SAS® 9.4 SQL プロシジャ ユーザーガイド

第 2 版

SAS® ドキュメント

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2013. *SAS® 9.4 SQL Procedure User's Guide, Second Edition*. Cary, NC: SAS Institute Inc.

SAS® 9.4 SQL Procedure User's Guide, Second Edition

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513-2414.

August 2014

目次

本書について . . . . . *vii*
SAS 9.4 SQL プロシジャの新機能 . . . . . *xiii*

1 部 SQL プロシジャの使用 1

1 章 • SQL プロシジャについて **3**
SQL について 3
SQL プロシジャについて 3
用語 4
PROC SQL と SAS DATA ステップとの比較 5
サンプルテーブルの注記 7

2 章 • 1 つのテーブルからのデータの取得 **19**
SELECT ステートメントの概要 20
テーブルの列の選択 22
列の新規作成 27
データの並べ替え 37
条件を満たす行の取得 44
データの要約 56
データのグループ化 64
グループ化されたデータのフィルタリング 69
クエリの検証 71

3 章 • 複数のテーブルからのデータの取得 **73**
はじめに 73
結合を使用し、複数のテーブルからデータを選択する 74
サブクエリを使用したデータの選択 96
結合とサブクエリの使用が必要な場合 102
セット演算子を使用したクエリの組み合わせ 103

4 章 • テーブルとビューの作成および更新 **111**
はじめに 112
テーブルの作成 112
テーブルへの行の挿入 116
テーブルのデータ値の更新 120
行の削除 123
列の変更 123
インデックスの作成 127
テーブルの削除 128
SAS ソフトウェアでの SQL プロシジャテーブルの使用 128
テーブルの一貫性制約の作成および使用 129
PROC SQL ビューの作成および使用 131

5 章 • SQL プロシジャを使用したプログラミング **139**
はじめに 140
PROC SQL オプションを使用し、クエリを作成、デバックする 140
クエリパフォーマンスの向上 144

列エイリアスの使用 . . . 148
DICTIONARY テーブルを使用し、SAS System の情報にアクセスする . . . 151
PROC SQL で SAS データセットオプションを使用する . . . 157
PROC SQL を SAS マクロ機能とともに使用する . . . 158
REPORT プロシジャを使用し、PROC SQL 出力をフォーマットする . . . 166
SAS/ACCESS を使用した DBMS へのアクセス . . . 168
PROC SQL で ODS (Output Delivery System)を使用する . . . 175

6 章 • PROC SQL を使用した問題の解決 . . . **177**
概要 . . . 178
重み付き平均の計算 . . . 178
テーブルの比較 . . . 180
欠損データ値の重ね合わせ . . . 182
小計内の百分率の計算 . . . 185
テーブルの重複する行のカウント . . . 187
テーブルの階層データの展開 . . . 189
複数列のデータの要約 . . . 192
要約レポートの作成 . . . 194
カスタマイズされた並べ替え順序の作成 . . . 197
テーブルの条件付き更新 . . . 200
別のテーブルの値を使用してテーブルを更新する . . . 203
マクロ変数の作成および使用 . . . 205
他の SAS プロシジャでの PROC SQL テーブルの使用 . . . 208

2 部 SQL プロシジャリファレンス 211

7 章 • SQL プロシジャ . . . **213**
概要 . . . 214
構文: SQL プロシジャ . . . 217
例: SQL プロシジャ . . . 267

8 章 • SQL プロシジャの構成要素 . . . **311**
概要 . . . 311
ディクショナリ . . . 312

3 部 付録 367

付録 1 • SQL マクロ変数とシステムオプション . . . **369**
ディクショナリ . . . 369

付録 2 • PROC SQL および ANSI 規格 . . . **383**

付録 3 • 「SQL プロシジャの使用」で示されているコード例 . . . **387**

付録 4 • 「SQL プロシジャリファレンス」で示されている例のデータセット . . . **433**
概要 . . . 433
Employees . . . 434
Houses . . . 434
Match_11 . . . 434
Proclib.Delay . . . 436
Proclib.Houses . . . 437

Proclib.March 437
Proclib.Paylist2 438
Proclib.Payroll 439
Proclib.Payroll2 442
Proclib.Schedule2 442
Proclib.Staff 442
Proclib.Staff2 445
Proclib.Superv2 446
Stores 446
Survey 446

推奨資料 **447**
用語集 **449**
キーワード **453**

本書について

SAS 言語の構文規則

SAS 言語の構文規則の概要

SAS では、SAS 言語要素の構文ドキュメントに共通の規則を使用しています。これらの規則により、SAS 構文の構成要素を簡単に識別できます。規則は、次の項目に分類されます。

- 構文の構成要素
- スタイル規則
- 特殊文字
- SAS ライブラリと外部ファイルの参照

構文のコンポーネント

言語要素の多くでは、その構文の構成要素はキーワードと引数から構成されます。キーワードのみ必要な言語要素もあります。また、キーワードに等号(=)が続く言語要素もあります。複数の引数を含む構文で区切り記号を使用する場合と使用しない場合を説明するために、引数の構文の形式が複数示されています。

キーワード
プログラムの作成ときに使用する SAS 言語要素名です。キーワードはリテラルであり、通常、構文の先頭の単語です。CALL ルーチンでは、最初の 2 つの単語がキーワードです。

これらの例の SAS 構文では、キーワードには太字が使用されています。

CHAR (*string, position*)

CALL RANBIN (*seed, n, p, x*);

ALTER (*alter-password*)

BEST *w.*

REMOVE <*data-set-name*>

この例では、CALL ルーチンの最初の 2 つの単語がキーワードです。

CALL RANBIN(*seed*, *n*, *p*, *x*)

引数なしで 1 つのキーワードから構成される SAS ステートメント構文もあります。

DO;
... SAS code ...

END;

2 つのキーワード値のいずれか 1 つの指定が必要なシステムオプションもあります。

DUPLEX | **NODUPLEX**

プロシジャステートメントによっては、ステートメント構文中に複数のキーワードが含まれます。

CREATE <UNIQUE> **INDEX** *index-name* **ON** *table-name* (*column-1* <, *column-2*, …>)

引数

数値定数、文字定数、変数、式のいずれかです。引数は、キーワードに続くか、キーワードの後ろの等号に続きます。SAS では、引数を使用して、言語要素を処理します。引数が必須の場合もオプションの場合もあります。構文では、オプションの引数は山かっこ(< >)で囲まれます。

この例では、*string* と *position* がキーワード CHAR に続きます。これらの引数は、CHAR 関数の必須引数です。

CHAR (*string, position*)

引数ごとに値が指定されます。この例の SAS コードでは、引数 *string* の値は 'summer'、引数 *position* の値は 4 です。

```
x=char('summer', 4);
```

この例では、*string* および *substring* は必須引数ですが、*modifiers* と *startpos* はオプションです。

FIND(*string, substring* <,*modifiers*> <,*startpos*>

argument(s)

引数は必ず 1 つ必要であり、複数の引数が許可されます。引数の間はスペースで区切ります。カンマ(,)などの区切り記号は、引数間に必要ありません。

たとえば、MISSING ステートメントは、この形式で複数の引数を含みます。

MISSING *character(s)*;

<LITERAL_ARGUMENT> *argument-1* <<LITERAL_ARGUMENT> *argument-2 ...* >

引数は必ず 1 つ必要であり、リテラル引数がこの引数に関連付けられます。リテラルと引数のペアは複数指定できます。リテラルと引数の間に区切り記号は必要ありません。省略記号(...)は、追加のリテラルと引数が許可されることを示します。

たとえば、BY ステートメントはこの引数を含みます。

BY <DESCENDING> *variable-1* <<DESCENDING> *variable-2* …>;

argument-1 <option(s)> <argument-2 <option(s)> ...>

引数は必ず 1 つ必要であり、1 つ以上のオプションがこの引数に関連付けられます。複数の引数と関連するオプションを指定できます。引数とオプションの間に区切り記号は必要ありません。省略記号(...)は、追加の引数と関連するオプションが許可されることを示します。

たとえば、FORMAT プロシジャの PICTURE ステートメントは、この形式で複数の引数を含みます。

PICTURE name <(*format-option(s)*)>
<*value-range-set-1* <(*picture-1-option(s)*)>
<*value-range-set-2* <(*picture-2-option(s)*)> …>>;

argument-1=value-1 <argument-2=value-2 ...>
引数には値を割り当てる必要があり、複数の引数を指定できます。省略記号(...)は、追加の引数が許可されることを示します。引数間に区切り記号は必要ありません。

たとえば、LABEL ステートメントは、この形式で複数の引数を含みます。

LABEL *variable-1=label-1 <variable-2=label-2 …>*;

argument-1 <, argument-2, ...>
引数は必ず 1 つ必要であり、カンマまたは別の区切り記号で区切って複数の引数を指定できます。省略記号(...)は、カンマで区切られた引数が続くことを示します。SAS ドキュメントでは両方の形式が使用されます。

次に、この形式で指定された複数の引数の例を示します。

AUTHPROVIDERDOMAIN (*provider-1:domain-1 <, provider-2:domain-2, …>*

INTO :*macro-variable-specification-1 <, :macro-variable-specification-2, …>*

注: 通常、SAS ドキュメントのサンプルコードは、小文字の固定幅フォントを使用して表記されます。コードの作成には、大文字も、小文字も、大文字と小文字の両方も使用できます。

書体に関する規則

SAS 構文の説明に使用されるスタイル規則には、大文字太字、大文字、斜体の規則も含まれます。

大文字太字
関数名やステートメント名などの SAS キーワードを示します。この例では、キーワード ERROR の表記には大文字太字が使用されています。

ERROR <*message*>;

大文字
リテラルの引数を示します。

この CMPMODEL=システムオプションの例では、BOTH、CATALOG、XML がリテラルです。

CMPMODEL=BOTH | CATALOG | XML |

斜体
ユーザー指定の引数または値を示します。斜体表記の項目は、ユーザー指定値であり、次のいずれかを表します。

- 非リテラル引数。この LINK ステートメントの例では、引数 *label* はユーザー指定値のため、斜体で表示されます。

  LINK *label*;

- 引数に割り当てられる非リテラル値。

  この FORMAT ステートメントの例では、引数 DEFAULT に変数の *default-format* が割り当てられます。

  FORMAT *variable(s)* <*format* > <DEFAULT = *default-format*>;

特殊文字

SAS 言語要素の構文には、次の特殊文字も使用されます。

=
等号は、一部の言語要素(システムオプションなど)のリテラル値を示します。

この MAPS システムオプションの例では、等号により MAPS の値が設定されます。

MAPS = *location-of-maps*

< >
山かっこはオプションの引数を示します。必須引数は山かっこで囲みません。

この CAT 関数の例では、少なくとも項目が 1 つ必要です。

CAT (*item-1* <, *item-2*, …>)

|
縦棒は、値グループから 1 つの値を選択できることを示します。縦棒で区切られている値は、相互排他です。

この CMPMODEL=システムオプションの例では、引数を 1 つのみ選択できます。

CMPMODEL=BOTH | CATALOG | XML

...
省略記号は、引数の繰り返しが可能なことを示します。引数と省略記号が山かっこで囲まれている場合、その引数はオプションです。繰り返される引数には、その引数の前や後ろに、区切り記号を入れる必要があります。

この CAT 関数の例では、複数の *item* 引数が許可され、カンマで区切る必要があります。

CAT (*item-1* <, *item-2*, …>)

'*value*'または"*value*"
一重引用符や二重引用符付きの引数は、その値にも一重引用符または二重引用符を付ける必要があることを示します。

この FOOTNOTE ステートメントの例では、引数 *text* に引用符が付けられています。

FOOTNOTE <*n*> <*ods-format-options* '*text*' | "*text*">;

;
セミコロンは、ステートメントまたは CALL ルーチンの終わりを示します。

この例では、各ステートメントがセミコロンで終了しています。

```
data namegame;
length color name $8;
color = 'black';
name = 'jack';
game = trim(color) || name;
run;
```

SAS ライブラリと外部ファイルへの参照

多くの SAS ステートメントなどの言語要素では、SAS ライブラリと外部ファイルを参照します。論理名(ライブラリ参照名またはファイル参照名)から参照を作成するのか、引用符付きの物理ファイル名を使用するかを選択できます。論理名を使用する場合、通常、参照の作成に SAS ステートメント(LIBNAME または FILENAME)を使用するのか、動作環境のコントロール言語を使用するのかを選択します。複数の方法を使用して、SAS ライブラリと外部ファイルを参照できます。動作環境によっては使用できない方法があります。

SAS ドキュメントでは、外部ファイルを使用する例には斜体のフレーズ *file-specification* を使用します。また、SAS ライブラリを使用する例には斜体フレーズ *SAS-library* を引用符で囲んで使用します。

```
infile file-specification obs = 100;
libname libref 'SAS-library';
```

SAS 9.4 SQL プロシジャの新機能

概要

新機能と拡張機能は次のとおりです。

- SAS SQL システムオプションおよび PROC SQL ステートメントオプションの追加
- DICTIONARY テーブルにおける拡張属性のサポートの追加
- BUFFERSIZE PROC SQL ステートメントオプションの置き換え

SAS SQL システムオプションおよび PROC SQL ステートメントオプションの追加

暗黙的なパススルー要求が失敗した場合に PROC SQL が SQL クエリを終了できるようにするために次の SQL オプションが追加されました。

- SQLIPONEATTEMPT system option
- IPONEATTEMPT | NOIPONEATTEMPT PROC SQL ステートメントオプション

詳細については、次を参照してください。“SQLIPONEATTEMPT System Option” (373 ページ)および“PROC SQL ステートメント” (220 ページ)

言語照合サポートの追加

SORTSEQ ステートメントオプションで言語照合サポートが追加されました。詳細については、“SORTSEQ=*sort-table* | LINGUISTIC” (229 ページ)を参照してください。

DICTIONARY テーブルにおける拡張属性のサポートの追加

DICTIONARY テーブルで拡張属性情報をコピーするためのサポートが追加されました。XATTRS DICTIONARY テーブルと Vxattr Sashelp ビューが追加されました。詳細

については、“DICTIONARY テーブルを使用し、SAS System の情報にアクセスする” (151 ページ)を参照してください。

BUFFERSIZE PROC SQL ステートメントオプションの置き換え

PROC SQL ステートメントの BUFFERSIZE オプションが、UBUFSIZE オプションで置き換えられました。SAS 9.4 より前のバージョンで使用されていた BUFFERSIZE オプションは、UBUFSIZE オプションと同じ機能を持つオプションであり、引き続きサポートされます。SAS 9.4 では、UBUFSIZE オプションの方が優先されます。詳細については、“PROC SQL ステートメント” (220 ページ)を参照してください。

1 部

SQL プロシジャの使用

1 章
SQL プロシジャについて 3

2 章
1 つのテーブルからのデータの取得 19

3 章
複数のテーブルからのデータの取得 73

4 章
テーブルとビューの作成および更新 111

5 章
SQL プロシジャを使用したプログラミング 139

6 章
PROC SQL を使用した問題の解決 177

1 章

SQL プロシジャについて

SQL について 3
SQL プロシジャについて 3
用語 4
テーブル 4
クエリ 5
ビュー 5
null 値 5
PROC SQL と SAS DATA ステップとの比較 5
サンプルテーブルの注記 7

SQL について

構造化照会言語(SQL)は、リレーショナルテーブルおよびリレーショナルデータベースのデータを検索し、更新するために標準化され、広く使用されている言語です。

関係は、集合の数学的概念に類似した数学的概念です。関係は、具体的には、行と列に配列された 2 次元テーブルで表されます。関係理論は、E.F.Codd(IBM の研究者)によって開発され、IBM の System R と呼ばれるプロトタイプに最初に実装されました。このプロトタイプは、SQL に基づく商用の IBM 製品に発展しました。構造化照会言語は、現在パブリックドメインであり、多くのベンダの製品に取り入れられています。

SQL プロシジャについて

SQL プロシジャは、Base SAS における構造化照会言語の実装です。PROC SQL は、Base SAS ソフトウェアに含まれており、任意の SAS データセット(テーブル)と共に使用できます。多くの場合、PROC SQL は、他の SAS プロシジャまたは DATA ステップの代替として使用できます。グローバルステートメント、データセットオプション、関数、入力形式、出力形式などの SAS 言語要素を、他の SAS プロシジャと同様に PROC SQL で使用できます。PROC SQL によって次のタスクを実行できます。

- レポートの生成
- 要約統計量の生成
- テーブルまたはビューからのデータ検索

- テーブルまたはビューのデータの結合
- テーブル、ビューおよびインデックスの作成
- PROC SQL テーブルのデータ値の更新
- データベース管理システム(DBMS)テーブルのデータの更新および検索
- 列の追加、変更または削除による PROC SQL テーブルの変更

PROC SQL は、対話的な SAS セッションまたはバッチプログラム内で使用できます。また、PROC SAL には、TITLE や OPTIONS などのグローバルステートメントを含めることができます。

用語

テーブル

PROC SQL テーブルは、SAS データファイルと同じものです。これは、DATA タイプの SAS ファイルです。PROC SQL テーブルは、行と列から成ります。行は SAS データファイルのオブザベーションに対応し、列は変数に対応します。次の表に、SQL、SAS および従来のデータ処理で使用されている同等の用語を示します。

表 1.1 *同等の用語の比較*

SQL 用語	SAS 用語	データ処理用語
テーブル	SAS データファイル	ファイル
行	オブザベーション	レコード
列	変数	フィールド

SAS DATA ステップまたは PROC SQL ステートメントを使用して、テーブルを作成および変更できます。これらは、4 章, “テーブルとビューの作成および更新” (111 ページ)で説明されています。他の SAS プロシジャおよび DATA ステップは、PROC SQL で作成されたテーブルを読み取り、更新できます。

SAS データファイルには、1 レベル名または 2 レベル名を付けることができます。通常、一時的な SAS データファイルには 1 レベル名のみが付けられ、このデータファイルは Work ライブラリに格納されます。PROC SQL は、User ライブラリが指定されていない限り、1 レベル名で指定されている SAS データファイルを Work ライブラリから読み取り、Work ライブラリに書き込むことを前提としています。User ライブラリは、LIBNAME ステートメントまたは SAS システムオプション USER=を使用して割り当てることができます。SAS データファイルおよびライブラリの操作方法の詳細については、次を参照してください。“Temporary and Permanent SAS Data Sets” (*Base SAS Procedures Guide*)

DBMS テーブルは、他のソフトウェアベンダのデータベース管理システムを使用して作成されたテーブルです。PROC SQL は、いくつかの制限付きで DBMS テーブルの接続、更新および変更を行うことができます。詳細については、“SAS/ACCESS を使用した DBMS へのアクセス” (168 ページ)を参照してください。

クエリ

クエリは、テーブル、ビューまたは DBMS からデータを検索します。クエリは、テーブルの行と列から成るクエリ結果を返します。PROC SQL では、SELECT ステートメントとその下位の句を使用してクエリを作成します。2 章, “1 つのテーブルからのデータの取得” (19 ページ)では、クエリの構築方法が説明されています。

ビュー

PROC SQL ビューは、実際には、テーブルが含むようなデータを含んでいません。PROC SQL ビューには、格納された SELECT ステートメントまたはクエリが含まれています。このクエリは、SAS プロシジャまたは DATA ステップでビューを使用するときに実行されます。ビューが実行されると、既存のテーブル、他のビューまたは SAS/ACCESS ビューから派生したデータが表示されます。他の SAS プロシジャおよび DATA ステップは、SAS データファイルを使用する場合と同様に PROC SQL ビューを使用できます。ビューの詳細については、次を参照してください。4 章, “テーブルとビューの作成および更新” (111 ページ)

注: クライアントとサーバー間で PROC SQL ビューを処理する場合、正しい結果が得られるかどうかは、クライアントとサーバー間のアーキテクチャの互換性に依存します。詳細については、“Accessing a SAS View” (*SAS/CONNECT User's Guide*)を参照してください。

null 値

SQL の ANSI 規格に従い、欠損値はヌル値と呼ばれます。これは、空白やゼロの値とは異なります。ただし、他の SAS 機能との互換性を保つために、PROC SQL は、欠損値を空白またはゼロ値と同じに扱い、これら 3 つすべてをヌル値と見なします。この重要な概念は、このドキュメントの複数の場所に現れます。

PROC SQL と SAS DATA ステップとの比較

PROC SQL は、DATA ステップならびに PRINT、SORT および SUMMARY プロシジャが提供する演算の一部を実行できます。次のクエリは、各大陸のすべての大国(100 万人を超える人口を有する国)の総人口を表示します。

```
proc sql;
   title 'Population of Large Countries Grouped by Continent';
   select Continent, sum(Population) as TotPop format=comma15.
      from sql.countries
      where Population gt 1000000
      group by Continent
      order by TotPop;
quit;
```

アウトプット 1.1 *SQL 出力の例*

Population of Large Countries Grouped by Continent

Continent	TotPop
Oceania	3,422,548
Australia	18,255,944
Central America and Caribbean	65,283,910
South America	316,303,397
North America	384,801,818
Africa	706,611,183
Europe	811,680,062
Asia	3,379,469,458

同じ結果を生成する SAS プログラムを次に示します。

```
title 'Large Countries Grouped by Continent';
proc summary data=sql.countries;
   where Population > 1000000;
   class Continent;
   var Population;
   output out=sumPop sum=TotPop;
run;

proc sort data=SumPop;
   by totPop;
run;

proc print data=SumPop noobs;
   var Continent TotPop;
   format TotPop comma15.;
   where _type_=1;
run;
```

アウトプット 1.2 *DATA ステップの出力例*

Large Countries Grouped by Continent

Continent	TotPop
Oceania	3,422,548
Australia	18,255,944
Central America and Caribbean	65,283,910
South America	316,303,397
North America	384,801,818
Africa	706,611,183
Europe	811,680,062
Asia	3,379,469,458

この例は、PROC SQL によって Base SAS ソフトウェアと同じ結果が得られることを示しています。ただし、多くの場合、そのステートメントは、より少なくかつ短くなります。この例で示した SELECT ステートメントは、合計、グループ化、並べ替え、行の選択を実行しています。また、PRINT プロシジャを使用しないでクエリの結果を表示しています。

PROC SQL は、RUN ステートメントを使用しないで実行されます。PROC SQL の呼び出し後、PROC ステートメントを再びサブミットしなくても、さらに SQL プロシジャステートメントをサブミットできます。プロシジャを終了するには、QUIT ステートメントを使用します。

サンプルテーブルの注記

すべての例に対して、次のグローバルステートメントが有効です。

```
libname sql 'SAS-library';
```

“SQL プロシジャの使用“セクション全体で使用されているテーブルには、地理的データや人口学的データが含まれています。これらのデータは、PROC SQL コードの例での使用のみを目的としています。これらのデータは、必ずしも最新でも正確でもないことにご注意ください。

これらのテーブルは、次のサイトから ZIP 形式で入手できます。http://support.sas.com/documentation/onlinedoc/base/index.html このページ上の *SAS SQL プロシジャユーザーガイド*を参照してください。SAS システムによりアクセス可能な場所に、ZIP ファイルをダウンロードして展開します。ZIP ファイルを展開すると、テーブルを含むトランスポートファイルになります。

SAS システムを起動します。次のコードをサブミットしてテーブルをインポートします。

```
/* Substitute the pathname of your Sasuser
   directory for 'your-Sasuser-directory-path'
   and the appropriate pathname and filename
```

```
   for 'your-downloaded-file-location'        */

      libname new 'your-Sasuser-directory-path';
      filename trans 'your-downloaded-file-location';

      proc cimport library=new infile=trans;
      run;

/* Assign a libref named SQL to provide access to some
   of the sample data sets.                          */

libname sql 'your-Sasuser-directory-path';
```

これらのデータセットの出力の一部を次の表に示します。

“SQL プロシジャの使用“セクションに示されているすべての SQL プログラム例は、付録 3, “「SQL プロシジャの使用」で示されているコード例” (387 ページ)にまとめられています。コードを SAS エディタにコピーアンドペーストする場合、HTML ページからコードをコピーすることで、そのコード内のスペーシングが保存されます。

Countries テーブルには、国に関するデータが含まれています。Area 列には、国の面積(平方マイル単位)が含まれています。UNDate 列には、該当する場合、国が国連に加盟した年が含まれています。

アウトプット 1.3 *Countries (部分的出力)*

COUNTRIES

Name	Capital	Population	Area	Continent	UNDate
Afghanistan	Kabul	17070323	251825	Asia	1946
Albania	Tirane	3407400	11100	Europe	1955
Algeria	Algiers	28171132	919595	Africa	1962
Andorra	Andorra la Vell	64634	200	Europe	1993
Angola	Luanda	9901050	481300	Africa	1976
Antigua and Barbuda	St. John's	65644	171	Central America	1981
Argentina	Buenos Aires	34248705	1073518	South America	1945
Armenia	Yerevan	3556864	11500	Asia	1992
Australia	Canberra	18255944	2966200	Australia	1945
Austria	Vienna	8033746	32400	Europe	1955
Azerbaijan	Baku	7760064	33400	Asia	1992
Bahamas	Nassau	275703	5400	Central America	1973
Bahrain	Manama	591800	300	Asia	1971
Bangladesh	Dhaka	1.2639E8	57300	Asia	1974
Barbados	Bridgetown	258534	200	Central America	1966

WorldCityCoords テーブルには、世界の都市の緯度と経度のデータが含まれています。西半球の都市は、負の経度座標を持ちます。南半球の都市は、負の緯度座標を持ちます。座標は、最も近い度に丸められます。

アウトプット 1.4 *WorldCityCoords (部分的出力)*

WORLDCITCOORDS

City	Country	Latitude	Longitude
Kabul	Afghanistan	35	69
Algiers	Algeria	37	3
Buenos Aires	Argentina	-34	-59
Cordoba	Argentina	-31	-64
Tucuman	Argentina	-27	-65
Adelaide	Australia	-35	138
Alice Springs	Australia	-24	134
Brisbane	Australia	-27	153
Darwin	Australia	-12	131
Melbourne	Australia	-38	145
Perth	Australia	-32	116
Sydney	Australia	-34	151
Vienna	Austria	48	16
Nassau	Bahamas	26	-77
Chittagong	Bangladesh	22	92

USCityCoords テーブルには、米国の都市の座標が含まれています。このテーブル内のすべての都市は西半球にあるため、これらすべての経度座標は負です。座標は、最も近い度に丸められます。

アウトプット 1.5 *USCityCoords (部分的出力)*

USCITYCOORDS

City	State	Latitude	Longitude
Albany	NY	43	-74
Albuquerque	NM	36	-106
Amarillo	TX	35	-102
Anchorage	AK	61	-150
Annapolis	MD	39	-77
Atlanta	GA	34	-84
Augusta	ME	44	-70
Austin	TX	30	-98
Baker	OR	45	-118
Baltimore	MD	39	-76
Bangor	ME	45	-69
Baton Rouge	LA	31	-91
Birmingham	AL	33	-87
Bismarck	ND	47	-101
Boise	ID	43	-116

The United States table contains data that is associated with the states. The Statehood column contains the date on which the state was admitted into the Union.

アウトプット 1.6 *United States (部分的出力)*

UNITEDSTATES

Name	Capital	Population	Area	Continent	Statehood
Alabama	Montgomery	4447100	52423	North America	14DEC1819
Alaska	Juneau	626932	656400	North America	03JAN1959
Arizona	Phoenix	5130632	114000	North America	14FEB1912
Arkansas	Little Rock	2447996	53200	North America	15JUN1836
California	Sacramento	31518948	163700	North America	09SEP1850
Colorado	Denver	3601298	104100	North America	01AUG1876
Connecticut	Hartford	3405565	5500	North America	09JAN1788
Delaware	Dover	707232	2500	North America	07DEC1787
District of Colum	Washington	612907	100	North America	21FEB1871
Florida	Tallahassee	13814408	65800	North America	03MAR1845
Georgia	Atlanta	8186453	59400	North America	02JAN1788
Hawaii	Honolulu	1183198	10900	Oceania	21AUG1959
Idaho	Boise	1293953	83600	North America	03JUL1890
Illinois	Springfield	11813091	57900	North America	03DEC1818
Indiana	Indianapolis	5769553	36400	North America	11DEC1816

PostalCodes テーブルには、郵便番号の短縮形が含まれています。

アウトプット 1.7 *PostalCodes (部分的出力)*

POSTALCODES

Name	Code
Alabama	AL
Alaska	AK
American Samoa	AS
Arizona	AZ
Arkansas	AR
California	CA
Colorado	CO
Connecticut	CT
Delaware	DE
District Of Columbia	DC
Florida	FL
Georgia	GA
Guam	GU
Hawaii	HI
Idaho	ID

WorldTemps テーブルには、さまざまな国際都市の平均最高気温と平均最低気温が含まれています。

アウトプット 1.8 *WorldTemps (部分的出力)*

WORLDTEMPS

City	Country	AvgHigh	AvgLow
Algiers	Algeria	90	45
Amsterdam	Netherlands	70	33
Athens	Greece	89	41
Auckland	New Zealand	75	44
Bangkok	Thailand	95	69
Beijing	China	86	17
Belgrade	Yugoslavia	80	29
Berlin	Germany	75	25
Bogota	Colombia	69	43
Bombay	India	90	68
Bucharest	Romania	83	24
Budapest	Hungary	80	25
Buenos Aires	Argentina	87	48
Cairo	Egypt	95	48
Calcutta	India	97	56

OilProd テーブルには、石油産出国の石油産出統計が含まれています。

アウトプット 1.9 *OilProd (部分的出力)*

OILPROD

Country	BarrelsPerDay
Algeria	1,400,000
Canada	2,500,000
China	3,000,000
Egypt	900,000
Indonesia	1,500,000
Iran	4,000,000
Iraq	600,000
Kuwait	2,500,000
Libya	1,500,000
Mexico	3,400,000
Nigeria	2,000,000
Norway	3,500,000
Oman	900,000
Saudi Arabia	9,000,000
United States of America	8,000,000

OilRsrvs テーブルは、石油産出国の石油備蓄量の概算値を示します。

アウトプット 1.10 *OilRsrvs (部分的出力)*

OILRSRVS

Country	Barrels
Algeria	9,200,000,000
Canada	7,000,000,000
China	25,000,000,000
Egypt	4,000,000,000
Gabon	1,000,000,000
Indonesia	5,000,000,000
Iran	90,000,000,000
Iraq	110,000,000,000
Kuwait	95,000,000,000
Libya	30,000,000,000
Mexico	50,000,000,000
Nigeria	16,000,000,000
Norway	11,000,000,000
Saudi Arabia	260,000,000,000
United Arab Emirates	100,000,000

Continents テーブルには、世界各国に関連する地理データが含まれています。

アウトプット 1.11 *Continents*

CONTINENTS

Name	Area	HighPoint	Height	LowPoint	Depth
Africa	11506000	Kilimanjaro	19340	Lake Assal	-512
Antarctica	5500000	Vinson Massif	16860		.
Asia	16988000	Everest	29028	Dead Sea	-1302
Australia	2968000	Kosciusko	7310	Lake Eyre	-52
Central America	.		.		.
Europe	3745000	El'brus	18510	Caspian Sea	-92
North America	9390000	McKinley	20320	Death Valley	-282
Oceania	.		.		.
South America	6795000	Aconcagua	22834	Valdes Peninsul	-131

Features テーブルには、海、湖、山など、さまざまなタイプの地勢を説明する統計値が含まれています。

アウトプット 1.12 *Features (部分的出力)*

FEATURES

Name	Type	Location	Area	Height	Depth	Length
Aconcagua	Mountain	Argentina	.	22834	.	.
Amazon	River	South America	.	.	.	4000
Amur	River	Asia	.	.	.	2700
Andaman	Sea		218100	.	3667	.
Angel Falls	Waterfall	Venezuela	.	3212	.	.
Annapurna	Mountain	Nepal	.	26504	.	.
Aral Sea	Lake	Asia	25300	.	222	.
Ararat	Mountain	Turkey	.	16804	.	.
Arctic	Ocean		5105700	.	17880	.
Atlantic	Ocean		33420000	.	28374	.
Baffin	Island	Arctic	183810	.	.	.
Baltic	Sea		146500	.	180	.
Baykal	Lake	Russia	11780	.	5315	.
Bering	Sea		873000	.	4893	.
Black	Sea		196100	.	3906	.

2 章

1 つのテーブルからのデータの取得

SELECT ステートメントの概要 **20**
SELECT ステートメントの使用方法 20
SELECT と FROM 句 20
WHERE 句 21
ORDER BY 句 21
GROUP BY 句 21
HAVING 句 21
SELECT ステートメントの並べ替え 22

テーブルの列の選択 **22**
テーブルのすべての列の選択 22
テーブルの特定の列の選択 23
クエリの結果から重複行を削除する 25
テーブルの構造の指定 27

列の新規作成 **27**
出力へのテキストの追加 27
値の計算 29
列のエイリアスを割り当てる 30
計算列をエイリアスで参照する 31
条件付き値の割り当て 32
欠損値の置換 35
列の属性の指定 36

データの並べ替え **37**
概要:並べ替え順序 37
列を基準に並べ替える 38
複数の列を基準に並べ替える 38
並べ替え順序の指定 39
計算列を基準に並べ替える 40
列の位置を基準に並べ替える 41
選択していない列を基準に並べ替える 42
異なるソートシーケンスの指定 43
欠損値を含む列の並べ替え 44

条件を満たす行の取得 **44**
単一の WHERE 句の使用 45
比較に基づく行の取得 45
複数の条件を満たす行の取得 47
その他の条件演算子の使用 49
切り捨て文字列の比較演算子の使用 53
欠損値を含む WHERE 句の使用 54

データの要約 **56**
概要:データの要約 56
集計関数の使用 56
WHERE 句を使用したデータの要約 57
合計の表示 58
複数の行から 1 行にデータを組み合わせる 59
要約統計量の再マージ 59
重複しない値に集計関数を使用する 61
欠損値を含むデータの要約 62

データのグループ化 **64**
1 つ列を基準にグループ化する 64
要約しないグループ化 64
複数の列を基準にグループ化する 65
データのグループ化と並べ替え 66
欠損値を含むグループ化 67

グループ化されたデータのフィルタリング **69**
概要:グループ化されたデータのフィルタリング 69
単一の HAVING 句の使用 69
HAVING と WHERE の選択 70
HAVING と集計関数の併用 70

クエリの検証 **71**

SELECT ステートメントの概要

SELECT ステートメントの使用方法

この章では、次のタスクの実行方法について説明します。

- SELECT ステートメントを使用した 1 つのテーブルからのデータの取得
- VALIDATE ステートメントを使用した SELECT ステートメントの正しさの検証

SELECT ステートメントを使用して、テーブルのデータまたは SAS データビューによって記述されたデータを取り出すことができます。

注: この章の例では、SAS データセットであるテーブルのデータを取り出します。ただし、ここで SAS データビューによって記述されたすべての操作を使用できます。

SELECT ステートメントは、PROC SQL の主要なツールです。SELECT ステートメントを使用して、テーブルのデータの列を、識別、取得および操作します。SELECT ステートメント内で複数のオプションの句を使用して、クエリに制限を設けることもできます。

SELECT と FROM 句

次の簡単な SELECT ステートメントは、十分に役立つ結果を生成します。

```
select Name
   from sql.countries;
```

SELECT ステートメントには SELECT 句と FROM 句を含める必要があり、PROC SQL クエリでは、その両方が必要です。SELECT ステートメントには、次の句が含まれます。

- Name 列を記述する SELECT 句

- Name 列が存在するテーブルを記述する FROM 句

WHERE 句

WHERE 句を使用して、テーブルの各行が満たす必要のある条件を指定することによって、取り出すデータを制限できます。PROC SQL の出力には、この条件を満たす行のみが含まれます。次の SELECT ステートメントには、5,000,000 人を超える人口を有する国のみにクエリ出力を制限する WHERE 句が含まれています。

```
select Name
   from sql.countries
   where Population gt 5000000;
```

ORDER BY 句

ORDER BY 句を使用して、テーブルからの出力を、1 つ以上の列によって並べ替えることができます。つまり、文字値をアルファベットの昇順または降順に並べることができ、数値を数値の昇順または降順に並べることができます。デフォルトの順序は昇順です。たとえば、前述の例を、人口の降順にデータを表示するように変更できます。

```
select Name
   from sql.countries
   where Population gt 5000000
   order by Population desc;
```

GROUP BY 句

GROUP BY 句を使用して、クエリ結果を行のサブセットに分割できます。GROUP BY 句を使用する場合、SELECT 句または HAVING 句で集計関数を使用して、データをグループ化する方法を PROC SQL に指示します。集計関数の詳細については、次を参照してください。“データの要約” (56 ページ)PROC SQL は、グループごとに個別に集計関数を計算します。集計関数を使用しない場合、PROC SQL は GROUP BY 句を ORDER BY 句と同様に扱い、どの集計関数もテーブル全体に対して適用されます。

次のクエリでは、SUM 関数を使用してそれぞれの大陸の総人口を表示しています。ここでは、GROUP BY 句が大陸別に国をグループ化し、ORDER BY 句がアルファベット順に大陸を並べ替えています。

```
select Continent, sum(Population)
   from sql.countries
   group by Continent
   order by Continent;
```

HAVING 句

HAVING 句は、GROUP BY 句と連動して、指定された条件に基づいてクエリ結果のグループを制限します。PROC SQL は、データをグループ化し、集計関数を適用してから、HAVING 条件を適用します。たとえば、次のクエリは、アジア大陸およびヨーロッパ大陸のみを含むようにグループを制限します。

```
select Continent, sum(Population)
   from sql.countries
   group by Continent
```

```
having Continent in ('Asia', 'Europe')
order by Continent;
```

SELECT ステートメントの並べ替え

SELECT ステートメントを作成する場合、次の順序で句を指定する必要があります。

1. SELECT
2. FROM
3. WHERE
4. GROUP BY
5. HAVING
6. ORDER BY

注: SELECT 句と FROM 句のみが必須です。

PROC SQL の SELECT ステートメントとその句については、次のセクションで詳細に説明します。

テーブルの列の選択

テーブルからデータを取り出す場合、基本的な SELECT ステートメントを変更することによって、1 つ以上の列を選択できます。

テーブルのすべての列の選択

テーブルのすべての列を選択するには、SELECT 句でアスタリスクを使用します。次の例では、Sql.USCityCoords テーブルのすべての列を選択しています。このテーブルには、アメリカの各都市の緯度と経度の値が含まれています。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'U.S. Cities with Their States and Coordinates';
   select *
      from sql.uscitycoords;
```

注: OUTOBS=オプションは、出力の行(オブザベーション)の数を制限します。OUTOBS=は、OBS=データセットオプションに類似しています。OUTOBS=は、このマニュアル全体で使用されており、例で表示される行の数を制限しています。

注: これらの例で使用されるテーブルでは、赤道から南の緯度の値は負になります。グリニッジ子午線から西の経度の値も負になります。

アウトプット 2.1 テーブルのすべての列の選択

U.S. Cities with Their States and Coordinates

City	State	Latitude	Longitude
Albany	NY	43	-74
Albuquerque	NM	36	-106
Amarillo	TX	35	-102
Anchorage	AK	61	-150
Annapolis	MD	39	-77
Atlanta	GA	34	-84
Augusta	ME	44	-70
Austin	TX	30	-98
Baker	OR	45	-118
Baltimore	MD	39	-76
Bangor	ME	45	-69
Baton Rouge	LA	31	-91

注: すべての列を選択した場合、PROC SQL は、テーブルに格納されている順序で列を表示します。

テーブルの特定の列の選択

テーブルの特定の列を選択するには、SELECT 句に列名を記述します。次の例では、Sql.USCityCoords テーブルの City 列のみを選択しています。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'Names of U.S. Cities';
   select City
      from sql.uscitycoords;
```

アウトプット 2.2 *1 つの列の選択*

Names of U.S. Cities

City
Albany
Albuquerque
Amarillo
Anchorage
Annapolis
Atlanta
Augusta
Austin
Baker
Baltimore
Bangor
Baton Rouge

複数の列を選択する場合、この例のように複数の列名をカンマで区切る必要があります。この例では、Sql.USCityCoords テーブルの City 列と State 列を選択しています。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'U.S. Cities and Their States';
   select City, State
      from sql.uscitycoords;
```

アウトプット 2.3 複数の列の選択

U.S. Cities and Their States

City	State
Albany	NY
Albuquerque	NM
Amarillo	TX
Anchorage	AK
Annapolis	MD
Atlanta	GA
Augusta	ME
Austin	TX
Baker	OR
Baltimore	MD
Bangor	ME
Baton Rouge	LA

注: 特定の列を選択した場合、PROC SQL は、SELECT 句に指定した順序で各列を表示します。

クエリの結果から重複行を削除する

場合によっては、列内の一意の値のみを検索する必要があることがあります。たとえば、アメリカの州が存在する一意の大陸を検索する場合、次のクエリを作成することから始めることができます。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'Continents of the United States';
   select Continent
      from sql.unitedstates;
```

アウトプット 2.4 *重複する値を含む列の選択*

Continents of the United States

Continent
North America
North America
North America
North America
North America
North America
North America
North America
North America
North America
North America
Oceania

SELECT 句で DISTINCT キーワードを使用して、結果から重複行を除去できます。前述の例と次のクエリを比べてください。このクエリでは、DISTINCT キーワードを使用して、大陸ごとに、Sql.UnitedStates テーブルに含まれる 1 つの行を出力しています。

```
libname sql 'SAS-library';

proc sql;
   title 'Continents of the United States';
   select distinct Continent
      from sql.unitedstates;
```

アウトプット 2.5 *重複する値の除去*

Continents of the United States

Continent
North America
Oceania

注: DISTINCT キーワードを使用して SELECT 句にすべてのテーブル列を指定した場合、PROC SQL は、重複行(つまり、すべての列の値が一致する行)を結果から除去します。

テーブルの構造の指定

テーブル内のすべての列のリストと、それらの属性を取得するには、DESCRIBE TABLE ステートメントを使用します。次の例では、Sql.UnitedStates テーブルの説明を生成しています。PROC SQL は、この説明をログに書き込みます。

```
libname sql 'SAS-library';

proc sql;
   describe table sql.unitedstates;
```

ログ 2.1 テーブルの構造を決定する部分のログ

```
注:SQL table SQL.UNITEDSTATES was created like: create table
SQL.UNITEDSTATES( bufsize=12288 ) ( Name char(35) format=$35. informat=$35.
label='Name', Capital char(35) format=$35. informat=$35. label='Capital',
Population num format=BEST8. informat=BEST8. label='Population', Area num
format=BEST8. informat=BEST8., Continent char(35) format=$35. informat=$35.
label='Continent', Statehood num );
```

列の新規作成

テーブルに格納された列の選択に加えて、クエリが存続する間存在する新しい列を作成できます。これらの列には、テキストまたは計算を含めることができます。PROC SQL は、作成した列を、テーブルの元の列と同様に出力します。

出力へのテキストの追加

クエリに文字列またはリテラルを含めることによって、出力にテキストを追加できます。次のクエリでは、追加の列として 2 つの文字列を出力に含めています。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'U.S. Postal Codes';
   select 'Postal code for', Name, 'is', Code
      from sql.postalcodes;
```

アウトプット 2.6 出力へのテキストの追加

U.S. Postal Codes

	Name		Code
Postal code for	Alabama	is	AL
Postal code for	Alaska	is	AK
Postal code for	American Samoa	is	AS
Postal code for	Arizona	is	AZ
Postal code for	Arkansas	is	AR
Postal code for	California	is	CA
Postal code for	Colorado	is	CO
Postal code for	Connecticut	is	CT
Postal code for	Delaware	is	DE
Postal code for	District Of Columbia	is	DC
Postal code for	Florida	is	FL
Postal code for	Georgia	is	GA

Name と Code という列見出しを表示しないようにするには、特殊文字で始まるラベルを各列に割り当てます。ラベルを割り当てると、PROC SQL は列名を出力しません。PROC SQL は、特殊文字で始まるラベルを出力しません。たとえば、次のクエリを使用して、前述の例で PROC SQL が表示していた列見出しを抑制できます。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'U.S. Postal Codes';
   select 'Postal code for', Name label='#', 'is', Code label='#'
      from sql.postalcodes;
```

アウトプット 2.7 出力での列見出しの抑制

U.S. Postal Codes

Postal code for	Alabama	is	AL
Postal code for	Alaska	is	AK
Postal code for	American Samoa	is	AS
Postal code for	Arizona	is	AZ
Postal code for	Arkansas	is	AR
Postal code for	California	is	CA
Postal code for	Colorado	is	CO
Postal code for	Connecticut	is	CT
Postal code for	Delaware	is	DE
Postal code for	District Of Columbia	is	DC
Postal code for	Florida	is	FL
Postal code for	Georgia	is	GA

値の計算

数値列から取り出した値を使用して、計算を実行できます。次の例では、Sql.WorldTemps テーブルの温度を華氏から摂氏に変換しています。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'Low Temperatures in Celsius';
   select City, (AvgLow - 32) * 5/9 format=4.1
      from sql.worldtemps;
```

注: この例では、FORMAT 属性を使用して、計算された出力の出力形式を変更します。詳細については、“列の属性の指定” (36 ページ)を参照してください。

アウトプット 2.8 値の計算

Low Temperatures in Celsius

City	
Algiers	7.2
Amsterdam	0.6
Athens	5.0
Auckland	6.7
Bangkok	20.6
Beijing	-8.3
Belgrade	-1.7
Berlin	-3.9
Bogota	6.1
Bombay	20.0
Bucharest	-4.4
Budapest	-3.9

列のエイリアスを割り当てる

列のエイリアスを指定することによって、PROC SQL クエリ内の任意の列に新しい名前を割り当てることができます。新しい名前は、SAS の命名規則に従う必要があります。この名前は、そのクエリが存続する間だけ、存続します。

エイリアスを使用して列に名前を付けると、その後のクエリで、そのエイリアスを使用して列を参照できます。PROC SQL は、エイリアスを出力で列見出しとして使用します。次の例では、前述の例の計算列に、LowCelsius というエイリアスを割り当てています。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'Low Temperatures in Celsius';
   select City, (AvgLow - 32) * 5/9 as LowCelsius format=4.1
      from sql.worldtemps;
```

アウトプット 2.9 *計算列へのエイリアスの割り当て*

Low Temperatures in Celsius

City	LowCelsius
Algiers	7.2
Amsterdam	0.6
Athens	5.0
Auckland	6.7
Bangkok	20.6
Beijing	-8.3
Belgrade	-1.7
Berlin	-3.9
Bogota	6.1
Bombay	20.0
Bucharest	-4.4
Budapest	-3.9

計算列をエイリアスで参照する

列のエイリアスを使用して、計算される値を参照する場合、エイリアスと共に CALCULATED キーワードを使用して、その値がクエリ内で計算されることを PROC SQL に知らせる必要があります。次の例では、計算される 2 つの値(LowC と HighC)を使用して、3 つ目の値(Range)を計算しています。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'Range of High and Low Temperatures in Celsius';
      select City, (AvgHigh - 32) * 5/9 as HighC format=5.1,
                   (AvgLow - 32) * 5/9 as LowC format=5.1,
                   (calculated HighC - calculated LowC)
                    as Range format=4.1
   from sql.worldtemps;
```

注: エイリアスを使用して、SELECT 句、WHERE 句または ORDER BY 句で計算列を参照できます。

アウトプット 2.10 計算列をエイリアスで参照する

Range of High and Low Temperatures in Celsius

City	HighC	LowC	Range
Algiers	32.2	7.2	25.0
Amsterdam	21.1	0.6	20.6
Athens	31.7	5.0	26.7
Auckland	23.9	6.7	17.2
Bangkok	35.0	20.6	14.4
Beijing	30.0	-8.3	38.3
Belgrade	26.7	-1.7	28.3
Berlin	23.9	-3.9	27.8
Bogota	20.6	6.1	14.4
Bombay	32.2	20.0	12.2
Bucharest	28.3	-4.4	32.8
Budapest	26.7	-3.9	30.6

注: このクエリでは、HighC 列、LowC 列および Range 列に対して 4.1 の数値出力形式を設定しているため、これらの列の値は、最も近い小数点 1 桁までの値に丸められます。この丸め処理のため、HighC 列と LowC 列の値の一部は、Range 列の範囲値の出力を反映していません。数値データの値を丸めると、この種の誤差が発生する場合があります。この問題を回避する場合、出力形式の小数点の桁数を追加できます。

詳細については、“列エイリアスの使用” (148 ページ)を参照してください。

条件付き値の割り当て

単一の CASE 式の使用

CASE 式を使用して、列内のデータ値の一部またはすべてを解釈し、変更できます。これによってデータはさらに役立ち、意味を持つようになります。

CASE 式を使用して条件に基づいて値を割り当てることによって、クエリ内で条件付き論理を使用できます。列名を使用できる場所であれば、どこでも CASE 式を使用できます。

次のテーブル(次の例で使用されます)は、場所 1 と場所 2 の間に存在する世界の気候帯(最も近い緯度に丸められています)を示しています。

表 2.1 世界の気候帯

気候帯	場所 1	場所 1 の緯度	場所 2	場所 2 の緯度
北寒帯	北極	90	北極圏	67

気候帯	場所 1	場所 1 の緯度	場所 2	場所 2 の緯度
北温帯	北極圏	67	北回帰線	23
熱帯	北回帰線	23	南回帰線	-23
南温帯	南回帰線	-23	南極圏	-67
南寒帯	南極圏	-67	南極	-90

この例では、CASE 式によって、Sql.WorldCityCoords テーブルの Latitude 列の値に基づいて、都市ごとに気候帯を決定しています。また、このクエリは ClimateZone というエイリアスを値に割り当てています。CASE 論理は、END キーワードを使用して閉じる必要があります。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'Climate Zones of World Cities';
   select City, Country, Latitude,
          case
             when Latitude gt 67 then 'North Frigid'
             when 67 ge Latitude ge 23 then 'North Temperate'
             when 23 gt Latitude gt -23 then 'Torrid'
             when -23 ge Latitude ge -67 then 'South Temperate'
             else 'South Frigid'
          end as ClimateZone
      from sql.worldcitycoords
      order by City;
```

アウトプット 2.11 単一の CASE 式の使用

Climate Zones of World Cities

City	Country	Latitude	ClimateZone
Abadan	Iran	30	North Temperate
Acapulco	Mexico	17	Torrid
Accra	Ghana	5	Torrid
Adana	Turkey	37	North Temperate
Addis Ababa	Ethiopia	9	Torrid
Adelaide	Australia	-35	South Temperate
Aden	Yemen	13	Torrid
Ahmenabad	India	22	Torrid
Algiers	Algeria	37	North Temperate
Alice Springs	Australia	-24	South Temperate
Amman	Jordan	32	North Temperate
Amsterdam	Netherlands	52	North Temperate

CASE-OPERAND フォームの使用

次の例のように、CASE-OPERAND フォームを使用して CASE 式を作成することもできます。この例では、州を選択し、それらの州を Continent 列の値に基づいて地域に割り当てています。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'Assigning Regions to Continents';
   select Name, Continent,
          case Continent
             when 'North America' then 'Continental U.S.'
             when 'Oceania' then 'Pacific Islands'
             else 'None'
          end as Region
      from sql.unitedstates;
```

注: CASE 式の CASE-OPERAND フォームを使用する場合、条件はすべて等式である必要があります。つまり、“単一の CASE 式の使用” (32 ページ)で使用されているような比較演算子や、他の種類の演算子を使用することはできません。

アウトプット 2.12 CASE-OPERAND フォームでの CASE 式の使用

Assigning Regions to Continents

Name	Continent	Region
Alabama	North America	Continental U.S.
Alaska	North America	Continental U.S.
Arizona	North America	Continental U.S.
Arkansas	North America	Continental U.S.
California	North America	Continental U.S.
Colorado	North America	Continental U.S.
Connecticut	North America	Continental U.S.
Delaware	North America	Continental U.S.
District of Columbia	North America	Continental U.S.
Florida	North America	Continental U.S.
Georgia	North America	Continental U.S.
Hawaii	Oceania	Pacific Islands

欠損値の置換

COALESCE 関数を使用して、列内の欠損値を、指定した新しい値に置き換えることができます。COALESCE 関数は、非欠損値を検出するまで、クエリで処理されるすべての行に対して各引数をチェックし、検出した値を返します。すべての引数が欠損値である場合、COALESCE 関数は欠損値を返します。たとえば、次のクエリは、Sql.Continents テーブルの LowPoint 列の欠損値を、**Not Available** という単語に置き換えます。

```
libname sql 'SAS-library';

proc sql;
   title 'Continental Low Points';
   select Name, coalesce(LowPoint, 'Not Available') as LowPoint
      from sql.continents;
```

アウトプット 2.13 COALESCE 関数を使用した欠損値の置換

Continental Low Points

Name	LowPoint
Africa	Lake Assal
Antarctica	Not Available
Asia	Dead Sea
Australia	Lake Eyre
Central America and Caribbean	Not Available
Europe	Caspian Sea
North America	Death Valley
Oceania	Not Available
South America	Valdes Peninsula

次の CASE 式は、同じ欠損値の置換を実行する別の方法を示しています。ただし、COALESCE 関数を使用したほうが、少ないコード行数で同じ結果を得ることができます。

```
libname sql 'SAS-library';

proc sql;
   title 'Continental Low Points';
   select Name, case
                   when LowPoint is missing then 'Not Available'
                   else Lowpoint
                end as LowPoint
      from sql.continents;
```

列の属性の指定

SAS データの表示方法を決定する次の列属性を指定できます。

- FORMAT=
- INFORMAT=
- LABEL=
- LENGTH=

これらの属性を指定しない場合、PROC SQL はテーブルにすでに保存されている属性を使用します。属性が保存されていない場合、PROC SQL はデフォルトの属性を使用します。

次の例では、**State** というラベルが Name 列に割り当てられ、COMMA10.という出力形式が Area 列に割り当てられています。

```
libname sql 'SAS-library';

proc sql outobs=12;
```

```
title 'Areas of U.S. States in Square Miles';
select Name label='State', Area format=comma10.
   from sql.unitedstates;
```

注: LABEL=キーワードの使用は任意です。たとえば、次の 2 つの SELECT 句は同じです。

```
select Name label='State', Area format=comma10.

select Name 'State', Area format=comma10.
```

アウトプット 2.14 *列の属性の指定*

Areas of U.S. States in Square Miles

State	Area
Alabama	52,423
Alaska	656,400
Arizona	114,000
Arkansas	53,200
California	163,700
Colorado	104,100
Connecticut	5,500
Delaware	2,500
District of Columbia	100
Florida	65,800
Georgia	59,400
Hawaii	10,900

データの並べ替え

概要:並べ替え順序

ORDER BY 句を使用して、選択されない列や計算列など、テーブル内の任意の列を指定してクエリ結果を並べ替えることができます。

ORDER BY 句を SELECT ステートメントに含めなければ、インデックスが存在する場合でも、出力行の特定の順序(照会されたテーブルでの行の出現順など)は保証されません。ORDER BY 句を指定しないと、出力行の順序は、PROC SQL の内部処理、SAS のデフォルトの照合順序、および使用するオペレーティングシステムによって決まります。したがって、結果テーブルを特定の順序で表示する場合は、ORDER BY 句を使用します。

詳細と例については、次を参照してください。“ORDER BY 句” (263 ページ)

列を基準に並べ替える

次の例では、Sql.Countries テーブルから国とそれらの人口を選択し、人口によって結果を並べ替えています。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'Country Populations';
   select Name, Population format=comma10.
      from sql.countries
      order by Population;
```

