEADER1      76      IN RPT1-HEADERS
RPT1-HEADER2      80      IN RPT1-HEADERS
RPT2              60      STRUCTURE
                      270
RPT2-DATA         109
SS REFERENCE
RPT2-HEADERS      85
RPT2-HEADER1      86      IN RPT2-HEADERS
RPT2-HEADER2      90      IN RPT2-HEADERS
RPT2-HEADER3      95      IN RPT2-HEADERS
SALARY            53      DECIMAL(8,2) IN RPT1
                      251
SALARY            107     IN RPT1-DATA
SALARY            ****    COLUMN
                      135 135 167 212
SALARY            134     DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
SEX               134     CHARACTER(1) COLUMN IN CORPDATA.EMPLOYEE
TOTAL-PROJ-COST   64      DECIMAL(12,2) IN RPT2
TOTAL-PROJ-COST   116     IN RPT2-DATA
WORK-DAYS         40      SMALL INTEGER PRECISION(4,0)
                      212
WORKDEPT          134     CHARACTER(3) COLUMN IN CORPDATA.EMPLOYEE

```

ソースにエラーは見つからなかった。

307 ソース・レコードが処理された。

***** リストの終わり *****

例: PL/I 内の SQL ステートメント

注: 重要な法律的事柄に関する情報について、214 ページの『コードについての特記事項』をご覧ください。

```

5722ST1 V5R3M0 040528          SQL PL/I プログラムの作成          PLIEX          03/11/27 21:51:19   ページ 1
ソース仕様タイプ . . . . .PLI
オブジェクト名 . . . . .CORPDATA/PLIEX
ソース・ファイル . . . . .CORPDATA/SRC
メンバー . . . . .PLIEX
TO ソース・ファイル. . . . .QTEMP/QSQLTEMP
オプション . . . . .*SRC          *XREF
ターゲット・リリース . . . V5R3M0
INCLUDE ファイル . . . . .*SRCFILE
コミット . . . . .*CHG
データのコピー可能 . . . . *YES
SQL カーソルのクローズ . . *ENDPGM
ブロック化可能 . . . . .*READ
PREPARE 遅延 . . . . .*NO
生成レベル . . . . .10
マージン . . . . .*SRCFILE
印刷装置ファイル . . . . .*LIBL/QSYSPRT
日付の形式 . . . . .*JOB
日付区切り記号 . . . . .*JOB
時刻の形式 . . . . .*HMS
時刻区切り記号 . . . . .*JOB
置き換え . . . . .*YES
リレーショナル・データベース*LOCAL
ユーザー . . . . .*CURRENT
RDB 接続方式. . . . .*DUM
省略時のコレクション . . . *NONE
動的省略時の
  コレクション . . . . .*NO
パッケージ名 . . . . .*PGMLIB/*PGM
パス . . . . .*NAMING
SQL 規則 . . . . .*DB2
ユーザー・プロファイル . . *NAMING
動的ユーザー・プロファイル*USER
ソート順序 . . . . .*JOB
言語 ID. . . . .*JOB
IBM SQL フラグづけ. . . . .*NOFLAG
ANS フラグ付け. . . . .*NONE
テキスト . . . . .*SRCMBRTXT
ソース・ファイルの CCSID. .65535
ジョブの CCSID. . . . .65535
10 進数結果オプション
  最大精度. . . . .31
  最大位取り. . . . .31
  除算の最小位取り. . . . .0
コンパイラ・オプション . . *NONE
03/11/27 21:51:00 にソース・メンバーが変更された。

1  /* A sample program which updates the salaries for those employees */          100
2  /* whose current commission total is greater than or equal to the */          200
3  /* value of COMMISSION. The salaries of those who qualify are */          300
4  /* increased by the value of PERCENTAGE, retroactive to RAISE_DATE. */          400
5  /* A report is generated showing the projects which these employees */          500
6  /* have contributed to, ordered by project number and employee ID. */          600
7  /* A second report shows each project having an end date occurring */          700
8  /* after RAISE_DATE (i.e. is potentially affected by the retroactive */          800
9  /* raises) with its total salary expenses and a count of employees */          900
10 /* who contributed to the project. */          1000
11 /****** */          1100
12                                     */          1200

```

図 5. SQL ステートメントを使用したサンプル PL/I プログラム (1/6)

```

5722ST1 V5R3M0 040528          SQL PL/I プログラムの作成          PLIEX          03/11/27 21:51:19   ページ 2
レコード *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8  SEQNBR 最終変更
13
14      PLIEX: PROC;
15
16      DCL RAISE_DATE CHAR(10);
17      DCL WORK_DAYS  FIXED BIN(15);
18      DCL COMMISSION FIXED DECIMAL(8,2);
19      DCL PERCENTAGE FIXED DECIMAL(5,2);
20
21      /* File declaration for sysprint */
22      DCL SYSPRINT FILE EXTERNAL OUTPUT STREAM PRINT;
23
24      /* Structure for report 1 */
25      DCL 1 RPT1,
26  1 %INCLUDE PROJECT (PROJECT, RECORD,,COMMA);
27      15 EMPNO      CHAR(6),
28      15 NAME       CHAR(30),
29      15 SALARY     FIXED DECIMAL(8,2);
30
31      /* Structure for report 2 */
32      DCL 1 RPT2,
33      15 PROJNO     CHAR(6),
34      15 PROJECT_NAME CHAR(36),
35      15 EMPLOYEE_COUNT FIXED BIN(15),
36      15 TOTL_PROJ_COST  FIXED DECIMAL(10,2);
37
38  2 EXEC SQL INCLUDE SQLCA;
39
40      COMMISSION = 2000.00;
41      PERCENTAGE = 1.04;
42      RAISE_DATE = '1982-06-01';
43      WORK_DAYS  = 253;
44      OPEN FILE(SYSPRINT);
45
46      /* Update the selected employee's salaries by the new percentage. */
47      /* If an error occurs during the update, ROLLBACK the changes. */
48  3 EXEC SQL WHENEVER SQLERROR GO TO UPDATE_ERROR;
49  4 EXEC SQL
50      UPDATE CORPDATA/EMPLOYEE
51      SET SALARY = SALARY * :PERCENTAGE
52      WHERE COMM >= :COMMISSION ;
53
54      /* Commit changes */
55  5 EXEC SQL
56      COMMIT;
57      EXEC SQL WHENEVER SQLERROR GO TO REPORT_ERROR;
58
59      /* Report the updated statistics for each project supported by one */
60      /* of the selected employees. */
61
62      /* Write out the header for Report 1 */
63      put file(sysprint)
64      edit('REPORT OF PROJECTS AFFECTED BY EMPLOYEE RAISES')
65      (col(22),a);
66      put file(sysprint)
67      edit('PROJECT','EMPID','EMPLOYEE NAME','SALARY')
68      (skip(2),col(1),a,col(10),a,col(20),a,col(55),a);
69
70  6 exec sql
71      declare c1 cursor for
72      select DISTINCT projno, EMPPROJACT.empno,
73      lastname||','||firstname, salary
74      from CORPDATA/EMPPROJACT, CORPDATA/EMPLOYEE
75      where EMPPROJACT.empno = EMPLOYEE.empno and
76      comm >= :COMMISSION
77      order by projno, empno;
78  7 EXEC SQL
79      OPEN C1;
80

```

図 5. SQL ステートメントを使用したサンプル PLI プログラム (2/6)