注: ORDER BY 句を使用した場合、出力の順序は変更されますが、テーブルに格納された行の順序は変更されません。

注: PROC SQL のデフォルトの並べ替え順序は、昇順です。

アウトプット 2.15 *列を基準に並べ替える*

Country Populations

Name	Population
Vatican City	1,010
Nauru	10,099
Tuvalu	10,099
Leeward Islands	12,119
Turks and Caicos Islands	12,119
Cayman Islands	23,228
San Marino	24,238
Liechtenstein	30,297
Gibraltar	30,297
Monaco	31,307
Saint Kitts and Nevis	41,406
Marshall Islands	54,535

複数の列を基準に並べ替える

ORDER BY 句で、複数の列名をカンマで区切って指定することで、複数の列によって並べ替えることができます。次の例では、2 つの列(Continent と Name)によって Sql.Countries テーブルを並べ替えています。

```
libname sql 'SAS-library';

proc sql outobs=12;
```

```
title 'Countries, Sorted by Continent and Name';
select Name, Continent
   from sql.countries
   order by Continent, Name;
```

アウトプット 2.16 *複数の列を基準に並べ替える*

Countries, Sorted by Continent and Name

Name	Continent
Bermuda	
Iceland	
Kalaallit Nunaat	
Algeria	Africa
Angola	Africa
Benin	Africa
Botswana	Africa
Burkina Faso	Africa
Burundi	Africa
Cameroon	Africa
Cape Verde	Africa
Central African Republic	Africa

注: この結果では、大陸が設定されていない国が最初に表示されています。これは、昇順の並べ替えの場合、PROC SQL が欠損値を最初に並べ替えるためです。

並べ替え順序の指定

結果を並べ替えるには、昇順の場合 ASC を指定し、降順の場合 DESC を指定します。ORDER BY 句では、列ごとに並べ替え順序を指定できます。

ORDER BY 句で複数の列を指定した場合、1 番目の列によって、結果の行の 1 番目の順序が決定されます。それ以降の列は、1 番目の並べ替えで同じ値だった行の順序を決定します。次の例では、Sql.Features テーブルを、地物の種類と名前で並べ替えています。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'World Topographical Features';
   select Name, Type
      from sql.features
      order by Type desc, Name;
```

注: PROC SQL のデフォルトの並べ替え順序が昇順であるため、ASC キーワードの指定は任意です。

アウトプット 2.17 並べ替え順序の指定

World Topographical Features

Name	Type
Angel Falls	Waterfall
Niagara Falls	Waterfall
Tugela Falls	Waterfall
Yosemite	Waterfall
Andaman	Sea
Baltic	Sea
Bering	Sea
Black	Sea
Caribbean	Sea
Gulf of Mexico	Sea
Hudson Bay	Sea
Mediterranean	Sea

計算列を基準に並べ替える

ORDER BY 句では、列のエイリアスを指定して、計算列によって並べ替えることができます。次の例では、人口密度を計算してから、計算された Density 列に対して並べ替えを実行しています。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'World Population Densities per Square Mile';
   select Name, Population format=comma12., Area format=comma8.,
          Population/Area as Density format=comma10.
      from sql.countries
      order by Density desc;
```

アウトプット 2.18 計算列を基準に並べ替える

World Population Densities per Square Mile

Name	Population	Area	Density
Hong Kong	5,857,414	400	14,644
Singapore	2,887,301	200	14,437
Luxembourg	405,980	100	4,060
Malta	370,633	100	3,706
Maldives	254,495	100	2,545
Bangladesh	126,387,850	57,300	2,206
Bahrain	591,800	300	1,973
Taiwan	21,509,839	14,000	1,536
Channel Islands	146,436	100	1,464
Barbados	258,534	200	1,293
Korea, South	45,529,277	38,300	1,189
Mauritius	1,128,057	1,000	1,128

列の位置を基準に並べ替える

列の位置を数値で指定して、SELECT 句内の任意の列によって並べ替えることができます。名前のかわりに位置を指定して、エイリアスが設定されていない計算列によって並べ替えることができます。次の例では、計算される Density 列に対してエイリアスを割り当てていません。そのかわり、ORDER BY 句内の列の位置 4 が、SELECT 句内の計算列の位置を参照しています。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'World Population Densities per Square Mile';
   select Name, Population format=comma12., Area format=comma8.,
          Population/Area format=comma10. label='Density'
      from sql.countries
      order by 4 desc;
```

注: PROC SQL は、列にラベルが割り当てられている場合、エイリアスが設定されていない列の見出しとして、そのラベルを使用します。

アウトプット 2.19 列の位置を基準に並べ替える

World Population Densities per Square Mile

Name	Population	Area	Density
Hong Kong	5,857,414	400	14,644
Singapore	2,887,301	200	14,437
Luxembourg	405,980	100	4,060
Malta	370,633	100	3,706
Maldives	254,495	100	2,545
Bangladesh	126,387,850	57,300	2,206
Bahrain	591,800	300	1,973
Taiwan	21,509,839	14,000	1,536
Channel Islands	146,436	100	1,464
Barbados	258,534	200	1,293
Korea, South	45,529,277	38,300	1,189
Mauritius	1,128,057	1,000	1,128

選択していない列を基準に並べ替える

クエリに含まれていない列によってクエリ結果を並べ替えることができます。たとえば、次のクエリでは、Sql.Countries テーブルのすべての行を返し、クエリに Population 列が含まれていないにもかかわらず、人口によってそれらの行を並べ替えています。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'Countries, Sorted by Population';
   select Name, Continent
      from sql.countries
      order by Population;
```

アウトプット 2.20 *選択していない列を基準に並べ替える*

Countries, Sorted by Population

Name	Continent
Vatican City	Europe
Tuvalu	Oceania
Nauru	Oceania
Leeward Islands	Central America and Caribbean
Turks and Caicos Islands	Central America and Caribbean
Cayman Islands	Central America and Caribbean
San Marino	Europe
Liechtenstein	Europe
Gibraltar	Europe
Monaco	Europe
Saint Kitts and Nevis	Central America and Caribbean
Marshall Islands	Oceania

異なるソートシーケンスの指定

SORTSEQ=は、ソートシーケンスを指定する PROC SQL ステートメントのオプションです。このソートシーケンスは、クエリに ORDER BY 句が含まれる場合に PROC SQL によって使用されます。このオプションは、使用するオペレーティングシステムのデフォルトのソートシーケンス以外のソートシーケンスを使用する場合にのみ使用します。使用可能な値には、ASCII、EBCDIC、英語以外のいくつかの言語などがあります。たとえば、EBCDIC ソートシーケンスをサポートするオペレーティングシステムでは、PROC SQL ステートメントで次のオプションを使用して、ソートシーケンスを EBCDIC に設定できます。

```
proc sql sortseq=ebcdic;
```

SAS 9.4 のメンテナンスリリース 3 では、SORTSEQ ステートメントオプションによって言語照合がサポートされます。詳細については、“SORTSEQ=*sort-table* | LINGUISTIC” (229 ページ)を参照してください。

注: SORTSEQ=は、ORDER BY 句にのみ影響を与えます。オペレーティングシステムによって設定されている WHERE 句のデフォルトの比較演算は、無効にされません。

動作環境の情報
使用しているオペレーティングシステムのデフォルトおよびその他の並べ替え順の詳細については、使用しているオペレーティングシステムに関する SAS のマニュアルを参照してください。

欠損値を含む列の並べ替え

PROC SQL は、NULL または欠損値を並べ替えてから、文字データまたは数値データを並べ替えます。したがって、昇順の並べ替えを指定した場合、クエリ結果には最初に欠損値が表示されます。

次の例では、Continents テーブルの行を LowPoint 列によって並べ替えています。

```
libname sql 'SAS-library';

proc sql;
   title 'Continents, Sorted by Low Point';
   select Name, LowPoint
      from sql.continents
      order by LowPoint;
```

3 つの大陸の LowPoint 列に欠損値が含まれているため、これらの大陸が最初に出力に表示されています。このクエリでは 2 番目の並べ替えを指定していないため、LowPoint 列の同じ値を持つ行(出力の最初の 3 行など)が特定の順序で表示されていないことに注意してください。通常、明示的に並べ替え順序を指定しなければ、PROC SQL の出力が特定の順序になることは保証されません。

アウトプット 2.21 *欠損値を含む列の並べ替え*

Continents, Sorted by Low Point

Name	LowPoint
Central America and Caribbean	
Antarctica	
Oceania	
Europe	Caspian Sea
Asia	Dead Sea
North America	Death Valley
Africa	Lake Assal
Australia	Lake Eyre
South America	Valdes Peninsula

条件を満たす行の取得

WHERE 句を使用して、ある条件を満たす行のみをテーブルから取り出すことができます。WHWRE 句には、選択されない列など、テーブル内の任意の列を含めることができます。

単一の WHERE 句の使用

次の例では、WHERE 句を使用して、ヨーロッパ大陸に存在するすべての国とそれらの人口を検索しています。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'Countries in Europe';
   select Name, Population format=comma10.
      from sql.countries
      where Continent = 'Europe';
```

アウトプット 2.22 単一の WHERE 句の使用

Countries in Europe

Name	Population
Albania	3,407,400
Andorra	64,634
Austria	8,033,746
Belarus	10,508,000
Belgium	10,162,614
Bosnia and Herzegovina	4,697,040
Bulgaria	8,887,111
Channel Islands	146,436
Croatia	4,744,505
Czech Republic	10,511,029
Denmark	5,239,356
England	49,293,170

比較に基づく行の取得

WHERE 句で比較演算子を使用して、データのさまざまなサブセットを選択できます。使用可能な比較演算子を次の表に示します。

表 2.2 比較演算子

記号	ニーモニック	定義	例
=	EQ	等しい	`where Name = 'Asia';`

記号	ニーモニック	定義	例
^=または~=または¬=または<>	NE	等しくない	**where Name ne 'Africa';**
>	GT	より大きい	**where Area > 10000;**
<	LT	より小さい	**where Depth < 5000;**
>=	GE	以上	**where Statehood >= '01jan1860'd;**
<=	LE	以下	**where Population <= 5000000;**

次の例では、人口が 5,000,000 人よりも多い州のみを含めることによって、Sql.UnitedStates テーブルをサブセット化しています。

```
libname sql 'SAS-library';

proc sql;
   title 'States with Populations over 5,000,000';
   select Name, Population format=comma10.
      from sql.unitedstates
      where Population gt 5000000
      order by Population desc;
```

アウトプット 2.23 比較に基づく行の取得

States with Populations over 5,000,000

Name	Population
California	31,518,948
New York	18,377,334
Texas	18,209,994
Florida	13,814,408
Pennsylvania	12,167,566
Illinois	11,813,091
Ohio	11,200,790
Michigan	9,571,318
New Jersey	7,957,196
North Carolina	7,013,950
Georgia	6,985,572
Virginia	6,554,851
Massachusetts	6,071,816
Indiana	5,769,553
Washington	5,307,322
Missouri	5,285,610
Tennessee	5,149,273
Wisconsin	5,087,770
Maryland	5,014,048

複数の条件を満たす行の取得

論理演算子(ブール演算子)を使用して、2 つ以上の式を含む WHERE 句を作成できます。使用可能な論理演算子を次の表に示します。

表 2.3 論理(ブール)演算子

記号	ニーモニック	定義	例
&	AND	前と後の条件が両方とも TRUE になる必要があることを指定します。	`Continent = 'Asia' and Population > 5000000`

記号	ニーモニック	定義	例
!または|または¦	OR	前または後の条件のいずれかが TRUE になる必要があることを指定します。	`Population < 1000000 or Population > 5000000`
^または~または¬	NOT	後の条件が FALSE になる必要があることを指定します。	`Continent not 'Africa'`

次の例では、2 つの式を使用して、アフリカに存在し、20,000,000 人よりも多い人口を有する国のみを含めています。

```
libname sql 'SAS-library';

proc sql;
   title 'Countries in Africa with Populations over 20,000,000';
   select Name, Population format=comma10.
      from sql.countries
      where Continent = 'Africa' and Population gt 20000000
      order by Population desc;
```

アウトプット 2.24 *複数の条件を満たす行の取得*

Countries in Africa with Populations over 20,000,000

Name	Population
Nigeria	99,062,003
Egypt	59,912,259
Ethiopia	59,291,170
South Africa	44,365,873
Congo, Democratic Republic of	43,106,529
Sudan	29,711,229
Morocco	28,841,705
Kenya	28,520,558
Tanzania	28,263,033
Algeria	28,171,132
Uganda	20,055,584

注: かっこを使用して、次に示すような複数の(つまり、複合的な)式を含む WHERE 句の可読性を改善できます。

```
where (Continent = 'Africa' and Population gt 2000000) or
      (Continent = 'Asia' and Population gt 1000000)
```

その他の条件演算子の使用

概要:その他の条件付き演算子の使用

WHERE 句では、さまざまな条件付き演算子を使用できます。使用可能なその他の演算子を次の表に示します。

表 2.4 条件付き演算子

演算子	定義	例
ANY	サブクエリから取得した一連の値のうちの少なくとも 1 つが特定の条件を満たす必要があることを指定します。	`where Population > any (select Population from sql.countries)`
ALL	サブクエリから取得したすべての値が特定の条件を満たす必要があることを指定します。	`where Population > all (select Population from sql.countries)`
BETWEEN-AND	値がある範囲(境界を含む)に含まれるかどうかを検証します。	`where Population between 1000000 and 5000000`
CONTAINS	指定された文字列が値に含まれるかどうかを検証します。	`where Continent contains 'America';`
EXISTS	サブクエリによって取得された値が存在するかどうかを検証します。	`where exists (select * from sql.oilprod);`
IN	値がリストの値のいずれかに一致するかどうかを検証します。	`where Name in ('Africa', 'Asia');`
IS NULL または IS MISSING	欠損値を検証します。	`where Population is missing;`
LIKE	値が指定されたパターンに一致するかどうかを検証します。[1]	`where Continent like 'A%';`
=*	値の発音が指定された値と似ているかどうかを検証します。	`where Name =* 'Tiland';`

注: これらすべての演算子の前に NOT 演算子を付加して、否定条件を作成できます。

1 パーセント記号(%)は、任意の数の文字に一致します。アンダーライン(_)は任意の 1 文字に一致します。

IN 演算子の使用

IN 演算子を使用すると、指定したリストに含まれる値を含めることができます。次の例では、IN 演算子を使用して、Sql.Features テーブル内の山と滝のみを含めています。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'World Mountains and Waterfalls';
   select Name, Type, Height format=comma10.
      from sql.features
      where Type in ('Mountain', 'Waterfall')
      order by Height;
```

アウトプット 2.25　IN 演算子の使用

World Mountains and Waterfalls

Name	Type	Height
Niagara Falls	Waterfall	193
Yosemite	Waterfall	2,425
Tugela Falls	Waterfall	3,110
Angel Falls	Waterfall	3,212
Kosciusko	Mountain	7,310
Pico Duarte	Mountain	10,417
Cook	Mountain	12,349
Matterhorn	Mountain	14,690
Wilhelm	Mountain	14,793
Mont Blanc	Mountain	15,771
Ararat	Mountain	16,804
Vinson Massif	Mountain	16,864

IS MISSING 演算子の使用

IS MISSING 演算子を使用して、値が欠損している列を含む行を識別できます。次の例では、大陸に存在しない国を選択しています。つまり、これらの国の Continent 列には欠損値が含まれています。

```
proc sql;
   title 'Countries with Missing Continents';
   select Name, Continent
      from sql.countries
      where Continent is missing;
```

注: IS NULL 演算子は、IS MISSING 演算子と同じであり、IS MISSING 演算子で置き換えることができます。

アウトプット 2.26 *IS MISSING 演算子の使用*

Countries with Missing Continents

Name	Continent
Bermuda	
Iceland	
Kalaallit Nunaat	

BETWEEN-AND 演算子の使用

値の範囲に基づいて行を選択するには、BETWEEN-AND 演算子を使用します。この例では、赤道から 5 度以内の緯度を含む国を選択します。

```
proc sql outobs=12;
   title 'Equatorial Cities of the World';
   select City, Country, Latitude
      from sql.worldcitycoords
      where Latitude between -5 and 5;
```

注: これらの例で使用されるテーブルでは、赤道から南の緯度の値は負になります。グリニッジ子午線から西の経度の値も負になります。

注: BETWEEN-AND 演算子は境界を含むため、BETWEEN-AND 式に指定した値も結果に含まれます。

アウトプット 2.27 BETWEEN-AND 演算子の使用

Equatorial Cities of the World

City	Country	Latitude
Belem	Brazil	-1
Fortaleza	Brazil	-4
Bogota	Colombia	4
Cali	Colombia	3
Brazzaville	Congo	-4
Quito	Ecuador	0
Cayenne	French Guiana	5
Accra	Ghana	5
Medan	Indonesia	3
Palembang	Indonesia	-3
Nairobi	Kenya	-1
Kuala Lumpur	Malaysia	4

LIKE 演算子の使用

LIKE 演算子を使用して、パターマッチングに基づいて行を選択できます。たとえば、次のクエリは、*Z* の文字で始まる任意の文字数の国名または *a* の文字で終わる 5 文字の国名を持つ Sql.Countries テーブル内のすべての国を返します。

```
libname sql 'SAS-library';

proc sql;
   title1 'Country Names that Begin with the Letter "Z"';
   title2 'or Are 5 Characters Long and End with the Letter "a"';
   select Name
      from sql.countries
      where Name like 'Z%' or Name like '____a';
```

アウトプット 2.28 *LIKE 演算子の使用*

Country Names that Begin with the Letter "Z" or Are 5 Characters Long and End with the Letter "a"

Name
China
Ghana
India
Kenya
Libya
Malta
Syria
Tonga
Zambia
Zimbabwe

パーセント記号(%)とアンダーライン(_)は、ワイルドカード文字です。LIKE 比較演算子を使用したパターンマッチングの詳細については、次を参照してください。7 章, “SQL プロシジャ” (213 ページ)

切り捨て文字列の比較演算子の使用

切り捨て文字列の比較演算子は、2 つの文字列の比較に使用されます。これらの演算子は、比較を実行する前に、長い文字列が短い文字列の長さと同じになるように PROC SQL によって切り捨てられる点が、従来の比較演算子とは異なります。切り捨ては内部で実行され、どちらのオペランドも永続的には変更されません。切り捨て文字列の比較演算子を次の表に示します。

表 2.5 *切り捨て文字列の比較演算子*

記号	定義	例
EQT	切り捨て文字列に等しい	`where Name eqt 'Aust';`
GTT	切り詰められた文字列より大きい	`where Name gtt 'Bah';`
LTT	切り詰められた文字列より小さい	`where Name ltt 'An';`
GET	切り詰められた文字列と等しいかより大きい	`where Country get 'United A';`
LET	切り詰められた文字列と等しいかより小さい	`where Lastname let 'Smith';`

記号	定義	例
NET	切り詰められた文字列と等しくない	`where Style net 'TWO';`

次の例では、**'New '**で始まる名前を持つアメリカの州のリストを返しています。

```
proc sql;
   title '"New" U.S. States';
   select Name
      from sql.unitedstates
      where Name eqt 'New ';
```

アウトプット 2.29 *切り捨て文字列の比較演算子の使用*

"New" U.S. States

Name
New Hampshire
New Jersey
New Mexico
New York

欠損値を含む WHERE 句の使用

WHERE 句で指定した列に欠損値が含まれる場合、クエリから予期しない結果が返される可能性があります。たとえば、次のクエリは、深さが 500 フィートよりも浅い Sql.Features テーブルのすべての地物を返します。

```
libname sql 'SAS-library';

/* incorrect output */

proc sql outobs=12;
   title 'World Features with a Depth of Less than 500 Feet';
   select Name, Depth
      from sql.features
      where Depth lt 500
      order by Depth;
```

アウトプット 2.30 欠損値を伴うWHERE 句の使用(不正な出力)

World Features with a Depth of Less than 500 Feet

Name	Depth
Kalahari	.
Nile	.
Citlaltepec	.
Lena	.
Mont Blanc	.
Borneo	.
Rub al Khali	.
Amur	.
Yosemite	.
Cook	.
Mackenzie-Peace	.
Mekong	.

しかし、PROC SQL が欠損値を非欠損値よりも小さいと見なすため、深さが示されない地物も結果に含まれています。この問題を回避するには、欠損値をチェックしてクエリ結果から欠損値を除外するように、次のように WHERE 式を修正します。

```
libname sql 'SAS-library';

/* corrected output */

proc sql outobs=12;
   title 'World Features with a Depth of Less than 500 Feet';
   select Name, Depth
      from sql.features
      where Depth lt 500 and Depth is not missing
      order by Depth;
```

アウトプット 2.31 *欠損値を伴うWHERE 句の使用(修正された出力)*

World Features with a Depth of Less than 500 Feet

Name	Depth
Baltic	180
Aral Sea	222
Victoria	264
Hudson Bay	305
North	308

データの要約

概要: データの要約

集計関数(要約関数)を使用して、テーブルのデータに関する統計量の要約を生成できます。集計関数は、1 つ以上の列のデータをまとめる方法を PROC SQL に指示します。集計関数の引数に 1 つの列を指定した場合、その列の値が計算されます。複数の引数を指定した場合、記述した引数(列)が計算されます。

注: SQL 集計関数内で複数の引数を使用した場合、その関数は SQL 集計(要約)関数とは見なされなくなります。類似する名前の Base SAS 関数が存在する場合、PROC SQL は、その Base SAS 関数を実行します。返される結果は、現在の行の値に基づきます。類似する名前の Base SAS 関数が存在しない場合、エラーが発生します。たとえば、AVG 関数に対して複数の引数を使用した場合、Base SAS には AVG 関数が存在しないため、エラーが発生します。

集計関数を使用するときに GROUP BY 句を使用しなければ、PROC SQL はその関数をテーブル全体に適用します。集計関数は、SELECT 句内または HAVING 句内で使用できます。

注: テーブル内のデータのグループごとの要約を生成する方法については、次を参照してください。“データのグループ化” (64 ページ)

集計関数の使用

使用可能な集計関数を次の表に示します。

表 2.6 *集計関数*

関数	定義
AVG、MEAN	値の平均
COUNT、FREQ、N	非欠損値の数
CSS	修正平方和

関数	定義
CV	変動係数(パーセント)
MAX	最大値
MIN	最小値
NMISS	欠損値の数
PRT	より大きなスチューデントの t の絶対値を得る確率
RANGE	値の範囲
STD	標準偏差
STDERR	平均の標準誤差
SUM	値の合計
SUMWGT	WEIGHT 変数値の合計 [1]
T	母集団の平均値がゼロに等しいという仮説を検定するためのスチューデントの t 値
USS	無修正平方和
VAR	分散

注: PROC SQL では、その他のほとんどの SAS 関数を使用できますが、それらは集計関数として扱われません。

WHERE 句を使用したデータの要約

概要:WHERE 句を使用したデータの要約

WHERE 句を使用することによって、集計関数(要約関数)を使用できます。使用可能な集計関数の完全な一覧については、次を参照してください。表 2.6 (56 ページ)

MEAN 関数と WHERE 句の併用

この例では、MEAN 関数を使用して、Sql.WorldTemps テーブル内の各国の年間平均気温を求めています。WHERE 句によって、平均気温が 75 度よりも高い国が返されます。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'Mean Temperatures for World Cities';
   select City, Country, mean(AvgHigh, AvgLow)
```

1 SQL プロシジャでは、それぞれの行の重みは 1 です。

```
      as MeanTemp
   from sql.worldtemps
   where calculated MeanTemp gt 75
   order by MeanTemp desc;
```

注: 計算列を参照するには、CALCULATED キーワードを使用する必要があります。

アウトプット 2.32 *MEAN 関数と WHERE 句の併用*

Mean Temperatures for World Cities

City	Country	MeanTemp
Lagos	Nigeria	82.5
Manila	Philippines	82
Bangkok	Thailand	82
Singapore	Singapore	81
Bombay	India	79
Kingston	Jamaica	78
San Juan	Puerto Rico	78
Calcutta	India	76.5
Havana	Cuba	76.5
Nassau	Bahamas	76.5

合計の表示

次の例では、SUM 関数を使用して、Sql.OilRsrvs テーブル内のすべての国についての合計石油埋蔵量を返しています。

```
libname sql 'SAS-library';

proc sql;
   title 'World Oil Reserves';
   select sum(Barrels) format=comma18. as TotalBarrels
      from sql.oilrsrvs;
```

注: この SUM 関数は、集計値以外の値が SELECT 句に含まれていないため、要求された合計について 1 行の出力を生成します。

アウトプット 2.33 *合計の表示*

World Oil Reserves

TotalBarrels
878,300,000,000

複数の行から 1 行にデータを組み合わせる

前述の例では、PROC SQL は、複数行のデータの情報をまとめて 1 行を出力していました。具体的には、世界各国の石油埋蔵量をまとめて、すべての国についての合計を求めていました。行をまとめること、つまりロールアップは、次の条件が存在する場合に実行されます。

- 集計関数内で指定された列のみが SELECT 句に含まれている。
- WHERE 句が存在し、SELECT 句で指定された列のみがそれに含まれる。

要約統計量の再マージ

次の例では、MAX 関数を使用して Sql.Countries テーブル内の最大の人口を求め、それを MaxPopulation という列に表示しています。MAX 関数などの集計関数を使用すると、すべての行について同じ計算が繰り返される場合があります。これは、PROC SQL がデータをマージし直すときに常に発生します。次の条件のいずれかが存在する場合、必ず再マージが発生します。

- SELECT 句が、集計関数を含む列を参照しており、かつ GROUP BY 句に記述されていない他の 1 つ以上の列を参照している。
- ORDER BY 句が、SELECT 句によって参照されていない列を参照している。

注: クエリがデータを再マージすると、PROC SQL は、データの再マージが実行されたことを示す注釈をログに表示します。

この例では、PROC SQL は、テーブル内の最大の人口である中国の人口を出力します。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'Largest Country Populations';
   select Name, Population format=comma20.,
          max(Population) as MaxPopulation format=comma20.
      from sql.countries
      order by Population desc;
```

アウトプット 2.34 要約統計量の再マージ

Largest Country Populations

Name	Population	MaxPopulation
China	1,202,215,077	1,202,215,077
India	929,009,120	1,202,215,077
United States	263,294,808	1,202,215,077
Indonesia	202,393,859	1,202,215,077
Brazil	160,310,357	1,202,215,077
Russia	151,089,979	1,202,215,077
Bangladesh	126,387,850	1,202,215,077
Japan	126,345,434	1,202,215,077
Pakistan	123,062,252	1,202,215,077
Nigeria	99,062,003	1,202,215,077
Mexico	93,114,708	1,202,215,077
Germany	81,890,690	1,202,215,077

場合によっては、集計関数を使用して、その計算結果を別の計算で使用できるようにする必要があることがあります。これを実行するには、PROC SQL の 1 つのクエリを作成するだけですみます。これによって、両方の計算が自動的に実行されます。このタイプの操作も、PROC SQL でのデータの再マージを引き起こします。

たとえば、世界の総人口に対する各国の人口の割合を求める場合、次のタスクを実行する 1 つのクエリを作成します。

- SUM 関数を使用して、世界の総人口を求めます。
- 各国の人口を、世界の総人口で割ります。

PROC SQL は、内部クエリを実行して合計を求め、次に別の内部クエリを実行し、各国の人口を合計で割ります。

```
libname sql 'SAS-library';

proc sql outobs=12;
   title 'Percentage of World Population in Countries';
   select Name, Population format=comma14.,
          (Population / sum(Population) * 100) as Percentage
          format=comma8.2
      from sql.countries
      order by Percentage desc;
```

アウトプット 2.35 集計関数の使用

Percentage of World Population in Countries

Name	Population	Percentage
China	1,202,215,077	21.10
India	929,009,120	16.30
United States	263,294,808	4.62
Indonesia	202,393,859	3.55
Brazil	160,310,357	2.81
Russia	151,089,979	2.65
Bangladesh	126,387,850	2.22
Japan	126,345,434	2.22
Pakistan	123,062,252	2.16
Nigeria	99,062,003	1.74
Mexico	93,114,708	1.63
Germany	81,890,690	1.44

重複しない値に集計関数を使用する

重複しない値のカウント

集計関数で DISTINCT を使用して、集計関数で列の一意の値のみを使用するように指定できます。

次のクエリは、Sql.Countries テーブル内の、一意の欠損していない大陸の数を返します。

```
libname sql 'SAS-library';

proc sql;
   title 'Number of Continents in the Countries Table';
   select count(distinct Continent) as Count
      from sql.countries;
```

アウトプット 2.36 COUNT 関数での DISTINCT の使用

Number of Continents in the COUNTRIES Table

Count
8

注: **select count(distinct *)**を使用して、テーブル内の一意の行をカウントすることはできません。PROC SQL はどの列の重複する値を削除するかがわからないため、このコードを実行するとエラーが発生します。

非欠損値のカウント

前述の例と、DISTINCT キーワードを使用しない次のクエリとを比べてください。このクエリは、Sql.Countries テーブル内に出現する欠損していない大陸を、重複する値を含めてすべてカウントします。

```
libname sql 'SAS-library';

proc sql;
   title 'Countries for Which a Continent is Listed';
   select count(Continent) as Count
      from sql.countries;
```

アウトプット 2.37 *COUNT 関数で DISTINCT を使用しない場合の影響*

Countries for Which a Continent is Listed

Count
205

すべての行のカウント

前述の 2 つの例では、Continent 列に欠損値が含まれる国は、COUNT 関数によって無視されます。テーブル内のすべての行の数を、大陸に存在しない国を含めてすべて取得するには、SELECT 句で次のコードを使用します。

```
proc sql;
   title 'Number of Countries in the Sql.Countries Table';
   select count(*) as Number
      from sql.countries;
```

アウトプット 2.38 *COUNT 関数を使用したテーブル内のすべての行のカウント*

Number of Countries in the SQL.COUNTRIES Table

Number
208

欠損値を含むデータの要約

概要:欠損値を含むデータの要約

欠損値を含むデータに対して集計関数を使用した場合、多くの集計関数が欠損値を無視するため、期待する情報が得られないことがあります。

欠損値が原因のエラーの検索

AVG 関数は、非欠損値のみの平均を返します。次次のクエリは、Sql.Features テーブル内の 3 つの地物(エンジェルフォール、アマゾン川およびナイル川)の平均長を計算します。

```
libname sql 'SAS-library';

/* unexpected output */

proc sql;
   title 'Average Length of Angel Falls, Amazon and Nile Rivers';
   select Name, Length, avg(Length) as AvgLength
      from sql.features
      where Name in ('Angel Falls', 'Amazon', 'Nile');
```

アウトプット 2.39 *欠損値によって発生するエラーの検出(予想外の出力)*

Average Length of Angel Falls, Amazon and Nile Rivers

Name	Length	AvgLength
Amazon	4000	4072.5
Angel Falls	.	4072.5
Nile	4145	4072.5

エンジェルフォールの長さが格納されていないため、平均には、アマゾン川とナイル川の値のみが含まれます。したがって、この平均には、予想外の出力結果が含まれています。

前述の例の結果と、次のクエリの結果を比べてください。このクエリには、欠損値を処理する COALESCE 式が含まれています。

```
/* modified output */

proc sql;
   title 'Average Length of Angel Falls, Amazon and Nile Rivers';
   select Name, Length, coalesce(Length, 0) as NewLength,
                avg(calculated NewLength) as AvgLength
      from sql.features
      where Name in ('Angel Falls', 'Amazon', 'Nile');
```

アウトプット 2.40 *欠損値によって発生するエラーの検出(変更後の出力)*

Average Length of Angel Falls, Amazon and Nile Rivers

Name	Length	NewLength	AvgLength
Amazon	4000	4000	2715
Angel Falls	.	0	2715
Nile	4145	4145	2715

データのグループ化

GROUP BY 句は、指定された 1 つ以上の列によってデータをグループ化します。GROUP BY 句を使用する場合、SELECT 句または HAVING 句で集計関数も使用して、グループごとのデータの要約方法を PROC SQL に指示します。PROC SQL は、グループごとに個別に集計関数を計算します。

1 つ列を基準にグループ化する

次の例では、すべての国の人口を合計して、各大陸の総人口を求めています。

```
libname sql 'SAS-library';

proc sql;
   title 'Total Populations of World Continents';
   select Continent, sum(Population) format=comma14. as TotalPopulation
      from sql.countries
      where Continent is not missing
      group by Continent;
```

注: 大陸が表示されない国は、WHERE 句によって除外されます。

アウトプット 2.41 *1 つ列を基準にグループ化する*

Total Populations of World Continents

Continent	TotalPopulation
Africa	710,529,592
Asia	3,381,858,879
Australia	18,255,944
Central America and Caribbean	66,815,930
Europe	813,481,724
North America	384,801,818
Oceania	5,342,368
South America	317,568,801

要約しないグループ化

集計関数を使用せずに GROUP BY 句を使用した場合、PROC SQL は GROUP BY 句を ORDER BY 句と同様に扱い、そのように処理されたことを知らせるメッセージがログに表示されます。次の例では、Sql.WorldTemps テーブル内の各都市の最高気温と最低気温の情報を、国別にグループ化することを試みています。

```
libname sql 'SAS-library';

proc sql outobs=12;
```

```
title 'High and Low Temperatures';
select City, Country, AvgHigh, AvgLow
   from sql.worldtemps
   group by Country;
```

この出力とログは、PROC SQL が GROUP BY 句を ORDER BY 句に変換していることを示しています。

アウトプット 2.42 *集計関数を使用しないグループ化*

High and Low Temperatures

City	Country	AvgHigh	AvgLow
Algiers	Algeria	90	45
Buenos Aires	Argentina	87	48
Sydney	Australia	79	44
Vienna	Austria	76	28
Nassau	Bahamas	88	65
Hamilton	Bermuda	85	59
Sao Paulo	Brazil	81	53
Rio de Janeiro	Brazil	85	64
Quebec	Canada	76	5
Montreal	Canada	77	8
Toronto	Canada	80	17
Beijing	China	86	17

ログ 2.2 *集計関数を使用しないグループ化(ログの一部)*

```
 WARNING:A GROUP BY clause has been transformed into an ORDER BY clause because
neither the SELECT clause nor the optional HAVING clause of the associated table-
expression referenced a summary function.
```

複数の列を基準にグループ化する

複数の列によってグループ化するには、GROUP BY 句内で、複数の列名をカンマで区切ります。任意の列を選択して、集計関数を使用できます。次の例では、Location と Type の両方によってグループ化し、Sql.Features テーブル内の各場所の砂漠と湖の合計面積(平方マイル)を生成しています。

```
libname sql 'SAS-library';

proc sql;
   title 'Total Square Miles of Deserts and Lakes';
   select Location, Type, sum(Area) as TotalArea format=comma16.
      from sql.features
```

```
        where type in ('Desert', 'Lake')
        group by Location, Type;
```

アウトプット 2.43 *複数の列を基準にグループ化する*

Total Square Miles of Deserts and Lakes

Location	Type	TotalArea
Africa	Desert	3,725,000
Africa	Lake	50,958
Asia	Lake	25,300
Australia	Desert	300,000
Canada	Lake	12,275
China	Desert	500,000
Europe - Asia	Lake	143,550
North America	Desert	140,000
North America	Lake	77,200
Russia	Lake	11,780
Saudi Arabia	Desert	250,000

データのグループ化と並べ替え

グループ化された結果を、ORDER BY 句を使用して並べ替えることができます。次の例では、前述の例に ORDER BY 句を追加して、Location 列の順序を昇順から降順に変更しています。

```
libname sql 'SAS-library';

proc sql;
   title 'Total Square Miles of Deserts and Lakes';
   select Location, Type, sum(Area) as TotalArea format=comma16.
      from sql.features
      where type in ('Desert', 'Lake')
      group by Location, Type
      order by Location desc;
```

アウトプット 2.44 ORDER BY 句を使用したグループ化

Total Square Miles of Deserts and Lakes

Location	Type	TotalArea
Saudi Arabia	Desert	250,000
Russia	Lake	11,780
North America	Lake	77,200
North America	Desert	140,000
Europe - Asia	Lake	143,550
China	Desert	500,000
Canada	Lake	12,275
Australia	Desert	300,000
Asia	Lake	25,300
Africa	Desert	3,725,000
Africa	Lake	50,958

欠損値を含むグループ化

欠損値が原因のグループ化のエラーの検索

列に欠損値が含まれる場合、PROC SQL は、欠損値を 1 つのグループとして扱います。これによって、予想外の結果が得られる場合があります。

この例では、Sql.Countries テーブルの Continent 列に欠損値がいくつか含まれるため、それらの欠損値がまとめられて 1 つのグループが形成され、Continent 列に欠損値を含む国の合計面積がそのグループに含められます。

```
libname sql 'SAS-library';

/* unexpected output */

proc sql outobs=12;
   title 'Areas of World Continents';
   select Name format=$25.,
          Continent,
          sum(Area) format=comma12. as TotalArea
      from sql.countries
      group by Continent
      order by Continent, Name;
```

バミューダ、アイスランドおよびグリーンランドが実際には同じ大陸に含まれていないため、この出力は正しくありません。しかし、PROC SQL は、これらがすべて Continent 列に欠損文字値を含んでいるため、これらをそのように処理します。

アウトプット 2.45 欠損値によって発生するグループ化エラーの検出(予想外の出力)

Areas of World Continents

Name	Continent	TotalArea
Bermuda		876,800
Iceland		876,800
Kalaallit Nunaat		876,800
Algeria	Africa	11,299,595
Angola	Africa	11,299,595
Benin	Africa	11,299,595
Botswana	Africa	11,299,595
Burkina Faso	Africa	11,299,595
Burundi	Africa	11,299,595
Cameroon	Africa	11,299,595
Cape Verde	Africa	11,299,595
Central African Republic	Africa	11,299,595

前述の例のクエリを修正するには、結果から欠損値を除外する WHERE 句を記述します。

```
/* modified output */

proc sql outobs=12;
   title 'Areas of World Continents';
   select Name format=$25.,
          Continent,
          sum(Area) format=comma12. as TotalArea
      from sql.countries
      where Continent is not missing
      group by Continent
      order by Continent, Name;
```

アウトプット 2.46 クエリの修正による欠損値に起因するエラーの回避(変更後の出力)

Areas of World Continents

Name	Continent	TotalArea
Algeria	Africa	11,299,595
Angola	Africa	11,299,595
Benin	Africa	11,299,595
Botswana	Africa	11,299,595
Burkina Faso	Africa	11,299,595
Burundi	Africa	11,299,595
Cameroon	Africa	11,299,595
Cape Verde	Africa	11,299,595
Central African Republic	Africa	11,299,595
Chad	Africa	11,299,595
Comoros	Africa	11,299,595
Congo	Africa	11,299,595

注: SUM 関数などの集計関数を使用すると、すべての行について同じ計算が繰り返される場合があります。これは、PROC SQL がデータをマージし直すときに常に発生します。再マージの詳細については、“要約統計量の再マージ” (59 ページ)を参照してください。

グループ化されたデータのフィルタリング

概要:グループ化されたデータのフィルタリング

GROUP BY 句と共に HAVING 句を使用して、グループ化されたデータをフィルタリングできます。HAVING 句は、WHERE 句が個々の行に影響を与えるのと同様の方法で、グループに影響を与えます。HAVING 句を使用すると、PROC SQL は、HAVING 式を満たすグループのみを表示します。

単一の HAVING 句の使用

次の例では、Sql.Features テーブル内の地物を種類によってグループ化し、次に島、大洋および海の数のみを表示しています。

```
libname sql 'SAS-library';

proc sql;
   title 'Numbers of Islands, Oceans, and Seas';
   select Type, count(*) as Number
      from sql.features
```

```
group by Type
having Type in ('Island', 'Ocean', 'Sea')
order by Type;
```

アウトプット 2.47 単一の HAVING 句の使用

Numbers of Islands, Oceans, and Seas

Type	Number
Island	6
Ocean	4
Sea	13

HAVING と WHERE の選択

HAVING 句と WHERE 句の違いを、下の表に示します。HAVING 句はデータのグループを操作するときに使用されるため、HAVING 句を含むクエリには、通常、次の要素も含まれます。

- GROUP BY 句
- 集計関数

ヒント HAVING 句は、グループに関して WHERE 句に似ています。

注: GROUP BY 句なしで HAVING 句を使用する場合、クエリが少なくとも 1 つの集計関数を参照しているならば、SQL プロシジャは、入力データを、それが単一のデータグループにより提供されるものであるかのように扱います。

表 2.7 HAVING 句と WHERE 句の違い

HAVING 句の特徴	WHERE 句の特徴
通常は、テーブルの行のグループを含めるため、または除外するための条件の指定に使用されます。	テーブルの個々の行を含めるため、または除外するための条件の指定に使用されます。
クエリ内で GROUP BY 句と共に使用された場合、必ずその GROUO BY 句の後に記述されます。	クエリ内で GROUP BY 句と共に使用された場合、必ずその GROUO BY 句よりも前に記述されます。
GROUP BY 句の影響を受けます。GROUP BY 句が存在しない場合、HAVING 句は WHERE 句と同様に扱われます。	GROUP BY 句の影響を受けません。
GROUP BY 句およびすべての集計関数の実行後に処理されます。	GROUP BY 句(存在する場合)およびすべての集計関数の実行前に処理されます。

HAVING と集計関数の併用

次のクエリは、16 カ国以上の国が存在するすべての大陸の人口を返します。

```
libname sql 'SAS-library';

proc sql;
   title 'Total Populations of Continents with More than 15 Countries';
   select Continent,
          sum(Population) as TotalPopulation format=comma16.,
          count(*) as Count
      from sql.countries
      group by Continent
      having count(*) gt 15
      order by Continent;
```

HAVING 式には、それぞれのグループ内の行の数をカウントする COUNT 関数が含まれています。

アウトプット 2.48 *COUNT 関数を含む HAVING 句の使用*

Total Populations of Continents with More than 15 Countries

Continent	TotalPopulation	Count
Africa	710,529,592	53
Asia	3,381,858,879	48
Central America and Caribbean	66,815,930	25
Europe	813,481,724	51

クエリの検証

VALIDATE ステートメントを使用して、クエリを PROC SQL にサブミットせずに、クエリの構文の正しさをチェックできます。PROC SQL は、構文が正しいかどうかを示すメッセージをログに表示します。

```
libname sql 'SAS-library';

proc sql;
   validate
      select Name, Statehood
         from sql.unitedstates
         where Statehood lt '01Jan1800'd;
```

ログ 2.3 *クエリの検証(ログの一部)*

```
3 proc sql; 4 validate 5 select Name, Statehood 6 from sql.unitedstates 7 where
Statehood lt '01Jan1800'd; NOTE:PROC SQL statement has valid syntax.
```

次の例は、無効なクエリと、それに対応するログメッセージを示しています。

```
libname sql 'SAS-library';

proc sql;
```

```
validate
   select Name, Statehood
   from sql.unitedstates
   where lt '01Jan1800'd;
```

ログ 2.4 *無効なクエリの検証(ログの一部)*

```
3 proc sql; 4 validate 5 select Name, Statehood 6 from sql.unitedstates 7 where
lt '01Jan1800'd; ------------ 22 76 ERROR 22-322:Syntax error, expecting one of
the following:!, !!, &, *, **, +, -, /, <, <=, <>, =, >, >=, ?, AND, CONTAINS,
EQ, GE, GROUP, GT, HAVING, LE, LIKE, LT, NE, OR, ORDER, ^=, |, ||, ~=.ERROR
76-322:Syntax error, statement will be ignored.注:The SAS System stopped
processing this step because of errors.
```

3 章

複数のテーブルからのデータの取得

はじめに **73**

結合を使用し、複数のテーブルからデータを選択する **74**
概要:結合を使用し、複数のテーブルからデータを選択する 74
内部結合 75
外部結合 85
特殊結合 88
結合に Coalesce 関数を使用する 91
DATA ステップのマッチマージと PROC SQL 結合の比較 92

サブクエリを使用したデータの選択 **96**
1 つの値のサブクエリ 96
複数の値のサブクエリ 97
相関するサブクエリ 98
値のグループの存在のテスト 99
複数のネストされたサブクエリの水準 100
サブクエリを使用し組み合わせる 101

結合とサブクエリの使用が必要な場合 **102**

セット演算子を使用したクエリの組み合わせ **103**
複数のクエリの結果の操作 103
両方のクエリから重複しない行を作成する(UNION) 104
クエリ 1 の結果のみに含まれる行を作成する(EXCEPT) 105
両方のクエリの結果に属する行を作成する(INTERSECT) 106
クエリの結果の連結(OUTER UNION) 107
1 回目または 2 回目のクエリから行を作成する 108

はじめに

この章では、次のタスクの実行方法について説明します。

- テーブルを一緒に結合することによって、複数のテーブルからデータを選択します。
- サブクエリを使用し、別のテーブルのデータ値に基づいて 1 つのテーブルからデータを選択します。
- セット演算子を使用して、複数のクエリの結果を結合します。

注: 特に断らない限り、この章で示される PROC SQL 操作は、テーブルだけでなくビューにも適用されます。ビューの詳細については、次を参照してください。4 章, “テーブルとビューの作成および更新” (111 ページ)

結合を使用し、複数のテーブルからデータを選択する

概要:結合を使用し、複数のテーブルからデータを選択する

レポートで必要となるデータが、複数のテーブルに存在する場合があります。それらのテーブルからデータを選択するために、クエリでテーブルを結合します。テーブルを結合することによって、データが 1 つのテーブルに含まれているかのように、複数のテーブルからデータを選択できるようになります。結合しても元のテーブルは変更されません。

最も基本的な結合のタイプは、単純に 2 つのテーブルを SELECT ステートメントの FROM 句に記述することです。次のクエリは、アウトプット 3.1 (74 ページ) に示す 2 つのテーブルを結合して、アウトプット 3.2 (75 ページ)を作成しています。

```
proc sql;
   title 'Table One and Table Two';
   select *
      from one, two;

proc sql;
  title 'Table One';
  select * from one;

  title 'Table Two';
  select * from two;

quit;
```

アウトプット 3.1 *テーブル 1、テーブル 2*

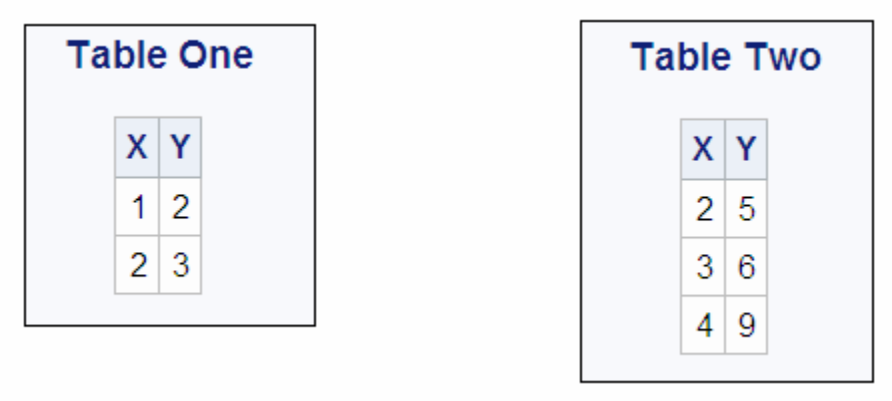


Table One

X	Y
1	2
2	3

Table Two

X	Y
2	5
3	6
4	9

アウトプット 3.2 *テーブル 1 とテーブル 2 のデカルト積*

Table One and Table Two

X	Y	X	Y
1	2	2	5
1	2	3	6
1	2	4	9
2	3	2	5
2	3	3	6
2	3	4	9

この方法でテーブルを結合すると、テーブルのデカルト積が返されます。最初のテーブルのそれぞれの行は、2 番目のテーブルのすべての行と結合されます。このクエリを実行すると、次のメッセージが SAS ログに書き込まれます。

ログ 3.1 *デカルト積のログメッセージ*

```
注:このクエリの実行には 1 つ以上のデカルト積結合の実行が含まれており、それらを最適化することはできません。
```

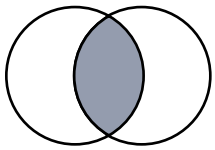
大きなテーブルのデカルト積は、巨大になることがあります。通常、必要なのはデカルト積のサブセットです。結合タイプを宣言することによってサブセットを指定します。

結合には、次の 2 つのタイプがあります。

- 内部結合は、FROM 句で指定された 1 つ以上の他のテーブルに、一致する行が 1 つ以上存在するすべての行を、結果テーブルで返します。
- 外部結合は、拡張された内部結合です。他のテーブルのどの行とも一致しない行が、結合結果に追加されます。外部結合には、右、左および完全の3種類があります。

内部結合

内部結合の概要



内部結合は、2 番目のテーブルの行に一致する、最初のテーブルの行のサブセットのみを返します。一致する値を照合するための列を、WHERE 句で指定できます。

次のコードは、前述のクエリに WHERE 句を追加したものです。この WHERE 句は、テーブル 2 の X 列の値に一致する X 列の値を持つテーブル 1 の行のみを出力で表示するように指定しています。このクエリの出力を、アウトプット 3.2 (75 ページ)と比較してください。

```
proc sql;
   title 'Table One and Table Two';
   select * from one, two
      where one.x=two.x;
```

アウトプット 3.3 *結合されたテーブル 1 とテーブル 2*

Table One and Table Two

X	Y	X	Z
2	3	2	5

それぞれのテーブルには一致する X 列の値が 1 つしかないため、出力には 1 つの行のみが含まれています。内部結合では、一致する行のみが選択されます。外部結合は、一致しない行を返すことができます。これについては、“外部結合” (85 ページ)で説明されています。

WHERE 句での列名には、その列のテーブル名の接頭語が付加されていることに注意してください。これは列名の修飾と呼ばれ、複数のテーブルに同じ名前の列が存在する列を指定する場合、必要になります。列名を修飾することによって、不明瞭な列参照が作成されるのを防ぎます。

テーブルのエイリアスの使用

テーブルのエイリアスは、テーブルの一時的な代替名です。テーブルのエイリアスは、FROM 句で指定します。テーブルのエイリアスは、結合において列名の修飾に使用され、テーブル名を省略することによってクエリを読みやすくします。

次の例では、OilProd テーブルと OilRsrvs テーブルを Country 列で結合することによって、各国の石油産出量とその国の石油備蓄量を比較しています。Country 列が両方のテーブルで共通しているため、それらの列はテーブルのエイリアスで修飾されています。列名にテーブル名の接頭語を付加することによって、列を修飾することもできます。

注: AS キーワードは任意です。

```
libname sql 'SAS-library';

proc sql outobs=6;
   title 'Oil Production/Reserves of Countries';
   select * from sql.oilprod as p, sql.oilrsrvs as r
      where p.country = r.country;
```

アウトプット 3.4 テーブルのエイリアスの使用による列名の省略

Oil Production/Reserves of Countries

Country	BarrelsPerDay	Country	Barrels
Algeria	1,400,000	Algeria	9,200,000,000
Canada	2,500,000	Canada	7,000,000,000
China	3,000,000	China	25,000,000,000
Egypt	900,000	Egypt	4,000,000,000
Indonesia	1,500,000	Indonesia	5,000,000,000
Iran	4,000,000	Iran	90,000,000,000

それぞれのテーブルの Country 列が表示されていることに注意してください。通常、結合が正しく機能すると判断したら、SELECT 句には一致する列のうちの 1 つのみを含めるようにします。

結合の出力の順序の指定

結合されたテーブルの出力を、いずれかのテーブルの 1 つ以上の列で並べ替えることができます。次の例の出力は、BarrelsPerDay 列の値の降順で並べ替えられています。BarrelsPerDay 列が OilProd テーブルにしか存在しないため、BarrelsPerDay を修飾する必要はありません。

```
libname sql 'SAS-library';

proc sql outobs=6;
   title 'Oil Production/Reserves of Countries';
   select p.country, barrelsperday 'Production', barrels 'Reserves'
      from sql.oilprod p, sql.oilrsrvs r
      where p.country = r.country
      order by barrelsperday desc;
```

アウトプット 3.5 結合されたテーブルの出力の並べ替え

Oil Production/Reserves of Countries

Country	Production	Reserves
Saudi Arabia	9,000,000	260,000,000,000
United States of America	8,000,000	30,000,000,000
Iran	4,000,000	90,000,000,000
Norway	3,500,000	11,000,000,000
Mexico	3,400,000	50,000,000,000
China	3,000,000	25,000,000,000

INNER JOIN キーワードを使用した内部結合の作成

INNER JOIN キーワードを使用してテーブルを結合できます。ON 句は、結合する列を指定するための WHERE 句に置き換わるものです。PROC SQL では、これらのキーワードは、主に他の結合(OUTER JOIN、RIGHT JOIN および LEFT JOIN)との互換性を保つために提供されています。INNER JOIN を ON 句と共に使用することで、FROM 句でテーブルを指定し、WHERE 句で結合する列を指定するのと同じ機能を利用できます。

次のコードは、前述のコードと同じ出力を生成しますが、INNER JOIN 構造を使用しています。

```
proc sql ;
   select p.country, barrelsperday 'Production', barrels 'Reserves'
      from sql.oilprod p inner join sql.oilrsrvs r
           on p.country = r.country
   order by barrelsperday desc;
```

比較演算子を使用してテーブルを結合する

WHERE 句で等号(=)以外の比較演算子を使用してテーブルを結合できます。比較演算子の詳細については、“比較に基づく行の取得” (45 ページ)を参照してください。この例では、エジプトのカイロよりも南に位置する、USCityCoords テーブル内のアメリカの都市がすべて選択されています。複合 WHERE 句では、WorldCityCoords テーブル内の都市、カイロを指定し、小なり記号(lt)演算子を使用して、USCityCoords と WorldCityCoords をそれらの Latitude 列で結合しています。

```
libname sql 'SAS-library';

proc sql;
   title 'US Cities South of Cairo, Egypt';
   select us.City, us.State, us.Latitude, world.city, world.latitude
      from sql.worldcitycoords world, sql.uscitycoords us
      where world.city = 'Cairo' and
            us.latitude lt world.latitude;
```

アウトプット 3.6 *比較演算子を使用したテーブルの結合*

US Cities South of Cairo, Egypt

City	State	Latitude	City	Latitude
Honolulu	HI	21	Cairo	30
Key West	FL	24	Cairo	30
Miami	FL	26	Cairo	30
San Antonio	TX	29	Cairo	30
Tampa	FL	28	Cairo	30

このクエリを実行すると、次のメッセージが SAS ログに書き込まれます。

ログ 3.2 *比較クエリのログメッセージ*

```
注:このクエリの実行には 1 つ以上のデカルト積結合の実行が含まれており、それらを最適化することはできません。
```

WHERE 句で一致する列を指定せずに、テーブルを結合するクエリを実行した場合、このメッセージが表示されることを思い出してください。PROC SQL は、不等号演算子を使用してテーブルを結合した場合にも、このメッセージを表示します。

結合への null 値の影響

ほとんどのデータベース製品は、ヌルを特別扱いし、結合ではそれらを照合しません。PROC SQL は、ヌルを欠損値として扱い、結合では一致として扱います。結合では、どのヌルも、他の同じタイプ(文字または数値)のヌルと一致します。

次の例では、テーブル 1 とテーブル 2 を B 列で結合しています。両方のテーブルの B 列にはヌル値が存在します。出力において、テーブル 1 の C 行のヌル値が、テーブル 2 のすべてのヌル値と一致していることに注目してください。恐らくこれは、目的とする結合結果ではありません。

```
proc sql;
   title 'One and Two Joined';
   select one.a 'One', one.b, two.a 'Two', two.b
      from one, two
      where one.b=two.b;
```

アウトプット 3.7 ヌル値を含むテーブルの結合

Table One

a	b
a	1
b	2
c	.
d	4

Table Two

a	b
a	1
b	2
c	.
d	4
e	.
f	.

One and Two Joined

One	b	Two	b
a	1	a	1
b	2	b	2
c	.	c	.
d	4	d	4
c	.	e	.
c	.	f	.

One and Two Joined

One	b	Two	b
a	1	a	1
b	2	b	2
c	.	c	.
d	4	d	4
c	.	e	.
c	.	f	.

結合で非欠損値のみを指定するには、IS NOT MISSING 演算子を使用します。

```
proc sql;
   select one.a 'One', one.b, two.a 'Two', two.b
      from one, two
      where one.b=two.b and
            one.b is not missing;
```

アウトプット 3.8 *ヌル値を含むテーブルの結合に IS NOT MISSING を追加した結果*

One and Two Joined

One	b	Two	b
a	1	a	1
b	2	b	2
d	4	d	4

複数列の結合の作成

複数の列の値の組み合わせで列を識別する場合、必要なすべての列を結合で使用します。たとえば、都市名は複数の国に存在する場合があります。正しい都市を選択するには、結合クエリの WHERE 句で、都市と国の両方の列を指定する必要があります。

この例では、Countries テーブルと WorldCityCoords テーブルを結合することで、首都の緯度と経度を表示しています。この例では、出力の行数を最小にするために、WHERE 式の最初の部分で、L の文字で始まる名前の首都を Countries テーブルから選択しています。

```
libname sql 'SAS-library';

proc sql;
   title 'Coordinates of Capital Cities';
   select Capital format=$12., Name format=$12.,
          City format=$12., Country format=$12.,
          Latitude, Longitude
      from sql.countries, sql.worldcitycoords
```

```
        where Capital like 'L%' and
                    Capital = City;
```

ロンドンは、Countries テーブル内で首都として 1 回現れます。しかし、WORLDCITYCOORDS では、ロンドンは 2 回現れます。1 回はイギリスの都市として、もう 1 回はカナダの都市としてです。WHERE 式で **`Capital = City`** のみを指定すると、次の不正な出力が得られます。

アウトプット 3.9 *首都の座標の選択(不正な出力)*

Coordinates of Capital Cities

Capital	Name	City	Country	Latitude	Longitude
La Paz	Bolivia	La Paz	Bolivia	-16	-69
London	England	London	Canada	43	-81
Lima	Peru	Lima	Peru	-13	-77
Lisbon	Portugal	Lisbon	Portugal	39	-10
London	England	London	England	51	0

この出力では、内部結合によって、イギリスのロンドンがカナダのロンドンとイギリスのロンドンの両方に不正に一致していること注目してください。国名の列(Countries.Name と WorldCityCoords.Country)も一緒に結合することで、行は正しく一致します。

```
libname sql 'SAS-library';

proc sql;
   title 'Coordinates of Capital Cities';
   select Capital format=$12., Name format=$12.,
          City format=$12., Country format=$12.,
          latitude, longitude
      from sql.countries, sql.worldcitycoords
      where Capital like 'L%' and
            Capital = City and
            Name = Country;
```

アウトプット 3.10 *首都の座標の選択(正しい出力)*

Coordinates of Capital Cities

Capital	Name	City	Country	Latitude	Longitude
La Paz	Bolivia	La Paz	Bolivia	-16	-69
Lima	Peru	Lima	Peru	-13	-77
Lisbon	Portugal	Lisbon	Portugal	39	-10
London	England	London	England	51	0

3 つ以上のテーブルからのデータの選択

必要なデータが 3 つ以上のテーブルに存在する場合があります。たとえば、アメリカの州都の座標を表示する場合、州都を含む UnitedStates テーブルと、アメリカの都市の座標を含む USCityCoords テーブルを結合する必要があります。正確な(前述した例と同様の)結合結果を得るために、都市を州と共に結合する必要があるため、都市と州の両方の列でテーブルを結合する必要があります。

UnitedStates.Capital 列と USCityCoords.City 列を結合することによる都市の結合は、直接的です。しかし、UnitedStates テーブルでは、Name 列に完全な州名が含まれています。USCityCoords テーブルでは、州は郵便番号で指定されています。したがって、2 つのテーブルをそれらの州の列で直接結合することは不可能です。この問題を解決するには、州名とその郵便番号の両方を含む PostalCodes テーブルを中間テーブルとして使用して、UnitedStates と USCityCoords の間に正しいリレーションシップを作成する必要があります。この正しい解決策によって、UnitedStates.Name 列と PostalCodes.Name 列が(完全な州名が一致して)結合され、PostalCodes.Code 列と USCityCoords.State 列が(州の郵便番号が一致して)結合されます。

```
libname sql 'SAS-library';

title 'Coordinates of State Capitals';
proc sql outobs=10;
   select us.Capital format=$15., us.Name 'State' format=$15.,
          pc.Code, c.Latitude, c.Longitude
      from sql.unitedstates us, sql.postalcodes pc,
           sql.uscitycoords c
      where us.Capital = c.City and
            us.Name = pc.Name and
            pc.Code = c.State;
```

アウトプット 3.11 3 つ以上のテーブルからのデータの選択

Coordinates of State Capitals

Capital	State	Code	Latitude	Longitude
Montgomery	Alabama	AL	32	-86
Juneau	Alaska	AK	58	-134
Phoenix	Arizona	AZ	33	-113
Little Rock	Arkansas	AR	35	-92
Sacramento	California	CA	38	-121
Denver	Colorado	CO	40	-105
Hartford	Connecticut	CT	42	-73
Dover	Delaware	DE	39	-76
Tallahassee	Florida	FL	31	-84
Atlanta	Georgia	GA	34	-84

自己結合を使用した 1 つのテーブル内のリレーションシップの表示

テーブル内の値の間の比較関係を表示する必要がある場合、同じテーブル内の列の結合が必要になることがあります。テーブルをそのテーブル自身と結合することを、自己結合または再帰結合とよびます。自己結合は、テーブルの内部コピーを作成し、このテーブルとそのコピーを結合する PROC SQL と考えることができます。

たとえば、次のコードでは、自己結合を使用して、他の都市の年間平均最低気温に等しい年間平均最高気温を持つ都市を選択しています。

```
libname sql 'SAS-library';

 proc sql;
    title "Cities' High Temps = Cities' Low Temps";
    select High.City format $12., High.Country format $12.,
           High.AvgHigh, ' | ',
           Low.City format $12., Low.Country format $12.,
           Low.AvgLow
       from sql.worldtemps High, sql.worldtemps Low
       where High.AvgHigh = Low.AvgLow and
             High.city ne Low.city and
             High.country ne Low.country;
```

WorldTemps テーブルには、2 つのエイリアス、**High** および **Low** が割り当てられていることに注目してください。概念的には、これによってテーブルのコピーが作成され、テーブルとそのコピーとの間で結合を作成できるようになります。WHERE 句では、最低気温に等しい最高気温を持つ行を選択しています。

また、この WHERE 句では、都市がその都市自身と結合されないようにしています (**City ne City** および **Country ne Country**)が、この場合、同じ都市で最高気温と最低気温が同じになることはまずありません。

アウトプット 3.12 テーブルとそれ自身との結合(自己結合)

Cities' High Temps = Cities' Low Temps

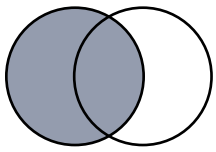
City	Country	AvgHigh		City	Country	AvgLow
Amsterdam	Netherlands	70	|	San Juan	Puerto Rico	70
Auckland	New Zealand	75	|	Lagos	Nigeria	75
Auckland	New Zealand	75	|	Manila	Philippines	75
Berlin	Germany	75	|	Lagos	Nigeria	75
Berlin	Germany	75	|	Manila	Philippines	75
Bogota	Colombia	69	|	Bangkok	Thailand	69
Cape Town	South Africa	70	|	San Juan	Puerto Rico	70
Copenhagen	Denmark	73	|	Singapore	Singapore	73
Dublin	Ireland	68	|	Bombay	India	68
Glasgow	Scotland	65	|	Nassau	Bahamas	65
London	England	73	|	Singapore	Singapore	73
Oslo	Norway	73	|	Singapore	Singapore	73
Reykjavik	Iceland	57	|	Caracas	Venezuela	57
Stockholm	Sweden	70	|	San Juan	Puerto Rico	70

外部結合

外部結合の概要

外部結合は、拡張された内部結合です。1 つのテーブルの他のテーブルのどの行とも一致しない行が、結合結果に追加されます。結果として得られる出力には、結合のソーステーブルの一致する行と一致しない行が含まれます。一致しない行の、一致しないテーブルの列には、ヌル値が設定されます。WHERE 句のかわりに ON 句を使用して、テーブルを結合するための 1 つ以上の列を指定します。ただし、続けて WHERE 句を使用して、クエリ結果のサブセットを作成できます。

左外部結合を使用して一致しない行を含める



左外部結合は、一致する行と、右側テーブル内のどの行とも一致しない左側テーブル(FROM 句に記述した最初のテーブル)の行を表示します。左外部結合は、LEFT JOIN キーワードおよび ON キーワードで指定します。

たとえば、国際都市の首都の座標を表示するには、左外部結合を使用して、首都を含む Countries テーブルと、都市の座標を含む WorldCityCoords テーブルを結合しま

す。左外部結合は、都市が WorldCityCoords に存在するかどうかに関わらず、すべての首都を表示します。内部結合を使用すると、一致する都市が WorldCityCoords に存在する首都のみが表示されます。

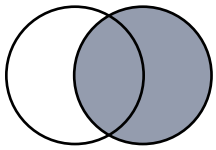
```
libname sql 'SAS-library';

proc sql outobs=10;
   title 'Coordinates of Capital Cities';
   select Capital format=$20., Name 'Country' format=$20.,
          Latitude, Longitude
      from sql.countries a left join sql.worldcitycoords b
           on a.Capital = b.City and
              a.Name = b.Country
      order by Capital;
```

アウトプット 3.13 *Countries と WorldCityCoords の左外部結合*

Coordinates of Capital Cities

Capital	Country	Latitude	Longitude
	Channel Islands	.	.
Abu Dhabi	United Arab Emirates	.	.
Abuja	Nigeria	.	.
Accra	Ghana	5	0
Addis Ababa	Ethiopia	9	39
Algiers	Algeria	37	3
Almaty	Kazakhstan	.	.
Amman	Jordan	32	36
Amsterdam	Netherlands	52	5
Andorra la Vella	Andorra	.	.

右外部結合を使用して一致しない行を含める



RIGHT JOIN キーワードと ON キーワードを使用して指定する右外部結合は、左外部結合の逆になります。つまり、出力には、一致するすべての行と共に、右側テーブル (FROM 句で 2 番目に記述したテーブル)の一致しない行が含まれます。次の例は、前の結合の例の逆になります。つまり、右外部結合を使用して WorldCityCoords テーブルのすべての都市を選択し、都市が国の首都である場合(つまり、都市が Countries テーブルに存在する場合)にのみ、その人口を表示しています。

```
libname sql 'SAS-library';

proc sql outobs=10;
```

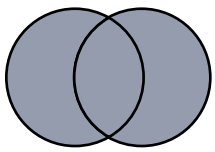
```
title 'Populations of Capitals Only';
select City format=$20., Country 'Country' format=$20.,
       Population
   from sql.countries right join sql.worldcitycoords
        on Capital = City and
           Name = Country
   order by City;
```

アウトプット 3.14 *Countries と WorldCityCoords の右外部結合*

Populations of Capitals Only

City	Country	Population
Abadan	Iran	.
Acapulco	Mexico	.
Accra	Ghana	17395511
Adana	Turkey	.
Addis Ababa	Ethiopia	59291170
Adelaide	Australia	.
Aden	Yemen	.
Ahmenabad	India	.
Algiers	Algeria	28171132
Alice Springs	Australia	.

完全外部結合を使用してすべての行を選択する



FULL JOIN キーワードと ON キーワードで指定される完全外部結合は、一致する行と一致しない行をすべて選択します。この例では、WorldCityCoords と Countires の City 列と Capital 列について、一致する行と一致しない行のうちの最初の 10 行表示しています。ここで、番号記号(#)は、ラベル内で行区切り文字として使用されています。

```
libname sql 'SAS-library';

proc sql outobs=10;
   title 'Populations and/or Coordinates of World Cities';
   select City '#City#(WorldCityCoords)' format=$20.,
          Capital '#Capital#(Countries)' format=$20.,
          Population, Latitude, Longitude
      from sql.countries full join sql.worldcitycoords
           on Capital = City and
              Name = Country;
```

アウトプット 3.15 Countries と WorldCityCoords の完全外部結合

Populations and/or Coordinates of World Cities

City (WORLDCITYCOORDS)	Capital (COUNTRIES)	Population	Latitude	Longitude
		146436	.	.
Abadan		.	30	48
	Abu Dhabi	2818628	.	.
	Abuja	99062003	.	.
Acapulco		.	17	-100
Accra	Accra	17395511	5	0
Adana		.	37	35
Addis Ababa	Addis Ababa	59291170	9	39
Adelaide		.	-35	138
Aden		.	13	45

特殊結合

特殊結合の概要

結合の 3 つのタイプ(クロス結合、和結合、および自然結合)は、標準的な結合タイプの特殊ケースです。

クロス結合を使用してすべての行の組み合わせを含める

クロス結合は、デカルト積です。つまり、2 つのテーブルの積を返します。デカルト積と同様に、クロス結合の出力は、WHERE 句によって制限できます。

この例は、テーブル 1 とテーブル 2 のクロス結合を示しています。

```
data one;
 input X Y $;
 datalines;
1 2
2 3
;


data two;
 input W Z $;
 datalines;
2 5
3 6
4 9
;
run;
```

```
proc sql;
  title 'Table One';
  select * from one;

  title 'Table Two';
  select * from two;

title;
quit;
```

アウトプット 3.16 *テーブル 1 とテーブル 2*

Table One

X	Y
1	2
2	3

Table Two

W	Z
2	5
3	6
4	9

```
proc sql;
   title 'Table One and Table Two';
   select *
      from one cross join two;
```

アウトプット 3.17 *クロス結合*

Table One and Table Two

X	Y	W	Z
1	2	2	5
1	2	3	6
1	2	4	9
2	3	2	5
2	3	3	6
2	3	4	9

従来のデカルト積と同様に、クロス積によってデカルト積に関する注釈が SAS ログに書き込まれます。

和結合を使用してすべての行を含める

和結合は、行の比較を試みることなく 2 つのテーブルを結合します。両方のテーブルのすべての列と行が結果に含まれます。和結合を使用したテーブルの結合は、

OUTER UNION セット演算子を使用したテーブルの結合に類似しています 。(“セット演算子を使用したクエリの組み合わせ” (103 ページ)を参照)和結合の出力は、WHERE 句によって制限できます。

次の例は、前にクロス結合を示すために使用されたのと同じテーブル 1 とテーブル 2 の和結合を示しています。

```
proc sql;
   select *
      from one union join two;
```

アウトプット 3.18 *和結合*

X	Y	W	Z
.		2	5
.		3	6
.		4	9
1	2	.	
2	3	.	

自然結合を使用して行を照合する

自然結合は、一致する行の判定で使用する列を自動的に各テーブルから選択します。自然結合では、PROC SQL は各テーブルで同じ名前とタイプを持つ列を特定し、それらの列の値が等しい行を一致する行として返します。ON 句は、暗に含まれています。

次の例は、“結合の出力の順序の指定” (77 ページ)の例と同じ結果を生成しています。

```
libname sql 'SAS-library';

proc sql outobs=6;
   title 'Oil Production/Reserves of Countries';
   select country, barrelsperday 'Production', barrels 'Reserve'
      from sql.oilprod natural join sql.oilrsrvs
      order by barrelsperday desc;
```

アウトプット 3.19 OilProd と OilRsrvs の自然内部結合

Oil Production/Reserves of Countries

Country	Production	Reserve
Saudi Arabia	9,000,000	260,000,000,000
United States of America	8,000,000	30,000,000,000
Iran	4,000,000	90,000,000,000
Norway	3,500,000	11,000,000,000
Mexico	3,400,000	50,000,000,000
China	3,000,000	25,000,000,000

自然結合を使用するメリットは、コーディングが簡素化されることです。ON 句は暗に含まれており、テーブルのエイリアスを使用して両方のテーブルに共通する列名を修飾する必要はありません。次の 2 つのクエリは、同じ結果を返します。

```
proc sql;
   select a.W, a.X, Y, Z
   from table1 a left join table2 b
   on a.W=b.W and a.X=b.X
   order by a.W;

proc sql;
   select W, X, Y, Z
   from table1 natural left join table2
   order by W;
```

共通する名前とタイプを持つ列が 1 つもないテーブルに対して自然結合を指定した場合、結果はデカルト積になります。WHERE 句を使用して、この出力を制限できます。

自然結合は、実行する内容について、決められた前提を適用します。そのため、自然結合を使用する前に、データについて完全に把握しておく必要があります。そうしないと、予期しない結果や不正な結果を招く場合があります。たとえば、2 つのテーブルにただ 1 つ共通する列が存在することを期待しているのに、実際には 2 つ存在するような場合です。FEEDBACK オプションを使用して、PROC SQL がどのようにクエリを実装しているかを正確に調べることができます。FEEDBACK オプションの詳細については“PROC SQL オプションを使用し、クエリを作成、デバックする” (140 ページ)を参照してください。

自然結合は、共通する列の等しい値を持つすべてのペアに基づいて結合することを前提にしています。不等号演算子または他の比較演算子に基づいて結合する場合は、標準的な内部結合または外部結合の構文を使用してください。

結合に Coalesce 関数を使用する

前述の例からわかるように、外部結合では、一致しない行には欠損値が含まれます。COALESCE 関数を使用して、テーブルの、データを含む行のみが表示されるように列を上書きできます。COALESCE が列のリストを引数として受け取り、最初に検出した非欠損値を返すことを思い出してください。

次の例は、COALESCE 関数を前述の例に追加して、Countries.Capital 列、WorldCityCoords.City 列、および Countries.Name 列を上書きしています。一部の島に

は首都が存在しないため、COALESCE に対して引数として Countries.Name が指定されています

```
libname sql 'SAS-library';

proc sql outobs=10;
   title 'Populations and/or Coordinates of World Cities';
   select coalesce(Capital, City,Name)format=$20. 'City',
          coalesce(Name, Country) format=$20. 'Country',
          Population, Latitude, Longitude
      from sql.countries full join sql.worldcitycoords
           on Capital = City and
           Name = Country;
```

アウトプット 3.20 *Countries と WorldCityCoords の完全外部結合での COALESCE の使用*

Populations and/or Coordinates of World Cities

City	Country	Population	Latitude	Longitude
Channel Islands	Channel Islands	146436	.	.
Abadan	Iran	.	30	48
Abu Dhabi	United Arab Emirates	2818628	.	.
Abuja	Nigeria	99062003	.	.
Acapulco	Mexico	.	17	-100
Accra	Ghana	17395511	5	0
Adana	Turkey	.	37	35
Addis Ababa	Ethiopia	59291170	9	39
Adelaide	Australia	.	-35	138
Aden	Yemen	.	13	45

COALESCE は、内部結合と外部結合の両方で使用できます。COALESCE の詳細については、“欠損値の置換” (35 ページ)を参照してください。

DATA ステップのマッチマージと PROC SQL 結合の比較

DATA ステップのマッチマージと PROC SQL の結合の比較の概要

多くの SAS ユーザーは、DATA ステップを使用したデータセットのマージについてよく知っています。このセクションでは、マージと結合について比較します。DATA ステップのマッチマージと、PROC SQL の結合は、同じ結果を生成します。ただし、マッチマージと結合の間には、結合する前にテーブルを並べ替える必要があるかどうかという大きな違いがあります。

すべての値が一致する場合

BY 変数のすべての値が一致し、重複する BY 変数が存在しない場合、内部結合を使用してマッチマージと同じ結果を生成できます。この結果を示すために、共通して

Flight 列を含む 2 つのテーブルを次に示します。Flight の値は、両方のテーブルで同じです。

```
FltSuper                    FltDest

Flight  Supervisor          Flight  Destination

  145   Kang                  145   Brussels
  150   Miller                150   Paris
  155   Evanko                155   Honolulu
```

FltSuper と FltDest は、一致する Flight 列によってすでに並べ替えられています。DATA ステップのマージによって、アウトプット 3.21 (93 ページ)が生成されます。

```
data fltsuper;
input Flight Supervisor $;
datalines;
145   Kang
150   Miller
155   Evanko
;
data fltdest;
input Flight Destination $;
datalines;
145   Brussels
150   Paris
155   Honolulu
;
run;

data merged;
   merge fltsuper fltdest;
   by Flight;
run;

proc print data=merged noobs;
   title 'Table Merged';
run;
```

アウトプット 3.21 *すべての値が一致する場合にマージされたテーブル*

Table MERGED

Flight	Supervisor	Destination
145	Kang	Brussels
150	Miller	Paris
155	Evanko	Honolulu

PROC SQL では、事前のデータの並べ替えは不要です。次の PROC SQL の結合によって、アウトプット 3.21 (93 ページ)に示した結果と同じ結果が得られます。

```
proc sql;
   title 'Table MERGED';
   select s.flight, Supervisor, Destination
      from fltsuper s, fltdest d
```

```
        where s.Flight=d.Flight;
```

一部の値のみ一致する場合

BY 変数の一部の値のみが一致する場合、外部結合を使用してマッチマージと同じ結果を生成できます。この結果を示すために、共通して Flight 列を含む 2 つのテーブルを次に示します。Flight の値は、両方のテーブルで同じではありません。

```
FltSuper                    FltDest

Flight  Supervisor          Flight  Destination

  145   Kang                  145   Brussels
  150   Miller                150   Paris
  155   Evanko                165   Seattle
  157   Lei
```

DATA ステップのマージによって、アウトプット 3.22 (94 ページ)が生成されます。

```
data merged;
   merge fltsuper fltdest;
   by flight;
run;
proc print data=merged noobs;
   title 'Table Merged';
run;
```

アウトプット 3.22 *一部の値が一致する場合にマージされたテーブル*

Table MERGED

Flight	Supervisor	Destination
145	Kang	Brussels
150	Miller	Paris
155	Evanko	
157	Lei	
165		Seattle

PROC SQL を使用して同じ結果を得るには、2 つのテーブルの一致しない行がクエリ結果に含まれるようにするために、外部結合を使用します。さらに、COALESCE 関数を使用して、両方のテーブルの Flight 列を上書きします。次の PROC SQL の結合によって、アウトプット 3.22 (94 ページ)に示した結果と同じ結果が得られます。

```
proc sql;
   select coalesce(s.Flight,d.Flight) as Flight, Supervisor, Destination
      from fltsuper s full join fltdest d
           on s.Flight=d.Flight;
```

値の位置が重要な場合

2 つのテーブルをマージし、値の位置が重要になる場合、DATA ステップのマージの使用が必要となることがあります。この考え方を説明するために、考慮すべき 2 つのテーブルを次に示します。

```
FltSuper                    FltDest

Flight  Supervisor          Flight  Destination

  145   Kang                145     Brussels
  145   Ramirez             145     Edmonton
  150   Miller              150     Paris
  150   Picard              150     Madrid
  155   Evanko              165     Seattle
  157   Lei
```

Flight 145 では、**Kang** が **Brussels** と一致し、**Ramirez** が **Edmonton** と一致しています。この DATA ステップでは、BY グループの値の位置に基づいてデータをマージしているため、Supervisor と Destination の値が適切に一致しています。DATA ステップのマージによって、アウトプット 3.23 (95 ページ)が生成されます。

```
data merged;
   merge fltsuper fltdest;
   by flight;
run;
proc print data=merged noobs;
   title 'Table Merged';
run;
```

アウトプット 3.23 *FltSuper テーブルと FltDest テーブルのマッチマージ*

Table MERGED

Flight	Supervisor	Destination
145	Kang	Brussels
145	Ramirez	Edmonton
150	Miller	Paris
150	Picard	Madrid
155	Evanko	
157	Lei	
165		Seattle

PROC SQL は、BY グループの値の位置に従って結合を処理しません。そのかわり PROC SQL は、データ値のみに従ってデータを処理します。FltSuper と FltDest の内部結合の結果を次に示します。

```
proc sql;
   title 'Table Joined';
   select *
      from fltsuper s, fltdest d
      where s.Flight=d.Flight;
```

アウトプット 3.24 *FltSuper テーブルと FltDest テーブルの PROC SQL による結合*

Table JOINED

Flight	Supervisor	Flight	Destination
145	Kang	145	Brussels
145	Kang	145	Edmonton
145	Ramirez	145	Brussels
145	Ramirez	145	Edmonton
150	Miller	150	Paris
150	Miller	150	Madrid
150	Picard	150	Paris
150	Picard	150	Madrid

PROC SQL は、デカルト積を作成してから、WHERE 句の条件を満たす行を表示します。WHERE 句は、管理者ごとに 2 つの行を返し、目的地ごとに 1 つの行を返します。Flight には重複する値が含まれており、他に一致する列がないため、**Kang** を **Brussels** のみに関連付けたり、**Ramirez** を **Edmonton** のみに関連付けたりなどする方法はありません。

DATA ステップのマッチマージの詳細については、*SAS ステートメント: リファレンス*を参照してください。

サブクエリを使用したデータの選択

テーブル結合によって、複数のテーブルが 1 つの新しいテーブルに結合されます。かっこで囲まれたサブクエリは、1 つのテーブルの行を別のテーブルの値に基づいて選択します。サブクエリ、つまり内部クエリは、別のクエリ式の一部としてネストされたクエリ式です。サブクエリは、それを含む句に応じて、1 つの値または複数の値を返すことができます。サブクエリは、WHERE 式と HAVING 式において最も多く使用されます。

1 つの値のサブクエリ

1 つの値のサブクエリは、1 つの行と列を返します。これは、比較演算子と共に WHERE 句または HAVING 句で使用できます。このサブクエリは 1 つの値のみを返す必要があります。そうしないと、クエリは失敗し、エラーメッセージがログに出力されます。

次のクエリは、WHERE 句でサブクエリを使用して、ベルギーよりも人口の多いアメリカの州を選択しています。このクエリは最初に評価され、外側のクエリにベルギーの人口を返します。

```
libname sql 'SAS-library';

proc sql;
   title 'U.S. States with Population Greater than Belgium';
```

```
select Name 'State' , population format=comma10.
   from sql.unitedstates
   where population gt
               (select population from sql.countries
                  where name = "Belgium");
```

サブクエリが実行されると、クエリは内部で次のように処理されます。

```
proc sql;
   title 'U.S. States with Population Greater than Belgium';
   select Name 'State', population format=comma10.
      from sql.unitedstates
      where population gt 10162614;
```

外側のクエリは、ベルギーよりも人口の多い州を表示します。

アウトプット 3.25 *1 つの値のサブクエリ*

U.S. States with Population Greater than Belgium

State	Population
California	31,518,948
Florida	13,814,408
Illinois	11,813,091
New York	18,377,334
Ohio	11,200,790
Pennsylvania	12,167,566
Texas	18,209,994

複数の値のサブクエリ

複数の値のサブクエリは、1 つの列の複数の値を返すことができます。これは、ANY または ALL によって変更される IN 演算子または比較演算子を含む WHERE 式または HAVING 式で使用されます。次の例は、石油産出国の人口を表示しています。このサブクエリは、まず OilProd テーブル内にあるすべての国を返します。次に外側のクエリは、Countries テーブル内の国と、サブクエリの結果を照合します。

```
libname sql 'SAS-library';

proc sql outobs=5;
   title 'Populations of Major Oil Producing Countries';
   select name 'Country', Population format=comma15.
      from sql.countries
      where Name in
            (select Country from sql.oilprod);
```

アウトプット 3.26 IN を使用した複数の値のサブクエリ

Populations of Major Oil Producing Countries

Country	Population
Algeria	28,171,132
Canada	28,392,302
China	1,202,215,077
Egypt	59,912,259
Indonesia	202,393,859

このクエリで NOT IN 演算子を使用した場合、クエリ結果には OilProd テーブルに含まれないすべての国が含まれます。

```
libname sql 'SAS-library';

proc sql outobs=5;
   title 'Populations of NonMajor Oil Producing Countries';
   select name 'Country', Population format=comma15.
      from sql.countries
      where Name not in
            (select Country from sql.oilprod);
```

アウトプット 3.27 NOT IN を使用した複数の値のサブクエリ

Populations of NonMajor Oil Producing Countries

Country	Population
Afghanistan	17,070,323
Albania	3,407,400
Andorra	64,634
Angola	9,901,050
Antigua and Barbuda	65,644

相関するサブクエリ

前述のサブクエリは、自己完結型の、外側のクエリとは独立して実行される単純なサブクエリでした。相関するサブクエリは、外側のクエリから 1 つ以上の値を渡される必要があります。このサブクエリは、実行が完了すると、その結果を外側のクエリに返します。相関するサブクエリは、1 つまたは複数の値を返すことができます。

この例では、アフリカ大陸の各国の主要な石油備蓄量をすべて選択しています。

```
libname sql 'SAS-library';

proc sql;
```

```
title 'Oil Reserves of Countries in Africa';
select * from sql.oilrsrvs o
   where 'Africa' =
          (select Continent from sql.countries c
            where c.Name = o.Country);
```

外側のクエリは、OilRsrvs テーブルから最初の行を選択し、次に Country 列の値 (**Algeria**)をサブクエリに渡しています。この時点で、サブクエリは内部で次のように処理されています。

```
(select Continent from sql.countries c
        where c.Name = 'Algeria');
```

このサブクエリは、Countries テーブルから国を選択します。次にサブクエリは、国の大陸を外側のクエリ内の WHERE 句に返します。大陸がアフリカの場合、その国が選択されて表示されます。次に外側のクエリは、OilRsrvs テーブルからその後の各行を選択し、Country の個々の値をサブクエリに渡します。サブクエリは、外側の WHERE 句で比較するための該当する Continent の値を外側のクエリに返します。

WHERE 句が=(等号)演算子を使用していることに注意してください。サブクエリが 1 つの値のみを返す場合、=(等号)演算子を使用できます。しかし、サブクエリが複数の値を帰す場合は、IN 演算子または比較演算子を ANY または ALL と共に使用する必要があります。サブクエリで使用できる演算子の詳細については、7 章, “SQL プロシジャ” (213 ページ)を参照してください。

アウトプット 3.28 *相関するサブクエリ*

Oil Reserves of Countries in Africa

Country	Barrels
Algeria	9,200,000,000
Egypt	4,000,000,000
Gabon	1,000,000,000
Libya	30,000,000,000
Nigeria	16,000,000,000

値のグループの存在のテスト

EXISTS 条件は、一連の値の存在をテストします。EXISTS 条件の値は、サブクエリによって行が生成された場合 true になり、行が生成されない場合 false になります。反対に、NOT EXISTS 条件の値は、サブクエリが空のテーブルを生成した場合に true になります。

次の例では、アウトプット 3.28 (99 ページ)と同じ結果を生成しています。EXISTS は、アフリカ大陸で石油備蓄を保有する国の存在をチェックします。前述の例では外側のクエリに含まれていた条件 **Continent = 'Africa'**が、ここではサブクエリの WHERE 句に含まれていることに注意してください。

```
libname sql 'SAS-library';

proc sql;
   title 'Oil Reserves of Countries in Africa';
```

```
select * from sql.oilrsrvs o
   where exists
          (select Continent from sql.countries c
             where o.Country = c.Name and
                   Continent = 'Africa');
```

アウトプット 3.29 *値のグループの存在のテスト*

Oil Reserves of Countries in Africa

Country	Barrels
Algeria	9,200,000,000
Egypt	4,000,000,000
Gabon	1,000,000,000
Libya	30,000,000,000
Nigeria	16,000,000,000

複数のネストされたサブクエリの水準

最も内側のサブクエリが次の外側のサブクエリで使用される 1 つ以上の値を返すように、サブクエリをネストできます。次に、外側のサブクエリの 1 つ以上の値が次の外側のクエリで使用される、などのように処理されます。評価は常に最も内側のサブクエリから始まり、外側に向かって処理されます。

次の例では、主要な石油備蓄を保有するアフリカ各国の都市を表示しています。

1. 最初に、最も内側のクエリが評価されます。このクエリはアフリカ大陸に存在する国を返します。
2. 外側のサブクエリが評価されます。このサブクエリは、内側のサブクエリが返した国のリストと OILRSRVS 内の国を比較することによって、主要な石油備蓄を保有するアフリカ各国のサブセットを返します。
3. 最後に、外側のクエリの WHERE 句は、WorldCityCoords テーブルに存在する、外側のサブクエリの結果に一致する国の都市の座標を表示します。

```
libname sql 'SAS-library';

proc sql;
   title 'Coordinates of African Cities with Major Oil Reserves';
   select * from sql.worldcitycoords
3    where country in
  2     (select Country from sql.oilrsrvs o
           where o.Country in
        1 (select Name from sql.countries c
             where c.Continent='Africa'));
```

アウトプット 3.30 複数のネストされたサブクエリの水準

Coordinates of African Cities with Major Oil Reserves

City	Country	Latitude	Longitude
Algiers	Algeria	37	3
Cairo	Egypt	30	31
Benghazi	Libya	33	21
Lagos	Nigeria	6	3

サブクエリを使用し組み合わせる

結合とサブクエリを 1 つのクエリ内で組み合わせることができます。たとえば、USCityCoords テーブル内の各都市に最も近い都市を検索するとします。クエリは、まず都市 A を選択し、都市 A から他のすべての都市への距離を計算し、最後に都市 A から最短距離にある都市を選択する必要があります。これは、USCityCoords テーブルとそれ自身を結合し(自己結合)、次にサブクエリ内の別の自己結合を使用して都市間の最短距離を決定することによって実行できます。

座標間の距離を決定する式を次に示します。

```
SQRT(((Latitude2-Latitude1)**2) + ((Longitude2-Longitude1)**2))
```

この式の結果は、厳密には地球の湾曲によるゆがみのため正確ではありませんが、ある都市が別の都市よりも近いかどうかを例として決定するには十分正確です。

```
libname sql 'SAS-library';

proc sql outobs=10;
   title 'Neighboring Cities';
   select a.City format=$10., a.State,
          a.Latitude 'Lat', a.Longitude 'Long',
          b.City format=$10., b.State,
          b.Latitude 'Lat', b.Longitude 'Long',
          sqrt(((b.latitude-a.latitude)**2) +
               ((b.longitude-a.longitude)**2)) as dist format=6.1
      from sql.uscitycoords a, sql.uscitycoords b
      where a.city ne b.city and
            calculated dist =
            (select min(sqrt(((d.latitude-c.latitude)**2) +
                             ((d.longitude-c.longitude)**2)))
                from sql.uscitycoords c, sql.uscitycoords d
                where c.city = a.city and
                      c.state = a.state and
                      d.city ne c.city)
            order by a.city;
```

アウトプット 3.31　*サブクエリを使用し組み合わせる*

Neighboring Cities

City	State	Lat	Long	City	State	Lat	Long	dist
Albany	NY	43	-74	Hartford	CT	42	-73	1.4
Albuquerqu	NM	36	-106	Santa Fe	NM	36	-106	0.0
Amarillo	TX	35	-102	Carlsbad	NM	32	-104	3.6
Anchorage	AK	61	-150	Nome	AK	64	-165	15.3
Annapolis	MD	39	-77	Washington	DC	39	-77	0.0
Atlanta	GA	34	-84	Knoxville	TN	36	-84	2.0
Augusta	ME	44	-70	Portland	ME	44	-70	0.0
Austin	TX	30	-98	San Antoni	TX	29	-98	1.0
Baker	OR	45	-118	Lewiston	ID	46	-117	1.4
Baltimore	MD	39	-76	Dover	DE	39	-76	0.0

外側のクエリは、テーブルとそのテーブル自身を結合し、テーブル A の最初の都市 A1 とテーブル B の都市 B2(都市 A1 以外の最初の都市)の間の距離を決定しています。次に PROC SQL は、サブクエリを実行します。このサブクエリは、別の自己結合を実行し、都市 A1 と、テーブル内の都市 A1 以外のすべての都市との間の最短距離を計算します。外側のクエリは、都市 A1 と都市 B2 の間の距離が、サブクエリが計算した最短距離に等しいかどうかを調べてテストしています。それらが等しい場合、都市 A1 と都市 B2 を含む行が、それらの座標と距離と共に出力されます。

結合とサブクエリの使用が必要な場合

複数のテーブルから情報を参照する場合は、常に結合またはサブクエリを使用します。結合とサブクエリは、多くの場合同じクエリ内で一緒に使用されます。多くの場合、データ検索問題は、結合、サブクエリ、またはその両方を使用して解決できます。結合とクエリを使用する場合のいくつかのガイドラインを次に示します。

- レポートで、複数のテーブルからのデータが必要な場合は、結合を実行する必要があります。FROM 句に複数のテーブル(またはビュー)を記述すると、それらのテーブルは必ず結合されます。
- テーブル内の異なる行の関連情報を結合する必要がある場合、そのテーブルをそのテーブル自身と結合できます。
- 目的の結果を得るために複数のクエリが必要な場合、サブクエリを使用します。それぞれのサブクエリは、そのクエリに含まれるテーブルのサブセットを返します。
- メンバであるかを調べる場合、通常はサブクエリを使用します。クエリで NOT EXISTS 条件が必要な場合、NOT EXISTS はサブクエリ内でのみ動作するため、サブクエリを使用する必要があります。この原則は、EXISTS 条件の場合にも当てはまります。

- 多くのクエリは、結合またはサブクエリとして定式化できます。PROC SQL クエリオプティマイザは一部のサブクエリを結合に変更しますが、通常は結合のほうが効率的に処理されます。

セット演算子を使用したクエリの組み合わせ

複数のクエリの結果の操作

PROC SQL は、次のセット演算子を使用して、さまざまな方法で 2 つ以上のクエリの結果を結合できます。

UNION
　両方のクエリ結果から、すべて一意の行を生成します。

EXCEPT
　最初のクエリ結果のみに含まれる行を生成します。

INTERSECT
　両方のクエリ結果に共通する行を生成します。

OUTER UNION
　クエリ結果を連結します。

演算子は、次の例のように、2 つのクエリの間で使用します。

```
select columns from table
set-operator
select columns from table;
```

最後の SELECT ステートメントの後のみにセミコロンを配置します。セット演算子は、個々の列名とは無関係に、参照されるテーブル内での列の位置に基づいて 2 つのクエリの列を結合します。2 つのクエリ内の同じ相対位置にある列は、データタイプが同じである必要があります。最初のクエリ内のテーブルの列名が、出力テーブルの列名になります。3 つ以上のクエリ結果でのセット演算子の使用の詳細については、7 章, “SQL プロシジャ” (213 ページ)を参照してください。次の任意指定のキーワードを使用して、集合演算をさらに制御できます。

ALL
　これは、重複行を抑制しません。ALL キーワードを指定すると、PROC SQL は重複行を削除するための 2 回目のデータのパススルーを実行しません。したがって、ALL を使用すると、使用しない場合に比べて効率的です。ALL は、OUTER UNION 演算子では許可されません。

CORRESPONDING (CORR)
　これは、両方のテーブルで同じ名前が付けられた列を上書きします。CORR は、EXCEPT、INTERSECT および UNION と共に使用すると、両方のテーブルに含まれない列の表示を抑制します。

例では、それぞれのセット演算子は、次の 2 つのテーブルに基づいて記述され使用されます。

アウトプット 3.32 集合演算で使用されるテーブルの例

Table A

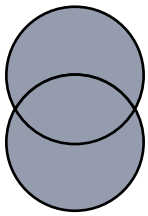
x	y
1	one
2	two
2	two
3	three

Table B

x	z
1	one
2	two
4	four

結合演算がテーブルを水平に結合するのに対して、集合演算はテーブルを垂直に結合します。そのため、それぞれのセクションに含まれる集合図は垂直に表示されています。

両方のクエリから重複しない行を作成する(UNION)



UNION 演算子は、2 つのクエリ結果を結合します。これは、両方のクエリ結果から、すべて一意の行を生成します。つまり、ある行が最初のテーブル、2 番目のテーブルまたは両方のテーブルに現れる場合、その行が返されます。UNION は、重複行を返しません。ある行が 2 回以上現れる場合、1 つの出現のみが返されます。

```
proc sql;
   title 'A UNION B';
   select * from sql.a
   union
   select * from sql.b;
```

アウトプット 3.33 両方のクエリから重複しない行を作成する(UNION)

A UNION B

x	y
1	one
2	two
3	three
4	four

ALL キーワードを使用して、出力に重複行を残すように要求できます。

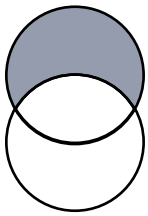
```
proc sql;
   title 'A UNION ALL B';
   select * from sql.a
   union all
   select * from sql.b;
```

アウトプット 3.34 *両方のクエリ結果からの行の生成(UNION ALL)*

A UNION ALL B

x	y
1	one
2	two
2	two
3	three
1	one
2	two
4	four

クエリ 1 の結果のみに含まれる行を作成する(EXCEPT)



EXCEP 演算子は、2 番目のクエリの結果を除く、最初のクエリの結果の行を返します。次の例では、3 と **three** の値を含む行が最初のクエリ結果(テーブル A)にのみ存在し、その行が EXCEPT によって返されます。

```
proc sql;
   title 'A EXCEPT B';
   select * from sql.a
   except
   select * from sql.b;
```

アウトプット 3.35　*クエリ 1 の結果のみに含まれる行を作成する(EXCEPT)*

A EXCEPT B

x	y
3	three

2 と two の値を含むテーブル A 内の重複行が出力に現れていないことに注意してください。EXCEPT は、2 番目のクエリ結果の行と一致しない重複行を返しません。ALL を追加すると、2 番目のクエリ結果に現れない重複行は残されます。

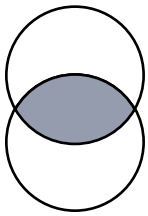
```
proc sql;
   title 'A EXCEPT ALL B';
   select * from sql.a
   except all
   select * from sql.b;
```

アウトプット 3.36　*最初のクエリ結果のみに含まれる行の生成(EXCEPT ALL)*

A EXCEPT ALL B

x	y
2	two
3	three

両方のクエリの結果に属する行を作成する(INTERSECT)



INTERSECT 演算子は、2 番目のクエリ結果にも現れる最初のクエリ結果の行を返します。

```
proc sql;
   title 'A INTERSECT B';
   select * from sql.a
   intersect
   select * from sql.b;
```

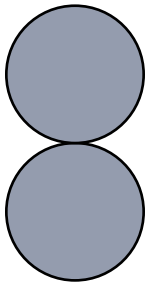
アウトプット 3.37　両方のクエリの結果に属する行を作成する(INTERSECT)

A INTERSECT B

x	y
1	one
2	two

INTERSECT ALL 演算子の出力には、最初のクエリで生成された行のうちの、2 番目のクエリで生成された行と 1 対 1 で一致する行が含まれます。この例では、INTERSECT ALL の出力は INTERSECT と同じです。

クエリの結果の連結(OUTER UNION)



OUTER UNION 演算子は、クエリの結果を連結します。次例では、テーブル A とテーブル B を連結しています。

```
proc sql;
   title 'A OUTER UNION B';
   select * from sql.a
   outer union
   select * from sql.b;
```

アウトプット 3.38　クエリ結果の連結(OUTER UNION)

A OUTER UNION B

x	y	x	z
1	one	.	
2	two	.	
2	two	.	
3	three	.	
.		1	one
.		2	two
.		4	four

OUTER UNION が 2 つのテーブルの列を上書きしていないことに注目してください。同じ位置の列を上書きするには、CORRESPONDING キーワードを使用します。

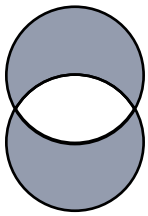
```
proc sql;
   title 'A OUTER UNION CORR B';
   select * from sql.a
   outer union corr
   select * from sql.b;
```

アウトプット 3.39 *クエリ結果の結合(OUTER UNION CORR)*

A OUTER UNION CORR B

x	y	z
1	one	
2	two	
2	two	
3	three	
1		one
2		two
4		four

1 回目または 2 回目のクエリから行を作成する



PROC SQL には、1 番目と 2 番目のテーブルから、両方に共通して現れない一意の行を返すキーワードはありません。この演算をシミュレーションできる 1 つの方法を次に示します。

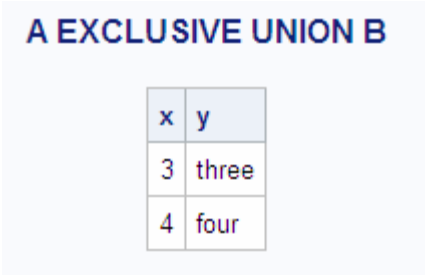
```
(query1 except query2)
union
(query2 except query1)
```

次の例は、この演算の使用方法を示しています。

```
proc sql;
   title 'A EXCLUSIVE UNION B';
   (select * from sql.a
         except
         select * from sql.b)
   union
   (select * from sql.b
         except
```

```
        select * from sql.a);
```

アウトプット 3.40 *1 回目または 2 回目のクエリから行を作成する*



A EXCLUSIVE UNION B

x	y
3	three
4	four

最初の EXCEPT は、1 番目のテーブル(テーブル A)のみから 1 つの一意の行を返します。2番目の EXCEPT は、2 番目のテーブル(テーブル B)のみから 1 つの一意の行を返します。中間にある UNION は、2 つの結果を結合します。したがって、このクエリは、2 番目のテーブルに含まれない 1 番目のテーブルの行に加えて、1 番目のテーブルに含まれない 2 番目のテーブルの行を返します。

4 章

テーブルとビューの作成および更新

はじめに **112**

テーブルの作成 **112**
列の定義からテーブルを作成する 112
テーブルをクエリの結果から作成する 113
既存のテーブルと同じテーブルの作成 115
既存のテーブルのコピー 115
データセットオプションの使用 115

テーブルへの行の挿入 **116**
SET 句を使用して行を挿入する 116
VALUES 句を使用して行を挿入する 117
クエリを使用して行を挿入する 118

テーブルのデータ値の更新 **120**
同一式で列のすべての行を更新する 120
異なる式で列の行を更新する 121
更新エラーの処理 122

行の削除 **123**

列の変更 **123**
列の追加 123
列の変更 125
列の削除 126

インデックスの作成 **127**
PROC SQL を使用したインデックスの作成 127
インデックスの作成のヒント 127
インデックスの削除 128

テーブルの削除 **128**

SAS ソフトウェアでの SQL プロシジャテーブルの使用 **128**

テーブルの一貫性制約の作成および使用 **129**

PROC SQL ビューの作成および使用 **131**
PROC SQL ビューの作成と使用の概要 131
ビューの作成 132
ビューの説明 132
ビューの更新 133
ビューに LIBNAME を埋め込む 133
ビューの削除 135
インラインビューの指定 135

SAS での SQL プロシジャビューの使用のヒント 136
SAS での SQL プロシジャビューの使用 137

はじめに

この章では、次のタスクの実行方法について説明します。

- テーブルの作成
- テーブルの更新
- 既存のテーブルの変更
- テーブルの削除
- インデックスの作成
- テーブル作成での一貫性制約の使用
- ビューの作成

テーブルの作成

CREATE TABLE ステートメントを使用して、列定義から行を含まないテーブルを作成することや、クエリ結果からテーブルを作成することができます。CREATE TABLE を使用して、既存のテーブルをコピーすることもできます。

列の定義からテーブルを作成する

列とそれらの属性を定義する CREATE TABLE ステートメントを使用して、行を含まないテーブルを新規作成できます。列の名前、タイプ、長さ、入力形式、出力形式、ラベルを指定できます。

次の CREATE TABLE ステートメントは、NewStates テーブルを作成します。

```
proc sql;
   create table sql.newstates
          (state char(2),           /* 2-character column for              */
                                    /* state abbreviation                  */

           date num                 /* column for date of entry into the US */
                informat=date9.     /* with an informat                    */
                format=date9.,      /* and format of DATE9.                */

           population num);         /* column for population               */
```

NewStates テーブルには 3 つの列があり、行数は 0 です。State の長さを変更するために、char(2)修飾子が使用されています。

テーブルが存在することを確認し、列の属性を調べるには、DESCRIBE TABLE ステートメントを使用します。次の DESCRIBE TABLE ステートメントは、CREATE TABLE ステートメントを SAS ログに書き込みます。

```
proc sql;
   describe table sql.newstates;
```

ログ 4.1 *列定義から作成されたテーブル*

```
1 proc sql; 2 describe table sql.newstates; NOTE:SQL table SQL.NEWSTATES was
created like:create table SQL.NEWSTATES( bufsize=4096 ) ( state char(2), date
num format=DATE9. informat=DATE9., population num );
```

DESCRIBE TABLE は、CREATE TABLE ステートメントを使用してテーブルを作成していなくても、CREATE TABLE ステートメントを SAS ログに書き込みます。DATASETS プロシジャ内で CONTENTS ステートメントを使用して、NewStates の説明を取得することもできます。

テーブルをクエリの結果から作成する

クエリ結果から PROC SQL テーブルを作成するには、AS キーワード付きの CREATE TABLE ステートメントを使用し、それを SELECT ステートメントの前に配置します。この方法でテーブルを作成すると、テーブルのデータは、クエリの FROM 句で参照されたテーブルまたはビューから生成されます。新しいテーブルの列名は、クエリの SELECT 句のリストに指定された名前になります。新しいテーブルの列の属性(タイプ、長さ、入力形式、出力形式、拡張属性)は、選択された元の列の属性と同じになります。

注: 複数テーブル結合や外部結合を使用して作成されるテーブルには、拡張属性はコピーされません。UNION、INTERSECT、またはマイナス演算子を使用する場合、その UNION、INTERSECT、またはマイナス演算子の前に記述されているテーブルに拡張属性が含まれている場合にのみ、拡張属性がコピーされます。

次の CREATE TABLE ステートメントは、Countries テーブルから Densities テーブルを作成します。新規作成されたテーブルは、そのテーブルへのクエリを実行しなければ SAS 出力には表示されません。OUTOBS オプションを使用すると、Densities テーブルのサイズが 10 行に制限されるため、注意してください。

```
libname sql 'SAS-library';

proc sql outobs=10;
   title 'Densities of Countries';
   create table sql.densities as
      select Name 'Country' format $15.,
             Population format=comma10.0,
             Area as SquareMiles,
             Population/Area format=6.2 as Density
         from sql.countries;

   select * from sql.densities;
```

アウトプット 4.1　*クエリ結果から作成されるテーブル*

Densities of Countries

Country	Population	SquareMiles	Density
Afghanistan	17,070,323	251825	67.79
Albania	3,407,400	11100	306.97
Algeria	28,171,132	919595	30.63
Andorra	64,634	200	323.17
Angola	9,901,050	481300	20.57
Antigua and Bar	65,644	171	383.88
Argentina	34,248,705	1073518	31.90
Armenia	3,556,864	11500	309.29
Australia	18,255,944	2966200	6.15
Austria	8,033,746	32400	247.96

次の DESCRIBE TABLE ステートメントは、CREATE TABLE ステートメントを SAS ログに書き込みます。

```
proc sql;
   describe table sql.densities;
```

ログ 4.2　*DENSITIES についての DESCRIBE TABLE ステートメントの SAS ログ*

```
注:SQL table SQL.DENSITIES was created like:create table
SQL.DENSITIES( bufsize=8192 ) ( Name char(35) format=$15. informat=$35.
label='Country', Population num format=COMMA10. informat=BEST8.
label='Population', SquareMiles num format=BEST8. informat=BEST8., Density num
format=6.2 );
```

この形式の CREATE TABLE ステートメントでは、列にラベルを割り当てず、エイリアスを割り当てて列の名前を変更しています。この例では、Area 列の名前は SquareMiles に変更され、計算された列には Density という名前が付けられています。ただし、Name 列の名前は変更されず、その表示ラベルは **Country** です。

既存のテーブルと同じテーブルの作成

既存のテーブルまたはビューと同じ列および属性を含む空のテーブルを作成するには、CREATE TABLE ステートメントで LIKE 句を使用します。次の例では、CREATE TABLE ステートメントによって、列数が 6 で行数が 0 の、Countries と同じ列属性を持つ NewCountries テーブルを作成しています。DESCRIBE TABLE ステートメントは、CREATETABLE ステートメントを SAS ログに書き込みます。

```
proc sql;
   create table sql.newcountries
      like sql.countries;

   describe table sql.newcountries;
```

ログ 4.3 *NewCountries についての DESCRIBE TABLE ステートメントの SAS ログ*

```
注:SQL table SQL.NEWCOUNTRIES was created like: create table
SQL.NEWCOUNTRIES( bufsize=65536 ) ( Name char(35) format=$35. informat=$35.,
Capital char(35) format=$35. informat=$35. label='Capital', Population num
format=BEST8. informat=BEST8. label='Population', Area num format=BEST8.
informat=BEST8., Continent char(30) format=$30. informat=$30. label='Continent',
UNDate num format=YEAR4.);
```

既存のテーブルのコピー

PROC SQL を使用してテーブルをコピーする簡単な方法は、テーブル全体を返すクエリと共に CREATE TABLE ステートメントを使用することです。次の例では、Countries に含まれるすべての列と行のコピーを含む Countries1 を作成しています。

```
create table countries1 as
   select * from sql.countries;
```

データセットオプションの使用

CREATE TABLE ステートメントでは、SAS データセットオプションを使用できます。次の CREATE TABLE ステートメントは、Countries から Countries2 を作成しています。DROP=オプションによって UNDate 列が削除されるため、UNDate は Countries2 に含まれません。

```
create table countries2 as
   select * from sql.countries(drop=UNDate);
```

テーブルへの行の挿入

テーブルにデータ値を挿入するには、INSERT ステートメントを使用します。INSERT ステートメントは、まず既存のテーブルに新しい行を追加し、次に指定した値をその行に挿入します。値は、SET 句または VALUES 句を使用して指定します。クエリの結果から得られた行を挿入することもできます。ほとんどの条件において、PROC SQL および SAS/ACCESS ビューを介してテーブルにデータを挿入できます。詳細については、“ビューの更新” (133 ページ)を参照してください。

SET 句を使用して行を挿入する

SET 句を使用して、名前によって値を列に割り当てます。列は、任意の順番で SET 句に記述できます。次の INSERT ステートメントでは、複数の SET 句を使用して 2 つの行を NewCountries に追加しています。

```
libname sql 'SAS-library';

/* Create the newcountries table.   */

proc sql;
   create table sql.newcountries
      like sql.countries;

/* Insert all of the rows from countries into newcountries based  */
/* on a population of 130000000.                                  */

proc sql;
   insert into sql.newcountries
   select * from sql.countries
      where population ge 130000000;

/* Insert 2 new rows in the newcountries table.     */
/* Print the table.                                 */

proc sql;
   insert into sql.newcountries
      set name='Bangladesh',
          capital='Dhaka',
          population=126391060
      set name='Japan',
          capital='Tokyo',
          population=126352003;

   title "World's Largest Countries";
   select name format=$20.,
          capital format=$15.,
          population format=comma15.0
      from sql.newcountries;
```

アウトプット 4.2 *SET 句を使用して挿入された行*

World's Largest Countries

Name	Capital	Population
Brazil	Brasilia	160,310,357
China	Beijing	1,202,215,077
India	New Delhi	929,009,120
Indonesia	Jakarta	202,393,859
Russia	Moscow	151,089,979
United States	Washington	263,294,808
Bangladesh	Dhaka	126,391,060
Japan	Tokyo	126,352,003

次の SET 句の特徴に注意してください。

- 他の SQL 句と同様に、列を区切る場合はコンマを使用します。さらに、最後の SET 句の後ろでのみセミコロンを使用する必要があります。
- 列のデータを省略した場合、その列の値は欠損値になります。
- 値が欠損していることを指定するには、文字値の場合は単一引用符で囲まれた空白を使用し、数値の場合はピリオドを使用します