```

5722ST1 V5R3M0 040528          SQL PL/I プログラムの作成          PLIEX          03/11/27 21:51:19   ページ 3
レコード *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8  SEQNBR 最終変更
81      /* Fetch and write the rows to SYSPRINT */          8100
82      8 EXEC SQL WHENEVER NOT FOUND GO TO DONE1;          8200
83      8300
84      DO UNTIL (SQLCODE ^= 0);          8400
85      9 EXEC SQL          8500
86          FETCH C1 INTO :RPT1.PROJNO, :rpt1.EMPNO, :RPT1.NAME,          8600
87          :RPT1.SALARY;          8700
88          PUT FILE(SYSPRINT)          8800
89          EDIT(RPT1.PROJNO,RPT1.EMPNO,RPT1.NAME,RPT1.SALARY)          8900
90          (SKIP,COL(1),A,COL(10),A,COL(20),A,COL(54),F(8,2));          9000
91      END;          9100
92      9200
93      DONE1:          9300
94      10 EXEC SQL          9400
95          CLOSE C1;          9500
96      9600
97      /* For all projects ending at a date later than 'raise_date' */          9700
98      /* (i.e. those projects potentially affected by the salary raises) */          9800
99      /* generate a report containing the project number, project name */          9900
100     /* the count of employees participating in the project and the */          10000
101     /* total salary cost of the project. */          10100
102     10200
103     /* Write out the header for Report 2 */          10300
104     PUT FILE(SYSPRINT) EDIT('ACCUMULATED STATISTICS BY PROJECT')          10400
105     (SKIP(3),COL(22),A);          10500
106     PUT FILE(SYSPRINT)          10600
107     EDIT('PROJECT','NUMBER OF','TOTAL')          10700
108     (SKIP(2),COL(1),A,COL(48),A,COL(63),A);          10800
109     PUT FILE(SYSPRINT)          10900
110     EDIT('NUMBER','PROJECT NAME','EMPLOYEES','COST')          11000
111     (SKIP,COL(1),A,COL(10),A,COL(48),A,COL(63),A,SKIP);          11100
112     11200
113     11 EXEC SQL          11300
114         DECLARE C2 CURSOR FOR          11400
115             SELECT EMPPROJECT.PROJNO, PROJNAME, COUNT(*),          11500
116                 SUM( (DAYS(EMENDATE) - DAYS(EMSTDATE)) * EMPTIME *          11600
117                     DECIMAL(( SALARY / :WORK_DAYS ),8,2) )          11700
118             FROM CORPDATA/EMPPROJECT, CORPDATA/PROJECT, CORPDATA/EMPLOYEE          11800
119             WHERE EMPPROJECT.PROJNO=PROJECT.PROJNO AND          11900
120                 EMPPROJECT.EMPNO =EMPLOYEE.EMPNO AND          12000
121                 PRENDATE > :RAISE_DATE          12100
122             GROUP BY EMPPROJECT.PROJNO, PROJNAME          12200
123             ORDER BY 1;          12300
124     EXEC SQL          12400
125         OPEN C2;          12500
126     12600
127     /* Fetch and write the rows to SYSPRINT */          12700
128     EXEC SQL WHENEVER NOT FOUND GO TO DONE2;          12800
129     12900
130     DO UNTIL (SQLCODE ^= 0);          13000
131     12 EXEC SQL          13100
132         FETCH C2 INTO :RPT2;          13200
133         PUT FILE(SYSPRINT)          13300
134         EDIT(RPT2.PROJNO,RPT2.PROJECT_NAME,EMPLOYEE_COUNT,          13400
135             TOTL_PROJ_COST)          13500
136         (SKIP,COL(1),A,COL(10),A,COL(50),F(4),COL(62),F(8,2));          13600
137     END;          13700
138     13800
139     DONE2:          13900
140     EXEC SQL          14000
141         CLOSE C2;          14100
142     GO TO FINISHED;          14200
143     14300
144     /* Error occured while updating table. Inform user and rollback */          14400
145     /* changes. */          14500
146     UPDATE_ERROR:          14600
147     13 EXEC SQL WHENEVER SQLERROR CONTINUE;          14700
148     PUT FILE(SYSPRINT) EDIT('*** ERROR Occurred while updating table.'||          14800
149     ' SQLCODE=' ,SQLCODE)(A,F(5));          14900

```

図 5. SQL ステートメントを使用したサンプル PL/I プログラム (3/6)

```

5722ST1 V5R3M0 040528          SQL PL/I プログラムの作成          PLIEX          03/11/27 21:51:19   ページ 4
150 14 EXEC SQL                  15000
151     ROLLBACK;                15100
152     GO TO FINISHED;          15200
153                               15300
154     /* Error occured while generating reports. Inform user and exit. */ 15400
155 REPORT_ERROR:                15500
156     PUT FILE(SYSPPRINT) EDIT('*** ERROR Occurred while generating '||
157     'reports. SQLCODE=',SQLCODE)(A,F(5)); 15600
158     GO TO FINISHED;          15700
159                               15800
160     /* All done */            15900
161 FINISHED:                     16000
162     CLOSE FILE(SYSPPRINT);    16100
163     RETURN;                   16200
164                               16300
165 END PLIEX;                    16400
                                   16500
                                   ***** ソースの終わり *****

```

図 5. SQL ステートメントを使用したサンプル PL/I プログラム (4/6)

相互参照
データ名

	定義	参照
ACTNO	74	SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPPROJECT
BIRTHDATE	74	DATE(10) COLUMN IN CORPDATA.EMPLOYEE
BONUS	74	DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
COMM	****	COLUMN 52 76
COMM	74	DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
COMMISSION	18	DECIMAL(8,2) 52 76
CORPDATA	****	COLLECTION 50 74 74 118 118 118
C1	71	CURSOR 79 86 95
C2	114	CURSOR 125 132 141
DEPTNO	26	CHARACTER(3) IN RPT1
DEPTNO	118	CHARACTER(3) COLUMN (NOT NULL) IN CORPDATA.PROJECT
DONE1	****	LABEL 82
DONE2	****	LABEL 128
EDLEVEL	74	SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMENDATE	74	DATE(10) COLUMN IN CORPDATA.EMPPROJECT
EMENDATE	****	COLUMN 116
EMPLOYEE	****	TABLE IN CORPDATA 50 74 118
EMPLOYEE	****	TABLE 75 120
EMPLOYEE_COUNT	35	SMALL INTEGER PRECISION(4,0) IN RPT2
EMPNO	27	CHARACTER(6) IN RPT1 86
EMPNO	****	COLUMN IN EMPPROJECT 72 75 77 120
EMPNO	****	COLUMN IN EMPLOYEE 75 120
EMPNO	74	CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJECT
EMPNO	74	CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMPPROJECT	****	TABLE 72 75 115 119 120 122
EMPPROJECT	****	TABLE IN CORPDATA 74 118
EMPTIME	74	DECIMAL(5,2) COLUMN IN CORPDATA.EMPPROJECT
EMPTIME	****	COLUMN 116
EMSTDATE	74	DATE(10) COLUMN IN CORPDATA.EMPPROJECT
EMSTDATE	****	COLUMN 116
FIRSTNME	****	COLUMN 73
FIRSTNME	74	VARCHAR(12) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
HIREDATE	74	DATE(10) COLUMN IN CORPDATA.EMPLOYEE
JOB	74	CHARACTER(8) COLUMN IN CORPDATA.EMPLOYEE
LASTNAME	****	COLUMN 73
LASTNAME	74	VARCHAR(15) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
MAJPROJ	26	CHARACTER(6) IN RPT1
MAJPROJ	118	CHARACTER(6) COLUMN IN CORPDATA.PROJECT
MIDINIT	74	CHARACTER(1) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
NAME	28	CHARACTER(30) IN RPT1 86
PERCENTAGE	19	DECIMAL(5,2) 51
PHONENO	74	CHARACTER(4) COLUMN IN CORPDATA.EMPLOYEE

図 5. SQL ステートメントを使用したサンプル PL/I プログラム (5/6)

```

相互参照
PRENDATE          26      DATE(10) IN RPT1
PRENDATE          ****   COLUMN
                   121
PRENDATE          118     DATE(10) COLUMN IN CORPDATA.PROJECT
PROJECT           ****   TABLE IN CORPDATA
                   118
PROJECT           ****   TABLE
                   119
PROJECT_NAME      34      CHARACTER(36) IN RPT2
PROJNAME          26      VARCHAR(24) IN RPT1
PROJNAME          ****   COLUMN
                   115 122
PROJNAME          118     VARCHAR(24) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PROJNO           26      CHARACTER(6) IN RPT1
PROJNO           ****   COLUMN
                   86
PROJNO           33      CHARACTER(6) IN RPT2
PROJNO           ****   COLUMN
                   72 77
PROJNO           74      CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJECT
PROJNO           ****   COLUMN IN EMPPROJECT
                   115 119 122
PROJNO           ****   COLUMN IN PROJECT
                   119
PROJNO           118     CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PRSTAFF          26      DECIMAL(5,2) IN RPT1
PRSTAFF          118     DECIMAL(5,2) COLUMN IN CORPDATA.PROJECT
PRSTDATE         26      DATE(10) IN RPT1
PRSTDATE         118     DATE(10) COLUMN IN CORPDATA.PROJECT
RAISE_DATE       16      CHARACTER(10)
                   121
REPORT_ERROR     ****   LABEL
                   57
RESPEMP          26      CHARACTER(6) IN RPT1
RESPEMP          118     CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
RPT1             25      STRUCTURE
RPT2             32      STRUCTURE
                   132
SALARY           29      DECIMAL(8,2) IN RPT1
                   87
SALARY           ****   COLUMN
                   51 51 73 117
SALARY           74      DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
SEX              74      CHARACTER(1) COLUMN IN CORPDATA.EMPLOYEE
SYSPRINT         22
TOTL_PROJ_COST   36      DECIMAL(10,2) IN RPT2
UPDATE_ERROR     ****   LABEL
                   48
WORK_DAYS        17      SMALL INTEGER PRECISION(4,0)
                   117
WORKDEPT         74      CHARACTER(3) COLUMN IN CORPDATA.EMPLOYEE
ソースにエラーは見つからなかった。
165 ソース・レコードが処理された。
***** リストの終わり *****

```

I 図 5. SQL ステートメントを使用したサンプル PLI プログラム (6/6)

例: RPG for iSeries プログラム内の SQL ステートメント

注: 重要な法律的事柄に関する情報について、214 ページの『コードについての特記事項』をご覧ください。