VALUES 句を使用して行を挿入する

VALUES 句を使用して、値を位置で列に割り当てます。次の INSERT ステートメントでは、複数の VALUES 句を使用して行を NewCountries に追加しています。NewCountries には 6 つの列があることを思い出してください。そのため、値または適切な欠損値を、6 つの列すべてに対して指定する必要があります。NewCountries の列の詳細については、“既存のテーブルと同じテーブルの作成” (115 ページ)の DESCRIBE TABLE ステートメントの結果を参照してください。

```
libname sql 'SAS-library';

proc sql;
   insert into sql.newcountries
      values ('Pakistan', 'Islamabad', 123060000, ., ' ', .)
      values ('Nigeria', 'Lagos', 99062000, ., ' ', .);
   title "World's Largest Countries";
   select name format=$20.,
          capital format=$15.,
          population format=comma15.0
      from sql.newcountries;
```

アウトプット 4.3 VALUES 句を使用して挿入された行

World's Largest Countries

Name	Capital	Population
Brazil	Brasilia	160,310,357
China	Beijing	1,202,215,077
India	New Delhi	929,009,120
Indonesia	Jakarta	202,393,859
Russia	Moscow	151,089,979
United States	Washington	263,294,808
Bangladesh	Dhaka	126,391,060
Japan	Tokyo	126,352,003
Pakistan	Islamabad	123,060,000
Nigeria	Lagos	99,062,000

VALUES 句の次の特徴に注意してください。

- 他の SQL 句と同様に、列を区切る場合はコンマを使用します。さらに、最後の VALUES 句の後ろでのみセミコロンを使用する必要があります。.
- 欠損値を指定しないで列のデータを省略した場合、エラーメッセージが表示され、行は挿入されません。
- 値が欠損していることを指定するには、文字値の場合は単一引用符で囲まれたスペースを使用し、数値の場合はピリオドを使用します。

クエリを使用して行を挿入する

クエリ結果の行をテーブルに挿入できます。次のクエリは、Countries テーブルから大きな国(人口 1 億 3000 万以上)の行を返します。この INSERT ステートメントでは、以前に “既存のテーブルと同じテーブルの作成” (115 ページ)で作成した空のテーブル NewCountries にデータを追加しています。

```
libname sql 'SAS-library';

proc sql;
   create table sql.newcountries
      like sql.countries;

proc sql;
   title "World's Largest Countries";
   insert into sql.newcountries
   select * from sql.countries
      where population ge 130000000;

   select name format=$20.,
          capital format=$15.,
          population format=comma15.0
```

```
      from sql.newcountries;
```

アウトプット 4.4 *クエリによって挿入された行*

World's Largest Countries

Name	Capital	Population
Brazil	Brasilia	160,310,357
China	Beijing	1,202,215,077
India	New Delhi	929,009,120
Indonesia	Jakarta	202,393,859
Russia	Moscow	151,089,979
United States	Washington	263,294,808

INSERT ステートメントにより指定された対象テーブルでは、Sql.Newcountries テーブルに挿入する具体的な列が指定されていないため、クエリ内のすべての列を選択する必要があります。対象ファイル内に存在するすべての列をクエリで選択していない場合、エラーが発生し、行は挿入されません。UNDO_POLICY=オプションを指定した場合でも、このエラーは回避できません。データ挿入中の PROC SQL でのエラー処理方法の詳細については、“更新エラーの処理” (122 ページ)を参照してください。

ソーステーブルから列のサブセットのクエリを使って行を挿入するには、INSERT ステートメントですべての列名をカンマで区切ったリストで指定し、かっこで囲みます。SELECT 句では、INSERT ステートメントの列に対応する列名を指定します。列の順番や番号は INSERT ステートメントと SELECT 句で一致していなければなりません。

```
proc sql;
  create table sql.newcountries
  like sql.countries;

proc sql;
   title "World's Largest Countries";
   insert into sql.newcountries (Name,Population)
   select Name,Population from sql.countries
      where population ge 130000000;

   select name format=$20., population format=comma15.0
      from sql.newcountries;
```

アウトプット 4.5 クエリによって挿入された少ない数の列を含む行

World's Largest Countries

Name	Population
Brazil	160,310,357
China	1,202,215,077
India	929,009,120
Indonesia	202,393,859
Russia	151,089,979
United States	263,294,808

INSERT ステートメント内に指定されているテーブル内の既存の列名数を超える数の列をクエリで選択した場合、エラーが発生します。

テーブルのデータ値の更新

UPDATE ステートメントを使用して、テーブル内のデータを変更できます。PROC SQL ビューおよび SAS/ACCESS ビューの元のテーブルのデータを変更することもできます。ビューの更新の詳細については、“ビューの更新” (133 ページ)を参照してください。UPDATE ステートメントは、既存の列のデータを更新しますが、新しい列を作成しません。新しい列を追加する場合は、“列の変更” (123 ページ)および “列の新規作成” (27 ページ)を参照してください。このセクションの例では、元の NewCountries テーブルを更新します。

同一式で列のすべての行を更新する

次の UPDATE ステートメントでは、NewCountries テーブルのすべての人口を 5%増やしています。

```
/* code for all examples in updating section */
libname sql 'SAS-library';

proc sql;
   create table sql.newcountries like sql.countries;
   insert into sql.newcountries
   select * from sql.countries
      where population ge 130000000;


proc sql;
   update sql.newcountries
      set population=population*1.05;
   title "Updated Population Values";
   select name format=$20.,
          capital format=$15.,
          population format=comma15.0
      from sql.newcountries;
```

アウトプット 4.6 *列のすべての行の更新*

Updated Population Values

Name	Capital	Population
Brazil	Brasilia	168,325,875
China	Beijing	1,262,325,831
India	New Delhi	975,459,576
Indonesia	Jakarta	212,513,552
Russia	Moscow	158,644,478
United States	Washington	276,459,548

異なる式で列の行を更新する

列のすべての値ではなく、一部の値を更新するには、UPDATE ステートメントで WHERE 式を使用します。それぞれ異なる WHERE 式を持つ複数の UPDATE ステートメントを使用できます。ただし、それぞれの UPDATE ステートメントには、1 つの WHERE 式しか含めることはできません。次の UPDATE ステートメントは、NewCountries テーブル内の異なる国の人口を、異なる値で増やしています。

```
libname sql 'SAS-library';

proc sql;
   create table sql.newcountries like sql.countries;
   insert into sql.newcountries
   select * from sql.countries
      where population ge 130000000;

proc sql;
   update sql.newcountries
      set population=population*1.05
         where name like 'B%';

   update sql.newcountries
      set population=population*1.07
         where name in ('China', 'Russia');

   title "Selectively Updated Population Values";
   select name format=$20.,
          capital format=$15.,
          population format=comma15.0
      from sql.newcountries;
```

アウトプット 4.7 *列の選択的更新*

Selectively Updated Population Values

Name	Capital	Population
Brazil	Brasilia	168,325,875
China	Beijing	1,286,370,132
India	New Delhi	929,009,120
Indonesia	Jakarta	202,393,859
Russia	Moscow	161,666,278
United States	Washington	263,294,808

次の CASE 式を使用して、同じ結果を得ることができます。

```
update sql.newcountries
   set population=population*
      case when name like 'B%' then 1.05
           when name in ('China', 'Russia') then 1.07
           else 1
      end;
```

WHEN 句が true の場合、対応する THEN 句は、SET 句によって式の実行に使用される値を返します。この例では、Name が *B* の文字で始まる場合、SET 式は「**population=population*1.05**」となります。

注意:

必ず、ELSE 句を指定してください。 ELSE 句を省略した場合、WHEN 句のいずれかに記述されていない各行は、更新中の列について欠損値を受け取ります。これは、CASE 式が SET 句に欠損値を提供し、Population 列に欠損値が掛けられ、その結果欠損値が生成されることにより発生します。

更新エラーの処理

テーブルでの行の更新中または挿入中に、更新または挿入を実行できないことを示すエラーメッセージが表示される場合があります。UNDO_POLICY=オプションを使用することによって、すでに実行された変更を永続化するかどうかを制御できます。

PROC SQL ステートメントと RESET ステートメントの UNDO _POLICY=オプションは、現在の INSERT ステートメントまたは UPDATE ステートメントによってエラー発生時点までに挿入または更新された行を、PROC SQL がどのように処理するかを指定します。

UNDO_POLICY = REQUIRED
これはデフォルトです。これは、エラー発生時点までのすべての更新または挿入を取り消します。

UNDO_POLICY = NONE
これは、どの更新、挿入も取り消しません。

UNDO_POLICY = OPTIONAL
これは、確実に取り消せるすべての更新または挿入を取り消します。

注: あるいは、SQLUNDOPOLICY システムオプションを設定することもできます。詳細については、“SQLUNDOPOLICY=システムオプション” (379 ページ)を参照してください。

行の削除

DELETE ステートメントは、テーブル内の 1 つ以上の行を削除します。PROC SQL ビューまたは SAS/ACCESS ビューの元のテーブル内の行も削除できます。ビューからの行の削除の詳細については、“ビューの更新” (133 ページ)を参照してください。次の DELETE ステートメントでは、*R* の文字で始まる国の名前を削除しています。

```
/* Create and populate Newcountries */
proc sql;
   create table sql.newcountries like sql.countries;
   insert into sql.newcountries
   select * from sql.countries
      where population ge 130000000;

proc sql;
   delete
      from sql.newcountries
      where name like 'R%';
```

SAS ログ内のメモに、削除された行数が示されます。

ログ 4.4 *DELETE ステートメントの SAS ログ*

```
注:1 row was deleted from SQL.NEWCOUNTRIES.
```

注: PROC SQL テーブルの場合、SAS は行内のデータを削除しますが、テーブルの領域は維持されます。

注意:

WHERE 句を省略した場合、DELETE ステートメントは、指定したテーブルまたはビューによって記述されたテーブルのすべての行を削除します。行は、テーブルを再作成するまでテーブルから削除されません。

列の変更

ALTER TABLE ステートメントは、既存のテーブルの列を追加、変更、および削除します。ALTER TABLE ステートメントはテーブルでのみ使用でき、ビューでは動作しません。テーブルの変更内容を示すメモが、SAS ログに表示されます。

列の追加

ADD 句は、新しい列を既存のテーブルに追加します。列名と列のデータタイプを指定する必要があります。長さ(LENGTH=)、出力形式(FORMAT=)、入力形式(INFORMAT=)、ラベル(LABEL=)を指定することもできます。次の ALTER TABLE ステートメントでは、数値データ列 Density を NewCountries テーブルに追加しています。

```
proc sql;
   create table sql.newcountries like sql.countries;
```

```
   insert into sql.newcountries
   select * from sql.countries
      where population ge 130000000;

proc sql;
   alter table sql.newcountries
      add density num label='Population Density' format=6.2;

   title "Population Density Table";
   select name format=$20.,
          capital format=$15.,
          population format=comma15.0,
          density
      from sql.newcountries;
```

アウトプット 4.8 *新しい列の追加*

Population Density Table

Name	Capital	Population	Population Density
Brazil	Brasilia	160,310,357	.
China	Beijing	1,202,215,077	.
India	New Delhi	929,009,120	.
Indonesia	Jakarta	202,393,859	.
Russia	Moscow	151,089,979	.
United States	Washington	263,294,808	.

新しい列が NewCountries に追加されますが、その列にはデータ値が含まれません。次の UPDATE ステートメントでは、Density の欠損値を、国ごとの適切な人口密度に変更しています。

```
proc sql;
   update sql.newcountries
      set density=population/area;

   title "Population Density Table";
   select name format=$20.,
          capital format=$15.,
          population format=comma15.0,
          density
      from sql.newcountries;
```

アウトプット 4.9　新しい列の値の入力

Population Density Table

Name	Capital	Population	Population Density
Brazil	Brasilia	160,310,357	48.78
China	Beijing	1,202,215,077	325.27
India	New Delhi	929,009,120	759.86
Indonesia	Jakarta	202,393,859	273.10
Russia	Moscow	151,089,979	22.92
United States	Washington	263,294,808	69.52

データ値の変更方法の詳細については、“テーブルのデータ値の更新” (120 ページ)を参照してください。

テーブルを再作成するときに、算術式を使用して Population Density 列を作成することによって、同じ更新結果を得ることもできます。

```
proc sql;
   create table sql.newcountries as
   select *, population/area as density
             label='Population Density'
             format=6.2
      from sql.newcountries;
```

算術式を使用した、列の作成の別の例については、“値の計算” (29 ページ)を参照してください。

列の変更

MODIFY 句を使用して、列の幅、入力形式、出力形式、ラベルを変更できます。列の名前を変更するには、RENAME=データセットオプションを使用してください。MODIFY 句を使用して列のデータタイプを変更することはできません。

次の MODIFY 句は、Population 列の出力形式を永続的に変更します。

```
proc sql;
   create table sql.newcountries like sqlcountries;
   create table sql.newcountries as
   select * from sql.countries
      where population ge 130000000;


proc sql;
   title "World's Largest Countries";
   alter table sql.newcountries
      modify population format=comma15.;
   select name, population from sql.newcountries;
```

アウトプット 4.10 *列の出力形式の変更*

World's Largest Countries

Name	Population
Brazil	160,310,357
China	1,202,215,077
India	929,009,120
Indonesia	202,393,859
Russia	151,089,979
United States	263,294,808

列を更新できるようにするために、まず列の幅(および出力形式)の変更が必要になる場合があります。たとえば、Name に長いテキスト文字列の接頭語を付加するには、まず Name の幅と出力形式を 35 から 60 に変更する必要があります。次のステートメントでは、Name 列を変更および更新しています。

```
proc sql;
   title "World's Largest Countries";
   alter table sql.newcountries
      modify name char(60) format=$60.;
   update sql.newcountries
      set name='The United Nations member country is '||name;

   select name from sql.newcountries;
```

アウトプット 4.11 *列の幅の変更*

World's Largest Countries

Name
The United Nations member country is Brazil
The United Nations member country is China
The United Nations member country is India
The United Nations member country is Indonesia
The United Nations member country is Russia
The United Nations member country is United States

列の削除

DROP 句は、テーブルから列を削除します。次の DROP 句は、NewCountries から UNDate を削除します。

```
proc sql;
   alter table sql.newcountries
      drop undate;
```

インデックスの作成

インデックスは、テーブルに関連付けられたファイルです。インデックスは、インデックス値による行へのアクセスを可能にします。インデックスによって、データの小さいサブセットへの素早いアクセスが可能になり、テーブル結合機能を強化できます。インデックスを作成することはできますが、PROC SQL に対してインデックスを使用するよう指示することはできません。PROC SQL は、インデックスを使用することが効率的かどうかを決定します。列によっては、インデックスの使用が適切でない場合があります。通常は、多くの一意の値を含む列または定常的に結合で使用される列に対して、インデックスを作成します。

PROC SQL を使用したインデックスの作成

1 つの列のみに適用される単一インデックスを作成できます。単一インデックスの名前は、インデックスを追加する列の名前と同じである必要があります。テーブル名の後のかっこ内に、列名を指定します。次の CREATE INDEX ステートメントでは、NewCountries の Area 列にインデックスを作成しています。

```
proc sql;
   create index area
      on sql.newcountries(area);
```

2 つ以上の列に適用される複合インデックスを作成することもできます。次の CREATE INDEX ステートメントでは、NewCountries の Name 列と Continent 列にインデックス Place を作成しています。

```
proc sql;
   create index places
      on sql.newcountries(name, continent);
```

インデックスが追加された列のそれぞれの値(または複合インデックスの列のそれぞれの値の組み合わせ)が必ず一意になるようにするには、UNIQUE キーワードを使用します。

```
proc sql;
   create unique index places
      on sql.newcountries(name, continent);
```

UNIQUE キーワードを使用すると、複数の行が同一のインデックス値を持つような変更をテーブルに加えることができなくなります。

インデックスの作成のヒント

- 複合インデックスの名前を、テーブル内のいずれかの列と同じ名前にすることはできません。
- 通常 2 つの列を使用してデータにアクセスする場合(たとえば、従業員データベースの姓および名の列を使用する場合)、それらの列に複合インデックスを作成する必要があります。
- ディスク領域と更新コストを削減するため、保持するインデックスの数は最小限にします。

- 比較的少数(15%未満の)の行を取得するクエリの場合は、インデックスを使用します。
- 通常は、小さなテーブルにインデックスを追加しても、パフォーマンスの向上は得られません。
- 通常は、異なる値の個数が少ない(6 または 7 未満の)列にインデックスを追加しても、パフォーマンスの向上は得られません。
- 単一インデックスと複合インデックスには、同じ列を使用できます。ただし、主キーに一貫性制約が設定されたテーブルの場合、主キーと同じ列に基づく複数のインデックスを作成しないでください。

インデックスの削除

テーブルからインデックスを削除するには、DROP INDEX ステートメントを使用します。次の DROP INDEX ステートメントでは、NewCountries からインデックス Place を削除しています。

```
proc sql;
   drop index places from sql.newcountries;
```

テーブルの削除

PROC SQL テーブルを削除するには、DROP TABLE ステートメントを使用します。

```
proc sql;
   drop table sql.newcountries;
```

SAS ソフトウェアでの SQL プロシジャテーブルの使用

PROC SQL テーブルは SAS データファイルであるため、それらを DATA ステップまたは他の SAS プロシジャへの入力として使用できます。たとえば、次の PROC MEANS ステップは、Countries 内のすべての国について平均面積を計算します。

```
proc means data=sql.countries mean maxdec=2;
   title "Mean Area for All Countries";
   var area;
run;
```

アウトプット 4.12 PROC MEANS での PROC SQL テーブルの使用

Mean Area for All Countries

The MEANS Procedure

Analysis Variable : Area
Mean
250998.76

テーブルの一貫性制約の作成および使用

一貫性制約は、テーブル内のデータの正確さ、完全性、または一貫性を保証するために指定するルールです。すべての一貫性制約は、一貫性制約が定義されているテーブルの列内のデータ値を挿入、削除、または変更した場合に適用されます。既存のデータを含むテーブルに制約が追加される前に、すべてのデータがチェックされ、制約を満たしているかどうかが判定されます。

一般的な一貫性制約を使用して、列内のデータが次のいずれかを満たしていることを確認できます。

- 非欠損
- 一意
- 非欠損かつ一意
- 指定した値のセットまたは範囲に含まれる

あるテーブルの指定された列(**主キー**と呼ぶ)の値を別のテーブルの指定された列の値にリンクする、参照一貫性制約を適用することもできます。2 番目のテーブルの列は、主キーにリンクされた場合、**外部キー**と呼ばれます。

参照制約を定義する際に、主キーの値が更新または削除されたときに実行されるアクションを選択することもできます。

- 主キーに一致する外部キーの値が存在する場合、主キーの値が更新または削除されるのを防ぐことができます。これがデフォルトです。
- 主キーの値に対する更新と削除を許可できます。デフォルトでは、影響を受けたすべての外部キーの値は、欠損値に変更されます。ただし、デフォルトのかわりに CASCADE オプションを指定して、外部キーの値を更新できます。現在、CASCADE オプションは、削除には適用されません。

更新と削除に対して、別々のアクションを選択できます。

注: ビューには、一貫性制約を定義できません。

次の例では、MyStates テーブルと、別の USPostal テーブルに、一貫性制約を作成しています。それらの制約は、次のとおりです。

- state は、両方のテーブルで一意かつ非欠損でなければならない。
- population は、0 よりも大きくなければならない。
- continent は、North America または Oceania のいずれかでなければならない。

```
proc sql;
   create table sql.mystates
      (state      char(15),
       population num,
       continent  char(15),

          /* contraint specifications */
       constraint prim_key    primary key(state),
       constraint population  check(population gt 0),
       constraint continent   check(continent in ('North America', 'Oceania')));

   create table sql.uspostal
      (name      char(15),
       code      char(2) not null,          /* constraint specified as    */
                                            /* a column attribute         */

       constraint for_key foreign key(name) /* links NAME to the          */
                  references sql.mystates   /* primary key in MYSTATES    */

                     on delete restrict     /* forbids deletions to STATE */
                                            /* unless there is no         */
                                            /* matching NAME value        */

                     on update set null);   /* allows updates to STATE,   */
                                            /* changes matching NAME      */
                                            /* values to missing          */
```

DESCRIBE TABLE CONSTRAINTS ステートメントは、テーブルの制約仕様のみを **Results** ウィンドウに表示します。

```
proc sql;
   describe table sql.mystates;
   describe table constraints sql.uspostal;
```

アウトプット 4.13 *一貫性制約を表示する PROC SQL DESCRIBE TABLE CONSTRAINTS Results ウィンドウ*

-----Alphabetic List of Integrity Constraints-----

#	Integrity Constraint	Type	Variables	Where Clause	Reference	On Delete	On Update
1	continent	Check		continent in ('North America', 'Oceania')			
2	population	Check		population>0			
3	prim_key	Primary Key	state				
	for_key	Referential			SQL.USPOSTAL	Restrict	Set Null

-----Alphabetic List of Integrity Constraints-----

#	Integrity Constraint	Type	Variables	Reference	On Delete	On Update
1	_NM0001_	Not Null	code			
2	for_key	Foreign Key	name	SQL.MYSTATES	Restrict	Set Null

一貫性制約は、ビューでは使用できません。一貫性制約の詳細については、*SAS 言語リファレンス: 解説編*を参照してください。

PROC SQL ビューの作成および使用

PROC SQL ビューの作成と使用の概要

PROC SQL ビューにはストアドクエリが含まれています。これは、SAS プロシジャや DATA ステップでビューを使用するときに実行されます。ビューは、次の理由により役立ちます。

- ビューは、ビューがアクセスするデータに比べて非常に小さいことが多いため、多くの場合、領域を節約します。
- ユーザーが頻繁にクエリをサブミットして不要な列や行を削除することがなくなります。
- 機密の列をユーザーから保護しながら、同じユーザーに同じテーブルの他の列を表示できます。
- データが実行時にテーブルから派生するため、入力データセットが常に最新であることが保証されます。
- 複雑な結合や複雑なクエリをユーザーから隠蔽します。

ビューの作成

PROC SQL ビューを作成するには、次の例に示すように、CREATE VIEW ステートメントを使用します。

```
libname sql 'SAS-library';

proc sql;
   title 'Current Population Information for Continents';
   create view sql.newcontinents as
   select continent,
          sum(population) as totpop  format=comma15. label='Total Population',
          sum(area) as totarea format=comma15. label='Total Area'
      from sql.countries
      group by continent;

   select * from sql.newcontinents;
```

アウトプット 4.14 SQL プロシジャビュー

Current Population Information for Continents

Continent	Total Population	Total Area
	384,772	876,800
Africa	710,529,592	11,299,595
Asia	3,381,858,879	12,198,325
Australia	18,255,944	2,966,200
Central America and Caribbean	66,815,930	291,463
Europe	813,481,724	9,167,250
North America	384,801,818	8,393,092
Oceania	5,342,368	129,600
South America	317,568,801	6,885,418

注: この例では、それぞれの列には名前があります。変数名を必要とするプロシジャでビューを使用しようとする場合、他のプロシジャで変数名として参照できる列のエイリアスを指定する必要があります。詳細については、“SAS での SQL プロシジャビューの使用” (137 ページ)を参照してください。

ビューの説明

DESCRIBE VIEW ステートメントは、PROC SQL ビューの説明を SAS ログに書き込みます。次の SAS ログは、“ビューの作成” (132 ページ)で作成された NewContinents ビューについて説明しています。

```
proc sql;
   describe view sql.newcontinents;
```

ログ 4.5 DESCRIBE VIEW ステートメントによる SAS ログ

```
注:SQL view SQL.NEWCONTINENTS is defined as:select continent, SUM(population) as
totpop label='Total Population' format=COMMA15.0, SUM(area) as totarea
label='Total Area' format=COMMA15.0 from SQL.COUNTRIES group by continent;
```

パスワードで保護された SAS ビューを定義するには、パスワードを指定する必要があります。複数のパスワードを使用して SAS ビューを作成した場合、そのビューの定義にアクセスするには、最も制限の強いパスワードを指定する必要があります。詳細については“DESCRIBE ステートメント” (246 ページ)を参照してください。

ビューの更新

PROC SQL ビューと SAS/ACCESS ビューを介して、INSERT ステートメント、DELETE ステートメントおよび UPDATE ステートメントを使用してデータを更新できます。ただし、次の条件があります。

- ビューを介して 1 つのテーブルのみを更新できます。元のテーブルを、セット演算子を使用して別のテーブルに結合またはリンクできません。ビューにサブクエリを含めることはできません。
- ビューが DBMS テーブルにアクセスする場合、外部のデータベース管理システム (Oracle など)によって適切な権限が与えられている必要があります。また、その DBMS に対応した SAS/ACCESS ソフトウェアをインストールしている必要があります。SAS/ACCESS ビューの詳細については、使用している DBMS の SAS/ACCESS のマニュアルを参照してください。
- 列のエイリアスを使用してビューの列を更新できますが、派生した列、つまり式によって生成された列は更新できません。次の例では、SquareMiles は更新できますが、Density は更新できません。

```
proc sql;
   create view mycountries as
      select Name,
             area as SquareMiles,
             population/area as Density
         from sql.countries;
```

- WHERE 句を含むビューを更新できます。WHERE 句は、UPDATE 句またはビューに含めることができます。その他の句(ORDER BY、HAVING など)を含むビューは、更新できません。

ビューに LIBNAME を埋め込む

USING LIBNAME 句を使用して、SAS LIBNAME ステートメントまたは SAS/ACCESS LIBNAME ステートメントをビューに埋め込むことができます。PROC SQL がビューを実行するときに、ストアドクエリがライブラリ参照名を割り当てます。SAS/ACCESS ライブラリ参照名の場合、PROC SQL は DBMS との接続を確立します。ライブラリ参照名の範囲はビューに対してローカルです。そのため、SAS セッション内の同一の名前が付けられたライブラリ参照名とは競合しません。クエリが終了すると、ライブラリ参照名の割り当ては解除されます。DBMS との接続が終了し、ライブラリ内のすべてのデータは使用できなくなります。

埋め込まれたライブラリ参照名のメリットは、engine-host オプションと DBMS 接続情報 (パスワードなど)をビューに格納できるということです。つまり、ライブラリ参照名を使用するときに、その情報を記憶して再入力する必要がありません。

注: USING LIBNAME 句は、SELECT ステートメント内の最後の句である必要があります。複数の句をカンマで区切って指定できます。

次の例では、ライブラリ参照名 OilInfo が割り当てられ、Oracle データベースとの接続が作成されています。

```
proc sql;
   create view sql.view1 as
      select *
         from oilinfo.reserves as newreserves
         using libname oilinfo oracle
            user=username
            pass=password
            path='dbms-path';
```

SAS/ACCESS LIBNAME ステートメントの詳細については、使用している DBMS の SAS/ACCESS のマニュアルを参照してください。

次の例では、SAS LIBNAME ステートメントをビューに埋め込んでいます。

```
proc sql;
   create view sql.view2 as
      select *
         from oil.reserves
         using libname oil 'SAS-library';
```

ビューの削除

ビューを削除するには、DROP VIEW ステートメントを使用します。

```
proc sql;
   drop view sql.newcontinents;
```

インラインビューの指定

場合によっては、テーブルまたはビューではなく、FROM 句でクエリを使用したほうがよいことがあります。ビューを作成し、それを FROM 句で参照できますが、その処理には 2 つのステップが含まれます。余分なステップを省くには、FROM 句で、インラインビューをかっこで囲んで指定します。

インラインビューは、FROM 句に現れるクエリです。インラインビューは、外側のクエリによってデータの選択に使用されるテーブルを内部で生成します。CREATE VIEW ステートメントで作成されるビューとは異なり、インラインビューには名前が割り当てられません。そのため、他のクエリや SAS プロシジャから、テーブルであるかのように参照することはできません。インラインビューは、それが定義されているクエリ内でのみ参照可能です。

次のクエリでは、カリブ海および中央アメリカのすべての国の人口を、インラインクエリで合計しています。WHERE 句は、この合計を個々の国の人口と比較しています。カリブ海と中央アメリカの総人口よりも人口の多い国のみが表示されます。

```
libname sql 'SAS-library';

proc sql;
   title 'Countries With Population GT Caribbean Countries';
   select w.Name, w.Population format=comma15., c.TotCarib
      from (select sum(population) as TotCarib format=comma15.
                  from sql.countries
                 where continent = 'Central America and Caribbean') as c,
           sql.countries as w
        where w.population gt c.TotCarib;
```

アウトプット 4.15 インラインビューの使用

Countries With Population GT Caribbean Countries

Name	Population	TotCarib
Bangladesh	126,387,850	66,815,930
Brazil	160,310,357	66,815,930
China	1,202,215,077	66,815,930
Germany	81,890,690	66,815,930
India	929,009,120	66,815,930
Indonesia	202,393,859	66,815,930
Japan	126,345,434	66,815,930
Mexico	93,114,708	66,815,930
Nigeria	99,062,003	66,815,930
Pakistan	123,062,252	66,815,930
Philippines	70,500,039	66,815,930
Russia	151,089,979	66,815,930
United States	263,294,808	66,815,930
Vietnam	73,827,657	66,815,930

SAS での SQL プロシジャビューの使用のヒント

- ビューで ORDER BY 句を使用することは避けてください。ORDER BY 句を指定した場合、ビューを参照するたびにデータの並べ替えが必要になります。
- 1 つのプログラムまたは複数のプログラムで何度もデータを使用する場合、ビューではなくテーブルを作成したほうが効率的です。1 つのプログラムで何度もビューを参照すると、参照のたびにデータへのアクセスが必要になります。
- ビューが、寄与している 1 つまたは複数のテーブルと同じ SAS ライブラリ内に常駐する場合、FROM 句で 1 レベル名を指定します。FROM 句の 1 つまたは複数のテーブルの、ライブラリ参照名のデフォルトは、ビューを含むライブラリのライブラリ参照名です。これによって、ビューおよびビューが寄与している 1 つまたは複数のテーブルを含む SAS ライブラリに、別のライブラリ参照名を割り当てる場合に、ビューを変更する必要がなくなります。これについてのヒントは、“ビューの作成” (132 ページ)で説明されているビューで使用されています。
- 構造が変更される可能性のあるテーブルに基づいてビューを作成することは避けてください。ビューは、存在しない列を参照している場合、有効ではありません。
- ビュー内のオブザベーションの数を制限するには、FIRSTOBS=や OBS=のようなシステムオプションではなく、FIRSTOBS=や OBS=のようなデータセットオプションを使用します。システムオプションを使用すると、オブザベーション数の制限がまず元のテーブルに適用された後、続いてビューに適用されます。つまり、オブザベーション数の削減が 2 回実施されます。データセットオプションを使用すると、オブザベーション数の制限はビューに対してのみ適用されます。

- クライアントとサーバー間で PROC SQL ビューを処理する場合、正しい結果が得られるかどうかは、クライアントとサーバー間のアーキテクチャの互換性に依存します。詳細については、“Accessing a SAS View” (*SAS/CONNECT User's Guide*)を参照してください。

SAS での SQL プロシジャビューの使用

PROC SQL ビューを、DATA ステップまたは他の SAS プロシジャへの入力として使用できます。SAS で PROC SQL ビューを使用する場合の構文は、PROC SQL テーブルを使用する場合と同じです。例については、“SAS ソフトウェアでの SQL プロシジャテーブルの使用” (128 ページ)を参照してください。

5 章

SQL プロシジャを使用したプログラミング

はじめに **140**

PROC SQL オプションを使用し、クエリを作成、デバックする **140**
概要:PROC SQL オプションを使用し、クエリを作成、デバックする 140
INOBS=と OUTOBS=オプションを使用した行の処理の制限 141
LOOPS=オプションを使用した反復の限定 141
NOEXEC オプションと VALIDATE ステートメントを使用した構文チェック 141
FEEDBACK オプションを使用した SELECT *の拡張 142
STIMER オプションを使用して PROC SQL の時間を設定する 142
RESET ステートメントを使用して PROC SQL オプションをリセットする 143

クエリパフォーマンスの向上 **144**
概要:クエリパフォーマンスの向上 144
インデックスを使用したパフォーマンスの改善 144
Set 操作に ALL キーワードを使用する 145
テーブルとビューの作成時に ORDER BY 句を指定しない 145
インラインビューと一時テーブルの使用 145
結合を使用したサブクエリの比較 145
WHERE 式と結合の併用 145
PUT 関数の最適化 146
DATE、TIME、DATETIME、TODAY 関数への参照の置換 147
要約関数の使用時にデータの再マージを無効化する 148

列エイリアスの使用 **148**
列エイリアスの概要 148
列エイリアスの拡張機能 148
CALCULATED キーワードと列エイリアスの使用 150

DICTIONARY テーブルを使用し、SAS System の情報にアクセスする **151**
ディクショナリテーブルについて 151
DICTIONARY テーブルと Sashelp ビューの情報の取得 153
DICTIONARY.Tables の使用 154
DICTIONARY.Columns の使用 155
DICTIONARY テーブルとパフォーマンス 156

PROC SQL で SAS データセットオプションを使用する **157**

PROC SQL を SAS マクロ機能とともに使用する **158**
概要:PROC SQL を SAS マクロ機能とともに使用する 158
PROC SQL のマクロ変数の作成 158
マクロ変数の値の連結 161
テーブル作成のマクロの定義 162
PROC SQL 自動マクロ変数の使用 163

REPORT プロシジャを使用し、PROC SQL 出力をフォーマットする **166**

SAS/ACCESS を使用した DBMS へのアクセス **168**
概要:SAS/ACCESS を使用した DBMS へのアクセス 168
LIBNAME ステートメントを使用した DBMS への接続 169
パススルー機能を使用した DBMS への接続 172
PROC SQL ビューと SAS/ACCESS ビューの更新 174

PROC SQL で ODS (Output Delivery System)を使用する **175**

はじめに

このセクションでは、次の実行方法について説明します。

- PROC SQL オプションを使用してクエリを作成し、デバッグします。
- クエリのパフォーマンスを改善します。
- DICTIONARY テーブルにアクセスし、それらを SAS の要素についての情報収集に役立てます。
- SAS マクロ機能で PROC SQL を使用します。
- REPORT プロシジャで PROC SQL を使用します。
- SAS/ACCESS ソフトウェアを使用して DBMS にアクセスします。
- SAS Output Delivery System (ODS)を使用して、PROC SQL の出力をフォーマットします。

PROC SQL オプションを使用し、クエリを作成、デバックする

概要:PROC SQL オプションを使用し、クエリを作成、デバックする

PROC SQL は、クエリの開発中に PROC SQL をさらに高度に制御できるオプションをサポートしています。

- INOBS=、OUTOBS=および LOOPS=の各オプションを使用して、PROC SQL が処理する行数と反復回数を制限することによって、クエリの実行時間を減らすことができます。
- EXEC ステートメントと VALIDATE ステートメントによって、クエリの構文を迅速にチェックできます。
- FEEDBACK オプションを指定すると、SELECT *ステートメントは、それが表す列のリストに展開されます。
- PROC SQL STIMER オプションを指定すると、クエリの実行回数が記録され、表示されます。

PROC SQL ステートメントでオプションを初期設定し、その後、RESET ステートメントを使用して現在の PROC SQL ステップを終了することなく同じオプションの設定を変更できます。

INOBS=と OUTOBS=オプションを使用した行の処理の制限

大きなテーブルに対するクエリを開発しているときに、PROC SQL が処理する行数を減らすことによって、クエリの実行時間を減らすことができます。WHERE ステートメントを使用してテーブルをサブセット化することは、これを実現する方法の 1 つです。他には、INOBS=オプションと OUTOBS=オプションを使用する方法があります。

INOBS=オプションは、PROC SQL が 1 つのソースから入力として受け取る行の数を制限します。たとえば、INOBS=10 を指定した場合、PROC SQL は、FROM 句で指定されたどのテーブルまたはビューについても、10 行のみを使用します。INOBS=10 を指定し、WHERE 句を使用しないで 2 つのテーブルを結合した場合、結果のテーブル(デカルト積)には、最大で 100 行が含まれます。INOBS=オプションは、SAS システムオプションの OBS=に類似しています。

OUTOBS=オプションは、PROC SQL が表示したりテーブルに書き込んだりする行の数を制限します。たとえば、OUTOBS=10 を指定し、クエリを使用してテーブルに値を挿入した場合、PROC SQL は、結果のテーブルに最大で 10 行を挿入します。OUTOBS=は、SAS データセットオプションの OBS=に類似しています。

単純なクエリでは、INOBS を使用した場合と OUTOBS を使用した場合とで、明らかな違いがないことがあります。しかし、通常は、正しいオプションを選択することは重要です。たとえば、INOBS=10 を指定して列の平均を取得した場合、列の 10 個の値のみの平均が返されます。

LOOPS=オプションを使用した反復の限定

LOOPS=オプションを指定すると、PROC SQL の内部ループでの反復回数が、指定した回数に制限されます。制限を設定することによって、クエリがコンピュータリソースを過剰に使用することを防ぎます。たとえば、結合一致条件を満たさない 3 つの大きなテーブルを結合しようとすると、巨大な内部テーブルが作成されて、処理の効率が悪くなる場合があります。LOOPS=オプションを使用することで、このような問題を防ぎます。

各 PROC SQL ステートメントの実行後に SQLOOPS マクロ変数でレポートされる反復回数を使用して、適切な LOOPS=オプションの値を評価できます。詳細については、“PROC SQL 自動マクロ変数の使用” (163 ページ)を参照してください。

INOBS=オプション、OUTOBS=オプションまたは LOOPS=オプションと共に PROMPT オプションを使用すると、これらのオプションで設定した制限に達した場合に、処理の停止または続行の選択を促すメッセージが表示されます。

NOEXEC オプションと VALIDATE ステートメントを使用した構文チェック

PROC SQL ステップの構文を、実際に実行せずにチェックするには、NOEXEC オプションまたは VALIDATE ステートメントを使用します。NOEXEC オプションを PROC SQL ステートメントで 1 回使用するだけで、その PROC SQL ステップに含まれるすべてのクエリの構文について、実行することなく正確さをチェックできます。SELECT ステートメントを実行せずに正確さをチェックするには、それぞれの SELECT ステートメントの前で VALIDATE ステートメントを指定する必要があります。構文が有効な場合、その効果についてのメッセージが SAS ログに書き込まれます。構文が無効な場合は、エラーメッセージが表示されます。SQLRC 自動マクロ変数には、構文の有効性を示すエラーコードが格納されます。PROC SQL での VALIDATE ステートメントの使用例については、次を参照してください。“クエリの検証” (71 ページ)SAS/AF アプリケーションでの VALIDATE ステートメントの使用例については、“PROC SQL 自動マクロ変数の使用” (163 ページ)を参照してください。

注: SAS がバッチまたは非対話型セッションで実行されている場合、PROC SQL EXEC オプションと ERRORSTOP オプションの間には相互関係があります。詳細については、7 章, “SQL プロシジャ” (213 ページ)を参照してください。

FEEDBACK オプションを使用した SELECT *の拡張

FEEDBACK オプションを指定すると、SELECT * (ALL)ステートメントは、それが表す列のリストに展開されます。すべての PROC SQL ビューは、元になるクエリに展開され、すべての式は、それらの評価順序を示すかっこで囲まれます。また、クエリに対して実行される PUT 関数の最適化が表示されます。FEEDBACK オプションによって、マクロとマクロ変数の展開された値も表示されます。

たとえば、次のクエリが SAS ログに展開されます。

```
libname sql 'SAS-library';

proc sql feedback;
   select * from sql.countries;
```

ログ 5.1 *展開された SELECT *ステートメント*

```
注:Statement transforms to:select COUNTRIES.Name, COUNTRIES.Capital,
COUNTRIES.Population, COUNTRIES.Area, COUNTRIES.Continent, COUNTRIES.UNDate from
SQL.COUNTRIES;
```

STIMER オプションを使用して PROC SQL の時間を設定する

特定の操作については、複数の方法で実行できます。たとえば、多くの場合、サブクエリと等価な結合が存在します。可読性や保守性などの要因を考慮するとしても、通常は最も高速に実行できるクエリが選択されます。SAS システムオプションの STIMER を指定すると、プロシジャ全体の累積時間が表示されます。PROC SQL の STIMER オプションを指定すると、PROC SQL ステップの個々のステートメントの実行時間が表示されます。これによって、クエリの最適化が可能になります。

注: PROC SQL の STIMER オプションが動作するには、SAS システムオプションの STIMER も指定する必要があります。

この例では、2 つのクエリの実行時間を比較します。これらのクエリは、どちらも、ベルギーよりも人口の多い UnitedStates テーブル内の州の名前と人口を表示します。最初のクエリでは結合を使用してこれを実行しますが、2 番目のクエリではサブクエリを使用します。ログ 5.2 (143 ページ)に、SAS ログに出力された STIMER の結果を示します。

```
libname sql 'SAS-library';

proc sql stimer ;
   select us.name, us.population
      from sql.unitedstates as us, sql.countries as w
      where us.population gt w.population and
            w.name = 'Belgium';

   select Name, population
      from sql.unitedstates
      where population gt
                  (select population from sql.countries
                     where name = 'Belgium');
```

ログ 5.2 *2 つのクエリの実行時間の比較*

```
4 proc sql stimer ; NOTE:SQL Statement used:real time 0.00 seconds cpu time 0.01
seconds 5 select us.name, us.population 6 from sql.unitedstates as us,
sql.countries as w 7 where us.population gt w.population and 8 w.name =
'Belgium'; NOTE:このクエリの実行には 1 つ以上のデカルト積結合の実行が含まれており、それらを最適化する
ことはできません。注:SQL Statement used: real time 0.10 seconds cpu time 0.05 seconds
9 10 select Name, population 11 from sql.unitedstates 12 where population gt 13
(select population from sql.countries 14 where name = 'Belgium'); NOTE:SQL
Statement used:real time 0.09 seconds cpu time 0.09 seconds
```

最初のクエリ(結合を使用)の CPU 時間 0.05 秒と、2 番目のクエリ(サブクエリを使用)の CPU 時間 0.09 秒を比べてください。クエリの実行時間に影響を与える多くの要因がありますが、通常は結合のほうが、それと等価なサブクエリよりも高速です。

RESET ステートメントを使用して PROC SQL オプションをリセットする

PROC SQL ステートメントのオプションを追加、削除または変更するには、RESET ステートメントを使用します。PROC SQL ステートメントと RESET ステートメントのオプションは、任意の順序で記述できます。オプションは、リセットされるまで有効です。

この例では、SELECT ステートメントが結果テーブルを SAS 出力に表示しないようにするために、最初に NOPRINT オプションを使用します。次に RESET ステートメントによって、NOPRINT オプションを PRINT オプション(デフォルト)に変更し、NUMBER オプションを追加して、結果テーブルに行番号を表示します。

```
proc sql noprint;
   title 'Countries with Population Under 20,000';
   select Name, Population from sql.countries;
reset print number;
   select Name, Population from sql.countries
      where population lt 20000;
```

アウトプット 5.1 *RESET ステートメントを使用して PROC SQL オプションをリセットする*

Countries with Population Under 20,000

Row	Name	Population
1	Leeward Islands	12119
2	Nauru	10099
3	Turks and Caicos Islands	12119
4	Tuvalu	10099
5	Vatican City	1010

クエリパフォーマンスの向上

概要:クエリパフォーマンスの向上

クエリのパフォーマンスを改善するには、次のような複数の方法があります。

- インデックスおよび複合インデックスの使用。
- 集合演算での ALL キーワードの使用。これは、結果テーブルに重複行が存在しないことがわかっている場合、または重複行が存在しても問題ない場合に使用します。
- テーブルまたはビューを作成する際の ORDER BY 句の省略。
- 一時テーブルではなく、インラインビューの使用(またはその逆)。
- サブクエリではなく、結合の使用。
- WHERE 式の使用。結合によって作成される結果テーブルのサイズを制限するために使用します。
- PROC SQL オプションまたは SAS システムオプション(あるいはその両方)の使用。クエリ内の PUT 関数を、それと論理的に等価な式で置き換えるために使用します。
- クエリ内の DATE 関数、DATETIME 関数および TODAY 関数への参照の、それらと等価な定数への置き換え。クエリを実行する前に行います。
- クエリ内で要約関数を使用する場合の、データの再マージの無効化。

インデックスを使用したパフォーマンスの改善

インデックスは、PROC SQL の CREATE INDEX ステートメント、または DATASETS プロシジャの MODIFY ステートメントと INDEX CREATE ステートメントを使用して作成します。インデックスは、SAS ライブラリの専用のメンバに格納されます。インデックスの SAS メンバタイプは、INDEX です。インデックスに格納された値は、元になるデータを変更した場合、自動的に更新されます。

インデックスを設定すると、特定のクラスを取得するときのパフォーマンスが向上します。たとえば、インデックスが作成された列を WHERE 式の定数値と比較すると、多くの場合クエリのパフォーマンスが向上します。また、外部テーブルへの相関参照で指定されている列にインデックスを作成すると、サブクエリの(したがってクエリの)パフォーマンスが向上します。複合インデックスを使用すると、複合インデックスに含めた列と定数値を比較するクエリのパフォーマンスが向上します。これらの比較は、AND 演算子を使用して連結されます。たとえば、CITY 列と STATE 列で複合インデックスを作成し、WHERE CITY='xxx' AND STATE='yy という WHERE 式を指定した場合、この複合インデックスによって、行のサブセットを選択する効率が向上します。また、インデックスを使用すると、のような形式の WHERE 句を含むクエリの効率も向上できます。

```
... where var1 in (select item1 from table1) ...
```

インデックスを使用することで、外部クエリに含まれている VAR1 の値を、内部クエリで見つけることができます。結合対象のいずれかのテーブルの列にインデックスを作成すると、テーブルの結合処理が向上します。この最適化は、等結合クエリの場合、つまり、WHERE 式で table1.X=table2.Y を指定した場合にのみ実行できます。

Set 操作に ALL キーワードを使用する

クエリを結合するために、UNION、OUTER UNION、EXCEPT、INTERSECT などのセット演算子を使用できます。オプションの ALL キーワードを指定すると、最後の処理で、結果テーブルから重複行が除去されなくなります。ALL キーワードは、結果テーブルに重複行が存在しないことがわかっている場合、または重複行が残っていても問題ない場合に使用する必要があります。

テーブルとビューの作成時に ORDER BY 句を指定しない

テーブルまたはビューを作成するときに ORDER BY 句を指定すると、このテーブルまたはビューを参照するクエリで別の ORDER BY 句を指定しない限り、データは常にその順序で表示されます。他の並べ替え処理と同様に、データを検索するときに ORDER BY 句を使用すると、特に大きなテーブルの場合、パフォーマンスがいくらか低下します。結果において出力の順序が重要でない場合、ORDER BY 句を使用しないでクエリを実行すると、通常は高速になります。

インラインビューと一時テーブルの使用

これは、多くの場合、問題を調べてクエリを複数のステップに分割し、中間結果を保持する一時テーブルを作成するときに役立ちます。問題が解決した後で、インラインビューを使用してクエリを 1 つに結合すると、さらに効率的です。ただし、状況によっては、一時テーブルを使用したほうが効率的な場合があります。どちらが効率的かを決定するには、両方の方法を試す必要があります。

結合を使用したサブクエリの比較

サブクエリは、多くの場合、結合によっても表現できます。通常、結合は、少なくともサブクエリと同程度に効率的に処理されます。PROC SQL は、相関する列の一意のセットごとに、一時的に結果の値を格納します。これによって、サブクエリを何度も計算する必要がなくなります。

WHERE 式と結合の併用

テーブルを結合する場合、WHERE 式を指定する必要があります。WHERE 式を使用しないで結合すると、デカルト積の乗数による影響のため、多くの場合、評価に時間がかかります。たとえば、それぞれ 1,000 行の 2 つのテーブルを、WHERE 式または ON 句を指定しないで結合すると、100 万行の結果テーブルが生成されます。

次に示すように、バランスの悪い WHERE 式(または ON 結合式)を等結合で指定して PROC SQL を実行した場合でも、正しい結果が得られます。ただし、処理効率は悪くなります。

```
where table1.columnA-table2.columnB=0
```

式がバランスを保つように、この句を次のように書き換えて、各テーブルの列を等号の両側に配置したほうが効率的です。

```
where table1.columnA=table2.columnB
```

PROC SQL は、各行を WHERE 式に対して評価する等結合条件が含まれない結合については、逐次的に処理します。つまり、等結合条件のない結合は、ソートマージやインデックス検索の手法を使用して評価されません。左外部結合および右外部結合を評価する速度は、通常、標準的な内部結合の速度と同等か、わずかに遅くなるだけです。完全外部結合の場合、通常は、結合に含まれる両方のテーブルを 2 回調べる必

要がありますが、PROC SQL は、できるだけ多くのデータをバッファに格納しようとします。したがって、小さいテーブルの場合、物理的にデータを 1 回読み込むだけで外部結合が処理されることがあります。

PUT 関数の最適化

PUT 関数の減少

PUT 関数を最適化してクエリのパフォーマンスを改善できる、複数の方法があります。データベースのテーブルを参照している場合、PUT 関数への参照を削除することによって、さらに多くのクエリをデータベースに渡すことができます。これによって、デフォルトの Base SAS エンジンでの SELECT ステートメントの評価を単純化することができます。

PUT 関数を最適化する場合に実行される可能性のある評価は、次の 5 つです。

- リテラル値を含む、PUT などの関数。
- SAS が提供する出力形式を含む WHERE 句または HAVING 句内の PUT 関数。
- ユーザー定義の出力形式を含む WHERE 句または HAVING 句内の PUT 関数。
- OTHER=句で定義されたユーザー定義の出力形式を含む、SELECT ステートメントのいずれかの部分の PUT 関数。
- データベース内に配置された PUT 関数。

PUT 関数最適化の制御

- PROC SQL の REDUCEPUT=オプションまたは SQLREDUCEPUT=システムオプションを指定した場合、SAS は、クエリを実行する前に PUT 関数を最適化します。

  次の SELECT ステートメントは、最適化されるクエリの例です。

  ```
  select x, y from sqllibb where (PUT(x, abc.) in ('yes', 'no'));
  select x from sqlliba where (PUT(x, udfmt.) = trim(left('small')));
  ```

- テーブルの行数がわからないときに暗黙的なパススルーを許可するデータベースの場合、PROC SQL は、データベースによってクエリを実行するために、最適化を許可します。PROC SQL の REDUCEPUT=オプションまたは SQLREDUCEPUT=システムオプショを DBMS、BASE または ALL に設定した場合、PROC SQL は、PROC SQL の REDUCEPUTOBS=オプションまたは SQLREDUCEPUTOBS=システムオプションの値を考慮して、PUT 関数を最適化するかどうかを決定します。PROC SQL の REDUCEPUTOBS=オプションまたは SQLREDUCEPUTOBS=システムオプションは、PROC SQL がクエリ内の PUT 関数の最適化を考慮するためにテーブルに存在する必要のある、行の最小数を指定します。暗黙的なパススルーを許可しないデータベースの場合、PROC SQL は最適化を実行せず、より多くのクエリが SAS によって実行されます。
- 出力形式(特に、ユーザー定義の出力形式)によっては、多くの出力形式値が含まれる場合があります。特定の PUT 関数式に対応する数に応じて、結果として得られる式には、多くの出力形式値が記述される場合があります。出力形式値の数が多すぎる場合、クエリのパフォーマンスが低下する可能性があります。PROC SQL の REDUCEPUT=オプションまたは SQLREDUCEPUT=システムオプショを DBMS、BASE または ALL に設定した場合、PROC SQL は、PROC SQL の REDUCEPUTVALUES=オプションまたは SQLREDUCEPUTVALUES=システムオプションの値を考慮して、クエリ内の PUT 関数を最適化するかどうかを決定します。暗黙的なパススルーを許可しないデータベースの場合、PROC SQL は最適化を実行せず、より多くのクエリが SAS によって実行されます。

詳細は、7 章, “SQL プロシジャ” (213 ページ)、の REDUCEPUT=、REDUCEPUTOBS=、REDUCEPUTVALUES=の各オプションおよび付録 1, “SQL マクロ変数とシステムオプション” (369 ページ)の SQLREDUCEPUT=、SQLREDUCEPUTOBS=、SQLREDUCEPUTVALUES=の各システムオプションを参照してください。

注: PROC SQL は、PUT 関数を最適化するかどうか決定しようとするときに、REDUCEPUTOBS=オプションと REDUCEPUTVALUES=オプション(または SQLREDUCEPUTOBS=システムオプションと SQLREDUCEPUTVALUES=システムオプション)の両方を考慮できます。

DBMS 内への PUT 関数と SAS 出力形式の配置

リレーショナルデータベース用の SAS/ACCESS ソフトウェアでは、出力形式をパブリッシュするマクロを使用して、PUT 関数の実装を SAS_PUT()という関数名でデータベースに配置または公開できます。SAS_PUT()関数は、他のプログラム関数と同様に、1 つ以上の入力パラメータを受け取って、出力値を返すことができます。SQLMAPPUTTO システムオプションのデフォルト値は、SAS_PUT です。SAS_PUT()関数をデータベースに配置すると、データベース内の標準 SQL 関数と同じように SAS_PUT()関数を使用できます。

さらに、SAS_PUT()関数は、データベースにサブミットされる SQL クエリでの、SAS 出力形式の使用をサポートします。SAS が提供する出力形式と、FORMAT プロシジャを使用して作成したカスタム出力形式の両方を、出力形式をパブリッシュするマクロを使用してデータベースに公開できます。

SAS 出力形式の使用をサポートする SAS_PUT()関数として、PUT 関数の実装をデータベースにパブリッシュし、SAS が提供する出力形式および FORMAT プロシジャを使用して作成したカスタム出力形式の両方をパッケージ化することにより、次のメリットが得られます。

- SQL クエリ全体をデータベース内部で処理できます。
- SAS 出力形式処理で、DBMS の拡張可能なアーキテクチャを利用できます。
- 結果が、フォーマットされたデータごとにグループ化され、データベースから抽出されます。

注: LIBNAME ステートメントの SQL_FUNCTIONS=オプションを使用して PUT 関数(たとえば、SAS_PUT())を再マッピングした場合、LIBNAME ステートメントの SQL_FUNCTIONS=オプションは、SQLMAPPUTTO=システムオプションよりも優先されます。詳細については、“SQL_FUNCTIONS= LIBNAME Option” (*SAS/ACCESS for Relational Databases: Reference*)を参照してください。

ヒント SQLREDUCEPUT=システムオプション(または PROC SQL REDUCEPUT=オプション)と SAS_PUT()関数の両方を使用すると、パフォーマンスが大幅に向上する場合があります。

in-database 出力形式をパブリッシュするマクロと SQLMAPPUTTO システムオプションの使用の詳細については、*SAS/ACCESS for Relational Databases: Reference* を参照してください。

DATE、TIME、DATETIME、TODAY 関数への参照の置換

PROC SQL の CONSTDATETIME オプションまたは SQLCONSTDATETIME システムオプションを設定した場合、PROC SQL は、クエリ内の DATE 関数、TIME 関数、DATETIME 関数、TODAY 関数を一度評価し、クエリを通じてそれらの値を使用します。これらの値を一度計算すると、クエリで関数を複数回使用した場合、またはクエリが日付や時刻の境界近くで関数を実行した場合に結果の一貫性が保たれます。これ

によって、より多くのクエリをデータベースに渡せるようになるため、データベーステーブルを参照するときのパフォーマンスが向上します。

詳細は、“SQLCONSTDATETIME System Option” (369 ページ)または *Base SAS プロシジャガイド*の CONSTDATETIME オプションを参照してください。

注: PROC SQL の REDUCEPUT オプションまたは SQLREDUCEPUT=システムオプションおよび PROC SQL の CONSTDATETIME オプションまたは SQLCONSTDATETIME システムオプションの両方を指定した場合、PROC SQL は、PUT 関数の値を決定するために、クエリを実行する前に、DATE 関数、TIME 関数、DATETIME 関数、TODAY 関数を、それぞれの値で置き換えます。

要約関数の使用時にデータの再マージを無効化する

SELECT 句または HAVING 句で要約関数を使用したときに、PROC SQL は、データを再マージする場合があります。データの再マージには、2 回のデータのパススルーが含まれます。PROC SQL の NOREMERGE オプションまたは NOSQLREMERGE システムオプションを設定すると、PROC SQL は、データの再マージを処理しなくなります。これによって、より多くのクエリをデータベースに渡せるようになるため、データベーステーブルを参照するときのパフォーマンスが向上します。

詳細は、*Base SAS プロシジャガイド*の PROC SQL ステートメントの REMERGE オプションおよび付録 1, “SQL マクロ変数とシステムオプション” (369 ページ)の SQLREMERGE システムオプションを参照してください。

列エイリアスの使用

列エイリアスの概要

列エイリアスは、列の一時的な代替名です。SELECT 句でエイリアスを指定して、列に名前を付けたり、名前を変更したりします。そうすることで、結果テーブルが明瞭になり、読みやすくなります。多くの場合、エイリアスは、算術演算式や要約関数の結果から得られる列に名前を付けるために使用されます。エイリアスは、1 語のみです。さらに長い名前が必要な場合、“column-modifier” (317 ページ)で説明されているように、LABEL= column-modifier を使用します。SELECT 句内で列エイリアスを列名と区別するために、列エイリアスを指定する場合、AS キーワードが必須です。

列のエイリアスは任意です。SELECT 句内の列名ごとにエイリアスを設定できます。列にエイリアスを割り当てると、他の句でエイリアスを使用してその列を参照できます。

PROC SQL ビューを作成するときに列のエイリアスを使用した場合、そのビューの各実行において、そのエイリアスが永続的な列名になります。

注: SAS SQL プロシジャの外部で使用される SQL コードの可搬性を最大化するために、WHERE 句、GROUP BY 句、または HAVING 句内の列エイリアスを参照するようなコードを記述しないでください。

列エイリアスの拡張機能

PROC SQL の開発スコープとそのエイリアス作成ルールは、最初の ANSI SQL 規格および ISO SQL 規格のスコープとルールよりも前から存在します。PROC SQL では、WHERE 句、ON 句、GROUP BY 句、HAVING 句、ORDER BY 句で列エイリアスを使用できます。ANSI SQL 規格および ISO SQL 規格では、列エイリアスに関連付けられている値は、ORDER BY 句が実行されるまで利用可能となる必要はありません。こ

のため、WHERE 句、GROUP BY 句、HAVING 句で列エイリアスが参照された際に、SQL プロセッサがその列エイリアスを解決できるかどうかは保証されません。ANSI SQL 規格および ISO SQL 規格では、列エイリアスは、ORDER BY 句が実行される際に参照専用で使用可能になることが必要とされるため、WHERE 句、GROUP BY 句、HAVING 句で列エイリアスを参照するようなコードは記述しないでください。

ORDER BY 句で発生する式の一部としてではなく、SQL 式で列エイリアスを参照した場合、式が機能しない場合があります。ORDER BY 句以外の文脈で列エイリアスを参照する場合、各列エイリアス参照の前に CALCULATED キーワードを入れる必要があります。詳細については、“CALCULATED キーワードと列エイリアスの使用” (150 ページ)を参照してください。

ANSI SQL 規格または ISO SQL 規格では、SELECT ステートメントを構成する 6 つの部分にそれぞれ概念的な実行順序が設定されています。6 つの部分すべてが存在する場合、順序は次のようになります。

1. FROM 部が最初に実行されます。
2. WHERE 部または ON 部が 2 番目に実行されます。
3. GROUP BY 部が 3 番目に実行されます。
4. HAVING 部が 4 番目に実行されます。
5. SELECT 部が 5 番目に実行されます。
6. ORDER BY 部が 6 番目に実行されます。

SQL クエリに必須となる部分は、SELECT 句と FROM 句です。それ以外の 4 つの部分はオプションであり、どのオプションを使用するかは実行するクエリの種類に応じて異なります。

SQL クエリの高レベルのテンプレートを次に示します。各部分の右端に示されているかっこで囲まれた番号は、各部分の実行順序を表しています。

```
select <SELECT list> (5)
   from <FROM clause> (1)
   where <WHERE clause> (2)
   group by <GROUP BY clause> (3)
   having <HAVING clause> (4)
   order by <ORDER BY clause>; (6)
```

次のコード例では、各 SELECT ステートメントの最初のエイリアスは、テーブル列の別名となります。2 番目のエイリアスは、計算式を意味します。最初と 2 番目の SQL ステートメントは、PROC SQL の予想結果を出力します。

Here is the preferred SQL code example because a column alias is not referenced in the WHERE clause. This code example is portable to other SQL processors.

```
select qty as Quantity, cost, cost+100 as ListPrice
   from calc
   where qty > 5;
```

This code example will work in PROC SQL, but it might not work with other SQL processors.

```
select qty as Quantity, cost, cost+100 as ListPrice
   from calc
   where Quantity > 5;
```

CALCULATED キーワードと列エイリアスの使用

PROC SQL 開発に対して実施された初期の拡張は、CALCULATED キーワードでした。CALCULATED キーワードは、PROC SQL のユーザーが計算式に関連付けられている列エイリアスを参照できるようにします。CALCULATED キーワードにより参照される列エイリアスは、WHERE 句、GROUP BY 句、HAVING 句、ORDER BY 句に記述できます。CALCULATED キーワードが列エイリアスを参照するために ORDER BY 句で使用されている場合、CALCULATED キーワードの使用は冗長になります。その列エイリアスは、ORDER BY 句が実行される前に解決されます。詳細については、“計算列をエイリアスで参照する” (31 ページ)を参照してください。

CALCULATED キーワードを使用して、2 番目のエイリアス(ListPrice)に関連付けられている値で行をサブセット化する PROC SQL のコード例を次に示します。

例のコード 5.1 *CALCULATED Keyword and the PROC SQL Use of Column Aliases*

```
/*-- PROC SQL use of the CALCULATED keyword --*/
select qty as Quantity, cost, cost+100 as ListPrice
   from calc
   where CALCULATED ListPrice > 1500;
```

Here is an ISO SQL standard- and ANSI SQL standard-approved way of accomplishing this task:

例のコード 5.2 *CALCULATED Keyword and the PROC SQL Use of Column Aliases*

```
/*-- PROC SQL use of the CALCULATED keyword --*/
select qty as Quantity, cost, cost+100 as ListPrice
   from calc
   where cost+100 > 1500;
```

上記のコード例には可搬性があります。

関連項目:

- “列の新規作成” (27 ページ)
- “列のエイリアスを割り当てる” (30 ページ)
- “計算列をエイリアスで参照する” (31 ページ)
- “CALCULATED” (313 ページ)
- “SELECT ステートメント” (252 ページ)
- “WHERE 句” (260 ページ)
- “GROUP BY 句” (261 ページ)
- “HAVING 句” (262 ページ)
- “ORDER BY 句” (263 ページ)

DICTIONARY テーブルを使用し、SAS System の情報にアクセスする

ディクショナリテーブルについて

DICTIONARY テーブルは、読み取り専用の、特殊な PROC SQL のテーブルまたはビューです。これらテーブルから、現在の SAS セッションに関連するすべての SAS ライブラリ、SAS データセット、SAS システムオプション、外部ファイルに関する情報を取得できます。たとえば、DICTIONARY.COLUMNS テーブルには、現在の SAS セッションでわかっているすべてのテーブルのすべての列に関する名前、タイプ、長さ、出力形式などの情報が含まれます。

PROC SQL は、自動的に DICTIONARY にライブラリ参照名を割り当てます。DICTIONARY テーブルから情報を取得するには、PROC SQL の SELECT ステートメントの FROM 句で、DICTIONARY.table-name を指定します。

DICTIONARY.table-name は、PROC SQL でのみ有効です。ただし、SAS では DICTIONARY テーブルに基づく PROC SQL ビューが提供されており、これらは、他の SAS プロシジャや DATA ステップでも使用できます。これらのビューは、Sashelp ライブラリに格納され、通常、“Sashelp ビュー”と呼ばれます。

DICTIONARY テーブルの例については、次を参照してください。“例 6: DICTIONARY テーブルからレポートを作成する” (279 ページ)

次の表は、使用可能な DICTIONARY テーブルについての説明と、各テーブルに関連する Sashelp ビューを示しています。

表 5.1 DICTIONARY テーブルおよび関連する Sashelp ビュー

DICTIONARY テーブル	Sashelp ビュー	説明
CATALOGS	Vcatalg	既知の SAS カタログに関する情報が含まれます。
CHECK_CONSTRAINTS	Vchkcon	既知のチェック制約に関する情報が含まれます。
COLUMNS	Vcolumn	既知のすべてのテーブルの列に関する情報が含まれます。
CONSTRAINT_COLUMN_USAGE	Vcncolu	一貫性制約によって参照される列に関する情報が含まれます。
CONSTRAINT_TABLE_USAGE	Vcntabu	一貫性制約が定義されているテーブルに関する情報が含まれます。
DATAITEMS	Vdatait	既知の Information Map データ項目に関する情報が含まれます。
DESTINATIONS	Vdest	既知の ODS 出力先に関する情報が含まれます。
DICTIONARIES	Vdctnry	すべての DICTIONARY テーブルに関する情報が含まれます。

DICTIONARY テーブル	Sashelp ビュー	説明
ENGINES	Vengine	SAS エンジンに関する情報が含まれます。
EXTFILES	Vextfl	既知の外部ファイルに関する情報が含まれます。
FILTERS	Vfilter	既知の Information Map フィルタに関する情報が含まれます。
FORMATS	Vformat Vcformat	現在使用可能な出力形式と入力形式に関する情報が含まれます。
FUNCTIONS	Vfunc	現在使用可能な関数に関する情報が含まれます。
GOPTIONS	Vgopt Vallopt	現在定義されているグラフオプション(SAS/GRAPH ソフトウェア)に関する情報が含まれます。Sashelp.Vallopt には、グラフオプションに加えて、SAS システムオプションも含まれています。
INDEXES	Vindex	既知のインデックスに関する情報が含まれます。
INFOMAPS	Vinfomp	既知の Information Map に関する情報が含まれます。
LIBNAMES	Vlibnam	現在定義されている SAS ライブラリに関する情報が含まれます。
MACROS	Vmacro	現在定義されているマクロ変数に関する情報が含まれます。
MEMBERS	Vmember Vsacces Vscatlg Vslib Vstable Vstabvw Vsview	現在 SAS ライブラリに定義されているすべてのオブジェクトに関する情報が含まれます。Sashelp.Vmember には、すべてのメンバタイプに関する情報が含まれています。他の Sashelp ビューは、特定のメンバタイプ(テーブルやビューなど)に固有のビューです。
OPTIONS	Voption Vallopt	SAS システムオプションに関する情報が含まれます。Sashelp.Vallopt には、SAS システムオプションに加えて、グラフオプションも含まれています。
REFERENTIAL_CONSTRAINTS	Vrefcon	参照制約に関する情報が含まれます。
REMEMBER	Vrememb	既知の保存情報が含まれます。
STYLES	Vstyle	既知の ODS スタイルに関する情報が含まれます。
TABLE_CONSTRAINTS	Vtabcon	既知のすべてのテーブルの一貫性制約に関する情報が含まれます。

DICTIONARY テーブル	Sashelp ビュー	説明
TABLES	Vtable	既知のテーブルに関する情報が含まれます。
TITLES	Vtitle	現在定義されているタイトルとフットノートに関する情報が含まれます。
VIEWS	Vview	既知のデータビューに関する情報が含まれます。
VIEW_SOURCES	Not available	SQL または DATASTEP ビューが参照するテーブル(または他のビュー)のリスト、および参照数の合計が含まれます。
XATTRS	Vxattr	拡張属性に関する情報を含んでいます。

DICTIONARY テーブルと Sashelp ビューの情報の取得

それぞれの DICTIONARY テーブルの定義を表示するには、DESCRIBE TABLE ステートメントをサブミットします。次の例では、DICTIONARY.Tables の定義を表示しています。

```
proc sql;
   describe table dictionary.tables;
```

この結果は、SAS ログに書き込まれます。

ログ 5.3 *DICTIONARY.Tables の定義*

```
注:create table DICTIONARY.TABLES ( libname char(8) label='Library Name',
memname char(32) label='Member Name', memtype char(8) label='Member Type',
dbms_memtype char(32) label='DBMS Member Type', memlabel char(256) label='Data
Set Label', typemem char(8) label='Data Set Type', crdate num format=DATETIME
informat=DATETIME label='Date Created', modate num format=DATETIME
informat=DATETIME label='Date Modified', nobs num label='Number of Physical
Observations', obslen num label='Observation Length', nvar num label='Number of
Variables', protect char(3) label='Type of Password Protection', compress
char(8) label='Compression Routine', encrypt char(8) label='Encryption', npage
num label='Number of Pages', filesize num label='Size of File', pcompress num
label='Percent Compression', reuse char(3) label='Reuse Space', bufsize num
label='Bufsize', delobs num label='Number of Deleted Observations', nlobs num
label='Number of Logical Observations', maxvar num label='Longest variable
name', maxlabel num label='Longest label', maxgen num label='Maximum number of
generations', gen num label='Generation number', attr char(3) label='Data Set
Attributes', indxtype char(9) label='Type of Indexes', datarep char(32)
label='Data Representation', sortname char(8) label='Name of Collating
Sequence', sorttype char(4) label='Sorting Type', sortchar char(8)
label='Charset Sorted By', reqvector char(24) format=$HEX48 informat=$HEX48
label='Requirements Vector', datarepname char(170) label='Data Representation
Name', encoding char(256) label='Data Encoding', audit char(8) label='Audit
Trail Active?', audit_before char(8) label='Audit Before Image?', audit_admin
char(8) label='Audit Admin Image?', audit_error char(8) label='Audit Error
Image?', audit_data char(8) label='Audit Data Image?', num_character num
label='Number of Character Variables', num_numeric num label='Number of Numeric
Variables' );
```

同様に、PROC SQL で DESCRIBE VIEW ステートメントを使用して、Sashelp ビューの定義を確認できます。次に例を示します。

```
proc sql;
   describe view sashelp.vstabvw;
```

ログ 5.4 Sashelp.Vstabvw の説明

```
注:SQL view SASHELP.VSTABVW is defined as:select libname, memname, memtype from
DICTIONARY.MEMBERS where (memtype='VIEW') or (memtype='DATA') order by libname
asc, memname asc;
```

DICTIONARY.Tables の使用

DICTIONARY テーブルは、PROC DATASETS などの他のソースの出力よりも簡単にデータを操作できるため、通常は SAS セッションの監視と管理に使用されます。DICTIONARY テーブルは、WHERE 句によるサブセット化、結果の並べ替え、PROC SQL ビューの作成を含め、他のテーブルの照会と同じ方法で照会できます。

なお、DICTIONARY テーブルの多数の文字値は、すべて大文字で格納されています。そのため、それに合わせてクエリを作成する必要があります。

DICTIONARY テーブルは、読み取り専用のオブジェクトです。そのため、DICTIONARY テーブルに対して、行や列の挿入、行属性の変更、一貫性制約の追加などは実行できません。

注: DICTIONARY.Tables および Sashelp.Vtable では、テーブルの読み取りがパスワードによって保護されている場合、そのテーブルについて表示される情報は、ライブラリ名、メンバ名、メンバタイプ、パスワード保護のタイプのみです。他のすべての情報は、欠損として設定されます。

注: ある SQL ビューが 1 つのライブラリに存在し、割り当てられていない別のライブラリの入力テーブルを含んでいる場合、その SQL ビューに関する情報を DICTIONARY.Tables を使用して取得しようとすると、エラーが発生します。

次のクエリでは、SELECT 句とサブセット化のための WHERE 句を使用して、SQL ライブラリに現れる永続的なテーブルとビューに関する情報を取得しています。

```
libname sql 'SAS-library';

proc sql;
   title 'All Tables and Views in the SQL Library';
   select libname, memname, memtype, nobs
      from dictionary.tables
      where libname='SQL';
```

アウトプット 5.2 このドキュメントで使用されテーブルとビュー

All Tables and Views in the SQL Library

Library Name	Member Name	Member Type	Number of Physical Observations
SQL	A	DATA	4
SQL	ACT	DATA	10
SQL	B	DATA	3
SQL	CAL	DATA	2
SQL	CONTINENTS	DATA	9
SQL	COUNTRIES	DATA	208
SQL	DENSITIES	DATA	10
SQL	FEATURES	DATA	74
SQL	HOL	DATA	2
SQL	MYSTATES	DATA	0
SQL	NEWCONTINENTS	VIEW	.
SQL	NEWCOUNTRIES	DATA	6
SQL	NEWSTATES	DATA	0
SQL	OILPROD	DATA	19
SQL	OILRSRVS	DATA	19
SQL	ONE	DATA	2
SQL	POSTALCODES	DATA	58
SQL	TWO	DATA	3
SQL	UNITEDSTATES	DATA	51
SQL	USCITYCOORDS	DATA	132
SQL	USPOSTAL	DATA	0
SQL	WOR	DATA	2
SQL	WORLDCITYCOORDS	DATA	212
SQL	WORLDTEMPS	DATA	59

DICTIONARY.Columns の使用

DICTIONARY テーブルは、特定の列を検索してレポートに含める場合に役立ちます。次のクエリでは、このドキュメントで使用されるテーブルのうち、Country 列を含むテーブルを表示しています。

```
libname sql 'SAS-library';
```

```
proc sql;
   title 'All Tables That Contain the Country Column';
   select libname, memname, name
      from dictionary.columns
      where name='Country' and
            libname='SQL';
```

アウトプット 5.3 *特定の列を見つけるための DICTONARY.Columns の使用*

All Tables That Contain the Country Column

Library Name	Member Name	Column Name
SQL	OILPROD	Country
SQL	OILRSRVS	Country
SQL	WORLDCITYCOORDS	Country
SQL	WORLDTEMPS	Country

DICTIONARY テーブルとパフォーマンス

DICTIONARY テーブルを照会すると、SAS は、このテーブルに関連する情報を収集する検索プロセスを起動します。この検索処理は、照会される DICTIONARY テーブルに応じて、ライブラリの検索、テーブルのオープン、ビューの実行などを行なえます。他の SAS プロシジャや DATA ステップとは異なり、PROC SQL は、検索処理を起動する前にクエリを最適化することによって、検索処理を軽減できます。したがって、SAS プロシジャまたは DATA ステップによって、Sashelp ビューを使用して DICTIONARY テーブルの情報にアクセスすることは可能ですが、多くの場合、かわりに PROC SQL を使用したほうが効率的です。

注: DICTIONARY テーブルでは、データセットオプションは使用できません。

たとえば、次のプログラムはどちらも同じ結果を生成しますが、PROC SQL ステップのほうが非常に高速に実行されます。これは、Sashelp.Vcolumn ビューが参照するテーブルが開かれる前に、WHERE 句が処理されるためです。

```
data mytable;
   set sashelp.vcolumn;
   where libname='WORK' and memname='SALES';
run;

proc sql;
   create table mytable as
      select * from sashelp.vcolumn
      where libname='WORK' and memname='SALES';
quit;
```

注: SAS は、DICTIONARY テーブルの情報をクエリ間で維持しません。
DICTIONARY テーブルに対するクエリごとに、新しい検索処理が起動されます。

ある行で何度も同じ DICTIONAR テーブルを照会する場合、必要な情報を含む一時的な SAS データセットを(DATA ステップの SET ステートメントまたは PROC SQL の CREATE TABLE AS ステートメントを使用して)作成し、それに対してクエリを実行することによって、さらに高速なパフォーマンスを得ることができます。

DICTIONARY.Tables または Sashelp.Vtable を照会した場合、SAS セッションに割り当てられたすべてのライブラリに含まれるテーブルとビューがすべて開かれ、要求された情報が検索されます。

WHERE 句を使用して、検索対象のライブラリを制限できます。ただし、この WHERE 句は、UPCASE などのほとんどの関数呼び出しを処理しません。

たとえば、**where UPCASE (libname) ='WORK'**を使用した場合、UPCASE 関数は、この WHERE 句の条件の最適化を妨げます。その結果、SAS セッションに割り当てられたすべてのライブラリが検索されます。すべてのライブラリを検索すると、SAS セッションに割り当てられているライブラリの数によっては、予想外に検索時間が増える場合があります。

すべてのライブラリ参照名と SAS テーブル名は、大文字で格納されます。LIBNAME と MEMNAME の値を大文字で指定し、UPCASE 関数を削除すれば、WHERE 句が最適化され、パフォーマンスが向上します。前述の例の場合は、コードを **where libname='WORK'**に変更します。

注: すべてのライブラリ参照名を検索すると、予期せぬ結果を引き起こす場合があります。すべてのライブラリ参照名を検索すると、SAS セッションに現在割り当てられていないライブラリ参照名を含むビューが存在する可能性があります。このビューをクエリで情報を取り出すためにオープンすると、エラーが発生します。

注: 外部データベースに割り当てられたライブラリのテーブル情報を照会する場合、LIBNAME ステートメントの PRESERVE_TAB_NAMES=YES オプションまたは PRESERVE_COL_NAMES=YES オプションを使用して、データベースで表示されるとおりにテーブル名と列名を指定するのであれば、UPCASE 関数を使用する必要はありません。

PROC SQL で SAS データセットオプションを使用する

PROC SQL でテーブルまたは SAS/ACCESS ビューを指定する際にはいつでも、テーブルまたは SAS/ACCESS ビューに対して、KEEP=や DROP=などの、ほとんどの SAS データセットオプションを適用できます。SQL プロシジャでは、スペースで区切られた SAS データセットオプションは、かっこで囲みます。データセットオプションは、テーブル名または SAS/ACCESS ビュー名の直後に記述します。次の PROC SQL ステップでは、RENAME=データセットオプションによって、Staff1 テーブルの LNAME を LASTNAME に名前変更しています。OBS=データセットオプションによって、Staff1 から読み取られる行の数を 15 行に制限しています。

```
proc sql;
   create table
          staff1(rename=(lname=lastname)) as
      select *
         from staff(obs=15);
```

SAS データセットオプションは、SQL ステートメントの引数と組み合わせることができます。次の PROC SQL ステップでは、PW=データセットオプションによって Test テーブルにパスワードを割り当て、ALTER=データセットオプションによって Staff1 テーブルに ALTER パスワードを割り当てています。

```
proc sql;
   create table test
      (a character, b numeric, pw=cat);
   create index staffidx on
      staff1 (lastname, alter=dog);
```

次の PROC SQL ステップでは、PW=データセットオプションによって ONE テーブルにパスワードを割り当てています。このパスワードは、行を挿入するとき、およびテーブルを更新するときに使用されます。

```
proc sql;
    create table one(pw=red, col1 num, col2 num, col3 num);
   quit;

proc sql;
   insert into one(pw=red, col1, col3)
   values(1, 3);
quit;
proc sql;
   update one(pw=red)
      set col2 = 22
         where col2 = . ;
quit;
```

DICTIONARY テーブルが読み取り専用のオブジェクトであるため、DICTIONARY テーブルでは SAS データセットオプションを使用できません。

PROC SQL ビューで使用できる SAS データセットオプションは、SAS パスワードの割り当ておよび指定を行うデータセットオプション(READ=、WRITE=、ALTER=、PW=)のみです。

SAS データセットオプションの詳細については、*SAS データセットオプション: リファレンス*を参照してください。

PROC SQL を SAS マクロ機能とともに使用する

概要:PROC SQL を SAS マクロ機能とともに使用する

マクロ機能は、SAS ソフトウェアの拡張とカスタマイズに使用できるプログラミングツールです。マクロ機能を使用することで、共通タスクまたは繰り返しタスクを実行するために入力する必要のあるテキストの量が減り、SQL プログラムの効率と利便性が向上します。

マクロ機能を使用して、文字列または SAS プログラムステートメントのグループに名前を割り当てることができます。これ以降は、テキスト自体ではなく、名前を使用して作業できます。SAS マクロ機能の詳細については、*SAS マクロ言語: リファレンス*を参照してください。

マクロ変数は、SAS コードのテキスト文字列を置き換える効率的な手段を提供します。作成して名前を付けたマクロ変数は、ユーザー定義マクロ変数と呼ばれます。SAS によって定義されたマクロ変数は、自動マクロ変数と呼ばれます。PROC SQL は、問題のトラブルシューティングに役立つ 6 つの自動マクロ変数(SQLOBS、SQLRC、SQLOOPS、SQLEXITCODE、SQLXRC、SQLXMSG)を生成します。詳細については、"PROC SQL 自動マクロ変数の使用" (163 ページ)を参照してください。

PROC SQL のマクロ変数の作成

概要:PROC SQL のマクロ変数の作成

他のソフトウェアベンダの SQL 製品では、SQL を別の言語に埋め込むことができます。その言語の変数(列)への参照は、ホスト変数参照と呼ばれます。これらは、名前

にコロンの接頭語を付加することによって、テーブル列への参照とは区別されます。ホスト変数には、SELECT 句に記述されたオブジェクト項目の値が格納されます。

現在 SAS で使用可能なホスト言語は、Base SAS ソフトウェアに含まれるマクロ言語のみです。列の値に対して計算を実行するときに、マクロ機能の:macro-variable を使用して結果を格納できます。次にその結果を、別の PROC SQL クエリまたは SAS プロシジャで、名前によって参照できます。ホスト変数は、SELECT ステートメントのサブクエリではなく、外側のクエリでのみ使用できます。ホスト変数は、CREATE ステートメントでは使用できません。

クエリによって複数行の出力を生成した場合、マクロ変数には、最初の行の値のみが含まれます。クエリの出力に行が存在しない場合、マクロ変数は変更されません。マクロ変数が存在しない場合、マクロ変数は作成されません。PROC SQL のマクロ変数の SQLOBS には、クエリが生成した行の数が格納されます。

注: SQL SELECT ステートメントの実行後、SQLOBS 自動マクロ変数に値が割り当てられます。

クエリ結果の先頭行からマクロ変数を作成する

INTO 句で 1 つのマクロ変数を指定すると、PROC SQL は、SELECT に記述された該当する列の最初の行の値のみを、マクロ変数に割り当てます。この例では、Country 列の最初の行の値を&country1 に割り当て、Barrels 列の最初の行の値を&barrels1 に割り当てます。NOPRINT オプションによって、PROC SQL がクエリの結果を表示しないようにします。%PUT ステートメントによって、マクロ変数の内容を SAS ログに書き込みます。

```
libname sql 'SAS-library';

proc sql noprint;
   select country, barrels
      into :country1, :barrels1
      from sql.oilrsrvs;

%put &country1 &barrels1;
```

ログ 5.5 *クエリ結果の先頭行からマクロ変数を作成する*

```
4 proc sql noprint; 5 select country, barrels 6 into :country1, :barrels1 7 from
sql.oilrsrvs; 8 9 %put &country1 &barrels1; Algeria 9,200,000,000 NOTE:PROCEDURE
SQL used:real time 0.12 seconds
```

集計関数の結果からマクロ変数を作成する

マクロ変数の便利な機能の 1 つは、データ値を SAS タイトルに表示できることです。次の例では、Worldtemps テーブルのサブセットを出力し、カナダの最高気温をタイトルに表示しています。

```
libname sql 'SAS-library';

proc sql outobs=12;
   reset noprint;
   select max(AvgHigh)
      into :maxtemp
      from sql.worldtemps
      where country = 'Canada';
reset print;
   title "The Highest Temperature in Canada: &maxtemp";
   select city, AvgHigh format 4.1
```

```
        from sql.worldtemps
        where country = 'Canada';
```

注: TITLE ステートメントでは、マクロ変数への参照を展開するために二重引用符を使用する必要があります。

注: デフォルトでは、大きな数値を含んでいるマクロ変数は、BEST8.出力形式を使用してフォーマットされます。この出力形式を使用すると、科学的記数法を使用して値が表示されます。科学的記数法を使用せずに値を表示する場合、別の出力形式(*w*.など) を使用します。

```
select sum(population) format=16.
   into :totpop from sql.countries;
```

アウトプット 5.4 *マクロ変数参照をタイトルに含める*

The Highest Temperature in Canada: 80

City	AvgHigh
Montreal	77.0
Quebec	76.0
Toronto	80.0

複数のマクロ変数の作成

SELECT ステートメントの結果の行ごとに、1 つの新しいマクロ変数を作成できます。マクロ変数の範囲を作成するには、INTO 句で THROUGH キーワード、THRU キーワードまたはハイフン(-)を使用します。

注: マクロ変数の範囲を指定した場合、SAS マクロ機能は、必要な数のマクロ変数のみを作成します。たとえば、**:var1-:var9999** を指定した場合、55 個の変数のみが必要であれば、**:var1-:var55** のみが作成されます。プログラムのその後の部分で、実際に作成された変数の個数を知る必要がある場合、SQLOBS 自動変数が役立ちます。この例では、SQLOBS には 55 の値が格納されます。

次の例では、Name 列の最初の 4 行と、Population 列の最初の 3 行の値をマクロ変数に割り当てています。%PUT ステートメントによって、結果を SAS ログに書き込みます。

```
libname sql 'SAS-library';

proc sql noprint;
   select name, Population
      into :country1 - :country4, :pop1 - :pop3
           from sql.countries;

%put &country1 &pop1;
%put &country2 &pop2;
%put &country3 &pop3;
%put &country4;
```

ログ 5.6 複数のマクロ変数の作成

```
4 proc sql noprint; 5 select name, Population 6 into :country1
- :country4, :pop1 - :pop3 7 from sql.countries; 8 9 %put &country1 &pop1;
Afghanistan 17070323 10 %put &country2 &pop2; Albania 3407400 11 %put &country3
&pop3; Algeria 28171132 12 %put &country4; Andorra
```

マクロ変数の値の連結

1 つの列のそれぞれの値を 1 つのマクロ変数に連結することができます。このフォームは、変数または定数のリストを作成する場合に役立ちます。SEPARATED BY キーワードを使用して、マクロ変数の値を区切る文字を指定します。

この例では、Countries テーブルの Name 列の最初の 5 つの値を、&countries マクロ変数に割り当てます。INOBS オプションによって、PROC SQL が使用する行を、Countries テーブルの最初の 5 行に制限します。マクロ変数の値は、カンマとスペースによって区切られます。

```
libname sql 'SAS-library';

proc sql noprint inobs=5;
   select Name
      into :countries separated by ', '
      from sql.countries;

%put &countries;
```

ログ 5.7 マクロ変数の値の連結

```
4 proc sql noprint inobs=5; 5 select Name 6 into :countries separated by ', ' 7
from sql.countries; WARNING:Only 5 records were read from SQL.COUNTRIES due to
INOBS= option.8 9 %put &countries; Afghanistan, Albania, Algeria, Andorra, Angola
```

マクロ変数を作成する前に、値の前と後の空白を切り取ります。空白を切り取らない場合は、INTO 句に NOTRIM を追加します。前述の例に NOTRIM を追加したコードを、次に示します。

```
libname sql 'SAS-library';

proc sql noprint inobs=5;
   select Name
      into :countries separated by ',' NOTRIM
      from sql.countries;

%put &countries;
```

ログ 5.8 マクロ変数の値の連結

```
1 proc sql noprint inobs=5; 2 select Name 3 into :countries separated by ','
NOTRIM 4 from sql.countries; WARNING:Only 5 records were read from SQL.COUNTRIES
due to INOBS= option.5 6   %put &countries;
Afghanistan                        ,Albania                             ,Algeria ,
Andorra                            ,Angola
```

テーブル作成のマクロの定義

マクロは、テーブルを作成するためのインターフェイスとして役立ちます。新しいテーブルを作成する場合や、既存のテーブルに行を追加する場合に、SAS マクロ機能を使用すると便利です。

次の例では、学術論文の審査の査読者を表示するテーブルを作成しています。テーブルには、1 つの題目につき 4 人以上を含めることはできません。この例で定義されたマクロは、査読者の数をチェックしてから、新しい査読者の名前をテーブルに挿入します。このマクロには、査読者の名前と学術論文の題目という 2 つのパラメータがあります。

```
libname sql 'SAS-library';

proc sql;
create table sql.referee
   (Name      char(15),
    Subject   char(15));

   /* define the macro */
%macro addref(name,subject);
%local count;

   /* are there three referees in the table? */
reset noprint;
   select count(*)
      into :count
      from sql.referee
      where subject="&subject";

%if &count ge 3 %then %do;
   reset print;
   title "ERROR: &name not inserted for subject - &subject..";
   title2 "        There are 3 referees already.";
   select * from sql.referee where subject="&subject";
   reset noprint;
   %end;

%else %do;
   insert into sql.referee(name,subject) values("&name","&subject");
   %put NOTE: &name has been added for subject - &subject..;
   %end;

%mend;
```

2 つのパラメータを指定して%ADDREF()マクロをサブミットし、査読者の名前をテーブルに追加します。マクロをサブミットするたびに、SAS ログにメッセージが書き込まれます。

```
%addref(Conner,sailing);
%addref(Fay,sailing);
%addref(Einstein,relativity);
%addref(Smythe,sailing);
%addref(Naish,sailing);
```

ログ 5.9 *テーブル作成のマクロの定義*

```
34 %addref(Conner,sailing); NOTE:1 row was inserted into SQL.REFEREE. NOTE:
注:Conner has been added for subject - sailing.35  %addref(Fay,sailing); NOTE:1
row was inserted into SQL.REFEREE. NOTE:注:Fay has been added for subject -
sailing.36  %addref(Einstein,relativity); NOTE:1 row was inserted into
SQL.REFEREE. NOTE:注:Einstein has been added for subject - relativity.
37  %addref(Smythe,sailing); NOTE:1 row was inserted into SQL.REFEREE. NOTE:
注:Smythe has been added for subject - sailing.38  %addref(Naish,sailing);
```

出力には、%ADDREF()マクロの実行ごとに 1 行が追加されます。テーブルに 3 人の査読者の名前が含まれると、それ以上査読者を受け取れないことを示すメッセージが SAS 出力に表示されます。

アウトプット 5.5 *SAS マクロ言語インターフェイスを使用して作成された結果テーブルとメッセージ*

ERROR: Naish not inserted for subject – sailing.
There are 3 referees already.

Name	Subject
Conner	sailing
Fay	sailing
Smythe	sailing

PROC SQL 自動マクロ変数の使用

PROC SQL は、各ステートメントの実行後に、特定の値を使用してマクロ変数を設定します。PROC SQL ステップの実行を続行するかどうか決めるために、これらのマクロ変数をマクロ内でテストできます。

各 PROC SQL ステートメントの実行後に、次のマクロ変数の値が更新されます。

SQLEXITCODE
: 一部のタイプの SQL による挿入が失敗したことにより発生した、最も高いレベルのリターンコードが含まれます。このリターンコードは、PROC SQL の終了時に SYSERR マクロ変数に書き込まれます。

SQLOBS
: SQL プロシジャステートメントが処理した行の数が含まれます。たとえば、SQLOBS マクロ変数には、SELECT ステートメントによってフォーマットされ、SAS 出力に表示された行の数、または DELETE ステートメントによって削除された行の数が格納されます。

  NOPRINT オプションを指定した場合、SQLOBS マクロ変数の値は、出力テーブル、単一のマクロ変数、マクロ変数のリスト、マクロ変数の範囲のうち、いずれが作成されたかによって変わります。

  - 出力テーブルが作成されなかった場合、マクロ変数のリストまたは範囲が作成され、SQLOBS に 1 の値が格納されます。
  - 出力テーブルが作成された場合、SQLOBS には、出力テーブルの行数が格納されます。
  - 単一のマクロ変数が作成された場合、SQLOBS には 1 の値が格納されます。

- マクロ変数のリストまたは範囲が作成された場合、SQLOBS には、マクロ変数のリストまたは範囲を作成するために処理された行の数が格納されます。

SQL ビューが作成された場合、SQLOBS には 0 の値が格納されます。

注: SQL SELECT ステートメントの実行後、SQLOBS 自動マクロ変数に値が割り当てられます。

SQLOOPS
PROC SQL の内部ループで処理された反復の回数が含まれます。反復回数は、クエリの複雑さに比例して増加します。詳細は、"LOOPS=オプションを使用した反復の限定" (141 ページ) および *Base SAS プロシジャガイド*の LOOPS=を参照してください。

SQLRC
SQL プロシジャステートメントが成功したことを示す次のステータス値が含まれます。

0
PROC SQL ステートメントが、エラーなしに正常終了しました。

4
PROC SQL ステートメントが警告を発行する状況を検出しました。ステートメントの実行は継続されました。

8
PROC SQL ステートメントがエラーを検出しました。その時点でステートメントの実行が停止しました。

12
PROC SQL ステートメントで、PROC SQL のバグを示す内部エラーが発生しました。SAS テクニカルサポートに報告してください。このエラーは、コンパイル時にのみ発生します。

16
PROC SQL ステートメントがユーザーエラーを検出しました。たとえば、1 つの値しか返せないサブクエリが 2 つ以上の行を評価した場合に、このエラーコードが使用されます。このエラーは、実行時にのみ発生します。

24
PROC SQL ステートメントでシステムエラーが発生しました。たとえば、ディスクがいっぱいのため、PROC SQL テーブルに書き込めない場合に、このエラーが使用されます。このエラーは、実行時にのみ発生します。

28
PROC SQL ステートメントで、PROC SQL のバグを示す内部エラーが発生しました。SAS テクニカルサポートに報告してください。このエラーは、実行時にのみ発生します。

SQLRC の値は、PROC SQL ステートメントの UNDO_POLICY=オプションまたは SQLUNDOPOLICY システムオプションの値に応じて変わる場合があります。

たとえば、CREATE UNIQUE INDEX ステートメントを使用して定義されたインデックスに重複する値を挿入しようとした場合、UNDO_POLICY=オプションまたは SQLUNDOPOLICY システムオプションの値に応じて、SQLRC のリターンコードは変わります。

- UNDO_POLICY=オプションまたは SQLUNDOPOLICY システムオプションを REQUIRED または OPTIONAL のいずれかに設定し、重複するインデック値を挿入しようとした場合、SAS は、更新の適用前と適用後のテーブルのコピーを作成し、維持しようとします。PROC SQL は、エラー条件を受け取ると、すぐに実行を停止します。SAS によってエラー条件が検出されると、PROC SQL にリターンコードが渡されます。SQLRC には、24 の値が格納されます。

- UNDO_POLICY=オプションまたは SQLUNDOPOLICY システムオプションを NONE に設定し、重複するインデック値を挿入しようとした場合、SAS は、更新の前後のテーブルのコピーを作成しません。SAS は、エラー条件を検出せず、リターンコードを PROC SQL に渡しません。PROC SQL は、更新処理を続行しようとします。SQLRC には、8 の値が格納されます。

SQLXMSG
パススルー機能によって返されたエラーに関する説明情報と、DBMS 固有のリターンコードが含まれます。

注: SQLXMSG マクロ変数の値には、特殊文字(&、%、/、*、;など)が含まれる場合があるため、次の値を出力する場合は%SUPERQ マクロ関数を使用してください。**`%put %superq(sqlxmsg);`** %SUPERQ 関数の詳細については、*SAS マクロ言語: リファレンス*を参照してください。

SQLXRC
パススルー機能によって返された DBMS 固有のリターンコードが含まれます。

PROC SQL によって生成されたマクロ変数は、%LET の適用範囲のルールに従います。マクロ変数の適用範囲の詳細については、*SAS マクロ言語: リファレンス*を参照してください。

SAS/AF ソフトウェアのユーザーは、SAS コンポーネント言語(SCL)プログラムで SYMGET 関数を使用して、これらの自動マクロ変数にアクセスできます。次の例では、SAS/AF ソフトウェアアプリケーションで VALIDATE ステートメントを使用し、コードブロックの構文をチェックしています。アプリケーションは、CREATE VIEW ステートメントを発行する前に、そのビューがアクセス可能であることをチェックします。

```
submit sql immediate;
   validate &viewdef;
end submit;

if symget('SQLRC') gt 4 then
   do;
      ... the view is not valid ...
   end;
else do;
   submit sql immediate;
      create view &viewname as &viewdef;
   end submit;
end;
```

次の例では、Countries テーブルからデータを取得しています。ただし、PROC SQL ステートメントで NOPRINT オプションを指定しているため、このテーブルは表示されません。%PUT マクロ言語ステートメントによって、3 つの自動マクロ変数の値を SAS ログに表示しています。%PUT ステートメントと SAS マクロ機能の詳細については、*SAS マクロ言語: リファレンス*を参照してください。

```
   libname sql 'SAS-library';

   proc sql noprint;
      select * from sql.countries;
   %put SQLOBS=*&sqlobs* SQLOOPS=*&sqloops* SQLRC=*&sqlrc*;
```

ログ 5.10 *PROC SQL 自動マクロ変数の使用*

```
SQLOBS=*1* SQLOOPS=*11* SQLRC=*0*
```

SQLOBS の値が 1 であることに注目してください。NOPRINT オプションを使用し、テーブルもマクロ変数も作成されなかった場合、1 つの行のみが処理されるため、SQLOBS によって 1 の値が返されます。

注: すべての自動マクロ変数の値を表示する場合、%PUT ステートメントで _AUTOMATIC_ オプションを使用できます。この表示は、サイトにインストールされている SAS 製品によって変わります。

REPORT プロシジャを使用し、PROC SQL 出力をフォーマットする

SQL は、制限された出力フォーマット機能を備えています。一部の SQL ベンダは、これらの制限に対処するために、自社製品に出力フォーマットステートメントを追加しています。SAS は、PROC SQL 出力の外観を改善するレポートツールを備えています。

たとえば、SQL では、出力に繰り返し出現する列の値の最初の出現のみを表示することはできません。次の例では、USCityCoords テーブル内の都市を表示しています。繰り返される State 列の値に注目してください。

```
libname sql 'SAS-library';

proc sql outobs=10;
   title 'US Cities';
   select State, City, latitude, Longitude
      from sql.uscitycoords
      order by state;
```

アウトプット 5.6 *繰り返される州の値を示す USCityCoords テーブル*

US Cities

State	City	Latitude	Longitude
AK	Sitka	57	-135
AK	Anchorage	61	-150
AK	Nome	64	-165
AK	Juneau	58	-134
AL	Mobile	31	-88
AL	Montgomery	32	-86
AL	Birmingham	33	-87
AR	Hot Springs	34	-93
AR	Little Rock	35	-92
AZ	Flagstaff	35	-112

次のコードでは、州のコードが州のグループごとに一度だけ表示されるように、PROC REPORT を使用して出力をフォーマットしています。環太平洋地域にある州の都市の

座標のみがレポートに表示されるように、WHERE 句によってデータをサブセット化しています。PROC REPORT の詳細については、*Base SAS プロシジャガイド*を参照してください。

```
libname sql 'SAS-library';

proc sql noprint;
   create table sql.cityreport as
   select *
      from sql.uscitycoords
      order by state;

proc report data=sql.cityreport
            headline nowd
            headskip;
   title 'Coordinates of U.S. Cities in Pacific Rim States';
   column state city ('Coordinates' latitude longitude);
   define state / order format=$2. width=5 'State';
   define city / order format=$15. width=15 'City';
   define latitude / display format=4. width=8 'Latitude';
   define longitude / display format=4. width=9 'Longitude';
   where state='AK' or
         state='HI' or
         state='WA' or
         state='OR' or
         state='CA';
run;
```

アウトプット 5.7 各州の値の最初の出現のみを表示する PROC REPORT 出力

Coordinates of U.S. Cities in Pacific Rim States

		Coordinates	
State	City	Latitude	Longitude
AK	Anchorage	61	-150
	Juneau	58	-134
	Nome	64	-165
	Sitka	57	-135
CA	El Centro	32	-115
	Fresno	37	-120
	Long Beach	34	-118
	Los Angeles	34	-118
	Oakland	38	-122
	Sacramento	38	-121
	San Diego	33	-117
	San Francisco	38	-122
	San Jose	37	-122
HI	Honolulu	21	-158
OR	Baker	45	-118
	Eugene	44	-124
	Klamath Falls	42	-122
	Portland	45	-123
	Salem	45	-123
WA	Olympia	47	-123
	Seattle	47	-122
	Spokane	48	-117

SAS/ACCESS を使用した DBMS へのアクセス

概要:SAS/ACCESS を使用した DBMS へのアクセス

リレーショナルデータベース用の SAS/ACCESS ソフトウェアでは、SAS ソフトウェアと他のベンダのデータベース管理システム(DBMS)のデータとの間のインターフェイスが提供されます。SAS/ACCESS ソフトウェアでは、SAS/ACCESS の LIBNAME ステート

メントと PROC SQL のパススルー機能によって、DBMS データへの動的アクセスが提供されます。LIBNAME ステートメントによって、SAS ライブラリ参照名を、スキーマやデータベースなどの DBMS オブジェクトに割り当てることができます。パススルー機能により、SAS セッションから離れることなく、SQL 構文を使用して DBMS を操作できます。

DBMS データにアクセスする場合、SAS/ACCESS の LIBNAME ステートメントを使用することをお勧めします。これは、DBMS データにアクセスする場合、通常はこの方法が最も高速で直接的なためです。LIBNAME ステートメントには、次のようなメリットがあります。

- DBMS の操作を実行するために必要な SAS コードの行数が、大幅に減ります。たとえば、1 つの LIBNAME ステートメントによって DBMS との接続を確立し、データの処理方法を指定し、SAS において DBMS テーブルを簡単に参照できます。
- DBMS データにアクセスし、操作するために、DBMS の SQL 言語を知っている必要はありません。PROC SQL などの SAS プロシジャまたは DATA ステッププログラムを、DBMS データを参照する任意のライブラリ参照名に対して使用できます。通常の SAS 構文を使用して、データの読み取り、挿入、更新、削除、追加に加え、DBMS テーブルの作成および削除も実行できます。
- LIBNAME ステートメントを使用すると、多くの LIBNAME のオプションとデータセットオプションによって、ロック処理、スプール処理、データタイプ変換など DBMS 操作をさらに制御できます。
- LIBNAME エンジンは、インデックスなどの DBMS の処理機能を活用するために、結合および WHERE 句の演算を DBMS に直接渡します。

これによって、結合および WHERE 句の処理は最適化されます。ただし、ANSI 規格に準拠しない SQL を使用する必要がある場合、これは推奨されません。
SAS/ACCESS の LIBNAME ステートメントは、ANSI 規格の SQL のみを受け付けますが、PROC SQL のパススルー機能は、DBMS が提供する SQL に対するすべての拡張を受け付けます。このアクセス方法の別のメリットは、クエリに要約関数(AVG や COUNT など)、GROUP BY 句、または式(COMPUTED 関数など)によって作成された列が含まれる場合、パススルー機能のステートメントによって、DBMS でのクエリの最適化が可能になる点です。

SAS/ACCESS ソフトウェアの詳細については、*SAS/ACCESS for Relational Databases: Reference* を参照してください。

LIBNAME ステートメントを使用した DBMS への接続

概要:LIBNAME ステートメントを使用した DBMS への接続

LIBNAME ステートメントを使用して、SAS データセットと同じように DBMS オブジェクトから読み取り、DBMS オブジェクトに書き込むことができます。LIBNAME ステートメントを使用して DBMS のテーブルまたはビューに接続すると、PROC SQL を使用して DBMS データを操作できます。

多くの DBMS では、SAS/ACCESS の LIBNAME ステートメントを使用して DBMS にライブラリ参照名を割り当てることによって、DBMS データに直接アクセスできます。DBMS にライブラリ参照名を割り当てておくと、2 レベルの SAS 名で DBMS テーブルを指定し、SAS データセットと同様にそのテーブルを操作できます。LIBNAME ステートメントを PROC SQL ビューに埋め込むこともできます。詳細については、“CREATE VIEW ステートメント” (242 ページ)を参照してください。

PROC SQL は、可能な場合は特定の演算を DBMS に渡すことによって、DBMS の機能を活用します。たとえば、PROC SQL は、結合を実行する前に、DBMS が結合を実行できるかどうかをチェックして決定します。結合が可能な場合、PROC SQL は、結合を DBMS に渡します。こうすることで、データの移動と変換が削減されるため、パフォ

ーマンスが向上します。DBMS が結合を実行できない場合、PROC SQL が結合を処理します。SAS/ACCESS の LIBNAME ステートメントを使用すると、多くの場合、DBMS 固有のコードを記述しなくても、SQL プロシジャのパススルー機能によるパフォーマンス上のメリットが得られます。

注: DBIDIRECTEXEC システムオプションを使用して、PROC SQL の CREATE TABLE AS SELECT ステートメント、CREATE VIEW ステートメント、DELETE ステートメント、INSERT ステートメント、UPDATE ステートメントを、実行のために直接データベースに送信できます。これによって、CPU と入出力のパフォーマンスを改善できます。詳細は、使用している DBMS 用の SAS/ACCESS のマニュアルを参照してください。

SAS/ACCESS の LIBNAME ステートメントを使用するには、使用している DBMS 用の SAS/ACCESS ソフトウェアがインストールされている必要があります。SAS/ACCESS の LIBNAME ステートメントの詳細については、使用している DBMS について SAS/ACCESS のマニュアルを参照してください。

DBMS テーブルのクエリ

この例では、PROC SQL を使用して、Oracle の PAYROLL テーブルを照会します。この PROC SQL では、すべてのジョブコードを取得し、ジョブコードごとに給与の総額を表示しています。

注: デフォルトでは、Oracle は出力結果を並べ替えません。出力結果の行の表示順序を指定するには、SELECT ステートメントで ORDER BY 句を使用する必要があります。

```
libname mydblib oracle user=user-id password=password
        path=path-name schema=schema-name;

proc sql;
   select jobcode label='Jobcode',
          sum(salary) as total
          label='Total for Group'
          format=dollar11.2
      from mydblib.payroll
      group by jobcode;
quit;
```

アウトプット 5.8 *DBMS テーブルの照会の出力*

The SAS System

Jobcode	Total for Group
PT1	$543,264.00
BCK	$232,148.00
ME3	$296,875.00
NA2	$157,149.00
ME2	$498,076.00
TA3	$476,155.00
TA1	$249,492.00
PT3	$221,009.00
FA1	$253,433.00
SCP	$128,162.00
PT2	$879,252.00
FA2	$447,790.00
TA2	$671,499.00
ME1	$228,002.00
FA3	$230,537.00
NA1	$210,161.00

DBMS テーブルの PROC SQL ビューの作成

PROC SQL ビューは、このビューの元になるファイルからデータ値を読み取る、格納されたクエリ式です。これには、DBMS データの SAS/ACCESS ビューを含めることができます。DBMS データの DATA ステップビューは、データの読み取りのみに使用できますが、DBMS データの PROC SQL ビューは、次の条件が満たされる場合、元になるデータの更新に使用できます。

- PROC SQL ビューが、ただ 1 つの DBMS テーブル(または、だた 1 つの DBMS テーブルに基づく DBMS ビュー)に基づいていること。
- PROC SQL ビューに、計算されたフィールドが含まれていないこと。

次の例では、LIBNAME ステートメントを使用して Oracle データベースに接続し、Oracle の SCHEDULE テーブルの一時的な PROC SQL ビューを作成し、PRINT プロシジャを使用してそのビューを出力しています。LIBNAME エンジンは、DBMS のインデックス機能と処理機能を活用するために、結合および WHERE 句の演算を DBMS に直接渡します。これによって、結合および WHERE 句の処理は最適化されます。

```
libname mydblib oracle user=user-id password=password
proc sql;
   create view LON as
   select flight, dates, idnum
```

```
        from mydblib.schedule
        where dest='LON';
quit;

proc print data=work.LON noobs;
run;
```

アウトプット 5.9 *PRINT プロシジャの出力*

The SAS System

FLIGHT	DATES	IDNUM
219	01MAR1998:00:00:00	1407
219	01MAR1998:00:00:00	1777
219	01MAR1998:00:00:00	1103
219	01MAR1998:00:00:00	1125
219	01MAR1998:00:00:00	1350
219	01MAR1998:00:00:00	1332
219	02MAR1998:00:00:00	1407
219	02MAR1998:00:00:00	1118
219	02MAR1998:00:00:00	1132
219	02MAR1998:00:00:00	1135
219	02MAR1998:00:00:00	1441
219	02MAR1998:00:00:00	1332
219	03MAR1998:00:00:00	1428
219	03MAR1998:00:00:00	1442
219	03MAR1998:00:00:00	1130
219	03MAR1998:00:00:00	1411
219	03MAR1998:00:00:00	1115
219	03MAR1998:00:00:00	1332

パススルー機能を使用した DBMS への接続

パススルー機能について

SQL プロシジャのパススルー機能によって、DBMS 固有の SQL ステートメントを、実行のために直接 DBMS に送信できます。パススルー機能では、SAS/ACCESS のインターフェイスエンジンを使用して DBMS に接続します。そのため、使用している DBMS 用の SAS/ACCESS ソフトウェアがインストールされている必要があります。

DBMS 固有の SQL ステートメントをサブミットします。たとえば、Sybase データベースには、Transact-SQL ステートメントを渡します。パススルー機能の基本構文は、すべて

の DBMS で同じです。ただし、DBMS への接続に使用されるステートメントと SQL ステートメントのみは、DBMS に固有です。

パススルー機能を使用して、次のタスクを実行できます。

- CONNECT ステートメントを使用して DBMS との接続を確立し、DISCONNECT ステートメントを使用して接続を終了します。
- クエリ以外の DBMS 固有の SQL ステートメントを、EXECUTE ステートメントを使用して DBMS に送信します。
- PROC SQL クエリで使用するためのデータを、SELECT ステートメントの FROM 句の CONNECTION TO 構成要素を使用して、DBMS から取得します。

パススルー機能のステートメントを、クエリで使用できます。あるいは、それらを PROC SQL ビューに格納できます。ビューを格納すると、それに対応する CONNECT ステートメントに指定したすべてのオプションも格納されます。そのため、SAS プログラムで PROC SQL ビューを使用すると、SAS は、DBMS への適切な接続を自動的に確立します。

詳細は、付録 1, “SQL マクロ変数とシステムオプション” (369 ページ)の CONNECT ステートメント、DISCONNECT ステートメント、EXECUTE ステートメント、CONNECTION TO ステートメント、および *SAS/ACCESS for Relational Databases: Reference* のリレーショナルデータベース用のパススルー機能を参照してください。

注: 複数の処理を実行する SAS プロシジャは、パススルー機能のステートメントが格納された PROC SQL ビューを操作できません。これは、パススルー機能では、最初のレコードを取得した後で、テーブルを再び開くことができないためです。この制限を回避するには、そのビューから SAS データセットを作成し、この SAS データセットを入力データセットとして使用します。

リターンコード

パススルー機能で使用可能な PROC SQL ステートメントを使用したときに、エラーが SAS ログに書き込まれます。パススルー機能によって生成されたリターンコードとメッセージは、SQLXRC マクロ変数と SQLXMSG マクロ変数を使用して参照できます。これらのマクロ変数は、“PROC SQL 自動マクロ変数の使用” (163 ページ)で説明されています。

パススルーの例

この例では、SAS/ACCESS によって、エイリアス **ora2** を使用して Oracle データベースに接続します。次に、PROC SQL を使用して Staff テーブルのすべての行を選択し、データの最初の 15 行を表示します。

```
proc sql outobs=15;
   connect to oracle as ora2 (user=user-id password=password);
   select * from connection to ora2 (select lname, fname, state from staff);
   disconnect from ora2;
quit;
```

アウトプット 5.10 パススルー機能の出力例

The SAS System

LNAME	FNAME	STATE
ADAMS	GERALD	CT
ALIBRANDI	MARIA	CT
ALHERTANI	ABDULLAH	NY
ALVAREZ	MERCEDES	NY
ALVAREZ	CARLOS	NJ
BAREFOOT	JOSEPH	NJ
BAUCOM	WALTER	NY
BANADYGA	JUSTIN	CT
BLALOCK	RALPH	NY
BALLETTI	MARIE	NY
BOWDEN	EARL	CT
BRANCACCIO	JOSEPH	NY
BREUHAUS	JEREMY	NY
BRADY	CHRISTINE	CT
BREWCZAK	JAKOB	CT

PROC SQL ビューと SAS/ACCESS ビューの更新

次の条件において、INSERT、DELETE および UPDATE ステートメントを使用して、PROC SQL ビューと SAS/ACCESS ビューを更新できます。

- ビューによって DBMS テーブルにアクセスする場合、外部のデータベース管理システム(たとえば、DB2)から、適切な権限が付与されている必要があります。また、その DBMS に対応した SAS/ACCESS ソフトウェアをインストールしている必要があります。SAS/ACCESS ビューの詳細については、使用している DBMS 用の SAS/ACCESS インターフェイスガイドを参照してください。
- ビューを介して 1 つのテーブルのみを更新できます。セット演算子を使用して、テーブルを別のテーブルと結合またはリンクすることはできません。ビューにサブクエリを含めることはできません。
- 列のエイリアスを使用してビューの列を更新できますが、派生した列、つまり式によって生成された列は更新できません。次の例の場合、SS 列は更新できますが、WeeklySalary 列は更新できません。

```
create view EmployeeSalaries as
   select Employee, SSNumber as SS,
          Salary/52 as WeeklySalary
          from employees;
```

- ORDER BY を含むビューは、更新できません。

注: SAS 9 以降、リレーショナル DBMS のデータにアクセスする方法として、PROC SQL ビュー、パススルー機能および SAS/ACCESS の LIBNAME ステートメントが推奨されています。SAS/ACCESS ビューは、推奨されていません。CV2VIEW プロシジャを使用して、既存の SAS/ACCESS ビューを PROC SQL ビューに変換できます。詳細については、"CV2VIEW" (*SAS/ACCESS for Relational Databases: Reference*)を参照してください。

PROC SQL で ODS (Output Delivery System)を使用する

Output Delivery System(ODS)を使用して、PostScript、HTML、リスト出力など、さまざまな出力形式で PROC SQL の出力を生成できます。ODS では、SAS プロシジャおよび SAS DATA ステップの生の出力に対して構造が定義されます。データとその出力構造定義の組み合わせは、*出力オブジェクト*と呼ばれます。出力オブジェクトは、表示、HTML、出力、プリンタなど、ODS のさまざまな出力先のいずれにも送信できます。新しい出力先を ODS に追加すると、それらは自動的に、PROC SQL、ODS をサポートする他のすべての SAS プロシジャ、および DATA ステップで使用できるようになります。ODS の詳細については、*SAS Output Delivery System: ユーザーガイド*を参照してください。

次の例では、HTML の出力先を開き、HTML 出力が含まれるファイルとして ODSOUT.HTM を指定しています。PROC SQL の出力は、ODSOUT.HTM に送信されます。

注: この例では、すべてのオペレーティングシステムで有効とは限らないファイル名が使用されます。使用しているオペレーティングシステムでこの例を正常に実行するために、ファイル仕様の変更が必要となる場合があります。

注: ブラウザによっては、ファイル名に HTM または HTML の拡張子が必要な場合があります。

```
libname sql 'SAS-library';

ods html body='odsout.htm';
   proc sql outobs=12;
      title 'Coordinates of U.S. Cities';
      select *
         from sql.uscitycoords;
```

アウトプット 5.11 アメリカの 都市の座標

Coordinates of U.S. Cities

City	State	Latitude	Longitude
Albany	NY	43	-74
Albuquerque	NM	36	-106
Amarillo	TX	35	-102
Anchorage	AK	61	-150
Annapolis	MD	39	-77
Atlanta	GA	34	-84
Augusta	ME	44	-70
Austin	TX	30	-98
Baker	OR	45	-118
Baltimore	MD	39	-76

Measurements are in metric tons.