```

ソース仕様タイプ . . . . .RPG
オブジェクト名 . . . . .CORPDATA/RPGEX
ソース・ファイル . . . . .CORPDATA/SRC
メンバー . . . . .RPGEX
TO ソース・ファイル . . . . .QTEMP/QSQLTEMP
オプション . . . . .*SRC *XREF
ターゲット・リリース . . . . .V5R3M0
INCLUDE ファイル . . . . .*SRCFILE
コミット . . . . .*CHG
データのコピー可能 . . . . .*YES
SQL カーソルのクローズ . . . . .*ENDPGM
ブロック化可能 . . . . .*READ
PREPARE 遅延 . . . . .*NO
生成レベル . . . . .10
マージン . . . . .*SRCFILE
印刷装置ファイル . . . . .*LIBL/QSYSVRT
日付の形式 . . . . .*JOB
日付区切り記号 . . . . .*JOB
時刻の形式 . . . . .*HMS
時刻区切り記号 . . . . .*JOB
置き換え . . . . .*YES
リレーショナル・データベース*LOCAL
ユーザー . . . . .*CURRENT
RDB 接続方式 . . . . .*DUM
省略時のコレクション . . . . .*NONE
動的省略時の
  コレクション . . . . .*NO
パッケージ名 . . . . .*PGMLIB/*PGM
パス . . . . .*NAMING
SQL 規則 . . . . .*DB2
ユーザー・プロファイル . . . . .*NAMING
動的ユーザー・プロファイル*USER
ソート順序 . . . . .*JOB
言語 ID . . . . .*JOB
IBM SQL フラグづけ . . . . .*NOFLAG
ANS フラグ付け . . . . .*NONE
テキスト . . . . .*SRCMBRTXT
ソース・ファイルの CCSID . . . . .65535
ジョブの CCSID . . . . .65535
10 進数結果オプション
  最大精度 . . . . .31
  最大位取り . . . . .31
  除算の最小位取り . . . . .0
コンパイラ・オプション . . . . .*NONE

```

03/11/27 22:21:08 にソース・メンバーが変更された。

```

1      H                               100
2      F* File declaration for QPRINT 200
3      F*                               300
4      FQPRINT 0 F 132          PRINTER 400
5      I*                               500
6      I* Structure for report 1.      600
7      I*                               700
8      1 IRPT1 E DSPROJECT          800
9      I          PROJNAME          PROJNM 900
10     I          RESPEMP          RESEM   1000
11     I          PRSTAFF          STAFF   1100
12     I          PRSTDATE          PRSTD   1200
13     I          PRENDATE          PREND   1300
14     I          MAJPROJ          MAJPRJ  1400
15     I*                               1500
16     I          DS                1600
17     I          1 6 EMPNO          1700
18     I          7 36 NAME          1800
19     I          P 37 412SALARY     1900
20     I*                               2000
21     I* Structure for report 2.      2100
22     I*                               2200
23     IRPT2 DS                2300
24     I          1 6 PRJNUM          2400
25     I          7 42 PNAME          2500
26     I          B 43 440EMPCNT     2600
27     I          P 45 492PRCOST     2700
28     I*                               2800
29     I          DS                2900
30     I          B 1 20WRKDAY        3000
31     I          P 3 62COMMI         3100
32     I          7 16 RDATE          3200
33     I          P 17 202PERCNT     3300

```

図 6. SQL ステートメントを使用したサンプル RPG for iSeries プログラム (1/6)


```

5722ST1 V5R3M0 040528          SQL RPG プログラムの作成          RPGEX          03/11/27 22:21:52   ページ 2
レコード *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8  SEQNBR 最終変更
34      2 C*                               3400
35      C                               Z-ADD253      WRKDAY      3500
36      C                               Z-ADD2000.00  COMMI      3600
37      C                               Z-ADD1.04    PERCNT     3700
38      C                               MOVE'1982-06-'RDATE  3800
39      C                               MOVE '01'    RDATE      3900
40      C                               SETON              LR      3901
41      C*                               4000
42      C* Update the selected projects by the new percentage. If an  4100
43      C* error occurs during the update, ROLLBACK the changes.    4200
44      C*                               4300
45      3 C/EXEC SQL WHENEVER SQLERROR GOTO UPDERR                    4400
46      C/END-EXEC                                                    4500
47      C*                               4600
48      4 C/EXEC SQL                                                    4700
49      C+ UPDATE CORPDATA/EMPLOYEE                                    4800
50      C+   SET SALARY = SALARY * :PERCNT                            4900
51      C+   WHERE COMM >= :COMMI                                     5000
52      C/END-EXEC                                                    5100
53      C*                               5200
54      C* Commit changes.                                           5300
55      C*                               5400
56      5 C/EXEC SQL COMMIT                                           5500
57      C/END-EXEC                                                    5600
58      C*                               5700
59      C/EXEC SQL WHENEVER SQLERROR GO TO RPTERR                    5800
60      C/END-EXEC                                                    5900
61      C*                               6000
62      C* Report the updated statistics for each employee assigned to  6100
63      C* selected projects.                                         6200
64      C*                               6300
65      C* Write out the header for report 1.                         6400
66      C*                               6500
67      C                               EXCPTRCA                       6600
68      6 C/EXEC SQL DECLARE C1 CURSOR FOR                             6700
69      C+   SELECT DISTINCT PROJNO, EMPPROJECT.EMPNO,                6800
70      C+   LASTNAME||', '||FIRSTNAME, SALARY                       6900
71      C+   FROM CORPDATA/EMPPROJECT, CORPDATA/EMPLOYEE             7000
72      C+   WHERE EMPPROJECT.EMPNO = EMPLOYEE.EMPNO AND              7100
73      C+   COMM >= :COMMI                                           7200
74      C+   ORDER BY PROJNO, EMPNO                                   7300
75      C/END-EXEC                                                    7400
76      C*                               7500
77      7 C/EXEC SQL                                                    7600
78      C+ OPEN C1                                                    7700
79      C/END-EXEC                                                    7800
80      C*                               7900
81      C* Fetch and write the rows to QPRINT.                        8000
82      C*                               8100
83      8 C/EXEC SQL WHENEVER NOT FOUND GO TO DONE1                    8200
84      C/END-EXEC                                                    8300
85      C                               SQLCOD      DOUNEO          8400
86      C/EXEC SQL                                                    8500
87      9 C+ FETCH C1 INTO :PROJNO, :EMPNO, :NAME, :SALARY            8600
88      C/END-EXEC                                                    8700
89      C                               EXCPTRCBB                       8800
90      C                               END                          8900
91      C                               DONE1      TAG              9000
92      C/EXEC SQL                                                    9100
93      10 C+ CLOSE C1                                                 9200
94      C/END-EXEC                                                    9300
95      C*                               9400
96      C* For all project ending at a date later than the raise date  9500
97      C* (i.e. those projects potentially affected by the salary raises) 9600
98      C* generate a report containing the project number, project name, 9700
99      C* the count of employees participating in the project and the   9800
100     C* total salary cost of the project.                           9900
101     C*                               10000
102     C* Write out the header for report 2.                          10100
103     C*                               10200
104     C                               EXCPTRCCE                      10300

```

図 6. SQL ステートメントを使用したサンプル RPG for iSeries プログラム (2/6)

```

5722ST1 V5R3M0 040528          SQL RPG プログラムの作成          RPGEX          03/11/27 22:21:52   ページ 3
レコード *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR 最終変更
105 11 C/EXEC SQL                               10400
106 C+ DECLARE C2 CURSOR FOR                    10500
107 C+ SELECT EMPPROJACT.PROJNO, PROJNAME, COUNT(*), 10600
108 C+ SUM((DAYS(EMENDATE) - DAYS(EMSTDATE)) * EMPTIME * 10700
109 C+ DECIMAL((SALARY/:WRKDAY),8,2))            10800
110 C+ FROM CORPDATA/EMPPROJACT, CORPDATA/PROJECT, CORPDATA/EMPLOYEE 10900
111 C+ WHERE EMPPROJACT.PROJNO = PROJECT.PROJNO AND 11000
112 C+ EMPPROJACT.EMPNO = EMPLOYEE.EMPNO AND    11100
113 C+ PRENDATE > :RDATE                        11200
114 C+ GROUP BY EMPPROJACT.PROJNO, PROJNAME      11300
115 C+ ORDER BY 1                               11400
116 C/END-EXEC                                  11500
117 C*                                           11600
118 C/EXEC SQL OPEN C2                         11700
119 C/END-EXEC                                  11800
120 C*                                           11900
121 C* Fetch and write the rows to QPRINT.      12000
122 C*                                           12100
123 C/EXEC SQL WHENEVER NOT FOUND GO TO DONE2   12200
124 C/END-EXEC                                  12300
125 C SQLCOD DOUNE0                            12400
126 C/EXEC SQL                                  12500
127 12 C+ FETCH C2 INTO :RPT2                  12600
128 C/END-EXEC                                  12700
129 C EXCPTRECD                                12800
130 C END                                       12900
131 C DONE2 TAG                                13000
132 C/EXEC SQL CLOSE C2                        13100
133 C/END-EXEC                                  13200
134 C RETRN                                     13300
135 C*                                           13400
136 C* Error ocured while updating table. Inform user and rollback 13500
137 C* changes.                                 13600
138 C*                                           13700
139 C UPDERR TAG                                13800
140 C EXCPTRECE                                13900
141 13 C/EXEC SQL WHENEVER SQLERROR CONTINUE   14000
142 C/END-EXEC                                  14100
143 C*                                           14200
144 14 C/EXEC SQL                               14300
145 C+ ROLLBACK                                14400
146 C/END-EXEC                                  14500
147 C RETRN                                     14600
148 C*                                           14700
149 C* Error ocured while generating reports. Inform user and exit. 14800
150 C*                                           14900
151 C RPTERR TAG                                15000
152 C EXCPTRECF                                15100
153 C*                                           15200
154 C* All done.                               15300
155 C*                                           15400
156 C FINISH TAG                                15500
157 OQPRINT E 0201 RECA                        15700
158 0 45 'REPORT OF PROJECTS AFFEC'            15800
159 0 64 'TED BY EMPLOYEE RAISES'            15900
160 0 E 01 RECA                                16000
161 0 7 'PROJECT'                              16100
162 0 17 'EMPLOYEE'                            16200
163 0 32 'EMPLOYEE NAME'                      16300
164 0 60 'SALARY'                              16400
165 0 E 01 RECB                                16500
166 0 PROJNO 6                                16600
167 0 EMPNO 15                                16700
168 0 NAME 50                                 16800
169 0 SALARYL 61                              16900
170 0 E 22 RECC                                17000
171 0 42 'ACCUMULATED STATISTIC'              17100
172 0 54 'S BY PROJECT'                      17200
173 0 E 01 RECC                                17300
174 0 7 'PROJECT'                              17400
175 0 56 'NUMBER OF'                          17500
176 0 67 'TOTAL'                              17600
177 0 E 02 RECC                                17700
178 0 6 'NUMBER'                              17800
179 0 21 'PROJECT NAME'                      17900
180 0 56 'EMPLOYEEES'                        18000
181 0 66 'COST'                              18100

```

図 6. SQL ステートメントを使用したサンプル RPG for iSeries プログラム (3/6)

```

5722ST1 V5R3M0 040528      SQL RPG プログラムの作成      RPGEX      03/11/27 22:21:52   ページ 4
182      0      E 01      RECD      57 'CODE='      18200
183      0      PRJNUM      6      18300
184      0      PNAME      45      18400
185      0      EMPCNTL     54      18500
186      0      PRCOSTL     70      18600
187      0      E 01      RECE      28 '*** ERROR Occurred while' 18700
188      0      52 ' updating table. SQLCODE' 18800
189      0      53 '='      18900
190      0      SQLCODL     62      19000
191      0      RECF      28 '*** ERROR Occurred while' 19100
192      0      E 01      RECF      52 ' generating reports. SQL' 19200
193      0      53 '='      19300
194      0      SQLCODL     67      19400
195      0      SQLCODL     67      19600
      * * * * * ソースの終わり * * * * *

```

図 6. SQL ステートメントを使用したサンプル RPG for iSeries プログラム (4/6)

相互参照

データ名

	定義	参照
ACTNO	68	SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPPROJECT
BIRTHDATE	48	DATE(10) COLUMN IN CORPDATA.EMPLOYEE
BONUS	48	DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
COMM	****	COLUMN 48 68
COMM	48	DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
COMMI	31	DECIMAL(7,2) 48 68
CORPDATA	****	COLLECTION 48 68 68 105 105 105
C1	68	CURSOR 77 86 92
C2	105	CURSOR 118 126 132
DEPTNO	8	CHARACTER(3) IN RPT1
DEPTNO	105	CHARACTER(3) COLUMN (NOT NULL) IN CORPDATA.PROJECT
DONE1	91	LABEL 83
DONE2	131	LABEL 123
EDLEVEL	48	SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMENDATE	68	DATE(10) COLUMN IN CORPDATA.EMPPROJECT
EMENDATE	****	COLUMN 105
EMPCNT	26	SMALL INTEGER PRECISION(4,0) IN RPT2
EMPLOYEE	****	TABLE IN CORPDATA 48 68 105
EMPLOYEE	****	TABLE 68 105
EMPNO	17	CHARACTER(6) 86
EMPNO	48	CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMPNO	****	COLUMN IN EMPPROJECT 68 68 68 105
EMPNO	****	COLUMN IN EMPLOYEE 68 105
EMPNO	68	CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJECT
EMPPROJECT	****	TABLE 68 68 105 105 105 105
EMPPROJECT	****	TABLE IN CORPDATA 68 105
EMPTIME	68	DECIMAL(5,2) COLUMN IN CORPDATA.EMPPROJECT
EMPTIME	****	COLUMN 105
EMSTDATE	68	DATE(10) COLUMN IN CORPDATA.EMPPROJECT
EMSTDATE	****	COLUMN 105
FINISH	156	LABEL
FIRSTNME	48	VARCHAR(12) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
FIRSTNME	****	COLUMN 68
HIREDATE	48	DATE(10) COLUMN IN CORPDATA.EMPLOYEE
JOB	48	CHARACTER(8) COLUMN IN CORPDATA.EMPLOYEE
LASTNAME	48	VARCHAR(15) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
LASTNAME	****	COLUMN 68
MAJPRJ	8	CHARACTER(6) IN RPT1
MAJPROJ	105	CHARACTER(6) COLUMN IN CORPDATA.PROJECT
MIDINIT	48	CHARACTER(1) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
NAME	18	CHARACTER(30) 86
PERCNT	33	DECIMAL(7,2) 48
PHONENO	48	CHARACTER(4) COLUMN IN CORPDATA.EMPLOYEE
PNAME	25	CHARACTER(36) IN RPT2
PRCOST	27	DECIMAL(9,2) IN RPT2
PREND	8	DATE(10) IN RPT1
PRENDATE	****	COLUMN 105
PRENDATE	105	DATE(10) COLUMN IN CORPDATA.PROJECT
PRJNUM	24	CHARACTER(6) IN RPT2

図 6. SQL ステートメントを使用したサンプル RPG for iSeries プログラム (5/6)

```

PROJECT          ****  TABLE IN CORPDATA
                  105
PROJECT          ****  TABLE
                  105
PROJNAME         ****  COLUMN
                  105 105
PROJNAME         105  VARCHAR(24) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PROJNM           8    VARCHAR(24) IN RPT1
PROJNO           8    CHARACTER(6) IN RPT1
                  86
PROJNO          ****  COLUMN
                  68 68
PROJNO           68  CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJECT
PROJNO          ****  COLUMN IN EMPPROJECT
                  105 105 105
PROJNO          ****  COLUMN IN PROJECT
                  105
PROJNO           105 CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PRSTAFF          105 DECIMAL(5,2) COLUMN IN CORPDATA.PROJECT
PRSTD            8    DATE(10) IN RPT1
PRSDATE          105 DATE(10) COLUMN IN CORPDATA.PROJECT
RDATE           32  CHARACTER(10)
                  105
RESEM            8    CHARACTER(6) IN RPT1
RESPEMP          105 CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
RPTERR          151 LABEL
                  59
RPT1             8    STRUCTURE
RPT2            23  STRUCTURE
                  126
SALARY           19  DECIMAL(9,2)
                  86
SALARY          ****  COLUMN
                  48 48 68 105
SALARY           48  DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
SEX              48  CHARACTER(1) COLUMN IN CORPDATA.EMPLOYEE
STAFF            8    DECIMAL(5,2) IN RPT1
UPDERR          139 LABEL
                  45
WORKDEPT         48  CHARACTER(3) COLUMN IN CORPDATA.EMPLOYEE
WRKDAY           30  SMALL INTEGER PRECISION(4,0)
                  105

```

ソースにエラーは見つからなかった。
196 ソース・レコードは処理された。

***** リストの終わり *****

1 図 6. SQL ステートメントを使用したサンプル RPG for iSeries プログラム (6/6)

例: ILE RPG for iSeries プログラム内の SQL ステートメント

注: 重要な法律的事柄に関する情報について、214 ページの『コードについての特記事項』をご覧ください。