6 章

PROC SQL を使用した問題の解決

概要 **178**

重み付き平均の計算 **178**
問題 178
背景情報 178
解法 179
動作 180

テーブルの比較 **180**
問題 180
背景情報 181
解法 181
動作 182

欠損データ値の重ね合わせ **182**
問題 182
背景情報 182
解法 183
動作 184

小計内の百分率の計算 **185**
問題 185
背景情報 185
解法 186
動作 187

テーブルの重複する行のカウント **187**
問題 187
背景情報 187
解法 188
動作 189

テーブルの階層データの展開 **189**
問題 189
背景情報 189
解法 190
動作 191

複数列のデータの要約 **192**
問題 192
背景情報 192
解法 193
動作 193

要約レポートの作成 **194**
問題 194
背景情報 194
解法 195
動作 196

カスタマイズされた並べ替え順序の作成 **197**
問題 197
背景情報 197
解法 198
動作 199

テーブルの条件付き更新 **200**
問題 200
背景情報 200
解法 201
動作 202

別のテーブルの値を使用してテーブルを更新する **203**
問題 203
背景情報 203
解法 204
動作 205

マクロ変数の作成および使用 **205**
問題 205
背景情報 206
解法 206
動作 207

他の SAS プロシジャでの PROC SQL テーブルの使用 **208**
問題 208
背景情報 208
解法 209
動作 210

概要

このセクションでは、PROC SQL が提供できる解法の例を示します。それぞれの例には、解決すべき問題の記述、問題を解決するために知っておく必要のある背景情報、PROC SQL 解法コード、解法の動作説明が含まれています。

重み付き平均の計算

問題

列の値の重み付き平均を計算します。

背景情報

Sample という 1 つの入力テーブルがあり、それには次のデータが含まれています。

```
data Sample;
   do i=1 to 10;
      Value=2983*ranuni(135);
      Weight=33*rannor(579);
      if mod(i,2)=0 then Gender='M';
         else Gender='F';
      output;
   end;
   drop i;

proc print data=Sample;
  title 'Sample Data for Weighted Average';
run;
```

アウトプット 6.1 *重み付き平均の入力テーブルの例*

Sample Data for Weighted Average

Obs	Value	Weight	Gender
1	2893.35	9.0868	F
2	56.13	26.2171	M
3	901.43	-4.0605	F
4	2942.68	-5.6557	M
5	621.16	24.3306	F
6	361.50	13.8971	M
7	2575.09	29.3734	F
8	2157.07	7.0687	M
9	690.73	-40.1271	F
10	2085.80	24.4795	M

重みの一部が負であることに注意してください。

解法

次の PROC SQL コードを使用して重み付き平均を取得します。その出力を下に示します。

```
proc sql;
   title 'Weighted Averages from Sample Data';
   select Gender, sum(Value*Weight)/sum(Weight) as WeightedAverage
      from (select Gender, Value,
                   case
                      when Weight gt 0 then Weight
                      else 0
                   end as Weight
              from Sample)
      group by Gender;
```

アウトプット 6.2 *重み付き平均の PROC SQL 出力*

Weighted Averages from Sample Data

Gender	WeightedAverage
F	1864.026
M	1015.91

動作

この解法では、インラインビューを使用して、Weight 列から負のデータ値を除去した一時テーブルを作成します。インラインビューは、次のタスクを実行するクエリです。

- Gender 列と Value 列を選択します。
- CASE 式を使用して Weight 列から値を選択します。Weight 値がゼロよりも大きい場合、それを取得します。Weight 値がゼロよりも小さい場合、その値のかわりにゼロが使用されます。

```
(select Gender, Value,
        case
           when Weight>0 then Weight
           else 0
        end as Weight
    from Sample)
```

クエリ内の最初の(つまり外側の)SELECT ステートメントは、次のタスクを実行します。

- Gender 列を選択します。
- インラインビューによって取得した結果から重み付き平均を計算します。

重み付き平均は、Value と Weight の積の合計を Weight の合計で割った値です。

```
select Gender, sum(Value*Weight)/sum(Weight) as WeightedAverage
```

最後にこのクエリは、GROUP BY 句を使用して、性別ごとに計算を実行するようにデータを結合します。

```
group by Gender;
```

テーブルの比較

問題

1 つのテーブルの 2 つのコピーがあります。コピーの 1 つは更新されています。変更された行を確認します。

背景情報

OldStaff テーブル、NewStaff テーブルという 2 つのテーブルがあります。NewStaff テーブルは、OldStaff テーブルのコピーです。NewStaff テーブルに対して変更が行われています。行われた変更を調べます。

アウトプット 6.3 *テーブル比較のための入力テーブルの例*

Old Staff Table

id	Last	First	Middle	Phone	Location
5463	Olsen	Mary	K.	661-0012	R2342
6574	Hogan	Terence	H.	661-3243	R4456
7896	Bridges	Georgina	W.	661-8897	S2988
4352	Anson	Sanford		661-4432	S3412
5674	Leach	Archie	G.	661-4328	S3533
7902	Wilson	Fran	R.	661-8332	R4454
0001	Singleton	Adam	O.	661-0980	R4457
9786	Thompson	Jack		661-6781	R2343

New Staff Table

id	Last	First	Middle	Phone	Location
5463	Olsen	Mary	K.	661-0012	R2342
6574	Hogan	Terence	H.	661-3243	R4456
7896	Bridges	Georgina	W.	661-2231	S2987
4352	Anson	Sanford		661-4432	S3412
5674	Leach	Archie	G.	661-4328	S3533
7902	Wilson	Fran	R.	661-8332	R4454
0001	Singleton	Adam	O.	661-0980	R4457
9786	Thompson	John	C.	661-6781	R2343
2123	Chen	Bill	W.	661-8099	R4432

解法

新しいバージョンのテーブル内の変更された行のみを表示するには、2 つの SELECT ステートメントの間で EXCEPT セット演算子を使用します。

```
proc sql;
   title 'Updated Rows';
   select * from newstaff
   except
   select * from oldstaff;
```

アウトプット 6.4 *変更された行*

Updated Rows

id	Last	First	Middle	Phone	Location
2123	Chen	Bill	W.	661-8099	R4432
7896	Bridges	Georgina	W.	661-2231	S2987
9786	Thompson	John	C.	661-6781	R2343

動作

EXCEPT 演算子は、最初のクエリの結果から 2 番目のクエリの結果を除いた行を返します。この例では、EXCEPT 演算子は、NewStaff テーブル内の追加または変更されている行のみを表示します。

注: OldStaff テーブルから削除された行は表示されません。

欠損データ値の重ね合わせ

問題

ボウラーが他のボーリングリーグのメンバーだったときのアベレージを分析して、新しいリーグのチームを結成しようとしています。可能であれば、それぞれのボウラーのリーグでの最新のアベレージを使用します。ただし、ボウラーが昨年はリーグに所属していなかった場合、一昨年のアベレージを使用します。

背景情報

League1 と League2 という、ボーリングのアベレージを含む 2 つのテーブルがあります。これらのテーブルのテーブル構造は、異なる 2 つのセクレタリでデータがコンパイルされているため、同じではありません。ただし、これらのテーブルには本質的に同じタイプのデータが含まれています。

```
data league1;
input @1 Fullname $20. @21 Bowler $4. @29 AvgScore 3.;
cards;
Alexander Delarge   4224    164
John T Chance       4425
Jack T Colton       4264
                    1412    141
Andrew Shepherd     4189    185
;
```

```
data league2;
input @1 FirstName $10. @12 LastName $15. @28 AMFNo $4. @38 AvgScore 3.;
cards;
Alex       Delarge         4224      156
Mickey     Raymond         1412
                           4264      174
Jack       Chance          4425
Patrick    O'Malley        4118      164
;

proc sql;
title 'Bowling Averages from League1';
select * from league1;
title 'Bowling Averages from League2';
select * from league2;
```

アウトプット 6.5 *欠損値を重ね合わせるための入力テーブルの例*

Bowling Averages from League1

Fullname	Bowler	AvgScore
Alexander Delarge	4224	164
John T Chance	4425	.
Jack T Colton	4264	.
	1412	141
Andrew Shepherd	4189	185

Bowling Averages from League2

FirstName	LastName	AMFNo	AvgScore
Alex	Delarge	4224	156
Mickey	Raymond	1412	.
		4264	174
Jack	Chance	4425	.
Patrick	O'Malley	4118	164

解法

次の PROC SQL コードは、2 つのテーブル(League1 と League2)の情報を結合します。プログラムは、可能であれば League1 テーブルのすべての値を使用し、欠損値を League2 テーブルの対応する値で置き換えます。出力結果を次に示します。

```
proc sql;
   title "Averages from Last Year's League When Possible";
   title2 "Supplemented when Available from Prior Year's League";
```

```
select coalesce(lastyr.fullname,trim(prioryr.firstname)
                ||' '||prioryr.lastname)as Name format=$26.,
       coalesce(lastyr.bowler,prioryr.amfno)as Bowler,
       coalesce(lastyr.avgscore,prioryr.avgscore)as Average format=8.
   from league1 as lastyr full join league2 as prioryr
        on lastyr.bowler=prioryr.amfno
   order by Bowler;
```

アウトプット 6.6 *欠損値の重ね合わせの PROC SQL 出力*

Averages from Last Year's League When Possible Supplemented when Available from Prior Year's League

Name	Bowler	Average
Mickey Raymond	1412	141
Patrick O'Malley	4118	164
Andrew Shepherd	4189	185
Alexander Delarge	4224	164
Jack T Colton	4264	174
John T Chance	4425	.

動作

この解法では、FULL JOIN を使用して League1 および League2 のすべての行を取得します。プログラムは、可能な場合、行の各列に値が存在するようにするために、各列に対して COALESCE 関数を使用します。かっこで囲まれた式のリストに対して COALESCE 関数を使用すると、検出された最初の非欠損値が返されます。次のコードは、League1 の行ごとに、Average として AvgScore 列を返します。

```
coalesce(lastyr.avgscore,prioryr.avgscore) as Average format=8.
```

この AvgScore の値が欠損している場合、COALESCE は、Average として League2 の AvgScore 列を返します。この AvgScore 列の値も欠損している場合、COALESCE は Average として欠損値を返します。

Name 列については、COALESCE 関数は、League1 の FullName の値が存在する場合、その値を返します。その値が存在しない場合、次のように TRIM 関数と連結演算子を使用して、League2 の FirstName 列と LastName 列を結合することによって値を取得します。

```
trim(prioryr.firstname)||' '||prioryr.lastname
```

最後にテーブルは、Bowler の値で並べ替えられます。Bowler 列は、次の COALESCE 関数の結果です。

```
coalesce(lastyr.bowler,prioryr.amfno)as Bowler
```

値がいずれかのテーブルから取得されるため、League1 の Bowler の値または League2 の AMFNo の値のいずれかで、確実に出力を並び替えることはできません。COALESCE 関数から得られる値によってのみ並べ替えることができます。

小計内の百分率の計算

問題

質問調査に対する回答を分析し、それぞれの州がどのように回答したかを判断します。次に、特定の州が寄与しているそれぞれの回答の百分率を計算します。たとえば、ノースカロライナ州からのすべてのいいえの回答の百分率などです。

背景情報

Survey という 1 つの入力テーブルがあり、これには次のデータ(最初の 10 行を示す)が含まれています。

```
data survey;
    input State $ Answer $ @@;
    datalines;
NY YES NY YES NY YES NY YES NY YES NY YES NY NO  NY NO  NY NO  NC YES
NC YES NC YES NC YES NC YES NC YES NC YES NC YES NC YES NC YES NC YES
NC YES NC YES NC YES NC YES NC YES NC YES NC YES NC YES NC YES NC NO
NC NO  NC NO  NC NO  NC NO  NC NO  NC NO  NC NO  NC NO  NC NO  NC NO
NC NO  NC NO  NC NO  NC NO  NC NO  NC NO  NC NO  NC NO  NC NO  NC NO
NC NO  NC NO  NC NO  PA YES PA YES PA YES PA YES PA YES PA YES PA YES
PA YES PA YES PA NO  PA NO  PA NO  PA NO  PA NO  PA NO  PA NO  PA NO
PA NO  PA NO  PA NO  PA NO  PA NO  PA NO  PA NO  PA NO  PA NO  PA NO
VA YES VA YES VA YES VA YES VA YES VA YES VA YES VA YES VA YES VA YES
VA YES VA YES VA YES VA YES VA YES VA YES VA YES VA YES VA YES VA NO
VA NO  VA NO  VA NO  VA NO  VA NO  VA NO  VA NO  VA NO  VA NO  VA NO
VA NO  VA NO  VA NO  VA NO  VA NO  VA NO
;


proc print data=Survey(obs=10);
  title 'Sample Data for Subtotal Percentages';
run;
```

アウトプット 6.7 小計の百分率を計算するための入力テーブル(部分的な出力)

Sample Data for Subtotal Percentages

Obs	State	Answer
1	NY	YES
2	NY	YES
3	NY	YES
4	NY	YES
5	NY	YES
6	NY	YES
7	NY	NO
8	NY	NO
9	NY	NO
10	NC	YES

解法

次の PROC SQL コードを使用して小計の百分率を計算します。

```
proc sql;
   title1 'Survey Responses';
   select survey.Answer, State, count(State) as Count,
          calculated Count/Subtotal as Percent format=percent8.2
   from survey,
        (select Answer, count(*) as Subtotal from survey
           group by Answer) as survey2
   where survey.Answer=survey2.Answer
   group by survey.Answer, State;
quit;
```

アウトプット 6.8 小計内の百分率を計算した PROC SQL 出力

Survey Responses

Answer	State	Count	Percent
NO	NC	24	38.71%
NO	NY	3	4.84%
NO	PA	18	29.03%
NO	VA	17	27.42%
YES	NC	20	37.04%
YES	NY	6	11.11%
YES	PA	9	16.67%
YES	VA	19	35.19%

動作

この解法では、サブクエリを使用して、回答ごとに小計を計算します。このコードは、サブクエリの結果と元のテーブルを結合し、次に計算された州の数を分子として使用し、サブクエリの小計を分母として使用して百分率を計算します。

このクエリは、それぞれの回答内の State に対して計算が実行されるように、GROUP BY 句を使用してデータを結合します。

```
group by survey.Answer, State;
```

テーブルの重複する行のカウント

問題

テーブル内の重複行の数をカウントし、それぞれの重複行の出現回数を示す出力列を生成します。

背景情報

Duplicates という 1 つの入力テーブルがあり、それには次のデータが含まれています。

```
data Duplicates;
   input LastName $ FirstName $ City $ State $;
   datalines;
Smith John Richmond Virginia
Johnson Mary Miami Florida
Smith John Richmond Virginia
Reed Sam Portland Oregon
Davis Karen Chicago Illinois
Davis Karen Chicago Illinois
```

```
Thompson Jennifer Houston Texas
Smith John Richmond Virginia
Johnson Mary Miami Florida
;

proc print data=Duplicates;
  title 'Sample Data for Counting Duplicates';
run;
```

アウトプット 6.9 *重複をカウントするための入力テーブルの例*

Sample Data for Counting Duplicates

Obs	LastName	FirstName	City	State
1	Smith	John	Richmond	Virginia
2	Johnson	Mary	Miami	Florida
3	Smith	John	Richmond	Virginia
4	Reed	Sam	Portland	Oregon
5	Davis	Karen	Chicago	Illinois
6	Davis	Karen	Chicago	Illinois
7	Thompson	Jennifer	Houston	Texas
8	Smith	John	Richmond	Virginia
9	Johnson	Mary	Miami	Florida

解法

次の PROC SQL コードを使用して重複行をカウントします。

```
proc sql;
   title 'Duplicate Rows in Duplicates Table';
   select *, count(*) as Count
      from Duplicates
      group by LastName, FirstName, City, State
      having count(*) > 1;
```

アウトプット 6.10 *重複のカウントの PROC SQL 出力*

Duplicate Rows in DUPLICATES Table

LastName	FirstName	City	State	Count
Davis	Karen	Chicago	Illinois	2
Johnson	Mary	Miami	Florida	2
Smith	John	Richmond	Virginia	3

動作

この解法では、次を実行するクエリを使用します。

- すべての列を選択します。
- すべての行をカウントします。
- 行を照合することによって、Duplicates テーブル内のすべての行をグループ化します。
- 重複していない行を除外します。

注: 正確な重複を検索するには、テーブル内のすべての列を GROUP BY 句に含める必要があります。

テーブルの階層データの展開

問題

テーブル内の行の間の階層関係を示す出力列を生成します。

背景情報

Employees という 1 つの入力テーブルがあり、それには次のデータが含まれています。

```
data Employees;
   input ID $ LastName $ FirstName $ Supervisor $;
   datalines;
1001 Smith John 1002
1002 Johnson Mary None
1003 Reed Sam None
1004 Davis Karen 1003
1005 Thompson Jennifer 1002
1006 Peterson George 1002
1007 Jones Sue 1003
1008 Murphy Janice 1003
1009 Garcia Joe 1002
;


proc print data=Employees;
  title 'Sample Data for Expanding a Hierarchy';
run;
```

アウトプット 6.11 *階層を展開するための入力テーブルの例*

Sample Data for Expanding a Hierarchy

Obs	ID	LastName	FirstName	Supervisor
1	1001	Smith	John	1002
2	1002	Johnson	Mary	None
3	1003	Reed	Sam	None
4	1004	Davis	Karen	1003
5	1005	Thompson	Jennifer	1002
6	1006	Peterson	George	1002
7	1007	Jones	Sue	1003
8	1008	Murphy	Janice	1003
9	1009	Garcia	Joe	1002

管理者が存在する各従業員の氏名と ID 番号と共に、その従業員の管理者の氏名と ID 番号を示す出力を作成します。

解法

次の PROC SQL コードを使用してデータを展開します。

```
proc sql;
   title 'Expanded Employee and Supervisor Data';
   select A.ID label="Employee ID",
          trim(A.FirstName)||' '||A.LastName label="Employee Name",
          B.ID label="Supervisor ID",
          trim(B.FirstName)||' '||B.LastName label="Supervisor Name"
      from Employees A, Employees B
      where A.Supervisor=B.ID and A.Supervisor is not missing;
```

アウトプット 6.12 階層の展開の PROC SQL 出力

Expanded Employee and Supervisor Data

Employee ID	Employee Name	Supervisor ID	Supervisor Name
1001	John Smith	1002	Mary Johnson
1009	Joe Garcia	1002	Mary Johnson
1006	George Peterson	1002	Mary Johnson
1005	Jennifer Thompson	1002	Mary Johnson
1004	Karen Davis	1003	Sam Reed
1008	Janice Murphy	1003	Sam Reed
1007	Sue Jones	1003	Sam Reed

動作

この解法では、自己結合(再帰結合)を使用して従業員とその管理者を突き合わせます。SELECT 句では、同じテーブルの 2 つのインスタンスに A と B というエイリアスが割り当てられ、それぞれのインスタンスのデータが取得されます。SELECT 句は、インスタンス A に対して次を実行します。

- ID 列を選択し、それに **Employee ID のラベルを割り当てます。**
- FirstName 列と LastName 列を選択して、それらを 1 つの出力列に連結し、それに **Employee Name というラベルを割り当てます。**

SELECT 句は、インスタンス B に対して次を実行します。

- ID 列を選択し、それに **Supervisor ID のラベルを割り当てます。**
- FirstName 列と LastName 列を選択して、それらを 1 つの出力列に連結し、それに **Supervisor Name のラベルを割り当てます。**

SELECT 句は、両方の連結で、TRIM 関数を使用して FirstName 列のデータの後に続くスペースを削除します。次に、そのデータを LastName 列のデータと間にスペースを 1 つ開けて連結し、氏名ごとに 1 つの文字値を生成します。

```
trim(A.FirstName)||' '||A.LastName label="Employee Name"
```

PROC SQL で WHERE 句が適用されると、2 つのテーブルインスタンスは結合されます。WHERE 句の条件は、テーブル B の従業員 ID に一致する管理者 ID を持つテーブル A 内の行のみを出力するように制限します。このオペレーションによって、管理者が存在しない従業員を除く元のテーブルの各従業員の管理者 ID と氏名が得られます。

```
 where A.Supervisor=B.ID and A.Supervisor is not missing;
```

注: Employees テーブルには欠損値がありませんが、予期しない結果が得られるのを避けるため、欠損値を調べて除外する必要があります。たとえば、空白の管理者 ID を持つ従業員と空白の ID を持つ従業員が存在する場合、それらによって誤った一致結果が生成されます。

複数列のデータの要約

問題

テーブル内の複数列の総計を生成します。

背景情報

Sales という 1 つの入力テーブルがあり、それには次のデータが含まれています。

```
data Sales;
   input Salesperson $ January February March;
   datalines;
Smith 1000 650 800
Johnson 0 900 900
Reed 1200 700 850
Davis 1050 900 1000
Thompson 750 850 1000
Peterson 900 600 500
Jones 800 900 1200
Murphy 700 800 700
Garcia 400 1200 1150
;

proc print data=Sales;
  title 'Sample Data for Summarizing Data from Multiple Columns';
run;
```

アウトプット 6.13 複数列のデータを要約するための入力テーブルの例

Sample Data for Summarizing Data from Multiple Columns

Obs	Salesperson	January	February	March
1	Smith	1000	650	800
2	Johnson	0	900	900
3	Reed	1200	700	850
4	Davis	1050	900	1000
5	Thompson	750	850	1000
6	Peterson	900	600	500
7	Jones	800	900	1200
8	Murphy	700	800	700
9	Garcia	400	1200	1150

月ごとの総売上および 3 か月間の総売上を示す出力を作成します。

解法

次の PROC SQL コードを使用して、月ごとの合計および総計を生成します。

```
proc sql;
   title 'Total First Quarter Sales';
   select sum(January)  as JanTotal,
          sum(February) as FebTotal,
          sum(March)    as MarTotal,
          sum(calculated JanTotal, calculated FebTotal,
              calculated MarTotal) as GrandTotal format=dollar10.
      from Sales;
```

アウトプット 6.14 複数列のデータを要約した PROC SQL 出力

Total First Quarter Sales

JanTotal	FebTotal	MarTotal	GrandTotal
6800	7500	8100	$22,400

動作

集合関数に対して引数として 1 つの列を指定すると、その列内の値が計算されることを思い出してください。複数列を指定した場合、それらの列の各行の値が計算されます。この解法では、SUM 関数を使用して月ごとの売上合計を計算します。次に、再び SUM 関数を使用して月ごとの合計を集計し、1 つの総計を求めます。

```
sum(calculated JanTotal, calculated FebTotal,
    calculated MarTotal) as GrandTotal format=dollar10.
```

総計計算を記述する別の方法として、次のようなネストされた関数を使用できます。

```
sum(sum(January), sum(February), sum(March))
   as GrandTotal format=dollar10.
```

要約レポートの作成

問題

詳細な売上情報を含むテーブルがあります。その詳細なテーブルから要約レポートを生成します。

背景情報

Sales という 1 つの入力テーブルがあり、それには詳細な売上情報が含まれています。第 1 四半期の売り上げごとに、サイト、製品、送り状番号、送り状金額、送り状日付を示す 1 つのレコードが存在します。

```
data sales;
  input Site $ Product $ Invoice $ InvoiceAmount InvoiceDate $;
  datalines;
V1009  VID010  V7679 598.5  980126
V1019  VID010  V7688 598.5  980126
V1032  VID005  V7771  1070  980309
V1043  VID014  V7780  1070  980309
V421   VID003  V7831  2000  980330
V421   VID010  V7832   750  980330
V570   VID003  V7762  2000  980302
V659   VID003  V7730  1000  980223
V783   VID003  V7815   750  980323
V985   VID003  V7733  2500  980223
V966   VID001  V5020  1167  980215
V98    VID003  V7750  2000  980223
;

proc sql;
  title 'Sample Data to Create Summary Sales Report';
  select * from sales;
quit;
```

アウトプット 6.15 売上要約レポートを作成するための入力テーブルの例

Sample Data to Create Summary Sales Report

Site	Product	Invoice	InvoiceAmount	InvoiceDate
V1009	VID010	V7679	598.5	980126
V1019	VID010	V7688	598.5	980126
V1032	VID005	V7771	1070	980309
V1043	VID014	V7780	1070	980309
V421	VID003	V7831	2000	980330
V421	VID010	V7832	750	980330
V570	VID003	V7762	2000	980302
V659	VID003	V7730	1000	980223
V783	VID003	V7815	750	980323
V985	VID003	V7733	2500	980223
V966	VID001	V5020	1167	980215
V98	VID003	V7750	2000	980223

このテーブルを使用して、製品ごとの四半期の月別売上を示す要約レポートを作成します。

解法

次の PROC SQL コードを使用して、四半期の月ごとに列を作成します。次に、要約関数 SUM を GROUP BY ステートメントと組み合わせて使用して、製品ごとの月別売上を累積計算します。

```
proc sql;
   title 'First Quarter Sales by Product';
   select Product,
          sum(Jan) label='Jan',
          sum(Feb) label='Feb',
          sum(Mar) label='Mar'
      from (select Product,
                   case
                      when substr(InvoiceDate,3,2)='01' then
                         InvoiceAmount end as Jan,
                   case
                      when substr(InvoiceDate,3,2)='02' then
                         InvoiceAmount end as Feb,
                   case
                      when substr(InvoiceDate,3,2)='03' then
                         InvoiceAmount end as Mar
               from work.sales)
      group by Product;
```

アウトプット 6.16 要約レポートの PROC SQL 出力

First Quarter Sales by Product

Product	Jan	Feb	Mar
VID001	.	1167	.
VID003	.	5500	4750
VID005	.	.	1070
VID010	1197	.	750
VID014	.	.	1070

注: 行列内の欠損値は、その月の特定の製品で売上が発生しなかったことを示しています。

動作

この解法では、インラインビューを使用して、InvoiceDate 列の月の部分に基づき、Jan、Feb および Mar という 3 つの一時列を作成します。このインラインビューは、次を実行するクエリです。

- Product 列を選択します。
- CASE 式を使用して、InvoiceDate 列の月の部分の値に応じて、送り状金額の値を 3 つの列(Jan、Feb または Mar)のうちのいずれかに割り当てます。

```
case
   when substr(InvoiceDate,3,2)='01' then
      InvoiceAmount end as Jan,
case
   when substr(InvoiceSate,3,2)='02' then
      InvoiceAmount end as Feb,
case
   when substr(InvoiceDate,3,2)='03' then
      InvoiceAmount end as Mar
```

クエリの最初(つまり外側)の SELECT ステートメントは、次を実行します。

- 製品を選択します。
- 要約関数 SUM を使用して、Jan、Feb および Mar の金額を累積計算します。
- GROUP BY ステートメントを使用して、製品ごとにテーブル内に行を生成します。

なお、ここで日付は、入力テーブルに文字列として格納されています。日付を SAS 日付として格納した場合、CASE 式を次のように記述できます。

```
case
   when month(InvoiceDate)=1 then
      InvoiceAmount end as Jan,
case
   when month(InvoiceDate)=2 then
      InvoiceAmount end as Feb,
case
   when month(InvoiceDate)=3 then
```

```
      InvoiceAmount end as Mar
```

カスタマイズされた並べ替え順序の作成

問題

データをアルファベット順ではなく論理的な順序で並べ替えます。

背景情報

Chores という 1 つの入力テーブルがあり、これには次のデータが含まれています。

```
data chores;
   input Project $ Hours Season $;
   datalines;
weeding 48 summer
pruning 12 winter
mowing 36 summer
mulching 17 fall
raking 24 fall
raking 16 spring
planting 8 spring
planting 8 fall
sweeping 3 winter
edging 16 summer
seeding 6 spring
tilling 12 spring
aerating 6 spring
feeding 7 summer
rolling 4 winter
;


proc sql;
title 'Garden Chores';
select * from chores;
quit;
```

アウトプット 6.17 カスタマイズされた並べ替えのための入力データの例

Garden Chores

Project	Hours	Season
weeding	48	summer
pruning	12	winter
mowing	36	summer
mulching	17	fall
raking	24	fall
raking	16	spring
planting	8	spring
planting	8	fall
sweeping	3	winter
edging	16	summer
seeding	6	spring
tilling	12	spring
aerating	6	spring
feeding	7	summer
rolling	4	winter

この仕事のリストを並べかえ、春から始まって 1 年を通じて進行するように、すべての仕事を季節ごとにグループ化します。単純に季節で並べ替えることによって、このリストはアルファベット順(fall、spring、summer、winter)で並べ替えられたように見えます。

解法

次の PROC SQL コードを使用して、春から冬にかけての季節を表す 1 から 4 までの値を格納するための新しい列(Sorter)を作成します。新しい列は、クエリを並べ替えるために使用されますが、表示では選択されません。

```
proc sql;
   title 'Garden Chores by Season in Logical Order';
   select Project, Hours, Season
      from (select Project, Hours, Season,
              case
                 when Season = 'spring' then 1
                 when Season = 'summer' then 2
                 when Season = 'fall' then 3
                 when Season = 'winter' then 4
                 else .
              end as Sorter
              from chores)
      order by Sorter;
```

アウトプット 6.18 カスタマイズされた並べ替え順序の PROC SQL 出力

Garden Chores by Season in Logical Order

Project	Hours	Season
tilling	12	spring
raking	16	spring
planting	8	spring
seeding	6	spring
aerating	6	spring
mowing	36	summer
feeding	7	summer
edging	16	summer
weeding	48	summer
raking	24	fall
mulching	17	fall
planting	8	fall
rolling	4	winter
pruning	12	winter
sweeping	3	winter

動作

この解法では、インラインビューを使用して、ORDER BY 列として使用できる一時列を作成します。このインラインビューは、次を実行するクエリです。

- Project 列、Hours 列および Season 列を選択します。
- CASE 式を使用して、各季節を新しい列 Sorter に再マッピングします。つまり、spring は 1、summer は 2、fall は 3、winter は 4 にマッピングされます。

```
(select project, hours, season,
        case
           when season = 'spring' then 1
           when season = 'summer' then 2
           when season = 'fall' then 3
           when season = 'winter' then 4
           else .
        end as sorter
    from chores)
```

クエリの最初(つまり外側)の SELECT ステートメントは、次を実行します。

- Project 列、Hours 列および Season 列を選択します。

- インラインビューで作成された Sorter 列内の、各季節に割り当てられた値によって、各行を並べ替えます。

Sorter 列が SELECT ステートメントに含まれていないことに注目してください。これによって、SELECT ステートメントに現れない列が ORDER BY ステートメントで使用されたことを示す注釈がログに記録されます。この場合、それが期待どおりの結果です。

テーブルの条件付き更新

問題

テーブル内の他の複数の列の値に基づいてテーブルの列の値を更新します。

背景情報

Incentives という 1 つのテーブルがあり、これには売上データについての情報が含まれています。販売員ごとに 1 つのレコードが存在し、このレコードには部門コード、基準賃金および 2 つの製品(道具と物置き棚)の売上が含まれています。

```
data incentives;
   input @1 Name $18. @20 Department $2. Payrate
            Gadgets Whatnots;
   datalines;
Lao Che            M2   8.00  10193  1105
Jack Colton        U2   6.00   9994  2710
Mickey Raymond     M1  12.00   6103  1930
Dean Proffit       M2  11.00   3000  1999
Antoinette Lily    E1  20.00   2203  4610
Sydney Wade        E2  15.00   4205  3010
Alan Traherne      U2   4.00   5020  3000
Elizabeth Bennett  E1  16.00  17003  3003
;



proc sql;
  title 'Sales Data for Incentives Program';
  select * from incentives;
quit;
```

アウトプット 6.19 テーブルを条件付きで変更するための入力データの例

Sales Data for Incentives Program

Name	Department	Payrate	Gadgets	Whatnots
Lao Che	M2	8	10193	1105
Jack Colton	U2	6	9994	2710
Mickey Raymond	M1	12	6103	1930
Dean Proffit	M2	11	3000	1999
Antoinette Lily	E1	20	2203	4610
Sydney Wade	E2	15	4205	3010
Alan Traherne	U2	4	5020	3000
Elizabeth Bennett	E1	16	17003	3003

各販売員の賃金を(道具と物置き棚の総売上に基づいて)増やし、部門コードに基づくいくつかの要因を考慮してテーブルを更新します。

特に 10,000 個を超える道具を売り上げた人には、すべて 1 時間あたり 5 ドルの追加手当が与えられます。5,000 個から 10,000 個までの道具を売り上げた販売員にもすべて奨励金が与えられますが、E 部門の販売員は他の部門よりも高い売上を期待されているため、他の部門の道具売上の奨励金が 1 時間当たり 3 ドルなのに対して、E 部門の奨励金は 2 ドルとなっています。物置き棚の売上が好調な販売員には、追加奨励金の権利も与えられます。物置き棚の売上に対するアルゴリズムは次のとおりです。2,000 個を超える物置き棚を売り上げたトップレベル(各部門のレベル 1)の販売員に対しては 1 時間当たり 0.5 ドルの追加手当が与えられ、2,000 個を超える物置き棚を売り上げたレベル 2 の販売員対しては 1 時間当たり 1 ドルの追加手当が与えられます。

解法

次の PROC SQL コードを使用して、Payrate 列に新しい値を作成します。実際には、Payrate は行ごとに 2 回更新されます。1 回目は道具の売上に基づき、2 回目は物置き棚の売上に基づきます。

```
proc sql;
   update incentives
   set payrate = case
                    when gadgets > 10000 then
                       payrate + 5.00
                    when gadgets > 5000 then
                       case
                          when department in ('E1', 'E2') then
                             payrate + 2.00
                          else payrate + 3.00
                       end
                    else payrate
                 end;
   update incentives
   set payrate = case
```

```
                    when whatnots > 2000 then
                       case
                          when department in ('E2', 'M2', 'U2') then
                             payrate + 1.00
                          else payrate + 0.50
                       end
                    else payrate
                 end;
   title 'Adjusted Payrates Based on Sales of Gadgets and Whatnots';
   select * from incentives;
```

アウトプット 6.20 *テーブルの条件付き更新の PROC SQL 出力*

Adjusted Payrates Based on Sales of Gadgets and Whatnots

Name	Department	Payrate	Gadgets	Whatnots
Lao Che	M2	13	10193	1105
Jack Colton	U2	10	9994	2710
Mickey Raymond	M1	15	6103	1930
Dean Proffit	M2	11	3000	1999
Antoinette Lily	E1	20.5	2203	4610
Sydney Wade	E2	16	4205	3010
Alan Traherne	U2	8	5020	3000
Elizabeth Bennett	E1	21.5	17003	3003

動作

この解法では、Incentive テーブルの Payrate 列に対する連続的な更新が実行されます。最初の更新ではネストされた CASE 式が使用されて、道具の売上に基づく区分の最初の判定が行われます。10,000 個を超える売上は 5 ドルの奨励金を必要とします。5,000 個から 10,000 個のまでの売上については、別の比較が必要になります。この比較は、ネストされた CASE 式を使用して部門コードをチェックし、2 ドルまたは 3 ドルの奨励金を選択することによって実現されます。

```
update incentives
set payrate = case
                 when gadgets > 10000 then
                    payrate + 5.00
                 when gadgets > 5000 then
                    case
                       when department in ('E1', 'E2') then
                          payrate + 2.00
                       else payrate + 3.00
                    end
                 else payrate
              end;
```

2回目の更新も同様ですが、さらに単純です。2,000 個を超える物置き棚の売り上げに対しては、部門のレベルに応じて、すべて 0.5 ドルまたは 1 ドルのいずれかの奨励金が与えられます。これも、ネストされた CASE 式によって実現されます。

```
update incentives
   set payrate = case
                    when whatnots > 2000 then
                       case
                          when department in ('E2', 'M2', 'U2') then
                             payrate + 1.00
                          else payrate + 0.50
                       end
                    else payrate
                 end;
```

別のテーブルの値を使用してテーブルを更新する

問題

更新された人口データを反映するために、Sql.United States テーブルを更新します。

背景情報

Sql.Newpop テーブルには、アメリカの一部の州の更新された人口データが含まれています。

```
libname sql 'SAS-library';

proc sql;
title 'Updated U.S. Population Data';
select state, population format=comma10. label='Population' from sql.newpop;
```

アウトプット 6.21 更新された人口データを含むテーブル

Updated U.S. Population Data

state	Population
Texas	20,851,820
Georgia	8,186,453
Washington	5,894,121
Arizona	5,130,632
Alabama	4,447,100
Oklahoma	3,450,654
Connecticut	3,405,565
Iowa	2,926,324
West Virginia	1,808,344
Idaho	1,293,953
Maine	1,274,923
New Hampshire	1,235,786
North Dakota	642,200
Alaska	626,932

解法

次の PROC SQL コードを使用して、Sql.UnitedStates テーブル内の州ごとの人口情報を更新します。

```
proc sql;
title 'UnitedStates';
update sql.unitedstates as u
   set population=(select population from sql.newpop as n
            where u.name=n.state)
         where u.name in (select state from sql.newpop);
select Name format=$17., Capital format=$15.,
       Population, Area, Continent format=$13., Statehood format=date9.
   from sql.unitedstates;

/* use this code to generate output so you don't
   overwrite the sql.unitedstates table */
options ls=84;
proc sql outobs=10;
title 'UnitedStates';
create table work.unitedstates as
   select * from sql.unitedstates;
update work.unitedstates as u
   set population=(select population from sql.newpop as n
```

```
            where u.name=n.state)
          where u.name in (select state from sql.newpop);
  select Name format=$17., Capital format=$15.,
          Population, Area, Continent format=$13., Statehood format=date9.
   from work.unitedstates
;
```

アウトプット 6.22 *更新された人口データを含む Sql.UnitedStates (部分的な出力)*

UNITEDSTATES

Name	Capital	Population	Area	Continent	Statehood
Alabama	Montgomery	4447100	52423	North America	14DEC1819
Alaska	Juneau	626932	656400	North America	03JAN1959
Arizona	Phoenix	5130632	114000	North America	14FEB1912
Arkansas	Little Rock	2447996	53200	North America	15JUN1836
California	Sacramento	31518948	163700	North America	09SEP1850
Colorado	Denver	3601298	104100	North America	01AUG1876
Connecticut	Hartford	3405565	5500	North America	09JAN1788
Delaware	Dover	707232	2500	North America	07DEC1787
District of Colum	Washington	612907	100	North America	21FEB1871
Florida	Tallahassee	13814408	65800	North America	03MAR1845

動作

UPDATE ステートメントは、Sql.UnitedStates テーブル(ここではエイリアス U を使用)の値を更新します。Sql.UnitedStates テーブルの行ごとに、SET 句内のインラインビューが 1 つの値を返します。Sql.NewPop 内に対応する行が存在する行の場合、この値は Sql.NewPop の Population 列の値になります。Sql.NewPop 内に対応する行が存在しない行の場合、この値は欠損値になります。どちらの場合も、返された値は Population 列に割り当てられます。

WHERE 句は、Sql.NewPop 内に対応する行が存在する Sql.UnitedStates の行のみを更新できるようにします。これは、それぞれの Name の値を、インラインビューから返された州名のリストに対してチェックすることによって行います。この WHERE 句がないと、Sql.NewPop 内に対応する行が存在しない行は、Population 値が欠損値に更新されます。

マクロ変数の作成および使用

問題

列の一意の値ごとに別々のデータセットを作成します。

背景情報

Sql.Features データセットには、世界中のさまざまな地勢に関する情報が含まれています。

```
libname sql 'SAS-library';

proc sql outobs=10;
title 'Features';
 select Name format=$15., Type,Location format =$15.,Area,
        Height, Depth, Length
 from sql.features;
```

アウトプット 6.23 *Features (部分的出力)*

FEATURES

Name	Type	Location	Area	Height	Depth	Length
Aconcagua	Mountain	Argentina	.	22834	.	.
Amazon	River	South America	.	.	.	4000
Amur	River	Asia	.	.	.	2700
Andaman	Sea		218100	.	3667	.
Angel Falls	Waterfall	Venezuela	.	3212	.	.
Annapurna	Mountain	Nepal	.	26504	.	.
Aral Sea	Lake	Asia	25300	.	222	.
Ararat	Mountain	Turkey	.	16804	.	.
Arctic	Ocean		5105700	.	17880	.
Atlantic	Ocean		33420000	.	28374	.

解法

地物のタイプごとに別々のデータセットを作成するために、データセットを手動で調べ、タイプの一意の値をすべて決定し、次にタイプごとに別々の DATA ステップを記述できます(あるいは、複数の OUTPUT ステートメントを使用して 1 つの DATA ステップを記述できます)。この方法は、大きなデータセットの場合、多くの手間がかかり、エラーが発生しやすいため、実用的ではありません。次の PROC SQL コードは、Type の一意の値をカウントし、それらの値を別々のマクロ変数に格納します。PROC SQL コードの下にある SAS マクロは、これらのマクロ変数を使用して、値ごとに SAS データセットを作成します。一意の値の個数やそれらの値が何であるかを前もって把握する必要はありません。

```
proc sql noprint;
   select count(distinct type)
      into :n
      from sql.features;
```

```
   select distinct type
      into :type1 - :type%left(&n)
      from sql.features;
quit;

%macro makeds;
   %do i=1 %to &n;
      data &&type&i (drop=type);
         set sql.features;
         if type="&&type&i";
      run;
   %end;
%mend makeds;
%makeds;
```

ログ 6.1 *列の一意の値ごとに別々のデータセットを作成した後の SAS ログ*

```
240 proc sql noprint; 241 select count(distinct type) 242 into :n 243 from
sql.features; 244 select distinct type 245 into :type1 - :type%left(&n) 246 from
sql.features; 247 quit; NOTE:PROCEDURE SQL used (Total process time): real time
0.04 seconds cpu time 0.03 seconds 248 249 %macro makeds; 250 %do i=1 %to &n;
251 data &&type&i (drop=type); 252 set sql.features; 253 if type="&&type&i"; 254
run; 255 %end; 256 %mend makeds; 257 %makeds; NOTE:There were 74 observations
read from the data set SQL.FEATURES. NOTE:注:The data set WORK.DESERT has 7
observations and 6 variables. NOTE:注:DATA statement used (Total process time):
real time 1.14 seconds cpu time 0.41 seconds NOTE:There were 74 observations
read from the data set SQL.FEATURES. NOTE:注:The data set WORK.ISLAND has 6
observations and 6 variables. NOTE:注:DATA statement used (Total process time):
real time 0.02 seconds cpu time 0.00 seconds NOTE:There were 74 observations
read from the data set SQL.FEATURES. NOTE:注:The data set WORK.LAKE has 10
observations and 6 variables. NOTE:注:DATA statement used (Total process time):
real time 0.01 seconds cpu time 0.01 seconds NOTE:There were 74 observations
read from the data set SQL.FEATURES. NOTE:注:The data set WORK.MOUNTAIN has 18
observations and 6 variables. NOTE:注:DATA statement used (Total process time):
real time 0.02 seconds cpu time 0.01 seconds
```

```
NOTE:There were 74 observations read from the data set SQL.FEATURES. NOTE:注:The
data set WORK.OCEAN has 4 observations and 6 variables. NOTE:注:DATA statement
used (Total process time): real time 0.01 seconds cpu time 0.01 seconds
NOTE:There were 74 observations read from the data set SQL.FEATURES. NOTE:注:The
data set WORK.RIVER has 12 observations and 6 variables. NOTE:注:DATA statement
used (Total process time): real time 0.02 seconds cpu time 0.02 seconds
NOTE:There were 74 observations read from the data set SQL.FEATURES. NOTE:注:The
data set WORK.SEA has 13 observations and 6 variables. NOTE:注:DATA statement
used (Total process time): real time 0.03 seconds cpu time 0.02 seconds
NOTE:There were 74 observations read from the data set SQL.FEATURES. NOTE:注:The
data set WORK.WATERFALL has 4 observations and 6 variables. NOTE:注:DATA
statement used (Total process time): real time 0.02 seconds cpu time 0.02 seconds
```

動作

この解法は、INTO 句を使用して、値をマクロ変数に格納します。最初の SELECT ステートメントは一意の変数をカウントし、その結果をマクロ変数 N に格納します。2 番目の SELECT ステートメントは、それぞれの一意の値に対応する一連のマクロ変数を作成し、それぞれの一意の値をマクロ変数のいずれかに格納します。%LEFT 関数の使用方法に注意してください。この関数は、N マクロ変数の値の先頭の空白を削除しています。

MAKEDS マクロは、PROC SQL ステップで作成されたすべてのマクロ変数を使用します。このマクロは、%DO ループを使用して一意の値ごとに DATA ステップを実行し、特定の Type の値を含む行を、同じ名前の SAS データセットに書き込みます。Type 変数は出力データセットから除去されます。

SAS マクロの詳細については、*SAS マクロ言語: リファレンス*を参照してください。

他の SAS プロシジャでの PROC SQL テーブルの使用

問題

ヨーロッパ各国の平均最高気温を摂氏温度でマップ上に表示します。

背景情報

Sql.Worldtemps テーブルには、世界中のさまざまな都市の平均最高気温と平均最低気温が含まれています。

```
proc sql outobs=10;
title 'WorldTemps';
 select City, Country,avghigh, avglow
  from sql.worldtemps
;
```

アウトプット 6.24 *WorldTemps (部分的出力)*

WORLDTEMPS

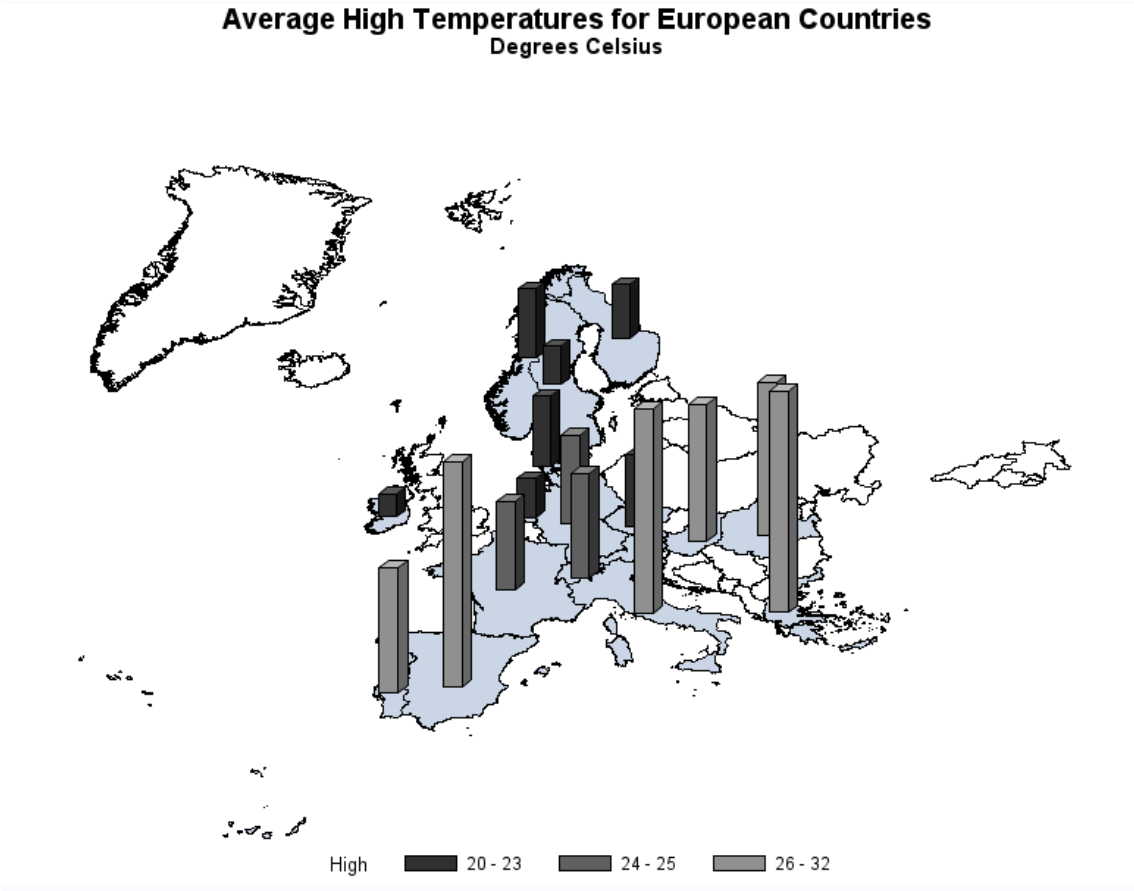
City	Country	AvgHigh	AvgLow
Algiers	Algeria	90	45
Amsterdam	Netherlands	70	33
Athens	Greece	89	41
Auckland	New Zealand	75	44
Bangkok	Thailand	95	69
Beijing	China	86	17
Belgrade	Yugoslavia	80	29
Berlin	Germany	75	25
Bogota	Colombia	69	43
Bombay	India	90	68

解法

次の PROC SQL コードと PROC GMAP コードを使用してマップを生成します。PROC GMAP を使用するには、SAS/GRAPH ソフトウェアのライセンスが必要です。

```
options fmtsearch=(sashelp.mapfmts);

proc sql;
   create table extremetemps as
   select country, round((mean(avgHigh)-32)/1.8) as High,
      input(put(country,$glcsmn.), best.) as ID
   from sql.worldtemps
   where calculated id is not missing and country in
      (select name from sql.countries where continent='Europe')
   group by country;
quit;

proc gmap map=maps.europe data=extremetemps all;
   id id;
   block high / levels=3;
   title 'Average High Temperatures for European Countries';
   title2 'Degrees Celsius'
run;
quit;
```

アウトプット 6.25 PROC GMAP の出力



動作

SAS システムオプション FMTSEARCH=は、SAS に対して、Sashelp.Mapfmts カタログからマップ関連のフォーマットを検索するように指定します。PROC SQL ステップで、Country 列、High 列および ID 列を含む一時テーブルが作成されます。`round((mean(avgHigh)-32)/1.8)`の計算は、次のように実行されます。

1. 複数の都市によって代表されている国の場合は、各都市の平均最高気温の平均値が、その国に対して使用されます。
2. その値は、華氏温度から摂氏温度に変換されます。
3. その結果は、最も近い温度に丸められます。

PUT 関数は、$GLCSMN 出力形式を使用して、国名を国別コードに変換します。INPUT 関数は、PUT 関数が文字値で返したこの国別コードを、GMAP プロシジャが理解できる数値に変換します。PUT 関数と INPUT 関数の詳細については、*SAS 関数と CALL ルーチン: リファレンス*を参照してください。

WHERE 句は、インラインビューが返したヨーロッパ各国のリストに対して Country 列の値をチェックすることによって、出力をヨーロッパ各国に限定します。また、ID の欠損値を含む行は除去されます。$GLCSMN.出力形式が国名を認識しない場合に、ID の欠損値が生成されることがあります。

GROUP BY 句は、テーブル全体に対してではなく国ごとに平均気温を計算できるようにするために必要です。

PROC GMAP ステップでは、ID 変数を使用して各国を識別し、各国の High の値を表すブロックを、マップ上のそれぞれの国に配置します。ALL オプションは、High の値を持たない国(この例の英国など)もマップ上に描画できるようにします。BLOCK ステートメントでは、LEVELS=オプションによって、グラフで使用される応答レベルの数を指定しています。GMAP プロシジャの詳細については、*SAS/GRAPH: Reference* を参照してください。

2 部

SQL プロシジャリファレンス

7 章
SQL プロシジャ *213*

8 章
SQL プロシジャの構成要素 *311*

7 章

SQL プロシジャ

概要 **214**
SQL プロシジャについて 214
PROC SQL テーブルについて 214
ビューについて 215
SQL プロシジャのコーディング規則 216
PROC SQL を使用したスレッド化処理 216