```

ソース仕様タイプ . . . . .RPG
オブジェクト名 . . . . .CORPDATA/RPGLEEX
ソース・ファイル . . . . .CORPDATA/SRC
メンバー . . . . .RPGLEEX
TO ソース・ファイル . . . . .QTEMP/QSQLTEMP1
オプション . . . . .*XREF
RPGプリプロセッサ・オプション . *NONE
プログラムの印刷 . . . . .*PRINT
ターゲット・リリース . . . V5R3M0
INCLUDE ファイル . . . . .*SRCFILE
コミット . . . . .*CHG
データのコピー可能 . . . . *YES
SQL カーソルのクローズ . . *ENDMOD
ブロック化可能 . . . . .*READ
PREPARE 遅延 . . . . .*NO
生成レベル . . . . .10
マージン . . . . .*SRCFILE
印刷装置ファイル . . . . .*LIBL/QSYSPRT
日付の形式 . . . . .*JOB
日付区切り記号 . . . . .*JOB
時刻の形式 . . . . .*HMS
時刻区切り記号 . . . . .*JOB
置き換え . . . . .*YES
リレーショナル・データベース *LOCAL
ユーザー . . . . .*CURRENT
RDB 接続方式 . . . . .*DUW
省略時のコレクション . . . *NONE
動的省略時の
  コレクション . . . . .*NO
パッケージ名 . . . . .*OBJLIB/*OBJ
パス . . . . .*NAMING
SQL 規則 . . . . .*DB2
作成オブジェクト・タイプ . *PGM
デバッグ・ビュー . . . . .*NONE
ユーザー・プロファイル . . *NAMING
動的ユーザー・プロファイル *USER
ソート順序 . . . . .*JOB
言語 ID . . . . .*JOB
IBM SQL フラグづけ . . . . .*NOFLAG
ANS フラグ付け . . . . .*NONE
テキスト . . . . .*SRCMBRTXT
ソース・ファイルの CCSID . . 65535
ジョブの CCSID . . . . .65535
10 進数結果オプション
  最大精度 . . . . .31
  最大位取り . . . . .31
  除算の最小位取り . . . . .0
コンパイラ・オプション . *NONE
03/11/27 23:10:12 にソース・メンバーが変更された。

```

```

1      H                                     100
2      F* File declaration for QPRINT        200
3      F*                                     300
4      FQPRINT    0    F 132      PRINTER    400
5      D*                                     500
6      D* Structure for report 1.           600
7      D*                                     700
8      1 DRPT1          E DS                EXTNAME(PROJECT) 800
9      D*                                     900
10     D              DS                    1000
11     D EMPNO          1      6            1100
12     D NAME           7      36           1200
13     D SALARY         37     41P 2        1300
14     D*                                     1400
15     D* Structure for report 2.           1500
16     D*                                     1600
17     DRPT2           DS                    1700
18     D PRJNUM         1      6            1800
19     D PNAME          7      42           1900
20     D EMPCNT         43     44B 0        2000
21     D PRCOST         45     49P 2        2100
22     D*                                     2200
23     D              DS                    2300
24     D WRKDAY         1      2B 0         2400
25     D COMMI          3      6P 2         2500
26     D RDATE          7      16          2600
27     D PERCNT        17     20P 2         2700
28     *                                     2800

```

図 7. SQL ステートメントを使用したサンプル ILE RPG for iSeries プログラム (1/6)

```

5722ST1 V5R3M0 040528          SQL ILE RPG オブジェクトの作成          RPGLEEX          03/11/27 23:10:56   ページ 2
レコード *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8  SEQNBR 最終変更   注記
29      2 C          Z-ADD      253          WRKDAY          2900
30      C          Z-ADD      2000.00        COMMI             3000
31      C          Z-ADD      1.04          PERCNT           3100
32      C          MOVE      '1982-06-'      RDATE            3200
33      C          MOVE      '01'          RDATE            3300
34      C          SETON                               LR             3400
35      C*                               3500
36      C* Update the selected projects by the new percentage. If an  3600
37      C* error occurs during the update, ROLLBACK the changes.     3700
38      C*                               3800
39      3 C/EXEC SQL WHENEVER SQLERROR GOTO UPDERR          3900
40      C/END-EXEC                                         4000
41      C*                                               4100
42      C/EXEC SQL                                         4200
43      4 C+ UPDATE CORPDATA/EMPLOYEE                      4300
44      C+   SET SALARY = SALARY * :PERCNT                4400
45      C+   WHERE COMM >= :COMMI                        4500
46      C/END-EXEC                                         4600
47      C*                                               4700
48      C* Commit changes.                                4800
49      C*                                               4900
50      5 C/EXEC SQL COMMIT                                5000
51      C/END-EXEC                                         5100
52      C*                                               5200
53      C/EXEC SQL WHENEVER SQLERROR GO TO RPTERR         5300
54      C/END-EXEC                                         5400
55      C*                                               5500
56      C* Report the updated statistics for each employee assigned to  5600
57      C* selected projects.                              5700
58      C*                                               5800
59      C* Write out the header for report 1.              5900
60      C*                                               6000
61      C          EXCEPT RECA                          6100
62      6 C/EXEC SQL DECLARE C1 CURSOR FOR                 6200
63      C+   SELECT DISTINCT PROJNO, EMPPROJACT.EMPNO,     6300
64      C+   LASTNAME||', '||FIRSTNME, SALARY             6400
65      C+   FROM CORPDATA/EMPPROJACT, CORPDATA/EMPLOYEE  6500
66      C+   WHERE EMPPROJACT.EMPNO = EMPLOYEE.EMPNO AND  6600
67      C+   COMM >= :COMMI                                6700
68      C+   ORDER BY PROJNO, EMPNO                       6800
69      C/END-EXEC                                         6900
70      C*                                               7000
71      7 C/EXEC SQL                                       7100
72      C+ OPEN C1                                         7200
73      C/END-EXEC                                         7300
74      C*                                               7400
75      C* Fetch and write the rows to QPRINT.            7500
76      C*                                               7600
77      8 C/EXEC SQL WHENEVER NOT FOUND GO TO DONE1       7700
78      C/END-EXEC                                         7800
79      C          SQLCOD      DOUNE      0              7900
80      C/EXEC SQL                                         8000
81      9 C+ FETCH C1 INTO :PROJNO, :EMPNO, :NAME, :SALARY  8100
82      C/END-EXEC                                         8200
83      C          EXCEPT RECB                          8300
84      C          END                                       8400
85      C          DONE1      TAG                          8500
86      C/EXEC SQL                                         8600
87      10 C+ CLOSE C1                                     8700
88      C/END-EXEC                                         8800
89      C*                                               8900
90      C* For all project ending at a date later than the raise date  9000
91      C* (i.e. those projects potentially affected by the salary raises) 9100
92      C* generate a report containing the project number, project name,  9200
93      C* the count of employees participating in the project and the  9300
94      C* total salary cost of the project.                9400
95      C*                                               9500
96      C* Write out the header for report 2.              9600
97      C*                                               9700
98      C          EXCEPT RECC                          9800
99      C/EXEC SQL                                         9900

```

12000

図 7. SQL ステートメントを使用したサンプル ILE RPG for iSeries プログラム (2/6)

```

5722ST1 V5R3M0 040528      SQL ILE RPG オブジェクトの作成      RPGLEEX      03/11/27 23:10:56 ページ 3
レコード *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8  SEQNBR 最終変更 注記
100 11 C+ DECLARE C2 CURSOR FOR 10000
101 C+ SELECT EMPPROJACT.PROJNO, PROJNAME, COUNT(*), 10100
102 C+ SUM((DAYS(EMENDATE) - DAYS(EMSTDATE)) * EMPTIME * 10200
103 C+ DECIMAL((SALARY/:WRKDAY),8,2)) 10300
104 C+ FROM CORPDATA/EMPPROJACT, CORPDATA/PROJECT, CORPDATA/EMPLOYEE 10400
105 C+ WHERE EMPPROJACT.PROJNO = PROJECT.PROJNO AND 10500
106 C+ EMPPROJACT.EMPNO = EMPLOYEE.EMPNO AND 10600
107 C+ PRENDATE > :RDATE 10700
108 C+ GROUP BY EMPPROJACT.PROJNO, PROJNAME 10800
109 C+ ORDER BY 1 10900
110 C/END-EXEC 11000
111 C* 11100
112 C/EXEC SQL OPEN C2 11200
113 C/END-EXEC 11300
114 C* 11400
115 C* Fetch and write the rows to QPRINT. 11500
116 C* 11600
117 C/EXEC SQL WHENEVER NOT FOUND GO TO DONE2 11700
118 C/END-EXEC 11800
119 C SQLCOD DOUNE 0 11900
120 C/EXEC SQL
121 12 C+ FETCH C2 INTO :RPT2 12100
122 C/END-EXEC 12200
123 C EXCEPT RECD 12300
124 C END 12400
125 C DONE2 TAG 12500
126 C/EXEC SQL CLOSE C2 12600
127 C/END-EXEC 12700
128 C RETURN 12800
129 C* 12900
130 C* Error occured while updating table. Inform user and rollback 13000
131 C* changes. 13100
132 C* 13200
133 C UPDERR TAG 13300
134 C EXCEPT RECE 13400
135 13 C/EXEC SQL WHENEVER SQLERROR CONTINUE 13500
136 C/END-EXEC 13600
137 C* 13700
138 14 C/EXEC SQL 13800
139 C+ ROLLBACK 13900
140 C/END-EXEC 14000
141 C RETURN 14100
142 C* 14200
143 C* Error occured while generating reports. Inform user and exit. 14300
144 C* 14400
145 C RPTERR TAG 14500
146 C EXCEPT RECF 14600
147 C* 14700
148 C* All done. 14800
149 C* 14900
150 C FINISH TAG 15000
151 OQPRINT E RECA 0 2 01 15100
152 O 42 'REPORT OF PROJECTS AFFEC' 15200
153 O 64 'TED BY EMPLOYEE RAISES' 15300
154 O E RECA 0 1 15400
155 O 7 'PROJECT' 15500
156 O 17 'EMPLOYEE' 15600
157 O 32 'EMPLOYEE NAME' 15700
158 O 60 'SALARY' 15800
159 O E RECB 0 1 15900
160 O PROJNO 6 16000
161 O EMPNO 15 16100
162 O NAME 50 16200
163 O SALARY L 61 16300
164 O E RECC 2 2 16400
165 O 42 'ACCUMULATED STATISTIC' 16500
166 O 54 'S BY PROJECT' 16600

```

図7. SQL ステートメントを使用したサンプル ILE RPG for iSeries プログラム (3/6)


```

5722ST1 V5R3M0 040528      SQL ILE RPG オブジェクトの作成      RPGLEEX      03/11/27 23:10:56   ページ 4
167      0      E      RECC      0 1      16700
168      0      7 'PROJECT'      16800
169      0      56 'NUMBER OF'      16900
170      0      67 'TOTAL'      17000
171      0      E      RECC      0 2      17100
172      0      6 'NUMBER'      17200
173      0      21 'PROJECT NAME'      17300
174      0      56 'EMPLOYEES'      17400
175      0      66 'COST'      17500
176      0      E      RECD      0 1      17600
177      0      PRJNUM      6      17700
178      0      PNAME      45      17800
179      0      EMPCNT      L 54      17900
180      0      PRCOST      L 70      18000
181      0      E      RECE      0 1      18100
182      0      28 '*** ERROR Occurred while'      18200
183      0      52 ' updating table. SQLCODE'      18300
184      0      53 '='      18400
185      0      SQLCOD      L 62      18500
186      0      E      RECF      0 1      18600
187      0      28 '*** ERROR Occurred while'      18700
188      0      52 ' generating reports. SQL'      18800
189      0      57 'CODE='      18900
190      0      SQLCOD      L 67      19000
***** ソースの終わり *****

```

図 7. SQL ステートメントを使用したサンプル ILE RPG for iSeries プログラム (4/6)

相互参照
データ名

データ名	定義	参照
ACTNO	62	SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPPROJECT
BIRTHDATE	42	DATE(10) COLUMN IN CORPDATA.EMPLOYEE
BONUS	42	DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
COMM	****	COLUMN
		42 62
COMM	42	DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
COMMI	25	DECIMAL(7,2)
		42 62
CORPDATA	****	COLLECTION
		42 62 62 99 99 99
C1	62	CURSOR
		71 80 86
C2	99	CURSOR
		112 120 126
DEPTNO	8	CHARACTER(3) IN RPT1
DEPTNO	99	CHARACTER(3) COLUMN (NOT NULL) IN CORPDATA.PROJECT
DONE1	85	
DONE1	****	LABEL
		77
DONE2	125	
DONE2	****	LABEL
		117
EDLEVEL	42	SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMENDATE	62	DATE(10) COLUMN IN CORPDATA.EMPPROJECT
EMENDATE	****	COLUMN
		99
EMPCNT	20	SMALL INTEGER PRECISION(4,0) IN RPT2
EMPLOYEE	****	TABLE IN CORPDATA
		42 62 99
EMPLOYEE	****	TABLE
		62 99
EMPNO	11	CHARACTER(6) DBCS-open
		80
EMPNO	42	CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMPNO	****	COLUMN IN EMPPROJECT
		62 62 62 99
EMPNO	****	COLUMN IN EMPLOYEE
		62 99
EMPNO	62	CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJECT
EMPPROJECT	****	TABLE
		62 62 99 99 99 99
EMPPROJECT	****	TABLE IN CORPDATA
		62 99
EMPTIME	62	DECIMAL(5,2) COLUMN IN CORPDATA.EMPPROJECT
EMPTIME	****	COLUMN
		99
EMSTDATE	62	DATE(10) COLUMN IN CORPDATA.EMPPROJECT
EMSTDATE	****	COLUMN
		99
FINISH	150	
FIRSTNME	42	VARCHAR(12) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
FIRSTNME	****	COLUMN
		62
HIREDATE	42	DATE(10) COLUMN IN CORPDATA.EMPLOYEE
JOB	42	CHARACTER(8) COLUMN IN CORPDATA.EMPLOYEE
LASTNAME	42	VARCHAR(15) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
LASTNAME	****	COLUMN
		62
MAJPROJ	8	CHARACTER(6) IN RPT1
MAJPROJ	99	CHARACTER(6) COLUMN IN CORPDATA.PROJECT
MIDINIT	42	CHARACTER(1) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
NAME	12	CHARACTER(30) DBCS-open
		80
PERCNT	27	DECIMAL(7,2)
		42
PHONENO	42	CHARACTER(4) COLUMN IN CORPDATA.EMPLOYEE
PNAME	19	CHARACTER(36) DBCS-open IN RPT2
PRCOST	21	DECIMAL(9,2) IN RPT2
PRENDATE	8	DATE(8) IN RPT1
PRENDATE	****	COLUMN
		99
PRENDATE	99	DATE(10) COLUMN IN CORPDATA.PROJECT
PRJNUM	18	CHARACTER(6) DBCS-open IN RPT2

図 7. SQL ステートメントを使用したサンプル ILE RPG for iSeries プログラム (5/6)

```

相互参照
PROJECT          ****  TABLE IN CORPDATA
                  99
PROJECT          ****  TABLE
                  99
PROJNAME         8      VARCHAR(24) IN RPT1
PROJNAME         ****  COLUMN
                  99 99
PROJNAME         99     VARCHAR(24) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PROJNO           8      CHARACTER(6) IN RPT1
                  80
PROJNO           ****  COLUMN
                  62 62
PROJNO           62     CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPPROJECT
PROJNO           ****  COLUMN IN EMPPROJECT
                  99 99 99
PROJNO           ****  COLUMN IN PROJECT
                  99
PROJNO           99     CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
PRSTAFF          8      DECIMAL(5,2) IN RPT1
PRSTAFF          99     DECIMAL(5,2) COLUMN IN CORPDATA.PROJECT
PRSTDATE         8      DATE(8) IN RPT1
PRSTDATE         99     DATE(10) COLUMN IN CORPDATA.PROJECT
RDATE            26     CHARACTER(10) DBCS-open
                  99
RESPEMP          8      CHARACTER(6) IN RPT1
RESPEMP          99     CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.PROJECT
RPTERR           145
RPTERR           ****  LABEL
                  53
RPT1             8      STRUCTURE
RPT2             17     STRUCTURE
                  120
SALARY           13     DECIMAL(9,2)
                  80
SALARY           ****  COLUMN
                  42 42 62 99
SALARY           42     DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
SEX              42     CHARACTER(1) COLUMN IN CORPDATA.EMPLOYEE
UPDERR           133
UPDERR           ****  LABEL
                  39
WORKDEPT         42     CHARACTER(3) COLUMN IN CORPDATA.EMPLOYEE
WRKDAY           24     SMALL INTEGER PRECISION(4,0)
                  99

```

ソースにエラーは見つからなかった。

190 ソース・レコードが処理された。

***** リストの終わり *****

I 図 7. SQL ステートメントを使用したサンプル ILE RPG for iSeries プログラム (6/6)

例: REXX プログラム内の SQL ステートメント

注: 重要な法律的事柄に関する情報について、214 ページの『コードについての特記事項』をご覧ください。