構文: SQL プロシジャ **217**
PROC SQL ステートメント 220
ALTER TABLE ステートメント 231
CONNECT ステートメント 235
CREATE INDEX ステートメント 236
CREATE TABLE ステートメント 237
CREATE VIEW ステートメント 242
DELETE ステートメント 245
DESCRIBE ステートメント 246
DISCONNECT ステートメント 247
DROP ステートメント 247
EXECUTE ステートメント 248
INSERT ステートメント 249
RESET ステートメント 251
SELECT ステートメント 252
UPDATE ステートメント 265
VALIDATE ステートメント 266

例: SQL プロシジャ **267**
例 1: テーブルを作成し、データを挿入する 267
例 2: テーブルをクエリの結果から作成する 269
例 3: PROC SQL テーブルのデータの更新 271
例 4: 2 つのテーブルを結合する 273
例 5: 2 つのテーブルを組み合わせる 276
例 6: DICTIONARY テーブルからレポートを作成する 279
例 7: 外部結合を実行する 281
例 8: ビューをクエリの結果から作成する 286
例 9: 3 つのテーブルを結合する 289
例 10: インラインビューをクエリする 293
例 11: SOUNDS-LIKE 演算子を使用して値を取得する 295
例 12: 2 つのテーブルを結合して新しい値を計算する 297
例 13: 列内の値の使用可能な組み合わせをすべて作成する 300
例 14: ケース行とコントロール行の照合 306
例 15: SAS マクロを使用して欠損値をカウントする 309

概要

SQL プロシジャについて

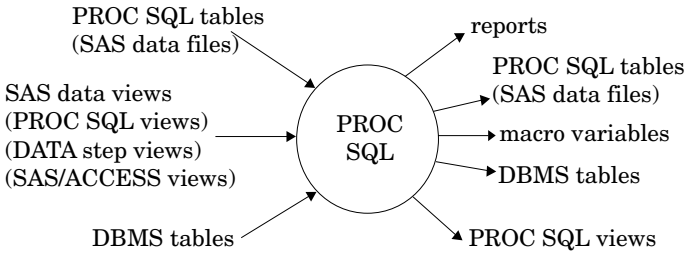
SQL プロシジャは、SAS での構造化照会言語(SQL)の実装です。SQL は、テーブル内およびそれらのテーブルに基づくビュー内のデータを検索および更新する、標準化され、広く使用されている言語です。

SAS SQL プロシジャを使用して、次を実行できます。

- テーブルまたはビューに格納されたデータを検索および操作します。
- テーブル、ビューを作成し、テーブルの列にインデックスを作成します。
- クエリ結果の行の値を含む SAS マクロ変数を作成します。
- テーブルの列のデータ値の追加や変更、行の挿入や削除を行います。列を追加、変更または削除して、テーブル自体を変更することもできます。
- DBMS 固有の SQL ステートメントをデータベース管理システム(DBMS)に送信し、DBMS のデータを検索します。

次の図に、PROC SQL で使用できるさまざまな入力と、プロシジャが生成できる出力をまとめます。

図 7.1 PROC SQL の入力と出力



PROC SQL テーブルについて

PROC SQL テーブルは、SAS データファイルと同期しており、DATA のメンバタイプが設定されています。DATA ステップおよびプロシジャへの入力として、PROC SQL テーブルを使用できます。

PROC SQL テーブルは、SAS データファイル、SAS ビューまたは DBMS テーブルから、PROC SQL のパススルー機能または SAS/ACCESS LIBNAME ステートメントを使用して作成されます。パススルー機能は、“パススルー機能を使用した DBMS への接続” (172 ページ)で説明されています。SAS/ACCESS LIBNAME ステートメントは、“LIBNAME ステートメントを使用した DBMS への接続” (169 ページ)で説明されています。

PROC SQL の用語では、テーブルの**行**は、SAS データファイルの**オブザベーション**と同じです。**列**は、変数と同じです。

ビューについて

SAS ビューは、後で使用するために名前を付けて格納される仮想データセットを定義します。ビューにはデータが含まれていませんが、別の場所に格納されたデータが記述または定義されています。SAS ビューには、次の 3 つのタイプがあります。

- PROC SQL ビュー
- SAS/ACCESS ビュー
- DATA ステップビュー

クエリ内で、テーブルと同じようにビューを参照できます。ビューのデータは、FROM 句に記述されたテーブルまたはビューから派生します。ビューによってアクセスするデータは、派生元のテーブルまたはビューに含まれるデータのサブセットまたはスーパーセットです。

PROC SQL ビューは、PROC SQL によって作成される VIEW タイプの SAS データセットです。PROC SQL ビューにはデータは含まれません。PROC SQL ビューは、その派生元のファイル(SAS データファイル、SAS/ACCESS ビュー、DATA ステップビュー、他の PROC SQL ビュー、DBMS データなど)からデータ値を読み取る、格納されたクエリ式です。PROC SQL ビューを実行すると、1 つ以上の派生元のファイルのサブセットまたはスーパーセットを出力できます。

SAS/ACCESS ビューと DATA ステップビューは、PROC SQL ビューに類似しており、両方とも VIEW メンバタイプのストアドプログラムとして PROC SQL ビューに含まれます。SAS/ACCESS ビューは、他のソフトウェアベンダの DBMS テーブル内のデータを表します。DATA ステップビューは、格納された DATA ステッププログラムです。

注: SAS System 9 で開始する場合は、リレーショナル DBMS データへのアクセス方法として、PROC SQL ビュー、パススルー機能および SAS/ACCESS LIBNAME ステートメントを推奨します。SAS/ACCESS ビューは推奨されていません。CV2VIEW プロシジャを使用して、既存の SAS/ACCESS ビューを PROC SQL ビューに変換できます。詳細については、“CV2VIEW” (*SAS/ACCESS for Relational Databases: Reference*)を参照してください。

PROC SQL ビューまたは SAS/ACCESS ビューを介して、特定の制限付きでデータを更新できます。“PROC SQL ビューと SAS/ACCESS ビューの更新” (174 ページ)を参照してください。

すべてのタイプのビューを、DATA ステップおよびプロシジャへの入力として使用できます。

注: この章では、**ビュー**という用語は、特に断らない限り PROC SQL ビュー、DATA ステップビューおよび SAS/ACCESS ビューを総称しています。

注: DATA ステップまたはプロシジャによって SQL ビューの内容が処理されるときに、ビューに格納されていない変数について情報を検索するために、参照されるデータセットが開かれている必要があります。そのデータセットに、現在の SAS コードでは定義されていないライブラリ参照名が関連付けられている場合、エラーが発生します。CREATE VIEW ステートメントで USING 句を指定することによって、このエラーを回避できます。詳細については、“CREATE VIEW ステートメント” (242 ページ)を参照してください。

注: クライアントとサーバー間で PROC SQL ビューを処理する場合、正しい結果が得られるかどうかは、クライアントとサーバー間のアーキテクチャの互換性に依存します。詳細については、“Accessing a SAS View” (*SAS/CONNECT User's Guide*)を参照してください。

SQL プロシジャのコーディング規則

PROC SQL は構造化照会言語を実装しているため、次に述べるように、他の Base SAS プロシジャとはやや異なった動作をします。

- PROC SQL ステートメントが実行されると、PROC SQL は、QUIT ステートメント、DATA ステップまたは別の SAS プロシジャが実行されるまで実行を継続します。したがって、PROC SQL ステートメントを各 SQL ステートメントで繰り返す必要はありません。SQL ステートメント間で QUIT ステートメント、DATA ステップまたは別の SAS プロシジャを実行した場合にのみ、PROC SQL ステートメントを繰り返す必要があります。
- SQL プロシジャステートメントは、複数の句に分かれています。たとえば、最も基本的な SELECT ステートメントには、SELECT 句と FROM 句が含まれています。SQL では、句に含まれる項目は、他の SAS コードにおけるような空白ではなく、カンマで区切られています。たとえば、SELECT 句で 3 つの列を記述する場合、それらの列はカンマで区切られます。
- また、データの検索に使用される SELECT ステートメントは、PROC SQL ステートメントで NOPRINT オプションを指定しない限り、自動的に出力データを**アウトプット**ウィンドウに出力します。したがって、PRINT プロシジャを指定しなくても、出力を表示したり、それをリストファイルに送信したりできます。
- ORDER BY 句は、データを列によって並べ替えます。さらに、テーブルを、PROC SQL で使用するために、変数によって事前に並べ替える必要はありません。したがって、PROC SQL プログラムでは、SORT プロシジャを使用する必要はありません。
- PROC SQL ステートメントは、それをサブミットすると実行されます。RUN ステートメントを指定する必要はありません。PROC SQL ステートメントの後に RUN ステートメントを記述した場合、SAS はその RUN ステートメントを無視し、通常どおりステートメントをサブミットします。

PROC SQL を使用したスレッド化処理

THREADS オプションは、PROC SQL での並列処理を有効化または無効化します。スレッド化処理は、処理操作における並列性の度合いを達成します。この並列性は、処理操作の完了までにかかる実時間の短縮と、その結果としての追加的な CPU リソースにかかるコストの制限を目的としています。詳細については“Support for Parallel Processing” (*SAS Language Reference: Concepts*)を参照してください。

SAS システムオプション CPUCOUNT=の値は、スレッド化された並べ替えの性能に影響を与えます。CPUCOUNT=は、スレッド化されたプロシジャにより使用可能なシステム CPU の数を指定します。

THREADS オプションの詳細については、“THREADS|NOTHREADS” (229 ページ)を参照してください。

詳細については、“CPUCOUNT= System Option” (*SAS System Options: Reference*)を参照してください。

構文: SQL プロシジャ

ヒント: 任意のグローバルステートメントを使用できます。詳細については、“Fundamental Concepts for Using Base SAS Procedures” (*Base SAS Procedures Guide*)を参照してください。

テーブル名またはビュー名を指定する際にはいつでも、データセットオプションを使用できます。詳細については、“PROC SQL で SAS データセットオプションを使用する” (157 ページ)を参照してください。

標準タイプは、構成要素の名前を示します。これについては、8 章, “SQL プロシジャの構成要素” (311 ページ)で説明されています。*view-name* は、任意のタイプの SAS ビューを示しています。

```
PROC SQL <option(s)>;
  ALTER TABLE table-name
    <ADD column-definition-1 <, column-definition-2, …>>
    <ADD CONSTRAINT constraint-name-1 constraint-specification-1
      <, constraint-name-2 constraint-specification-2, …>>
    <ADD constraint-specification-1 <, constraint-specification-2, …>>
    <DROP column-1 <, column-2, …>>
    <DROP CONSTRAINT constraint-name-1 <, constraint-name-2, …>>
    <DROP FOREIGN KEY constraint-name>
    <DROP PRIMARY KEY>
    <MODIFY column-definition-1 <, column-definition-2, …>>
  ;
  CREATE <UNIQUE> INDEX index-name
    ON table-name (column-1 <, column-2, …>);
  CREATE TABLE table-name
    (column-specification-1 <, column-specification-2 | constraint-specification-1, …>);
  CREATE TABLE table-name LIKE table-name-2;
  CREATE TABLE table-name AS query-expression
    <ORDER BY order-by-item-1 <, order-by-item-2, …>>;
  CREATE VIEW proc-sql-view <(column-name-list)> AS query-expression
    <ORDER BY order-by-item-1 <, order-by-item-2, …>>
    <USING libname-clause-1 <, libname-clause-2, …>> ;
  DELETE FROM table-name | sas/access-view | proc-sql-view
    <AS alias>
    <WHERE sql-expression>;
  DESCRIBE TABLE table-name-1 <, table-name-2, …>;
  DESCRIBE VIEW proc-sql-view-1 <, proc-sql-view-2, …>;
  DESCRIBE TABLE CONSTRAINTS table-name-1 <, table-name-2, …>;
  DROP INDEX index-name-1 <, index-name-2, …> FROM table-name;
  DROP TABLE table-name-1 <, table-name-2, …>;
  DROP VIEW view-name-1 <, view-name-2, …>;
  INSERT INTO table-name | sas/access-view | proc-sql-view
    <(column-1<, column-2, …>)>
      SET column1=sql-expression-1 <, column-2=sql-expression-2, …>
        <SET column1=sql-expression-1 <, column-2=sql-expression-2, …> …>;
  INSERT INTO table-name | sas/access-view | proc-sql-view
    <(column-1 <, column-2, …>)>
      VALUES (value-1 <, value-2, …>)
        <VALUES (value-1 <, value-2, …>) …>;
  INSERT INTO table-name | sas/access-view | proc-sql-view
    <(column-1 <, column-2, …>)> query-expression;
  RESET <option(s)>;
```

SELECT <DISTINCT | UNIQUE> *object-item-1* <, *object-item-2*, …>
<INTO *macro-variable-specification-1* <, *macro-variable-specification-2*, …>>
FROM *from-list*
<WHERE *sql-expression*>
<GROUP BY *group-by-item-1* <, *group-by-item-2*, …>>
<HAVING *sql-expression*>
<ORDER BY *order-by-item-1* <, *order-by-item-2* <ASC | DESC>, …>>;
UPDATE *table-name* | *sas/access-view* | *proc-sql-view* <AS *alias*>
SET *column-1*=*sql-expression-1* <, *column-2*=*sql-expression-2*, …>
<SET *column-1*=*sql-expression-1* <, *column-1*=*sql-expression-2*, …>>
<WHERE *sql-expression*>;
VALIDATE *query-expression*;
<QUIT;>

DBMS に接続して、DBMS 固有の非クエリSQL ステートメントを送信するには、次の形式を使用します。

PROC SQL;
CONNECT TO *dbms-name* <AS *alias*>
<(*connect-statement-argument-1*=*value-1*
<*connect-statement-argument-2*=*value-2* …>)>
<(*database-connection-argument-1*=*value-1*
<*database-connection-argument-2*=*value-2* …>)>;
EXECUTE (*dbms-SQL-statement*)
BY *dbms-name* | *alias*;
<DISCONNECT FROM *dbms-name* | *alias*;>
<QUIT;>

DBMS に接続して、DBMS データにクエリを行うには、次の形式を使用します。

PROC SQL;
CONNECT TO *dbms-name* <AS *alias*>
<(*connect-statement-argument-1*=*value-1*
<*connect-statement-argument-2*=*value-2* …>)>
<(*database-connection-argument-1*=*value-1*
<*database-connection-argument-2*=*value-2* …>)>;
SELECT *column-list*
FROM CONNECTION TO *dbms-name* | *alias*
(*dbms-query*)
optional PROC SQL clauses;
<DISCONNECT FROM *dbms-name* | *alias*;>
<QUIT;>

ステートメント	タスク	例
“PROC SQL ステートメント”	テーブルおよびテーブルに基づくビューでの、データの作成、維持、検索、更新	Ex. 1
“ALTER TABLE ステートメント”	列の変更、追加または削除	Ex. 3
“CONNECT ステートメント”	DBMS との接続の確立	

ステートメント	タスク	例
“CREATE INDEX ステートメント”	列でのインデックスの作成	
“CREATE TABLE ステートメント”	PROC SQL テーブルの作成	Ex. 2
“CREATE VIEW ステートメント”	PROC SQL ビューの作成	Ex. 8
“DELETE ステートメント”	行の削除	Ex. 5
“DESCRIBE ステートメント”	テーブルまたはビューの定義の表示	Ex. 6
“DISCONNECT ステートメント”	DBMS との接続の終了	
“DROP ステートメント”	テーブル、ビューまたはインデックスの削除	
“EXECUTE ステートメント”	DBMS 固有の非クエリ SQL ステートメントの DBMS への送信	
“INSERT ステートメント”	行の追加	Ex. 1
“RESET ステートメント”	プロシジャを再開せずにプロシジャ環境に影響を与える、オプションのリセット	Ex. 5
“SELECT ステートメント”	行の選択と実行	
“UPDATE ステートメント”	値の変更	Ex. 3
“VALIDATE ステートメント”	クエリの正確さの検証	

PROC SQL ステートメント

PROC SQL ステートメント

構文

PROC SQL <*option(s)*>;

オプション引数の要約

Control execution

CONSTDATETIME | NOCONSTDATETIME
DQUOTE=ANSI|SAS
ERRORSTOP|NOERRORSTOP
EXEC|NOEXEC
EXITCODE
INOBS=*n*
IPASSTHRU|NOIPASSTHRU
IPONEATTEMPT | NOIPONEATTEMPT
LOOPS=*n*
NOCONSTDATETIME
NOERRORSTOP
NOEXEC
NOIPASSTHRU
NOIPONEATTEMPT
NOPROMPT
NOREMERGE
NOSTIMER
NOTHREADS
OUTOBS=*n*
PROMPT|NOPROMPT
REDUCEPUT=ALL|NONE|DBMS|BASE
REDUCEPUTOBS=*n*
REDUCEPUTVALUES=*n*
REMERGE|NOREMERGE
STIMER|NOSTIMER
STOPONTRUNC
THREADS|NOTHREADS
UNDO_POLICY=NONE|OPTIONAL|REQUIRED

Control output

DOUBLE|NODOUBLE
FEEDBACK|NOFEEDBACK
FLOW<=*n* <*m*>>|NOFLOW
NODOUBLE
NOFEEDBACK
NOFLOW
NONUMBER
NOPRINT
NOSORTMSG
NOWARNRECURS
NUMBER | NONUMBER
PRINT|NOPRINT
SORTMSG|NOSORTMSG

SORTSEQ=*sort-table* | LINGUISTIC
UBUFSIZE=*n* | *n*K | *n*M | *n*G
WARNRECURS|NOWARNRECURS

オプション引数

CONSTDATETIME | NOCONSTDATETIME
SQL プロシジャがクエリを実行する前に、クエリ内の DATE 関数、TIME 関数、DATETIME 関数、TODAY 関数への参照を等価の定数値で置き換えるかどうかを指定します。これらの値を一度計算すると、クエリで関数を複数回使用した場合、またはクエリが日付や時刻の境界近くで関数を実行した場合に結果の一貫性が保たれます。

NOCONSTDATETIME オプションを設定すると、PROC SQL は、オブザベーションを処理するたびにクエリ内のこれらの関数を評価します。

デフォルト CONSTDATETIME

操作 CONSTDATETIME オプションと REDUCEPUT=オプション (226 ページ)の両方が指定された場合、PROC SQL は、PUT 関数値を決定するために、クエリを実行する前に DATE 関数、TIME 関数、DATETIME 関数、TODAY 関数を、それぞれの値で置き換えます。

ヒント あるいは、SQLCONSTDATETIME システムオプションを設定することもできます。SQLCONSTDATETIME システムオプションで指定した値は、PROC SQL CONSTDATETIME オプションが設定されない限り、すべての SQL プロシジャステートメントに対して有効です。CONSTDATETIME オプションの値は、SQLCONSTDATETIME システムオプションよりも優先されます。RESET ステートメントを使用して、CONSTDATETIME オプションを設定またはリセットすることもできます。ただし、CONSTDATETIME オプションの値を変更しても、SQLCONSTDATETIME システムオプションの値は変更されません。詳細については、“SQLCONSTDATETIME System Option” (369 ページ)を参照してください。

DOUBLE|NODOUBLE
レポートの行間を 2 行に設定するかどうかを指定します。

デフォルト NODOUBLE

例 “例 5: 2 つのテーブルを組み合わせる” (276 ページ)

DQUOTE=ANSI|SAS
PROC SQL が二重引用符(" ")で囲まれた値を変数として扱うか文字列として扱うかを指定します。DQUOTE=ANSI を指定すると、PROC SQL は、二重引用符で囲まれた値を変数として扱います。この機能によって、次の値をテーブル名、列名またはエイリアスとして使用できるようになります。

- AS、JOIN、GROUP などの予約語。
- 通常は SAS で許可されない、DBMS 名などの名前。

二重引用符で囲まれた値には、任意の文字を含めることができます。

DQUOTE=SAS を指定すると、二重引用符で囲まれた値は文字列として扱われます。

デフォルト SAS

ERRORSTOP|NOERRORSTOP
PROC SQL でエラーが発生した場合に、PROC SQL が実行を停止するかどうかを指定します。バッチまたは非対話型セッションにおいて ERRORSTOP を指定すると、PROC SQL は、エラーが発生するとステートメントの実行を停止しますが、構文のチェックは続行します。

NOERRORSTOP を指定すると、PROC SQL は、エラーが発生した後もステートメントを実行し、構文のチェックを続行します。

デフォルト 対話的型 SAS セッションの場合は NOERRORSTOP、バッチまたは非対話型セッションの場合は ERRORSTOP。

操作 このオプションは、EXEC オプションが有効な場合にのみ有効です。

ヒント ERRORSTOP は、SAS がバッチまたは非対話型実行モードで実行されている場合にのみ有効です。

NOERRORSTOP は、エラーが発生した後もバッチジョブでの SQL プロシジャステートメントの実行を継続する場合に役立ちます。

EXEC|NOEXEC
構文の正確さがチェックされた後にステートメントを実行するかどうかを指定します。

デフォルト EXEC

ヒント NOEXEC は、ステートメントを実行しないで SQL ステートメントの構文を確認する場合に役立ちます。

参照項目 ERRORSTOP (223 ページ)

EXITCODE
PROC SQL がすべてのの SQL ステートメントのエラーコードをクリアするかどうかを指定します。エラーコードは SQLEXITCODE マクロ変数に割り当てられています。

デフォルト 0

ヒント 終了コードは、PROC SQL ステートメントの間で“RESET ステートメント” (251 ページ)を使用して、デフォルト値にリセットできます。

参照項目 “PROC SQL 自動マクロ変数の使用” (163 ページ)

FEEDBACK|NOFEEDBACK
ビューの参照が展開された後、または特定の他のステートメント変換が実行された後に、PROC SQL が SAS ログに PROC SQL ステートメントを表示するかどうかを指定します。

このオプションには次の効果があります。

- 任意のアスタリスク(たとえば、`SELECT *`)が、それが表す修飾された列のリストに展開されます。
- PROC SQL ビューが、その元になるクエリに展開されます。
- マクロ変数が解決されます。
- すべての式の前後にかっこが表示され、それらの式の評価順序が詳細に示されます。
- コメントが削除されます。

デフォルト NOFEEDBACK

FLOW<=*n* <*m*>>|NOFLOW
n 個よりも長い文字の列が複数行に改行されることを指定します。PROC SQL は、列の幅を *n* に設定し、*n* 個よりも長い文字の列を複数行に改行することを指定します。FLOW=*n m* を指定した場合、PROC SQL は、適切なレイアウトを得るために、これらの値の範囲内で列の幅を調整します。引数なしで FLOW を指定した場合は、FLOW=12 200 を指定した場合と同じになります。

デフォルト NOFLOW

INOBS=*n*
PROC SQL が 1 つのソースから検索する行(オブザベーション)の数を制限します。

ヒント INOBS=は、大きなテーブルに関するクエリのデバッグを行う場合に役立ちます。

IPASSTHRU|NOIPASSTHRU
暗黙のパススルーを有効にするか無効にするかを指定します。

PROC SQL を呼び出すときに、暗黙のパススルーが有効になります。1 つのクエリまたは一連のクエリに対して、これを無効にできます。暗黙のパススルーを無効にしたほうがよい場合の主な理由は次のとおりです。

- DBMS は SQL2 の NULL 値の意味を使用し、その動作は SAS の欠損値とはやや異なります。
- PROC SQL のクエリ最適化処理が改善する場合があります。

デフォルト IPASSTHRU

参照項目 *SAS/ACCESS for Relational Databases: Reference* の、使用している DBMS のパススルー機能についてのドキュメント。

IPONEATTEMPT | NOIPONEATTEMPT
暗黙的なパススルー要求が失敗した場合に、PROC SQL で SQL クエリによる処理の続行を許可するかどうかを指定します。

デフォルト NOIPONEATTEMPT

LOOPS=*n*
PROC SQL での内部ループの反復回数を *n* に制限します。ループ回数を知るには、それぞれの SQL ステートメントの実行後に SQLOOPS マクロ変数でレポートされる反復回数を使用します。制限を設定することによって、クエリがコンピュータリソースを過剰に使用しないようにします。たとえば、結合の一致条件を満たさない 3 つの大きなテーブルを結合しようとすると、巨大な内部テーブルが作成されて、実行の効率が悪くなる場合があります。

参照項目 “PROC SQL 自動マクロ変数の使用” (163 ページ)

NOCONSTDATETIME

参照項目 “CONSTDATETIME | NOCONSTDATETIME” (222 ページ)

NODOUBLE

参照項目 “DOUBLE|NODOUBLE” (222 ページ)

NOERRORSTOP

参照項目 “ERRORSTOP|NOERRORSTOP” (223 ページ)

NOEXEC

参照項目 “EXEC|NOEXEC” (223 ページ)

NOFEEDBACK

参照項目 “FEEDBACK|NOFEEDBACK” (223 ページ)

NOFLOW

参照項目 “FLOW<=*n* <*m*>>|NOFLOW” (224 ページ)

NOIPASSTHRU

参照項目 “IPASSTHRU|NOIPASSTHRU” (224 ページ)

NOIPONEATTEMPT

参照項目 “IPONEATTEMPT | NOIPONEATTEMPT” (224 ページ)

NONUMBER

参照項目 “NUMBER | NONUMBER” (225 ページ)

NOPRINT

参照項目 “PRINT|NOPRINT” (226 ページ)

NOPROMPT

参照項目 “PROMPT|NOPROMPT” (226 ページ)

NOREMERGE

参照項目 “REMERGE|NOREMERGE” (228 ページ)

NOSORTMSG

参照項目 “SORTMSG|NOSORTMSG” (228 ページ)

NOSTIMER

参照項目 “STIMER|NOSTIMER” (229 ページ)

NOTHREADS

参照項目 “THREADS|NOTHREADS” (229 ページ)

NOWARNRECURS

参照項目 “WARNRECURS|NOWARNRECURS” (231 ページ)

NUMBER | NONUMBER

SELECT ステートメントが ROW と呼ばれる列を含むかどうかを指定します。ROW は、行を検索するときのデータの行(またはオブザベーション)番号です。

デフォルト NONUMBER

例 “例 4: 2 つのテーブルを結合する” (273 ページ)

OUTOBS=*n*
出力の行(オブザベーション)の数を制限します。たとえば、OUTOBS=10 を指定し、クエリ式を使用して値をテーブルに挿入した場合、SQL プロシジャは最大で 10 行を挿入します。同様に、OUTOBS=10 によって出力は 10 行に制限されます。

PRINT|NOPRINT
SELECT ステートメントの結果を出力するかどうかを指定します。

デフォルト PRINT

操作 NOPRINT は、SQLOBS 自動マクロ変数の値に影響を与えます。詳細については、“PROC SQL 自動マクロ変数の使用” (163 ページ)を参照してください。

ヒント NOPRINT は、テーブルから値を選択してそれをマクロ変数に入力し、何も表示しないようにする場合に役立ちます。

PROMPT|NOPROMPT
INOBS=オプション、OUTOBS=オプションおよび LOOPS=オプションの影響を変更します。PROMPT オプションを指定した場合、INOBS=、OUTOBS=または LOOPS=で指定した制限に達すると、PROC SQL は、停止または続行を選択するためのプロンプトを表示します。プロンプトは、同じ上限に達するたびに繰り返し表示されます。

デフォルト NOPROMPT

REDUCEPUT=ALL|NONE|DBMS|BASE
クエリでの PUT 関数の最適化に使用されるエンジンタイプを指定します。PUT 関数は、論理的に等価な式で置き換えられます。エンジンタイプには、次のいずれかの値を指定できます。

ALL
すべての PUT 関数の最適化を考慮することを指定します。これは、クエリがデータのアクセスにどのエンジンを使用するかにかかわりません。

NONE
どの PUT 関数も最適化しないことを指定します。

DBMS
SAS/ACCESS エンジンが実行するクエリ内のすべての PUT 関数について最適化を考慮することを指定します。

要件 PUT 関数に渡す最初の引数は、テーブルから取得される変数である必要があります。テーブルには、SAS/ACCESS エンジンを使用してアクセスする必要があります。

BASE
SAS/ACCESS エンジンまたは Base SAS エンジンが実行するクエリ内のすべての PUT 関数について最適化を考慮することを指定します。

デフォルト DBMS

操作 REDUCEPUT=オプションと CONSTDATETIME オプションの両方が指定された場合、PROC SQL は、PUT 関数値を決定するために、クエリを実行する前に DATE 関数、TIME 関数、DATETIME 関数、TODAY 関数をそれぞれの値で置き換えます。

クエリに WHERE 句または HAVING 句も含まれる場合、WHERE 句または HAVING 句の評価が簡略化されます。

ヒント あるいは、SQLREDUCEPUT=システムオプションを設定することもできます。SQLREDUCEPUT=システムオプションで指定した値は、REDUCEPUT=オプションが設定されない限り、すべての SQL プロシジャステートメントに対して有効です。REDUCEPUT=オプションの値は、SQLREDUCEPUT=システムオプションよりも優先されます。RESET ステートメントを使用して、REDUCEPUT=オプションを設定またはリセットすることもできます。ただし、REDUCEPUT=オプションの値を変更しても、SQLREDUCEPUT=システムオプションの値は変更されません。詳細については、“SQLREDUCEPUT=システムオプション” (374 ページ)を参照してください。

REDUCEPUTOBS=*n*

REDUCEPUT=オプションを DBMS、BASE または ALL に設定した場合、PROC SQL がクエリ内の PUT 関数の最適化を考慮するためにテーブルに存在する必要のあるオブザベーションの最小数を指定します。

デフォルト 0。これは、PROC SQL が PUT 関数を最適化するために使用するテーブル内のオブザベーションの最小数がないことを示します。

範囲 0 から 2^{63}–1(約 9.2x10 の 18 乗)まで

要件 *n* は整数である必要があります。

操作 REDUCEPUTOBS=オプションは、テーブル内のオブザベーションの数を記録する DBMS に対してのみ動作します。使用している DBMS がオブザベーションの数を記録しなくても、テーブルの行カウントを作成すれば、REDUCEPUTOBS=オプションは動作します。

ヒント あるいは、SQLREDUCEPUTOBS=システムオプションを設定することもできます。SQLREDUCEPUTOBS=システムオプションで指定した値は、REDUCEPUTOBS=オプションが設定されない限り、すべての SQL プロシジャステートメントに対して有効です。REDUCEPUTOBS=オプションの値は、SQLREDUCEPUTOBS=システムオプションよりも優先されます。RESET ステートメントを使用して、REDUCEPUTOBS=オプションを設定またはリセットすることもできます。ただし、REDUCEPUTOBS=オプションの値を変更しても、SQLREDUCEPUTOBS=システムオプションの値は変更されません。詳細については、“SQLREDUCEPUTOBS=システムオプション” (376 ページ)を参照してください。

REDUCEPUTVALUES=*n*

REDUCEPUT=オプションを DBMS、BASE または ALL に設定した場合、PROC SQL がクエリ内の PUT 関数の最適化を考慮するために PUT 関数式に存在できる SAS 出力形式値の最大数を指定します。

デフォルト 100

範囲 100 から 3,000 まで

要件 *n* は、整数である必要があります。

操作 PUT 関数式内の SAS 出力形式値の数がこの値を超えた場合、PROC SQL は PUT 関数を最適化しません。

ヒント あるいは、SQLREDUCEPUTVALUES=システムオプションを設定することもできます。SQLREDUCEPUTVALUES=システムオプションで指定した値は、REDUCEPUTVALUES=オプションが設定されない限り、すべての SQL プロシジャステートメントに対して有効です。REDUCEPUTVALUES=オプションの値は、SQLREDUCEPUTVALUES=システムオプションよりも優先されます。RESET ステートメントを使用して、REDUCEPUTVALUES=オプションを設定またはリセットすることもできます。ただし、REDUCEPUTVALUES=オプションの値を変更しても、SQLREDUCEPUTVALUES=システムオプションの値は変更されません。詳細については、“SQLREDUCEPUTVALUES=システムオプション” (377 ページ)を参照してください。

REDUCEPUTVALUES=の値は、個別の最適化で使用されます。たとえば、WHERE 句に PUT 関数が存在し、SELECT ステートメントには別の PUT 関数が存在し、値を含む両方の関数にユーザー定義の出力形式が設定されている場合、REDUCEPUTVALUES=の値が句とステートメントで別々に適用されます。

REMERGE|NOREMERGE

データの再マージを使用するクエリを PROC SQL が処理できるかどうかを指定します。PROC SQL の再マージ機能は、クエリを完了するために、テーブルのパススルーを 2 回実行します。つまり、最初のパススルーで作成されたデータを、2 番目のパススルーで使用します。NOREMERGE システムオプションが設定されると、PROC SQL はデータの再マージを処理できません。NOREMERG オプションを設定して再マージを実行しようとすると、エラーが SAS ログに書き込まれます。

デフォルト REMERGE

ヒント あるいは、SQLREMERGE システムオプションを設定することもできます。SQLREMERGE システムオプションで指定した値は、PROC SQL REMERGE オプションが設定されない限り、すべての SQL プロシジャステートメントに対して有効です。REMERGE オプションの値は、SQLREMERGE システムオプションよりも優先されます。RESET ステートメントを使用して、REMERGE オプションを設定またはリセットすることもできます。ただし、REMERGE オプションの値を変更しても、SQLREMERGE システムオプションの値は変更されません。詳細については、“SQLREMERGE システムオプション” (378 ページ)を参照してください。

参照項目 “データの再マージ” (359 ページ)

SORTMSG|NOSORTMSG

PROC SORT を使用してテーブルを内部的に並べ替えることができる ORDER BY のような特定の操作の場合、並べ替えに関する情報が PROC SORT から要求されるかどうか、およびその情報がログに表示されるかどうかを指定します。

デフォルト NOSORTMSG

SORTSEQ=*sort-table* | LINGUISTIC
クエリに ORDER BY 句が含まれる場合に使用される照合順序を指定します。このオプションは、システムまたはインストールの、デフォルトの照合順序以外の照合順序が必要な場合にのみ使用します。

sort-table では、PROC TRANTAB で作成した変換テーブルが示されます。

LINGUISTIC では、照合順序がセッションロケールにより決定されることと、デフォルト照合オプションが使用されたことが示されます。SORTSEQ システムオプションに対して LINGUISTIC の値が指定された場合、PROC SQL はその設定に従います。ただし、SORTSEQ プロシジャオプションの設定は、SORTSEQ システムオプションの設定よりも優先されます。

参照項目 “SORTSEQ= System Option: UNIX, Windows, and z/OS” (*SAS National Language Support (NLS): Reference Guide*)

“Specifying Linguistic Collation” (*SAS National Language Support (NLS): Reference Guide*)

STIMER|NOSTIMER
PROC SQL が、タイミング情報を、プロシジャ全体の累積値ではなくステートメントごとに SAS ログに書き込むかどうかを指定します。このオプションが動作するには、SAS システムオプションの STIMER も指定する必要があります。オペレーティングシステムによっては、SAS を呼出すときにこのシステムオプションを指定する必要があります。このシステムオプションを単独で使用すると、ステートメントごとのタイミング情報ではなく、SQL プロシジャ全体のタイミング情報が表示されます。

デフォルト NOSTIMER

STOPONTRUNC
切り捨てエラーが発生した場合に、列のサイズよりも大きいデータを含む行を挿入または更新しないことを指定します。これは、INSERT ステートメントまたは UPDATE ステートメントで SET 句を使用した場合にのみ適用されます。

THREADS|NOTHREADS
システムオプションが制限されていない限り、特定の PROC SQL の呼び出しに対して、SAS システムオプションの THREADS|NOTHREADS をオーバーライドします。(制限事項を参照。)特定のクエリで使用するために、THREADS|NOTHREADS を RESET ステートメントで指定することもできます。THREADS を指定すると、PROC SQL は、大量のデータを伴う並べ替え動作のパフォーマンス向上を目的として、並列処理を使用します。並列処理の詳細については、*SAS 言語リファレンス: 解説編*を参照してください。

デフォルト SAS システムオプションの THREADS|NOTHREADS の値。

制限事項 サイト管理者は、制限対象オプションテーブルを作成できます。制限対象オプションテーブルは、起動時に構築される SAS システムオプション値を指定します。この値は、オーバーライドできません。THREADS | NOTHREADS システムオプションを制限対象オプションテーブルに記述すると、このオプションを設定するどのような試みも無視され、警告メッセージが SAS ログに書き込まれます。

操作 PROC SQL ステートメントまたは RESET ステートメントで THREADS|NOTHREADS が指定された場合、この PROC SQL の呼び出しでは、この

オプションをデフォルト(つまり、SAS システムオプションの THREADS|NOTHREADS の値)にリセットする方法はありません。

参照項目 “THREADS System Option” (*SAS System Options: Reference*)

UNDO_POLICY=NONE|OPTIONAL|REQUIRED
データの更新中にエラーが発生した場合に PROC SQL が更新済みデータをどう処理するかを指定します。UNDO_POLICY=を使用して、変更を永続化するかどうかを制御できます。

NONE
すべての更新または挿入を維持します。

OPTIONAL
確実に破棄できるすべての更新または挿入を破棄します。

REQUIRED
エラー発生時点までに実行されたすべての挿入または更新を破棄します。場合によっては、確実に UNDO 操作を実行できるとは限りません。たとえば、プログラムが SAS/ACCESS ビューを使用する場合、同時に行われた他の変更による影響を破棄しなければ、INSERT ステートメントおよび UPDATE ステートメントによる影響を破棄できないことがあります。その場合、PROC SQL はエラーメッセージを発行し、ステートメントを実行しません。また、SAS データセットが SAS/SHARE サーバーを介してアクセスされ、CNTLLEV=RECORD データセットオプションを使用して開かれた場合、変更を確実に破棄することはできません。

このオプションは、他のユーザーが新しく挿入された行を更新することを許可します。挿入中にエラーが発生した場合、PROC SQL は、別のユーザーが更新したレコードを削除できます。その場合、ステートメントは実行されず、エラーメッセージが発行されます。

デフォルト REQUIRED

ヒント SPD エンジンを使用してデータセットを更新している場合、UNDO_POLICY=NONE に設定することによって処理のパフォーマンスを大幅に向上できます。ただし、NONE がアプリケーションに適した設定であることを確認してください。

あるいは、SQLUNDOPOLICY システムオプションを設定することもできます。SQLUNDOPOLICY=システムオプションで指定した値は、PROC SQL の UNDO_POLICY=オプションが設定されない限り、すべての SQL プロシジャステートメントに対して有効です。UNDO_POLICY=オプションの値は、SQLUNDOPOLICY=システムオプションよりも優先されます。RESET ステートメントを使用して、UNDO_POLICY=オプションを設定またはリセットすることもできます。ただし、UNDO_POLICY=オプションの値を変更しても、SQLUNDOPOLICY=システムオプションの値は変更されません。プロシジャの完了後は、SQLUNDOPOLICY=システムオプションの値に戻ります。詳細については、“SQLUNDOPOLICY=システムオプション” (379 ページ)を参照してください。

UBUFSIZE=*n* | *n*K | *n*M | *n*G
PROC SQL のページ化されたメモリサブシステムにおける内部的な一時バッファのページサイズを指定します。PROC SQL は、このサブシステムを使用して、結合、集計、交差などの操作を実装します。出力は、1(バイト)、1024(キロバイト)、1,048,576(メガバイト),、1,073,741,824(ギガバイト)のうちのいずれかの倍数で表さ

れます。たとえば、値 `65536` は 65536 バイトのページサイズを指定します。同じく、値 `64k` も 65536 バイトのページサイズを指定します 。

UBUFSIZE を、特定のクエリで使用するために RESET ステートメントで指定することもできます。

注: SAS 9.4 より前のバージョンで使用されていた BUFFERSIZE オプションは、UBUFSIZE オプションと同じ機能を持つオプションであり、引き続きサポートされます。SAS 9.4 では、UBUFSIZE オプションの方が優先されます。

デフォルト 0。これを指定すると、SAS はオペレーティングシステムの最小最適ページサイズを使用します。

WARNRECURS|NOWARNRECURS
再帰的参照に対する警告を SAS ログに表示するかどうかを指定します。

NOWARNRECURS は、再帰的参照を、警告のかわりに注として SAS ログに表示することを指定します。

デフォルト WARNRECURS

詳細

注: PROC SQL ステートメントの間で“RESET ステートメント” (251 ページ)を使用して、オプションを追加、削除、または変更できます。

ALTER TABLE ステートメント

既存のテーブルの列の追加、列の削除および列属性の変更を実行します。既存のテーブルの一貫性制約を追加、変更および削除します。

制限事項: ALTER TABLE ステートメントでは、どのタイプのビューも使用できません。

UPDATE 処理をサポートしないエンジンによってアクセスされたテーブルに対しては、ALTER TABLE を使用できません。

ALTER TABLE ステートメントでは、少なくとも 1 つの ADD 句、DROP 句または MODIFY 句を使用する必要があります。

参照項目: “例 3: PROC SQL テーブルのデータの更新” (271 ページ)

構文

ALTER TABLE *table-name*

<ADD *column-definition-1* <, *column-definition-2*, …>>

<ADD CONSTRAINT *constraint-name-1 constraint-specification-1* <, *constraint-name-2 constraint-specification-2*, …>>

<ADD *constraint-specification-1* <, *constraint-specification-2*, …>>

<DROP *column-1* <, *column-2*, …>>

<DROP CONSTRAINT *constraint-name-1* <, *constraint-name-2*, …>>

<DROP FOREIGN KEY *constraint-name*>

<DROP PRIMARY KEY>

<MODIFY *column-definition-1* <, *column-definition-2*, …>>

;

必須引数

table-name

- ALTER TABLE ステートメントでは、変更対象のテーブルの名前を参照します。
- REFERENCES 句では、外部キーが参照する主キーを含むテーブルの名前を参照します。

table-name としては、1 レベルの名前、2 レベルの *libref.table* の名前または単一引用符で囲まれた物理パス名が可能です。

オプション引数

<ADD *column-definition-1*<, *column-definition-2*, …>>
それぞれの column-definition で指定された、1 つ以上の列を追加します。

参照項目 “column-definition” (316 ページ)

<ADD CONSTRAINT *constraint-name-1 constraint-specification-1*<, *constraint-name-2 constraint-specification-2*, …>>
constraint-specification で指定された一貫性制約を追加し、それに *constraint-name* を割り当てます。

参照項目 “*constraint-name*” (232 ページ)

“*constraint-specification*” (232 ページ)

constraint-name
指定している制約の名前を指定します。制約名は、有効な SAS 名でなければなりません。

注: *constraint-name* の値として、PRIMARY、FOREIGN、MESSAGE、UNIQUE、DISTINCT、CHECK、NOT という名前は使用できません。

constraint-specification
制約の定義であり、次の形式で指定します。

constraint <MESSAGE='*message-string*' <MSGTYPE=*message-type*>>

constraint
これは、次のいずれかの一貫性制約です。

CHECK *(WHERE-clause)*
table-name 内のすべての行が *WHERE-clause* の条件を満たすことを指定します。*WHERE-clause* には、SAS WHERE 句を指定します。WHERE キーワードを WHERE 句に含めないでください。

DISTINCT (*column-1*<, *column-2*, …>)
各 *column* の値がユニークである必要があることを指定します。この制約は、UNIQUE と同じです。

FOREIGN KEY (*column-1*<, *column-2*,>) REFERENCES *table-name* <ON DELETE *referential-action*> <ON UPDATE *referential-action*>
外部キー、つまり別のテーブル(REFERENCES に対して指定された *table-name*)の主キー変数の値にリンクされた値を持つ一連の *columns*。*referential-actions* は、外部キーから参照される主キー列の値が更新または削除された場合に実行されます。

column には、*table-name* 内の列名を指定します。

referential-action には、次のアクションを指定できます。

CASCADE
主キーのデータ値の更新を可能にし、それに対応する外部キーの値を同じ値に更新します。参照アクションは、現在更新に対してのみサポートされています。

RESTRICT
外部キーに一致する場合は、主キーのデータ値が更新または削除されないようにします。この参照アクションはデフォルトです。

SET NULL
主キーのデータ値の更新を可能にし、更新された値に一致するすべての外部キーの値を NULL に設定します。

制限事項 主キー制約と外部キー制約を重複して定義した場合、データファイル内の変数は主キー定義と外部キー定義の両方に含まれます。全く同じ変数を使用する場合、それらの変数は異なる順序で定義する必要があります。外部キーの更新および削除の参照アクションは、両方とも RESTRICT である必要があります。

参照項目 "*table-name*" (232 ページ)

NOT NULL (*column*)
column に、NULL または特殊欠損値などの欠損値が含まれないことを指定します。

PRIMARY KEY (*column-1* <, *column-2*, …>)
1 つ以上の主キー列、つまり欠損値を含まず、ユニークな値を含む列を指定します。

制限事項 主キー制約と外部キー制約を重複して定義した場合、データファイル内の変数は主キー定義と外部キー定義の両方に含まれます。全く同じ変数を使用する場合、それらの変数は異なる順序で定義する必要があります。

注 新しい主キー制約の定義に使用される変数に NOT NULL 制約が存在する場合、その新しい主キーの制約は既存の主キーの制約を置き換えます。

主キー制約で既に定義されている変数に NOT NULL 制約を定義するよう試みた場合、その NOT NULL 制約の定義は失敗します。

UNIQUE (*column-1* <, *column-2*, …>)
各 *column* の値がユニークである必要があることを指定します。この制約は、DISTINCT と同じです。

message-string
一貫性制約が満たされない場合にログに書き込まれるエラーメッセージのテキストを指定します。*message-string* の最大長は、250 文字です。

message-type
一貫性制約が満たされない場合の、SAS ログでのエラーメッセージの表示方法を指定します。*message-type* には、次の値のいずれかを指定できます。

NEWLINE
MESSAGE=に対して指定したテキストが、一貫性制約のデフォルトのエラーメッセージと共に表示されます。

USER
MESSAGE=に対して指定したテキストのみが表示されます。

<ADD *constraint-specification-1*< , *constraint-specification-2*, …>>
constraint-specification で指定された一貫性制約を追加し、それにデフォルト名を割り当てます。デフォルトの制約名には、次の表に示す形式があります。

デフォルト名	制約タイプ
NM*xxxx*	NOT NULL
UN*xxxx*	UNIQUE
CK*xxxx*	CHECK
PK*xxxx*	PRIMARY KEY
FK*xxxx*	FOREIGN KEY

これらのデフォルト名において、*xxxx* は、0001 から始まるカウンタです。

参照項目 “*constraint-specification*” (232 ページ)

<DROP *column-1*< , *column-2*, …>>
それぞれの *column* をテーブルから削除します。*column* には、*table-name* 内の列名を指定します。

<DROP CONSTRAINT *constraint-name-1*<, *constraint-name-2* ,…>>
それぞれの *constraint-name* が参照する一貫性制約を削除します。一貫性制約の名前を検索するには、DESCRIBE TABLE CONSTRAINTS 句を使用します。(“DESCRIBE ステートメント” (246 ページ)を参照してください。)

参照項目 “*constraint-name*” (232 ページ)

<DROP FOREIGN KEY *constraint-name*>
constraint-name が参照する外部キー制約を削除します。

注: DROP FOREIGN KEY 句は、DB2 拡張です。

参照項目 “*constraint-name*” (232 ページ)

<DROP PRIMARY KEY>
table-name から主キー制約を削除します。

注: DROP PRIMARY KEY 句は、DB2 拡張です。

<MODIFY *column-definition-1*< , *column-definition-2*, …>>
それぞれの column-definition で指定した 1 つ以上の列の属性を変更します。

参照項目 “column-definition” (316 ページ)

詳細

新しい列の初期値の指定

ALTER TABLE ステートメントによってテーブルに列を追加すると、テーブルのすべての行において、その列の値は欠損値に初期化されます。UPDATE ステートメントを使用して、1 つ以上の新しい列に値を追加します。

列の属性の変更

列がすでにテーブルに存在する場合、MODIFY 句を使用して、長さ、入力形式、出力形式、ラベルの各列属性を変更できます。テーブル内の値は、指定された長さ属性に合うように、必要に応じて(文字データの場合)空白が切り捨てられるか、空白が埋め込まれます。

文字列から数値列(およびその逆)へは変更できません。列のデータタイプを変更するには、列を削除してから、列(およびそのデータ)を再び追加するか、DATA ステップを使用します。

注: 数値列の長さは、ALTER TABLE ステートメントを使用して変更できません。かわりに、DATA ステップを使用してください。

列名の変更

ALTER TABLE ステートメントでは、RENAME=データセットオプションを使用して列名を変更できません。ただし、CREATE TABLE ステートメントまたは SELECT ステートメントでは、RENAME=データセットオプションを使用できます。RENAME=データセットオプションの詳細については、*SAS データセットオプション: リファレンス* の SAS データセットオプションのセクションを参照してください。

変更された列に対するインデックスの作成

列の属性を変更するときに、その列にインデックスが定義されていれば、変更された列の値にも引き続きインデックスが定義されます。ALTER TABLE ステートメントを使用して列を削除した場合、その列が関与するすべての(単一および複合)インデックスも削除されます。インデックスの作成と使用の詳細については、"CREATE INDEX ステートメント" (236 ページ)を参照してください。

一貫性制約

既存のテーブルの一貫性制約を変更するには、ALTER TABLE を使用します。新しいテーブルに一貫性制約を付加するには、CREATE TABLE ステートメントを使用します。一貫性制約の詳細については、*SAS 言語リファレンス: 解説編* の SAS ファイルのセクションを参照してください。

CONNECT ステートメント

SAS/ACCESS ソフトウェアがサポートする DBMS との接続を確立します。

要件 SAS/ACCESS ソフトウェアが必要です。このステートメントの詳細については、SAS/ACCESS のドキュメントを参照してください。

参照項目: "パススルー機能を使用した DBMS への接続" (172 ページ)

構文

CONNECT TO *dbms-name* <AS *alias*>
<(*connect-statement-argument-1*=*value-1* <*connect-statement-argument-2*=*value-2* …>)>
<(*database-connection-argument-1*=*value-1* <*database-connection-argument-2*=*value-2* …>)>;

CONNECT USING *libref* <AS *alias*>;

必須引数

dbms-name
接続先の DBMS を指定します(たとえば、ORACLE や DB2)。

libref
LIBNAME ステートメントで DBMS 接続がすでに確立されているライブラリ参照名を指定します。ライブラリ参照名の接続パラメータは、CONNECT ステートメントを使用して SQL プロシジャ内で再利用できます。

オプション引数

AS *alias*
1 文字から 32 文字までのエイリアスを指定します。*alias* の前に AS キーワードを記述する必要があります。一部の DBMS では、複数の接続が可能です。任意の AS 句を使用して、後で参照するために接続に名前を付けることができます。

connect-statement-argument=value
データベースへの複数の接続、共有される接続、一意の接続などを作成できるかどうかを示す引数の値を指定します。これらの引数は任意ですが、含める場合はかっこで囲む必要があります。これらの引数の詳細については、*SAS/ACCESS for Relational Databases: Reference* を参照してください。

database-connection-argument=value
DBMS に接続するために PROC SQL で必要となる、DBMS 固有の引数の値を指定します。このれらの引数は、ほんとんどのデータベースで任意ですが、含める場合はかっこで囲む必要があります。詳細は、使用している DBMS 用の SAS/ACCESS のマニュアルを参照してください。

CREATE INDEX ステートメント

テーブルの列にインデックスを作成します。

制限事項: CREATE INDEX は、UPDATE 処理をサポートしないエンジンによってアクセスされたテーブルに対しては使用できません。

構文

CREATE <UNIQUE> **INDEX** *index-name*
ON *table-name* (*column-1* <, *column-2*, …>);

必須引数

INDEX *index-name*
作成するインデックスに名前を付けます。1 つの列にのみインデックスを作成する場合は、*index-name* は *column* と同じである必要があります。複数の列にインデックスを作成する場合は、*index-name* をテーブル内のどの列とも同じ名前にすることはできません。

ON *table-name*
PROC SQL テーブルを指定します。

オプション引数

UNIQUE
同じインデックス値を持つ複数の行を作成する可能性のあるテーブルに対するあらゆる変更を、SAS が拒否するようにします。インデックスを一意にすることで、1 つの列または複合グループの列のデータが、テーブル内のすべての行に対して一意であり続けることが保証されます。各行が一意のインデックス値を持つ場合、NULL 値または欠損値を含む列に対して一意のインデックスを定義できます。

列
table-name の列を指定します。

詳細

PROC SQL でのインデックス作成

インデックスには、テーブルの列の値と、インデックス値によってテーブル内の行にアクセスできるシステムの方向が格納されます。1 または複数の列にインデックスを定義することで、SAS は、特定の状況において、より素早く効率的にテーブル内の行を見つけることができます。インデックスによって、PROC SQL は、次の種類のクエリをより効率的に実行できるようになります。

- インデックスが作成された列に対する比較。
- IN サブクエリ。内部のサブクエリの列にインデックスを作成。
- 相関サブクエリ。相関参照を使用して比較される列にインデックスを作成。
- 結合クエリ。結合式は等号比較とし、結合式内のすべての列について、結合対象テーブルのいずれかでインデックスを作成。

SAS は、変更が PROC SQL または他のソースのいずれから発生した場合でも、テーブルのすべての変更に対してインデックスを維持します。したがって、列の定義を変更したり、列の値を更新した場合、列には引き続き同じインデックスが定義されます。ただし、インデックスが作成された列をテーブルから削除した場合は、そのインデックスも削除されます。

インデックスには、単一インデックスと複合インデックスがあります。*単一インデックス*は、テーブル内の 1 つの列に対して作成されます。単一インデックスには、その列と同じ名前を付ける必要があります。*複合インデックス*は、2 つ以上の列に対して定義された 1 つのインデックス名です 列の順序は任意に指定できます。各列のデータ型を一致させる必要はありません。複合インデックスには、テーブル内の列と同じ名前を付けることはできません。複合インデックスを削除した場合、その複合インデックスに含まれるすべての列について、インデックスが削除されます。

インデックスの管理

DATASETS プロシジャで CONTENTS ステートメントを使用して、テーブルのインデックス名と、それらが定義されている列を表示できます。DICTIONARY テーブルの INDEXS、TABLES および COLUMNS を使用して、インデックスに関する情報を表示することもできます。詳細については、“DICTIONARY テーブルを使用し、SAS System の情報にアクセスする” (151 ページ)を参照してください。

どのような場合にインデックスを使用するかについての詳細説明と、インデックスが BY グループ処理を扱う SAS ステートメントに与える影響については、*SAS 言語リファレンス: 解説編*の SAS ファイルのセクションを参照してください。

CREATE TABLE ステートメント

PROC SQL テーブルを作成します。

参照項目: “例 1: テーブルを作成し、データを挿入する” (267 ページ)
“例 2: テーブルをクエリの結果から作成する” (269 ページ)

構文

CREATE TABLE *table-name*
(*column-specification-1* <, *column-specification-2* | *constraint-specification-1*, …>);

CREATE TABLE *table-name* **LIKE** *table-name-2*;

CREATE TABLE *table-name* **AS** *query-expression*
<ORDER BY *order-by-item-1* <, *order-by-item-2*, …>>;

必須引数

table-name

- CREATE TABLE ステートメントでは、作成されるテーブルの名前を参照します。*table-name* の直後のかっこ内にデータセットオプションを配置して、それらを使用できます。詳細については、“PROC SQL で SAS データセットオプションを使用する” (157 ページ)を参照してください。
- REFERENCES 句では、外部キーが参照する主キーを含むテーブルの名前を参照します。

column-specification

列の定義と、その列の制約(オプション)を指定します。次の形式を使用します。

column-definition <column-constraint>

column-definition

“column-definition” (316 ページ)を参照してください。

column-constraint

次のいずれかを指定できます。

CHECK (*WHERE-clause*)

table-name 内のすべての行が *WHERE-clause* の条件を満たすことを指定します。*WHERE-clause* には、SAS WHERE 句を指定します。WHERE キーワードを WHERE 句に含めないでください。