```

レコード *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
1  /*****
2  /* A sample program which updates the salaries for those employees */
3  /* whose current commission total is greater than or equal to the */
4  /* value of COMMISSION. The salaries of those who qualify are */
5  /* increased by the value of PERCENTAGE, retroactive to RAISE_DATE. */
6  /* A report is generated and dumped to the display which shows the */
7  /* projects which these employees have contributed to, ordered by */
8  /* project number and employee ID. A second report shows each */
9  /* project having an end date occurring after RAISE_DATE (i.e. is */
10 /* potentially affected by the retroactive raises) with its total */
11 /* salary expenses and a count of employees who contributed to the */
12 /* project. */
13 /*****
14
15
16 /* Initialize RC variable */
17 RC = 0
18
19 /* Initialize HV for program usage */
20 COMMISSION = 2000.00;
21 PERCENTAGE = 1.04;
22 RAISE_DATE = '1982-06-01';
23 WORK_DAYS = 253;
24
25 /* Create the output file to dump the 2 reports. Perform an OVRDBF */
26 /* to allow us to use the SAY REXX command to write to the output */
27 /* file. */
28 ADDRESS '*COMMAND',
29 'DLTF FILE(CORPDATA/REPORTFILE)'
30 ADDRESS '*COMMAND',
31 'CRTPF FILE(CORPDATA/REPORTFILE) RCDLEN(80)'
32 ADDRESS '*COMMAND',
33 'OVRDBF FILE(STDOUT) TOFILE(CORPDATA/REPORTFILE) MBR(REPORTFILE)'
34
35 /* Update the selected employee's salaries by the new percentage. */
36 /* If an error occurs during the update, ROLLBACK the changes. */
37 3 SIGNAL ON ERROR
38 ERRLOC = 'UPDATE_ERROR'
39 UPDATE_STMT = 'UPDATE CORPDATA/EMPLOYEE ',
40 'SET SALARY = SALARY * ? ',
41 'WHERE COMM >= ? '
42 EXECSQL,
43 'PREPARE S1 FROM :UPDATE_STMT'
44 4 EXECSQL,
45 'EXECUTE S1 USING :PERCENTAGE,',
46 ':COMMISSION '
47 /* Commit changes */
48 5 EXECSQL,
49 'COMMIT'
50 ERRLOC = 'REPORT_ERROR'
51
52 /* Report the updated statistics for each project supported by one */
53 /* of the selected employees. */
54
55 /* Write out the header for Report 1 */
56 SAY ' '
57 SAY ' '
58 SAY ' '
59 SAY ' REPORT OF PROJECTS AFFECTED BY EMPLOYEE RAISES'
60 SAY ' '
61 SAY 'PROJECT EMPID EMPLOYEE NAME SALARY'
62 SAY '-----'
63 SAY ' '
64
65 SELECT_STMT = 'SELECT DISTINCT PROJNO, EMPPROJECT.EMPNO, ',
66 'LASTNAME||', '||FIRSTNAME, SALARY ',
67 'FROM CORPDATA/EMPPROJECT, CORPDATA/EMPLOYEE ',
68 'WHERE EMPPROJECT.EMPNO = EMPLOYEE.EMPNO AND ',
69 'COMM >= ? ',
70 'ORDER BY PROJNO, EMPNO '
71 EXECSQL,
72 'PREPARE S2 FROM :SELECT_STMT'
73 6 EXECSQL,
74 'DECLARE C1 CURSOR FOR S2'

```

図 8. SQL ステートメントを使用したサンプル REXX プロシージャ (1/3)

```

レコード *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
75 7 EXECSQL,
76 'OPEN C1 USING :COMMISSION'
77
78 /* Handle the FETCH errors and warnings inline */
79 SIGNAL OFF ERROR
80
81 /* Fetch all of the rows */
82 DO UNTIL (SQLCODE <> 0)
83 9 EXECSQL,
84 'FETCH C1 INTO :RPT1.PROJNO, :RPT1.EMPNO,',
85 ' :RPT1.NAME, :RPT1.SALARY '
86
87 /* Process any errors that may have occurred. Continue so that */
88 /* we close the cursor for any warnings. */
89 IF SQLCODE < 0 THEN
90 SIGNAL ERROR
91
92 /* Stop the loop when we hit the EOF. Don't try to print out the */
93 /* fetched values. */
94 8 IF SQLCODE = 100 THEN
95 LEAVE
96
97 /* Print out the fetched row */
98 SAY RPT1.PROJNO ' ' RPT1.EMPNO ' ' RPT1.NAME ' ' RPT1.SALARY
99 END;
100
101 10 EXECSQL,
102 'CLOSE C1'
103
..+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
104 /* For all projects ending at a date later than 'raise_date' */
105 /* (i.e. those projects potentially affected by the salary raises) */
106 /* generate a report containing the project number, project name */
107 /* the count of employees participating in the project and the */
108 /* total salary cost of the project. */
109
110 /* Write out the header for Report 2 */
111 SAY ' '
112 SAY ' '
113 SAY ' '
114 SAY ' ACCUMULATED STATISTICS BY PROJECT'
115 SAY ' '
116 SAY 'PROJECT PROJECT NAME NUMBER OF TOTAL'
117 SAY 'NUMBER EMPLOYEES COST'
118 SAY '-----'
119 SAY ' '
120
121
122 /* Go to the common error handler */
123 SIGNAL ON ERROR
124
125 SELECT_STMT = 'SELECT EMPPROJECT.PROJNO, PROJNAME, COUNT(*), ',
126 ' SUM( (DAYS(EMENDATE) - DAYS(EMSTDATE)) * EMPTIME * ',
127 ' DECIMAL(( SALARY / ? ),8,2) ) ',
128 'FROM CORPDATA/EMPPROJECT, CORPDATA/PROJECT, CORPDATA/EMPLOYEE',
129 'WHERE EMPPROJECT.PROJNO = PROJECT.PROJNO AND ',
130 ' EMPPROJECT.EMPNO = EMPLOYEE.EMPNO AND ',
131 ' PRENDATE > ? ',
132 'GROUP BY EMPPROJECT.PROJNO, PROJNAME ',
133 'ORDER BY 1 '
134 EXECSQL,
135 'PREPARE S3 FROM :SELECT_STMT'
136 11 EXECSQL,
137 'DECLARE C2 CURSOR FOR S3'
138 EXECSQL,
139 'OPEN C2 USING :WORK_DAYS, :RAISE_DATE'
140
141 /* Handle the FETCH errors and warnings inline */
142 SIGNAL OFF ERROR
143
144 /* Fetch all of the rows */
145 DO UNTIL (SQLCODE <> 0)

```

図 8. SQL ステートメントを使用したサンプル REXX プロシージャ (2/3)

```

レコード *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
146      12 EXECSQL,
147          'FETCH C2 INTO :RPT2.PROJNO, :RPT2.PROJNAME, ',
148          '          :RPT2.EMPCOUNT, :RPT2.TOTAL_COST '
149
150      /* Process any errors that may have occurred. Continue so that */
151      /* we close the cursor for any warnings. */
152      IF SQLCODE < 0 THEN
153          SIGNAL ERROR
154
155      /* Stop the loop when we hit the EOF. Don't try to print out the */
156      /* fetched values. */
157      IF SQLCODE = 100 THEN
158          LEAVE
159
160      /* Print out the fetched row */
161      SAY RPT2.PROJNO ' ' RPT2.PROJNAME ' ',
162          RPT2.EMPCOUNT ' ' RPT2.TOTAL_COST
163 END;
164
165 EXECSQL,
166     'CLOSE C2'
167
168 /* Delete the OVRDBF so that we will continue writing to the output */
169 /* display. */
170 ADDRESS '*COMMAND',
171     'DLTOVR FILE(STDOUT)'
172
173 /* Leave procedure with a successful or warning RC */
174 EXIT RC
175
176
177 /* Error occurred while updating the table or generating the */
178 /* reports. If the error occurred on the UPDATE, rollback all of */
179 /* the changes. If it occurred on the report generation, display the */
180 /* REXX RC variable and the SQLCODE and exit the procedure. */
181 ERROR:
182
183     13 SIGNAL OFF ERROR
184
185 /* Determine the error location */
186 SELECT
187     /* When the error occurred on the UPDATE statement */
188     WHEN ERRLOC = 'UPDATE_ERROR' THEN
189         DO
190             SAY '*** ERROR Occurred while updating table.',
191                 'SQLCODE = ' SQLCODE
192             14 EXECSQL,
193                 'ROLLBACK'
194             END
195         /* When the error occurred during the report generation */
196         WHEN ERRLOC = 'REPORT_ERROR' THEN
197             SAY '*** ERROR Occurred while generating reports. ',
198                 'SQLCODE = ' SQLCODE
199         OTHERWISE
200             SAY '*** Application procedure logic error occurred '
201         END
202     END
203
204 /* Delete the OVRDBF so that we will continue writing to the */
205 /* output display. */
206 ADDRESS '*COMMAND',
207     'DLTOVR FILE(STDOUT)'
208
209 /* Return the error RC received from SQL. */
210 EXIT RC
211
                * * * * * ソースの終わり * * * * *

```

SQL を使用したサンプル・プログラムにより作成される報告書

前記の各サンプル・プログラムにより作成される報告書は次のとおりです。

REPORT OF PROJECTS AFFECTED BY RAISES

PROJECT	EMPID	EMPLOYEE NAME	SALARY
AD3100	000010	HAAS, CHRISTINE	54860.00
AD3110	000070	PULASKI, EVA	37616.80
AD3111	000240	MARINO, SALVATORE	29910.40
AD3113	000270	PEREZ, MARIA	28475.20
IF1000	000030	KWAN, SALLY	39780.00
IF1000	000140	NICHOLLS, HEATHER	29556.80
IF2000	000030	KWAN, SALLY	39780.00
IF2000	000140	NICHOLLS, HEATHER	29556.80
MA2100	000010	HAAS, CHRISTINE	54860.00
MA2100	000110	LUCCHESSE, VICENZO	48360.00
MA2110	000010	HAAS, CHRISTINE	54860.00
MA2111	000200	BROWN, DAVID	28849.60
MA2111	000220	LUTZ, JENNIFER	31033.60
MA2112	000150	ADAMSON, BRUCE	26291.20
OP1000	000050	GEYER, JOHN	41782.00
OP1010	000090	HENDERSON, EILEEN	30940.00
OP1010	000280	SCHNEIDER, ETHEL	27300.00
OP2010	000050	GEYER, JOHN	41782.00
OP2010	000100	SPENSER, THEODORE	27196.00
OP2012	000330	LEE, WING	26384.80
PL2100	000020	THOMPSON, MICHAEL	42900.00

ACCUMULATED STATISTICS BY PROJECT

PROJECT NUMBER	PROJECT NAME	NUMBER OF EMPLOYEES	TOTAL COST
AD3100	ADMIN SERVICES	1	19623.11
AD3110	GENERAL ADMIN SYSTEMS	1	58877.28
AD3111	PAYROLL PROGRAMMING	7	66407.56
AD3112	PERSONNEL PROGRAMMING	9	28845.70
AD3113	ACCOUNT PROGRAMMING	14	72114.52
IF1000	QUERY SERVICES	4	35178.99
IF2000	USER EDUCATION	5	55212.61
MA2100	WELD LINE AUTOMATION	2	114001.52
MA2110	W L PROGRAMMING	1	85864.68
MA2111	W L PROGRAM DESIGN	3	93729.24
MA2112	W L ROBOT DESIGN	6	166945.84
MA2113	W L PROD CONT PROGS	5	71509.11
OP1000	OPERATION SUPPORT	1	16348.86
OP1010	OPERATION	5	167828.76
OP2010	SYSTEMS SUPPORT	2	91612.62
OP2011	SCP SYSTEMS SUPPORT	2	31224.60
OP2012	APPLICATIONS SUPPORT	2	41294.88
OP2013	DB/DC SUPPORT	2	37311.12
PL2100	WELD LINE PLANNING	1	43576.92

第 13 章 ホスト言語プリコンパイラー用の DB2 UDB for iSeries CL コマンドの説明

DB2 UDB for iSeries は、以下のプログラム言語でコード化されたプログラムのプリコンパイル・コマンドを提供します。

- COBOL
- ILE COBOL
- ILE C
- C++
- PL/I
- RPG
- ILE RPG

CRTSQLCBL (SQL COBOL プログラム作成) コマンド

SQL COBOL プログラム作成 (CRTSQLCBL) コマンドは構造化照会言語 (SQL) プリコンパイラーを呼び出すもので、このプリコンパイラーが SQL ステートメントを含む COBOL ソース・プログラムをプリコンパイルし、一時ソース・メンバーを作成してから、任意で COBOL コンパイラーを呼び出してプログラムのコンパイルを行います。

このコマンドの詳細については、「CL」に関するトピックの『CRTSQLCBL コマンド』を参照してください。

CRTSQLCBLI (SQL ILE COBOL オブジェクト作成) コマンド

SQL ILE COBOL オブジェクト作成 (CRTSQLCBLI) コマンドは構造化照会言語 (SQL) プリコンパイラーを呼び出すもので、このプリコンパイラーが SQL ステートメントを含む COBOL ソース・プログラムをプリコンパイルし、一時ソース・メンバーを作成してから、任意選択で ILE COBOL コンパイラーを呼び出してモジュール、プログラム、またはサービス・プログラムの作成を行います。

このコマンドの詳細については、「CL」に関するトピックの『CRTSQLCBLI コマンド』を参照してください。

CRTSQLCI (SQL ILE C オブジェクト作成) コマンド

SQL ILE C オブジェクト作成 (CRTSQLCI) コマンドは構造化照会言語 (SQL) プリコンパイラーを呼び出すもので、このプリコンパイラーが SQL ステートメントを含む C ソース・プログラムをプリコンパイルし、一時ソース・メンバーを作成してから、任意で ILE C コンパイラーを呼び出してモジュールの作成、プログラムの作成、またはサービス・プログラムの作成を行います。

このコマンドの詳細については、「CL」に関するトピックの『CRTSQLCI コマンド』を参照してください。

CRTSQLCPPI (SQL C++ オブジェクト作成) コマンド

SQL C++ オブジェクト作成 (CRTSQLCPPI) コマンドは、構造化照会言語 (SQL) プリコンパイラーを呼び出すものです。この SQL プリコンパイラーは、SQL ステートメントを含む C++ ソースをプリコンパイルし、一時ソース・メンバーを作成し、それから任意で C++ コンパイラーを呼び出してモジュールを作成します。

このコマンドの詳細については、「CL」に関するトピックの『CRTSQLCPPI コマンド』を参照してください。

CRTSQLPLI (SQL PL/I 作成) コマンド

SQL PL/I 作成 (CRTSQLPLI) コマンドは構造化照会言語 (SQL) プリコンパイラーを呼び出すもので、このプリコンパイラーが SQL ステートメントを含む PL/I ソース・プログラムをプリコンパイルし、一時ソース・メンバーを作成してから、任意選択で PL/I コンパイラーを呼び出してプログラムのコンパイルを行います。

このコマンドの詳細については、「CL」に関するトピックの『CRTSQLPLI コマンド』を参照してください。

CRTSQLRPG (SQL RPG 作成) コマンド

SQL RPG 作成 (CRTSQLRPG) コマンドは構造化照会言語 (SQL) プリコンパイラーを呼び出すもので、このプリコンパイラーが SQL ステートメントを含む RPG ソース・プログラムをプリコンパイルし、一時ソース・メンバーを作成してから、任意選択で RPG コンパイラーを呼び出してプログラムのコンパイルを行います。

このコマンドの詳細については、「CL」に関するトピックの『CRTSQLRPG コマンド』を参照してください。

CRTSQLRPGI (SQL ILE RPG オブジェクト作成) コマンド

SQL ILE RPG オブジェクト作成 (CRTSQLRPGI) コマンドは構造化照会言語 (SQL) プリコンパイラーを呼び出すもので、このプリコンパイラーが SQL ステートメントを含む RPG ソース・プログラムをプリコンパイルし、一時ソース・メンバーを作成してから、任意選択で ILE RPG コンパイラーを呼び出してモジュールの作成、プログラムの作成、またはサービス・プログラムの作成を行います。

このコマンドの詳細については、「CL」に関するトピックの『CRTSQLRPGI コマンド』を参照してください。

付録. 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

- | 〒106-0032
- | 東京都港区六本木 3-2-31
- | IBM World Trade Asia Corporation
- | Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態を提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

- | IBM は、お客様が提供するいかなる情報も、お客様に対してなら義務も負うことのない、自ら適切と信
- | ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

- | IBM Corporation
- | Software Interoperability Coordinator, Department 49XA
- | 3605 Highway 52 N
- | Rochester, MN 55901
- | U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性があります。その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確認できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者をお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があります、単に目標を示しているものです。

表示されている IBM の価格は IBM が小売り価格として提示しているもので、現行価格であり、通知なしに変更されるものです。卸価格は、異なる場合があります。

本書はプランニング目的としてのみ記述されています。記述内容は製品が使用可能になる前に変更になる場合があります。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほめかしたり、保証することはできません。お客様は、IBM のアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. _年を入れる_. All rights reserved.

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

プログラミング・インターフェース情報

- | 本書には、プログラムを作成するユーザーが DB2 UDB for iSeries のサービスを使用するためのプログラミング・インターフェースが記述されています。

商標

以下は、IBM Corporation の商標です。

DB2
IBM
iSeries
COBOL/400
Distributed Relational Database Architecture
DRDA
Language Environment
RPG/400

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

資料に関するご使用条件

お客様がダウンロードされる資料につきましては、以下の条件にお客様が同意されることを条件にその使用が認められます。

個人使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商業上の使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態を提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。

これらの資料の著作権はすべて、IBM Corporation に帰属しています。

お客様が、このサイトから資料をダウンロードまたは印刷することにより、これらの条件に同意されたものとさせていただきます。

コードについての特記事項

本書には、プログラミングの例が含まれています。

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作使用权を許諾します。お客様は、このサンプル・コードから、お客様独自の特別のニーズに合わせた類似のプログラムを作成することができます。

すべてのサンプル・コードは、例として示す目的でのみ、IBM により提供されます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

ここに含まれるすべてのプログラムは、現存するままの状態を提供され、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。



Printed in Japan