DISTINCT

列の値がユニークである必要があることを指定します。この制約は、UNIQUE と同じです。

NOT NULL

列に、NULL 値や、特殊欠損値などの欠損値が含まれないことを指定します。

PRIMARY KEY

列が主キー列、つまり、欠損値を含まず一意の値を含む列であることを指定します。

制限事項 主キー制約と外部キー制約を重複して定義した場合、データファイル内の変数は主キー定義と外部キー定義の両方に含まれます。全く同じ変数を使用する場合、それらの変数は異なる順序で定義する必要があります。

注 新しい主キー制約の定義に使用される変数に NOT NULL 制約が存在する場合、その新しい主キーの制約は既存の主キーの制約を置き換えます。

主キー制約で既に定義されている変数に NOT NULL 制約を定義するよう試みた場合、その NOT NULL 制約の定義は失敗します。

REFERENCES *table-name* **<ON DELETE** *referential-action***> <ON UPDATE** *referential-action***>**
列が外部キー、つまり別のテーブル(REFERENCES で指定した *table-name*)の主キー変数の値にリンクされた値を持つ列であることを指定します。*referential-actions* は、外部キーから参照される主キー列の値が更新または削除された場合に実行されます。

referential-action には、次のアクションを指定できます。

CASCADE
主キーのデータ値の更新を可能にし、それに対応する外部キーの値を同じ値に更新します。参照アクションは、現在更新に対してのみサポートされています。

RESTRICT
一致する外部キーの値が存在する場合にのみ発生します。この参照アクションはデフォルトです。

SET NULL
一致するすべての外部キーの値を NULL に設定します。

制限事項 主キー制約と外部キー制約を重複して定義した場合、データファイル内の変数は主キー定義と外部キー定義の両方に含まれます。全く同じ変数を使用する場合、それらの変数は異なる順序で定義する必要があります。外部キーの更新および削除の参照アクションは、両方とも RESTRICT である必要があります。

UNIQUE
列の値がユニークである必要があることを指定します。この制約は、DISTINCT と同じです。

注: *column-constraint* を指定した場合、SAS は自動的に制約に名前を割り当てます。制約名には、次の形式があります。ここで、*xxxx* は 0001 から始まるカウンタです。

デフォルト名	制約タイプ
CK*xxxx*	CHECK
FK*xxxx*	FOREIGN KEY
NM*xxxx*	NOT NULL
PK*xxxx*	PRIMARY KEY
UN*xxxx*	UNIQUE

table-name-2
table-name2 と同じ列名および列属性を持つ *table-name* を作成します。ただし、行は含まれません。

query-expression
クエリの結果から *table-name* を作成します。

参照項目 “query-expression” (339 ページ)

オプション引数

constraint-specification

1 つまたは複数の列に関する制約を次の形式で指定します。

CONSTRAINT *constraint-name constraint* <MESSAGE='*message-string*' <MSGTYPE=*message-type*>>

constraint-name

指定している制約の名前を指定します。制約名は、有効な SAS 名でなければなりません。

注: constraint-name の値として、PRIMARY、FOREIGN、MESSAGE、UNIQUE、DISTINCT、CHECK、NOT という名前は使用できません。

constraint

次のいずれかの制約タイプを指定します。

CHECK ***WHERE-clause***

table-name 内のすべての行が *WHERE-clause* の条件を満たすことを指定します。*WHERE-clause* には、SAS WHERE 句を指定します。WHERE キーワードを WHERE 句に含めないでください。

DISTINCT (*column-1* <, *column-2*, …>)

各 *column* の値がユニークである必要があることを指定します。この制約は、UNIQUE と同じです。

FOREIGN KEY (*column-1* <, *column-2*, …>)
REFERENCES *table-name* <ON DELETE *referential-action*> <ON UPDATE *referential-action*>

外部キー、つまり別のテーブル(REFERENCES に対して指定された *table-name*)の主キー変数の値にリンクされた値を持つ一連の *columns*。*referential-actions* は、外部キーから参照される主キー列の値が更新または削除された場合に実行されます。*referential-actions* には、次のアクションを指定できます。

CASCADE

主キーのデータ値の更新を可能にし、それに対応する外部キーの値を同じ値に更新します。参照アクションは、現在更新に対してのみサポートされています。

RESTRICT

一致する外部キーの値が存在する場合にのみ発生します。この参照アクションはデフォルトです。

SET NULL

一致するすべての外部キーの値を NULL に設定します。

制限事項

主キー制約と外部キー制約を重複して定義した場合、データファイル内の変数は主キー定義と外部キー定義の両方に含まれます。全く同じ変数を使用する場合、それらの変数は異なる順序で定義する必要があります。外部キーの更新および削除の参照アクションは、両方とも RESTRICT である必要があります。

NOT NULL (*column*)

column に、NULL または特殊欠損値などの欠損値が含まれないことを指定します。

PRIMARY KEY (*column-1* <, *column-2*, …>)

1 つ以上の主キー列、つまり欠損値を含まず、ユニークな値を含む列を指定します。

制限事項 主キー制約と外部キー制約を重複して定義した場合、データファイル内の変数は主キー定義と外部キー定義の両方に含まれます。全く同じ変数を使用する場合、それらの変数は異なる順序で定義する必要があります。

注 新しい主キー制約の定義に使用される変数に NOT NULL 制約が存在する場合、その新しい主キーの制約は既存の主キーの制約を置き換えます。

主キー制約で既に定義されている変数に NOT NULL 制約を定義するよう試みた場合、その NOT NULL 制約の定義は失敗します。

UNIQUE (*column-1* <, *column-2*, …>)
各 *column* の値がユニークである必要があることを指定します。この制約は、DISTINCT と同じです。

message-string
一貫性制約が満たされない場合にログに書き込まれるエラーメッセージのテキストを指定します。*message-string* の最大長は、250 文字です。

message-type
一貫性制約が満たされない場合の、SAS ログでのエラーメッセージの表示方法を指定します。

NEWLINE
MESSAGE=に対して指定したテキストが、一貫性制約のデフォルトのエラーメッセージと共に表示されます。

USER
MESSAGE=に対して指定したテキストのみが表示されます。

ORDER BY *order-by-item-1* <, *order-by-item-2*, …>
それぞれの *order-by-item* の値で *table-name* 内の行を並べ替えます。

参照項目 “ORDER BY 句” (263 ページ)

詳細

行を含まないテーブルの作成

- CREATE TABLE ステートメントの最初のフォームは、SAS がサポートするテーブルに自動的に SQL データタイプをマッピングするテーブルを作成します。既存のテーブルに存在しない列を含む新しいテーブルを作成する場合、このフォームを使用します。これは、別の SQL ベースデータベースの SQL アプリケーションの SQL ステートメントを実行する場合にも役立ちます。
- 2 番目のフォームは、LIKE 句を使用して、別のテーブルと同じ列名および列属性を持つテーブルを作成します。新しいテーブルで任意の列を削除するには、CREATE TABLE ステートメントで DROP=データセットオプションを指定できます。テーブルの作成時に、指定した列が削除されます。インデックスは、新しいテーブルにコピーされません。

  これらのフォームは、両方とも行を含まないテーブルを作成します。INSERT ステートメントを使用して行を追加できます。列属性の変更、列の追加または削除を実行するには、ALTER TABLE ステートメントを使用します。

クエリ式からのテーブルの作成

- CREATE TABLE ステートメントの 3 番目のフォームは、任意のクエリ式の結果をテーブルに格納し、その出力を表示しません。これは、他のテーブルのサブセットまたはスーパーセットとなる一時的テーブルを作成する場合に便利な方法です。

  このフォームを使用すると、ステートメントを実行したときに、1 つのテーブルが物理的に作成されます。新しく作成されたこのテーブルは、クエリ式内の元になるテーブルに対するその後の変更を反映しません。継続的に最新データにアクセスする場合は、テーブルのかわりにクエリ式からビューを作成してください。詳細については、“CREATE VIEW ステートメント” (242 ページ)を参照してください。

  注意:

  再帰的なテーブル参照は、データの整合性の問題を引き起こす恐れがあります。
  CREATE TABLE AS ステートメントの対象テーブルを再帰的に参照することは可能ですが、それを行うと、データの一貫性の問題や、不正な結果を招く恐れがあります。次のような構造は避けてください。`proc sql; create table a as select var1, var2 from a;`

- 拡張属性を含んでいる既存のテーブルから新しいテーブルを作成すると、それらの拡張属性の一部または全部が新しいテーブルにコピーされます。拡張属性を含んでいる既存のテーブルのすべての列が新しいテーブルに含まれている場合、それらの拡張属性のすべてが新しいテーブルにコピーされます。新しいテーブルに既存のテーブルの列のサブセットのみが含まれている場合、列のサブセットに含まれている拡張属性のみが新しいテーブルにコピーされます。

  注: 複数テーブル結合や外部結合を使用して作成されるテーブルには、拡張属性はコピーされません。UNION、INTERSECT、またはマイナス演算子を使用する場合、その UNION、INTERSECT、またはマイナス演算子の前に記述されているテーブルに拡張属性が含まれている場合にのみ、拡張属性がコピーされます。

一貫性制約

テーブルを新規作成するときに、一貫性制約を付加できます。一貫性制約を変更するには、ALTER TABLE ステートメントを使用します。

一貫性制約を列指定に含めるときに、次の相互関係が一貫性制約に適用されます。

- 複合主キーを指定できません。
- 列指定で指定したチェック制約が WHERE 句内の同じ列を参照する必要はありません。
- 複数の一貫性制約を指定できます。
- 制約に対して、MSGTYPE=オプションと MESSAGE=オプションを指定できます。

詳細については、“Understanding Integrity Constraints” (*SAS Language Reference: Concepts*)を参照してください。

CREATE VIEW ステートメント

クエリ式から PROC SQL ビューを作成します。

参照項目: “ビューについて” (215 ページ)
“例 8: ビューをクエリの結果から作成する” (286 ページ)

構文

CREATE VIEW *proc-sql-view* <(*column-name-list*)> **AS** *query-expression*
<ORDER BY *order-by-item-1* <, *order-by-item-2*, …>>
<USING *libname-clause-1* <, *libname-clause-2*, …>> ;

必須引数

proc-sql-view
作成する PROC SQL ビューの名前を指定します。

参照項目 “ビューについて” (215 ページ)

query-expression
“query-expression” (339 ページ)を参照してください。

オプション引数

column-name-list
これは、カンマで区切ったビューの列名のリストです。SELECT 句で指定した列名またはエイリアスのかわりに使用されます。このリストの名前は、SELECT 句に指定した列の順序で、列に割り当てられます。このリストの列名の数が SELECT 句の列の数と等しくない場合、警告が SAS ログに書き込まれます。

ORDER BY ***order-by-item-1 <, order-by-item-2, …>***
“ORDER BY 句” (263 ページ)を参照してください。

USING ***libname-clause-1 <, libname-clause-2 , …>***
SAS/ACCESS LIBNAME ステートメントをビューの内部に埋め込むことにより、DBMS 接続情報を格納します。*libname-clause* には、次のいずれかを指定できます。

LIBNAME *libref <engine> ' SAS-library' <option(s)> <engine-host-option(s)>*

LIBNAME *libref SAS/ACCESS-engine-name<SAS/ACCESS-engine-connection-option(s)> <SAS/ACCESS-engine-LIBNAME-option(s)>*

参照項目 “LIBNAME Statement” (*SAS Statements: Reference*)

“LIBNAME Statement Syntax for Relational Databases” (*SAS/ACCESS for Relational Databases: Reference*)

詳細

ビューで取得したデータの並べ替え

PROC SQL では、CREATE VIEW ステートメントで ORDER BY 句を指定できます。ORDER BY 句を使用してビューにアクセスし、その ORDER BY 句が結果の順序に直接影響を与える場合、結果のデータは並べ替えられて、ORDER BY 句で指定したとおりに表示されます。ただし、ORDER BY 句が結果の順序に直接影響を与えない場合(たとえば、結合の一部として指定された場合)、PROC SQL は、パフォーマンス向上のため、その ORDER BY 句を無視します。

注: ORDER 句が省略された場合、インデックスが存在する場合でも、出力行が特定の順序(照会されたテーブルでの行の出現順など)になることは保証されません。ORDER BY 句を指定しないと、出力行の順序は、PROC SQL の内部処理、SAS のデフォルトの照合順序、および使用するオペレーティングシステムによって決まります。したがって、結果を特定の順序で表示する場合は、ORDER BY 句を使用してください。

注: ビューを作成するときに、PROC SQL ステートメントで NUMBER オプションを指定した場合、出力に ROW 行が表示されます。ただし、その後のクエリで、ROW 列によって並べ替えることはできません。“NUMBER | NONUMBER” (225 ページ)の説明を参照してください。

ライブラリ参照名と格納されたビュー

テーブルとビューが同じ SAS ライブラリに存在する場合、CREATE VIEW ステートメントの FROM 句で、ライブラリ参照名を使用せずにテーブル名を単独で参照できます。次に例を示します。

```
create view proclib.view1 as
   select *
      from invoice
      where invqty>10;
```

このビューでは、VIEW1 と INVOICE は、Proclib が参照する SAS ライブラリに永続的に格納されます。INVOICE のライブラリ参照名の指定は任意です。

ビューの更新

いくつかの制限付きで、ビューを構成するデータを更新できます。“PROC SQL ビューと SAS/ACCESS ビューの更新” (174 ページ)を参照してください。

埋め込まれた LIBNAME ステートメント

USING 句を使用すると、SAS/ACCESS LIBNAME ステートメントをビューの内部に埋め込むことによって、ビューに DBMS 接続情報を格納できます。このビューを PROC SQL で実行すると、格納されたクエリによって、ライブラリ参照名の割り当てと、LIBNAME ステートメント内の情報に基づく DBMS への接続が行われます。ライブラリ参照名の範囲は、そのビューに限定されます。このため、同一の SAS セッション中に同じ名前のライブラリ参照名があっても衝突しません。クエリが終了すると、DBMS との接続が終了し、ライブラリ参照名の割り当ても解除されます。

USING 句は、CREATE VIEW ステートメントの最後の句である必要があります。複数の LIBNAME ステートメントを、カンマで区切って指定できます。次の例では、接続が確立された後で、ACCREC というライブラリ参照名が ORACLE データベースに割り当てられます。

```
create view proclib.view1 as
   select *
      from accrec.invoices as invoices
      using libname accrec oracle
         user=username
pass=password
         path='dbms-path';
```

SAS/ACCESS LIBNAME ステートメントの詳細については、使用している DBMS に関する SAS/ACCESS のマニュアルを参照してください。

注: SAS System 9 で開始する場合は、リレーショナル DBMS データへのアクセス方法として、PROC SQL ビュー、パススルー機能および SAS/ACCESS LIBNAME ステートメントを推奨します。SAS/ACCESS ビューは推奨されていません。CV2VIEW プロシジャを使用して、既存の SAS/ACCESS ビューを PROC SQL ビューに変換できます。詳細については、“CV2VIEW” (*SAS/ACCESS for Relational Databases: Reference*)を参照してください。

USING 句を使用して、SAS LIBNAME ステートメントをビューに埋め込むことができます。これによって、SAS のライブラリ参照名情報をビューに格納できます。埋め込まれた SAS/ACCESS LIBNAME ステートメントと同様に、ライブラリ参照名の範囲はビュ

一に限定されています。そのため、同一の SAS セッション中の同じ名前のライブラリ参照名とは競合しません。

```
create view work.tableview as
   select * from proclib.invoices
      using libname proclib
'SAS-library';
```

DELETE ステートメント

FROM 句で指定したテーブルまたはビューから、1 つ以上の行を削除します。

制限事項: DELETE FROM は、UPDATE 処理をサポートしないエンジンによってアクセスされたテーブルには使用できません。

参照項目: “例 5: 2 つのテーブルを組み合わせる” (276 ページ)

構文

DELETE FROM *table-name* | *sas/access-view* | *proc-sql-view*
<AS *alias*>
<WHERE *sql-expression*>;

必須引数

table-name
行を削除するテーブルを指定します。*table-name* としては、1 レベルの名前、2 レベルの *libref.table* の名前または単一引用符で囲まれた物理パス名が可能です。

注意:
再帰的なテーブル参照は、データの整合性の問題を引き起こす恐れがあります。
DELETE ステートメントの対象テーブルを再帰的に参照することは可能ですが、それを行うと、データの一貫性の問題や、不正な結果を招く恐れがあります。次のような構造は避けてください。

```
proc sql;
   delete from a
      where var1 > (select min(var2) from a);
```

sas/access-view
行を削除する SAS/ACCESS ビューを指定します。

proc-sql-view
行を削除する PROC SQL ビューを指定します。*proc-sql-view* には、1 レベル名、2 レベルの *libref.view* の名前、または単一引用符で囲んだ物理パス名が可能です。

オプション引数

AS *alias*
table-name、*sas/access-view* または *proc-sql-view* にエイリアスを割り当てます。

WHERE *sql-expression*
“sql-expression” (347 ページ)を参照してください。

詳細

ビューを介した行の削除

いくつかの制限付きで、ビューの元になるテーブルから 1 つ以上の行を削除できます。“PROC SQL ビューと SAS/ACCESS ビューの更新” (174 ページ)を参照してください。

注意:

WHERE 句を省略した場合、DELETE ステートメントは、指定したテーブルまたはビューによって記述されたテーブルのすべての行を削除します。行は、テーブルを再作成するまでテーブルから削除されません。

DESCRIBE ステートメント

PROC SQL の定義を SAS ログに表示します。

制限事項: PROC SQL ビューは、DESCRIBE VIEW ステートメントで許可される唯一のタイプのビューです。

参照項目: “例 6: DICTIONARY テーブルからレポートを作成する” (279 ページ)

構文

DESCRIBE TABLE *table-name-1* <, *table-name-2*, …>;

DESCRIBE VIEW *proc-sql-view-1* <, *proc-sql-view-2*, …>;

DESCRIBE TABLE CONSTRAINTS *table-name-1* <, *table-name-2*, …>;

必須引数

table-name

PROC SQL テーブルを指定します。*table-name* としては、1 レベルの名前、2 レベルの *libref.table* の名前または単一引用符で囲まれた物理パス名が可能です。

proc-sql-view

PROC SQL ビューを指定します。*proc-sql-view* には、1 レベル名、2 レベルの *libref.view* の名前、または単一引用符で囲んだ物理パス名が可能です。

詳細

- DESCRIBE TABLE ステートメントは、テーブルが元々どのように作成されたか(たとえば、DATA ステップを使用するなど)に関わらず、DESCRIBE TABLE ステートメントで指定されたテーブルについての SAS ログに、CREATE TABLE ステートメントを書き込みます。該当する場合、SAS データセットオプションがテーブル定義に含まれます。テーブルの列にインデックスを定義した場合、それらのインデックスの CREATE INDEX ステートメントも SAS ログに書き込まれます。

  SAS/ACCESS ソフトウェアがサポートする DBMS にテーブルを転送するときに、このステートメントは、テーブルがどのように定義されているかを知るのに役に立ちます。テーブルの詳細情報を調べるには、FEEDBACK オプションまたは DATASETS プロシジャの CONTENTS ステートメントを使用します。

- DESCRIBE VIEW ステートメントは、ビュー定義を SAS ログに書き込みます。DESCRIBE VIEW ステートメントで、別のビューに基づくか別のビューから派生した PROC SQL ビューを使用する場合、PROC SQL ステートメントで FEEDBACK オプションを使用することをお勧めします。このオプションを指定すると、元になるビ

ューがどのように定義されているかが SAS ログに表示され、このビュー定義で使用されているすべての式が展開されます。DATASETS プロシジャの CONTENTS ステートメントをビューで使用して、詳細を調べることもできます。

パスワードで保護された SAS ビューを定義するには、パスワードを指定する必要があります。複数のパスワードを使用して SAS ビューを作成した場合、そのビューの定義にアクセスするには、最も制限の強いパスワードを指定する必要があります。たとえば、読み取りおよび書き込み保護の両方を備えた SAS ビューを定義するには、書き込みパスワードを指定する必要があります。同様に、読み取りおよび変更保護の両方を備えた SAS ビューを定義するには、変更パスワードを指定する必要があります。(これは、変更の方が読み取りよりも制限が強いためです)。詳細については、“Using Passwords with Views” (*SAS Language Reference: Concepts*)を参照してください。

- DESCRIBE TABLE CONSTRAINTS ステートメントは、指定された 1 つ以上のテーブルに対して定義された一貫性制約を表示します。ただし、主キー制約を参照する外部キーデータセット変数の名前は、主キー制約の DESCRIBE TABLE の出力の一部としては表示されません。

DISCONNECT ステートメント

SAS/ACCESS インターフェイスがサポートする DBMS との接続を終了します。

要件 SAS/ACCESS ソフトウェアが必要です。このステートメントの詳細については、SAS/ACCESS のドキュメントを参照してください。

参照項目: “パススルー機能を使用した DBMS への接続” (172 ページ)

構文

DISCONNECT FROM *dbms-name* | *alias*;

必須引数

dbms-name
接続を終了する DBMS(たとえば、DB2 や Oracle)を指定します。指定する名前は、CONNECT ステートメントで指定した名前と一致している必要があります。

alias
CONNECT ステートメントで定義したエイリアスを指定します。

詳細

- 暗黙の COMMIT が実行されてから、DISCONNECT ステートメントによって DBMS 接続が終了します。DISCONNECT ステートメントをサブミットしないと、PROC SQL の終了時に、暗黙の DISCONNECT アクションと COMMIT アクションが実行されて、DBMS との接続が切断されます。
- QUIT ステートメント、別の SAS プロシジャまたは DATA ステップをサブミットするまで、PROC SQL の実行は継続します。

DROP ステートメント

テーブル、ビューまたはインデックスを削除します。

制限事項: UPDATE 処理をサポートしないエンジンによってアクセスされたテーブルに対しては、DROP TABLE または DROP INDEX を使用できません。

構文

DROP TABLE *table-name-1* <, *table-name-2*, …>;

DROP VIEW *view-name-1* <, *view-name-2*, …>;

DROP INDEX *index-name-1* <, *index-name-2*, …> **FROM** *table-name*;

必須引数

table-name
PROC SQL テーブルを指定します。*table-name* としては、1 レベルの名前、2 レベルの *libref.table* の名前または単一引用符で囲まれた物理パス名が可能です。

view-name
いずれかのタイプの SAS ビュー(PROC SQL ビュー、SAS/ACCESS ビューまたは DATA ステップビュー)を指定します。*view-name* としては、1 レベルの名前、2 レベルの *libref.view* の名前、または単一引用符で囲まれた物理パス名が可能です。

index-name
table-name に存在するインデックスを指定します。

詳細

- ビュー定義が参照するテーブルを削除し、そのビューを実行しようとした場合、テーブルが存在しないことを示すエラーメッセージが SAS ログに書き込まれます。そのため、クエリおよびビュー内の、削除したテーブルおよびビューへの参照は、すべて削除してください。
- インデックスが作成されている列を含むテーブルを削除した場合、すべてのインデックスが自動的に削除されます。複合インデックスを削除した場合、そのインデックス内に記述されたすべての列のインデックスが削除されます。
- DROP ステートメントを使用して、パススルー機能または SAS/ACCESS LIBNAME ステートメントに関連付けられた外部データベース内のテーブルまたはビューを削除できます。ただし、SAS/ACCESS ビューに記述された外部データベースのテーブルまたはビューは削除できません。

EXECUTE ステートメント

DBMS 固有の SQL ステートメントを、SAS/ACCESS インターフェイスがサポートする DBMS に送信します。

要件 SAS/ACCESS ソフトウェアが必要です。このステートメントの詳細については、SAS/ACCESS のドキュメントを参照してください。

参照項目: “パススルー機能を使用した DBMS への接続” (172 ページ)
使用している DBMS の SQL に関するドキュメント

構文

EXECUTE (*dbms-SQL-statement*)
BY *dbms-name* | *alias*;

必須引数

dbms-SQL-statement
DBMS 固有の SQL ステートメントです。ただし、DBMS 固有の動的 SQL によって実行できる SELECT ステートメントを除きます。SQL ステートメントには、セミコロンを含めることができます。この SQL ステートメントは、データソースに応じて大文字と小文字が区別される場合があります。そのためこのステートメントは、入力したとおりに正確にデータソースに渡されます。

dbms-name
DBMS ステートメントの送信先の DBMS(たとえば、Oracle や DB2)を指定します。

alias
CONNECT ステートメントで定義したオプションのエイリアスを指定します。なお、*alias* は、BY キーワードの後に記述する必要があります。

詳細

- 使用する DBMS が複数の接続をサポートしている場合、CONNECT ステートメントで定義したエイリアスを使用できます。このエイリアスによって、EXECUTE ステートメントは特定の DBMS 接続に向けられます。
- ステートメントの実行後、DBMS によって生成されたすべてのリターンコードとメッセージは、マクロ変数 SQLXRC および SQLXMSG で参照できます。

例

次の例では、接続後、EXECUTE ステートメントを使用してテーブルを削除し、テーブルを作成し、データの行を挿入しています。

```
proc sql;
    execute(drop table ' My Invoice ') by db;
    execute(create table ' My Invoice '(
       ' Invoice Number ' LONG not null,
       ' Billed To '  VARCHAR(20),
       ' Amount '  CURRENCY,
       ' BILLED ON '  DATETIME)) by db;
    execute(insert into  ' My Invoice '
       values( 12345, 'John Doe', 123.45, #11/22/2003#)) by db;
quit;
```

INSERT ステートメント

新規または既存のテーブルまたはビューに、行を追加します。

制限事項: INSERT INTO は、UPDATE 処理をサポートしないエンジンによってアクセスされたテーブルに対しては使用できません。

参照項目: “例 1: テーブルを作成し、データを挿入する” (267 ページ)

構文

INSERT INTO *table-name | sas/access-view | proc-sql-view* <(*column-1*<, *column-2*, …>)>
SET *column1=sql-expression-1* <, *column-2=sql-expression-2*, …>
<SET *column1=sql-expression-1* <, *column-2=sql-expression-2*, …> …>;

INSERT INTO *table-name* | *sas/access-view* | *proc-sql-view* <(*column-1* <, *column-2*, …>)>
VALUES (*value-1* <, *value-2*, …>)
<VALUES (*value-1* <, *value-2*, …>) …>;
INSERT INTO *table-name* | *sas/access-view* | *proc-sql-view*
<(*column-1* <, *column-2*, …>)> *query-expression*;

必須引数

table-name
行を挿入する PROC SQL テーブルを指定します。*table-name* としては、1 レベルの名前、2 レベルの *libref.table* の名前または単一引用符で囲まれた物理パス名が可能です。

sas/access-view
行を挿入する SAS/ACCESS ビューを指定します。

proc-sql-view
行を挿入する PROC SQL ビュー指定します。*proc-sql-view* には、1 レベル名、2 レベルの *libref.view* の名前、または単一引用符で囲んだ物理パス名が可能です。

column
行を挿入する列を指定します。

sql-expression
“sql-expression” (347 ページ)を参照してください。

制限事項 SET 句の式では、論理演算(AND、OR または NOT)は使用できません。

value
データ値です。

注意:
再帰的なテーブル参照は、データの整合性の問題を引き起こす恐れがあります。
INSERT ステートメントの対象テーブルを再帰的に参照することは可能ですが、それを行うと、データの一貫性の問題や、不正な結果を招く恐れがあります。次のような構造は避けてください。

```
proc sql;
   insert into a
      select var1, var2
      from a
      where var1 > 0;
```

query-expression
“query-expression” (339 ページ)を参照してください。

詳細

値の挿入方法

- 最初の INSERT ステートメントでは、SET 句を使用して列の値を設定または変更しています。1 つの INSERT ステートメントに対して、1 つ以上の SET 句を使用することができ、1 つの SET 句で、1 つ以上の列の値を設定できます。複数の SET 句を使用する場合は、コンマで区切らずに指定します。オプションの列のリストを指定した場合、挿入する列のリストに指定した列の値のみを設定できます。
- 2 番目の INSERT ステートメントでは、VALUES 句を使用しています。VALUES 句は、テーブルに値のリストを挿入するときに使用します。通常は、テーブル内のす

べての列に値が設定されます。ただし、列名のリストを指定した場合は、そのリストに指定された列にのみ値が設定されます。挿入する行ごとに、VALUES 句を 1 つ指定します。複数の VALUES 句を使用する場合は、コンマで区切らずに指定します。VALUES 句に指定した値は、INSERT ステートメントの列リストの列名の順序に基づいて設定されます。列リストを指定しなかった場合は、テーブルの列の順序に基づいて設定されます。

- INSERT ステートメントの 3 番目のフォームは、クエリ式の結果をテーブルに挿入します。列リストが指定されていない場合、テーブル内の列の順序に一致します。クエリ式内の値の順序は、INSERT ステートメントの列リスト内の列名の順序に一致します。

注: INSERT ステートメントにオプションの列名のリストを含めた場合、それらの列にのみ INSERT ステートメントによって値が設定されます。テーブルに含まれていてもリストに含まれない列には、欠損値が設定されます。

ビューを介した行の挿入

いくつかの制限付きで、1 つ以上の行をビューを介してテーブルに挿入できます。"PROC SQL ビューと SAS/ACCESS ビューの更新" (174 ページ)を参照してください。

インデックスが作成された列への値の追加

テーブルの列にインデックスが定義されていて、そのテーブルに新しい行を挿入した場合、インデックスに値が追加されます。次を使用して、インデックスについての情報を表示できます。

- DATASETS プロシジャの CONTENTS ステートメント。詳細については、"CONTENTS Statement" (*Base SAS Procedures Guide*)を参照してください。
- DICTIONARY.INDEXES テーブル。詳細については、"DICTIONARY テーブルを使用し、SAS System の情報にアクセスする" (151 ページ)を参照してください。

インデックスの作成と使用の詳細については、"CREATE INDEX ステートメント" (236 ページ)を参照してください。

RESET ステートメント

プロシジャを再実行せずに PROC SQL オプションをリセットします。

参照項目: "例 5: 2 つのテーブルを組み合わせる" (276 ページ)

構文

RESET <*option(s)*>;

オプション引数

option(s)
プロシジャを再起動せずに追加、削除、変更を行いたい PROC SQL オプションを指定します。

参照項目 オプションの一覧については"PROC SQL ステートメント" (220 ページ)を参照してください。

SELECT ステートメント

テーブルおよびビューから、列と行のデータを選択します。

制限事項: SELECT ステートメントの句は、下記に示す順序で記述する必要があります。

参照項目: “query-expression” (339 ページ)
“table-expression” (364 ページ)

構文

SELECT <DISTINCT | UNIQUE> *object-item-1* <, *object-item-2*, …>
<INTO *macro-variable-specification-1* <, *macro-variable-specification-2*, …>>
FROM *from-list*
<WHERE *sql-expression*>
<GROUP BY *group-by-item-1* <, *group-by-item-2*, …>>
<HAVING *sql-expression*>
<ORDER BY *order-by-item-1* <, *order-by-item-2* <ASC | DESC>, …>>;

SELECT 句

出力に表示される列を記述します。

参照項目 “列エイリアスの使用” (148 ページ)
“column-definition” (316 ページ)
“例 1: テーブルを作成し、データを挿入する” (267 ページ)
“例 2: テーブルをクエリの結果から作成する” (269 ページ)

構文

SELECT <DISTINCT> *object-item-1* <, *object-item-2*, …>

必須引数

alias
列に一時的な代替名を割り当てます。

DISTINCT
重複行を除去します。引数 DISTINCT は UNIQUE と同一です。

別名 UNIQUE

注 引数 UNIQUE は DISTINCT と同一ですが、これは ANSI 規格ではありません。

DISTINCT は、内部の値または格納された値を処理しますが、必ずしも表示された値が処理されるわけではありません。数値精度によっては、同じに見える値を含む複数の行が返される場合があります。

ヒント　行のすべての値が別の行の値と同じである場合、その行は重複していると見なされます。引数 DISTINCT は、SELECT リスト内のすべての列に適用されます。存在する値の組合せごとに 1 行が表示されます。

可能な場合、PROC SQL は、SELECT DISTINCT ステートメントを処理するときにインデックスファイルを使用します。

例　“例 13: 列内の値の使用可能な組み合わせをすべて作成する” (300 ページ)

object-item
次のいずれかを指定できます。

*
FROM 句に記述されたテーブルまたはビューのすべての列を表します。

case-expression <**AS** *alias*>
CASE 式から列を生成します。

参照項目　“CASE 式” (314 ページ)

column-name <**AS** *alias*> <*column-modifier-1*<*column-modifier-2*…>>
1 つの列に名前を付けます。

参照項目　“column-name” (319 ページ)
“column-modifier” (317 ページ)

sql-expression <**AS** *alias*> <*column-modifier-1*<*column-modifier-2*…>>
SQL 式から列を生成します。

参照項目　“sql-expression” (347 ページ)
“column-modifier” (317 ページ)

*table-name.**
table-name で指定した PROC SQL テーブルのすべての列を指定します。

*table-alias.**
table-alias で指定したエイリアスが設定された PROC SQL テーブルのすべての列を指定します。

*view-name.**
view-name で指定した SAS ビューのすべての列を指定します。

*view-alias.**
view-alias で指定したエイリアスが設定された SAS ビューのすべての列を指定します。

詳細

アスタリスク(*)表記

アスタリスク(*)は、FROM 句に記述された 1 つ以上のテーブルのすべての列を表します。アスタリスクにテーブル名の接頭語を付加しない場合、FROM 句内のすべてのテーブルのすべての列が含まれます。接頭語を付加した場合(たとえば、table-name.*、table-alias.*など)、そのテーブルのすべての列のみが含まれます。

注: SELECT *の構文を使用して出力テーブルを作成した場合に、FROM 句に記述された複数のテーブルに同じ名前の列が存在すると、警告が表示されます。次のいずれかの対策を行って、この警告を回避できます。

- 重複する列名を除去するために、目的の列を、SELECT ステートメント内で同時に個別に記述します。

- RENAME=データセットオプションと DROP=データセットオプションを使用します。この例では、ID 列の名前が **tmpid** に変更されています。

```
proc sql;
   create table all(drop=tmpid) as
      select * from
         one, two(rename=(id=tmpid))
            where one.id=two.tmpid;
quit;
```

テーブルのエイリアスを使用する場合、テーブル名の後に RENAME=データセットオプションを記述し、その後にテーブルのエイリアスを記述します。名前を変更した列を最終的な出力テーブルで保持する場合、DROP=データセットオプションを省略できます。

INTO 句

別の PROC SQL クエリまたは SAS ステートメントで後で使用するために、1 つ以上の列の値を格納します。

制限事項 INTO 句は、CREATE TABLE ステートメントでは使用できません。

参照項目 “PROC SQL 自動マクロ変数の使用” (163 ページ)

構文

INTO *macro-variable-specification-1* <, *macro-variable-specification-2*, …>

必須引数

macro-variable-specification
次のいずれかを指定できます。

macro-variable **<SEPARATED BY '***character(s)***' <NOTRIM>>**
返された値を 1 つのマクロ変数に格納します。

macro-variable **<TRIMMED>**
返された値を 1 つのマクロ変数に格納します。

macro-variable-1–macro-variable-n **<NOTRIM>**
返された値を一連のマクロ変数に格納します。

ヒント マクロ変数の範囲を指定すると、SAS マクロ機能は、必要な数のマクロ変数のみを作成します。たとえば、**:var1-:var9999** を指定した場合、55 個の変数のみが必要であれば、**:var1-:var55** のみが作成されます。プログラムのその後の部分で、実際に作成された変数の個数を知る必要がある場合、SQLOBS 自動変数が役立ちます。この例の場合、SQLOBS には 55 の値が設定されます。

macro-variable-1 **- <NOTRIM>**
返された値を一連のマクロ変数に格納します。

ヒント 必要な変数の個数がわからない場合、上限を指定しないでマクロ変数の範囲を作成できます。プログラムのその後の部分で、実際に作成された変数の個数を知る必要がある場合、SQLOBS マクロ変数を使用できます。

macro-variable
返された行の値を格納する SAS マクロ変数を指定します。

NOTRIM
一連のマクロ変数に格納された値、または単一のマクロ変数に格納された複数の値から、先頭または末尾の空白が削除されるのを防ぎます。

SEPARATED BY '*character*'
行の値を区切る文字を指定します。

TRIMMED
単一のマクロ変数に格納された値から、先頭または末尾の空白を切り取ります。

詳細

- INTO 句は、SELECT ステートメントの外側のクエリでのみ使用し、サブクエリでは使用しません。
- 値を単一のマクロ変数に格納した場合、PROC SQL は、値の先頭または末尾の空白を維持します。TRIMMED オプションを使用して、単一のマクロ変数に格納された値から、先頭または末尾の空白を切り取ることができます。ただし、値を一連のマクロ変数に格納した場合、または SEPARATED BY オプションを使用して複数の値を単一のマクロ変数に格納した場合、PROC SQL は、NOTRIM オプションを指定しない限り、値の先頭または末尾の空白を切り取ります。
- 複数行の出力をマクロ変数に格納できます。PROC SQL マクロ変数の SQLOBS を使用して、クエリ式によって生成された行の数を特定できます。SQLOBS の詳細については、次を参照してください。“PROC SQL 自動マクロ変数の使用” (163 ページ)

  注: SQLOBS マクロ変数には、SELECT ステートメントを実行した後の値が割り当てられます。
- INTO 句で代入された値には、デフォルトで BEST8.出力形式が適用されます。大きな数値は、科学的記数法を使用して表示されます。科学的記数法を使用せずに値を表示する場合、別の出力形式(*w.*など) を使用します。

```
select sum(population format=16.
   into :totpop from sql.countries;
```

例: INTO 句

例 1: INTO 句

これらの例では、“Proclib.Houses” (437 ページ)テーブルを使用します。

```
title 'Proclib.Houses Table';
proc sql;
   select * from proclib.houses;
```

アウトプット 7.1 *Proclib.HOUSES テーブル*

PROCLIB.HOUSES Table

Style	SqFeet
CONDO	900
CONDO	1000
RANCH	1200
RANCH	1400
SPLIT	1600
SPLIT	1800
TWOSTORY	2100
TWOSTORY	3000
TWOSTORY	1940
TWOSTORY	1860

macro-variable-specification を使用して、次を実行できます。

- 結果の最初の行に基づいてマクロ変数を作成できます。

```
proc sql noprint;
   select style, sqfeet
      into :style, :sqfeet
      from proclib.houses;

%put &style &sqfeet;
```

結果が次のように SAS ログに書き込まれます。

```
1 proc sql noprint; 2 select style, sqfeet 3 into :style, :sqfeet 4 from proclib.houses;
```

- TRIMMED オプションを使用して、単一のマクロ変数に格納された値から、先頭または末尾の空白を削除できます。

```
proc sql noprint;
   select distinct style, sqfeet
      into :s1, :s2 TRIMMED
      from proclib.houses;
%put &s1 &s2;
%put There were &sqlobs distinct values.;
```

次の結果が SAS ログに書き込まれます。

```
1 proc sql noprint; 2 select distinct style, sqfeet 3 into :s1, :s2 TRIMMED 4 from procl
```

- SELECT ステートメントの結果に含まれる行ごとに、1 つの新しいマクロ変数を作成できます。この例は、ある列の値を別の列の値よりも多く要求する方法を示しています。INTO 句では、ハイフンは、マクロ変数の範囲を意味するために使用されます。ハイフンのかわりに、THROUGH キーワードまたは THRU キーワードを使用できます。

次の PROC SQL ステップでは、Proclib.HOUSES テーブルの最初の 4 行の値をマクロ変数に格納しています。

```
proc sql noprint;
select distinct Style, SqFeet
   into :style1 - :style3, :sqfeet1 - :sqfeet4
   from proclib.houses;

%put &style1 &sqfeet1;
%put &style2 &sqfeet2;
%put &style3 &sqfeet3;
%put &sqfeet4;
```

これらの%PUT ステートメントは、結果を SAS ログに書き込みます。

```
1 proc sql noprint; 2 select distinct style, sqfeet 3 into :style1 - :style3, :sqfeet1 -
```

- INTO 句でハイフンを使用して、上限を設けないで範囲を指定できます。

```
proc sql noprint;
select distinct Style, SqFeet
   into :style1 - , :sqfeet1 -
   from proclib.houses;

%put &style1 &sqfeet1;
%put &style2 &sqfeet2;
%put &style3 &sqfeet3;
%put &sqfeet4;
```

結果が次のように SAS ログに書き込まれます。

```
1 proc sql noprint; 2 select distinct Style, SqFeet 3 into :style1 - , :sqfeet1 - 4 from
```

- 1 つの列のそれぞれの値を 1 つのマクロ変数に連結することができます。このフォームは、変数または定数のリストを作成する場合に役立ちます。SQLOBS マクロ変数は、クエリによって処理されるデータのうちの重複しない変数の個数を示すのに役立ちます。

```
proc sql noprint;
   select distinct style
      into :s1 separated by ','
      from proclib.houses;
%put &s1;
%put There were &sqlobs distinct values.;
```

結果が次のように SAS ログに書き込まれます。

```
3 proc sql noprint; 4 select distinct style 5 into :s1 separated by ',' 6 from proclib.h
```

- 一連のマクロ変数名を作成するために、次の例に示すように、変数名の先頭でゼロを使用できます。

```
proc sql noprint;
   select SqFeet
      into :sqfeet01 - :sqfeet10
   from proclib.houses;

%put &sqfeet01 &sqfeet02 &sqfeet03 &sqfeet04 &sqfeet05;
%put &sqfeet06 &sqfeet07 &sqfeet08 &sqfeet09 &sqfeet10;
```

結果が次のように SAS ログに書き込まれます。

```
11 proc sql noprint; 12 select sqfeet 13 into :sqfeet01 - :sqfeet10 14 from proclib.hous
```

- マクロ変数に格納される値から先頭と末尾の空白が切り取られるのを防ぐことができます。デフォルトでは、一連のマクロ変数に値を格納したとき、または(SEPARATED BY オプションを使用して)単一のマクロ変数に複数の値を格納したときに、PROC SQL は、値の先頭と末尾の空白を切り取ってからマクロ変数を作成します。先頭と末尾の空白が切り取られないようにする場合、次の例に示すように、NOTRIM オプションを使用します。

```
proc sql noprint;
   select style, sqfeet
      into :style1 - :style4 notrim,
           :sqfeet separated by ',' notrim
      from proclib.houses;

%put *&style1* *&sqfeet*;
%put *&style2* *&sqfeet*;
%put *&style3* *&sqfeet*;
%put *&style4* *&sqfeet*;
```

次に示すような出力が SAS ログに書き込まれます。

```
3 proc sql noprint; 4 select style, sqfeet 5 into :style1 - :style4 notrim, 6 :sqfeet se
```

FROM 句

ソーステーブルまたはソースビューを指定します。

参照項目 “例 1: テーブルを作成し、データを挿入する” (267 ページ)
“例 4: 2 つのテーブルを結合する” (273 ページ)
“例 9: 3 つのテーブルを結合する” (289 ページ)
“例 10: インラインビューをクエリする” (293 ページ)

構文

FROM *from-list*

必須引数

from-list
次のいずれかを指定できます。

table-name <<**AS**> *alias*>
単一の PROC SQL テーブル名を指定します。*table-name* としては、1 レベルの名前、2 レベルの *libref.table* の名前または単一引用符で囲まれた物理パス名が可能です。*alias* には、FROM 句で指定したテーブル、ビュー、インラインビューの一時的な代替名を指定します。

view-name <<**AS**> *alias*>
1 つの SAS ビューに名前を付けます。*view-name* としては、1 レベルの名前、2 レベルの *libref.view* の名前、または単一引用符で囲まれた物理パス名が可能です。

joined-table
結合を指定します。

参照項目 “joined-table” (323 ページ)

(*query-expression*) <<**AS**> *alias*> <(*column-1* <, *column-2*, …>)>
インラインビューを指定します。

alias には、FROM 句で指定したテーブル、ビュー、インラインビューの一時的な代替名を指定します。

column には、出力に表示される列の名前を指定します。指定した列名は、出力の列と位置によって対応付けられます。

参照項目 “query-expression” (339 ページ)

CONNECTION TO
DBMS テーブルを指定します。

参照項目 “CONNECTION TO” (320 ページ)

注 *table-name* および *view-name* では、*table-name* または *view-name* の直後でかっこで囲んでデータセットオプションを使用できます。詳細については、“PROC SQL で SAS データセットオプションを使用する” (157 ページ)を参照してください。

詳細

テーブルのエイリアス

テーブルのエイリアスは、FROM 句で指定したテーブルの一時的な代替名です。複数のテーブルで共通する列を区別するには、テーブルのエイリアスの接頭語を列名に付加します。再帰結合(テーブルとそのテーブル自身との結合)で指定する列名には、それらの列がテーブルのどのコピーに属するかを区別するために、テーブルのエイリアスの接頭語を付加する必要があります。他のタイプの結合では、列名がテーブル間で重複している場合、テーブルのエイリアスまたはテーブル名の接頭語を列名に付加する必要があります。

テーブルのエイリアスを他のテーブル名と区別するために、多くの場合、オプションの AS キーワードが使用されます。

インラインビュー

FROM 句自体にクエリ式を含め、それにオプションのテーブルのエイリアスを設定できます。このタイプのネストされたクエリ式をインラインビューと呼びます。インラインビューは、CREATE VIEW ステートメントで有効な任意のクエリ式です。PROC SQL は、多くのレベルのネストをサポートできますが、1 つのクエリでのテーブル数は 256 個に制限されます。256 テーブルの制限には、FROM 句で指定されたビューに寄与する、元になるテーブルの個数も含まれます。

インラインビューを使用して、プログラムステップを省くことができます。ビューを作成してそれを別のクエリで参照するのではなく、FROM 句でインラインとしてビューを指定できます。

インラインビューの特徴は、次のとおりです。

- インラインビューには、永続的な名前を割り当てることはできませんが、エイリアスを指定できます。
- インラインビューが定義されているクエリ内でのみ、インラインビューに参照できます。別のクエリで参照することはできません。
- インラインビューで ORDER BY 句を使用することはできません。
- インラインビューのオブジェクト項目のリストに、またはエイリアスの後のかっこで囲まれた名前のリストを使用して、インラインビューの列名を割り当てることができます。この構文は、列名を変更する場合に役立ちます。例については、“例 10: インラインビューをクエリする” (293 ページ)を参照してください。
- インラインビューをクエリの他の部分と視覚的に区別するために、任意の数のかっこの組みでインラインビューを囲むことができます。ただし、インラインビューに対してエイリアスを指定する場合、そのインラインビューの最も外側のかっこの組みの外側で、エイリアス指定を記述する必要があります。

WHERE 句

指定された条件に基づいて出力をサブセット化します。

参照項目 “例 4: 2 つのテーブルを結合する” (273 ページ)
“例 9: 3 つのテーブルを結合する” (289 ページ)

構文

WHERE *sql-expression*

必須引数

sql-expression
“sql-expression” (347 ページ)を参照してください。

詳細

- 条件が満たされた場合(つまり、条件が TRUE に決定された場合)、結果テーブルにそれらの行が表示されます。そうでない場合、行は表示されません。
- 1 つの列のみを指定する要約関数を使用することはできません。

  次の例では、MAX は要約関数です。したがって、MAX のコンテキストは GROUP BY 句です。この関数を、データのグループ化、つまり要約に使用することはできません。

  ```
  where max(measure1) > 50;
  ```

  ただし、次の WHERE 句は動作します。

  ```
  where max(measure1,measure2) > 50;
  ```

  この場合、MAX は SAS 関数です。同じ行内の 2 つの列の値を比較しているため、この関数は WHERE 句で動作します。その結果、この関数を使用してデータをサブセット化できます。

GROUP BY 句

要約するためのデータのグループ化方法を指定します。

参照項目 “例 8: ビューをクエリの結果から作成する” (286 ページ)
“例 12: 2 つのテーブルを結合して新しい値を計算する” (297 ページ)

構文

GROUP BY *group-by-item-1* <, *group-by-item-2*, …>

必須引数

group-by-item
次のいずれかを指定できます。

integer
列の位置に等しい正の整数。

column-name
列名または列のエイリアス。

参照項目 “column-name” (319 ページ)
“列エイリアスの使用” (148 ページ)

sql-expression
“sql-expression” (347 ページ)を参照してください。

詳細

- 複数の *group-by-item* を指定して、さらに詳細なレポートを取得できます。複数の項目のグループ化と PROC ステップの BY ステートメントは、どちらも同じ方法で

評価されます。複数の *group-by-item* を指定した場合、最初の group-by-item が最上位のグループを決定します。

- GROUP BY 句の列名(つまり、SELECT 句のオブジェクト項目)は、整数で代用できます。たとえば、*group-by-item* が 2 である場合、結果は SELECT 句リストの 2 番目の列の値によってグループ化されます。整数を使用すると、コードを短くすることができ、SELECT 句リスト内の名前が付けられていない式の値によってグループ化できます。浮動小数点値(たとえば、2.3)を使用した場合、PROC SQL は小数点次の部分を無視します。
- PROC SQL が自動的に並べ替えを実行するため、グループごとの値の順序でデータを並べ替える必要はありません。ORDER BY 句を使用して、結果テーブルに表示する行の順序を指定できます。
- 要約関数を含まないクエリで GROUP BY 句を指定した場合、その句は ORDER BY 句に変換され、その影響を示すメッセージが SAS ログに書き込まれます。
- 式が返す値によって出力をグループ化できます。たとえば、X が数値変数の場合、次の出力は X の値の整数部分によってグループ化されます。

```
select x, sum(y)
from table1
group by int(x);
```

同様に、Y が文字変数の場合、次の出力は Y の値の 2 番目の文字によってグループ化されます。

```
select sum(x), y
from table1
group by substring(y from 2 for 1);
```

数値リテラル(および数値リテラルの関数)のみ、または文字リテラル(および文字リテラルの関数)のみを含む式は無視されます。

注: 前述の例のような式を使用すると、SAS では、要約統計量と元のデータが再マージされます。統計量とデータの再マージは、予期しない結果を招く可能性があります。詳細については、“データの再マージ” (359 ページ)を参照してください。

GROUP BY 句内の式を要約関数にすることはできません。たとえば、次の GROUP BY 句は無効です。

```
group by sum(x)
```

HAVING 句

指定された条件に基づいて、グループ化されたデータをサブセット化します。

参照項目 “列エイリアスの使用” (148 ページ)
“例 8: ビューをクエリの結果から作成する” (286 ページ)
“例 12: 2 つのテーブルを結合して新しい値を計算する” (297 ページ)

構文

HAVING *sql-expression*

必須引数

sql-expression
"sql-expression" (347 ページ)を参照してください。

詳細

HAVING 句は、少なくとも 1 つの要約関数とオプションの GROUP BY 句と共に使用されて、テーブル内のデータのグループを要約します。HAVING 句は、クエリ内のグループごとに TRUE または FALSE として評価される、任意の有効な SQL 式です。あるいは、クエリに再マージ対象のデータが含まれる場合、HAVING 式は、各グループに含まれる行ごとに評価されます。クエリには、1 つ以上の要約関数を指定する必要があります。

通常、GRUP BY 句は HAVING 式と共に使用されて、評価対象の 1 つ以上のグループを定義します。GROUP BY 句を省略した場合、要約関数と HAVING 句は、テーブルを 1 つのグループとして扱います。

次の PROC SQL ステップでは、Proclib.Payroll テーブル("例 2: テーブルをクエリの結果から作成する" (269 ページ) に示されているもの)を使用して、Gender によって行をグループ化し、最年長の従業員を性別に決定しています。SAS では、日付は整数で格納されます。誕生日を表す整数値が小さいほど、年齢は高くなります。式 **birth=min(birth)** は、テーブルの行ごとに評価されます。最小の誕生日が検出されると、式は TRUE になり、その行が出力に含まれます。

```
proc sql;
   title 'Oldest Employee of Each Gender';
   select *
      from proclib.payroll
      group by gender
      having birth=min(birth);
```

注: 要約関数から返される値が、GROUP BY 句に含まれない列の値と比較されるため、このクエリには再マージ対象のデータが含まれます。要約関数とデータの再マージの詳細については、"データの再マージ" (359 ページ)を参照してください。

ORDER BY 句

結果テーブルに行を表示する順序を指定します。

参照項目 "列エイリアスの使用" (148 ページ)
"例 11: SOUNDS-LIKE 演算子を使用して値を取得する" (295 ページ)
"query-expression" (339 ページ)

構文

ORDER BY *order-by-item-1* <ASC | DESC> <, *order-by-item-2* <ASC | DESC>, …>;

必須引数

order-by-item
次のいずれかを指定できます。

integer
列の位置を表します。

column-name
列名または列のエイリアス。

参照項目 “column-name” (319 ページ)

sql-expression
“sql-expression” (347 ページ)を参照してください。

ASC
データを昇順に並べ替えます。これは、デフォルトの順序です。ASC と DESC のどちらも指定されない場合、データは昇順に並べ替えられます。

DESC
データを降順に並べ替えます。

詳細

- ORDER BY 句は、そのクエリで指定された順序に従ってクエリ式の結果を並べ替えます。デフォルトでは、昇順にソートされます。SORTSEQ=オプションを使用して、出力の照合順序を変更できます。“PROC SQL ステートメント” (220 ページ)を参照してください。
- 返される出力行の順序は、ORDER BY 句に指定した列についてのみ保証されます。

  注: ORDER BY 句は、生成される行の順序が一意に決定されることを保証しません。SQL の ANSI 規格では、SQL の実装において、ORDER BY 句が安定的であるかどうかを規定できます。クエリの ORDER BY 句で参照される値の組合せが、並べ替えられるすべての行に対して一意である場合、ORDER BY 句によって生成される行の順序は必ず一意に決定されます。しかし、ORDER BY 句が一意の値の組合せを参照せず、ORDER BY 句が安定的でない場合、行の順序は一意に決定されません。
- ORDER 句が省略された場合、インデックスが存在する場合でも、出力行が特定の順序(照会されたテーブルでの行の出現順など)になることは保証されません。ORDER BY 句を指定しないと、出力行の順序は、PROC SQL の内部処理、SAS のデフォルトの照合順序、および使用するオペレーティングシステムによって決まります。
- 複数の *order-by-item* をコンマで区切って指定した場合、1 番目の order-by-item が、主要な並べ替え順序を決定します。
- ORDER BY 句の列名(つまり、SELECT 句のオブジェクト項目)を、整数で代用できます。たとえば、*order-by-item* が 2(整数)の場合、結果は 2 番目の列の値によって並べ替えられます。クエリ式にセット演算子(たとえば、UNION)が含まれる場合、整数を使用して順序を指定します。そうすることで、テーブル式での列への曖昧な参照を避けることができます。なお、整数のかわりに浮動小数点値(たとえば、2.3)を使用すると、PROC SQL は小数点次の部分を無視します。
- ORDER BY 句では、クエリ式の FROM 句で指定したテーブルまたはビューの任意の列を、その列がクエリの SELECT 句に含まれているかどうかに関わらず指定できます。たとえば、次のクエリは、1990 年から 1995 年にかけての各国の人口変化の降順の値によって並べ替えたレポートを生成します。

```
proc sql;
   select country
      from census
      order by pop95-pop90 desc;

NOTE: The query as specified involves
      ordering by an item that
      doesn't appear in its SELECT clause.
```

- 式が返す値によって、出力を並べ替えることができます。たとえば、X が数値変数の場合、次の出力は X の値の整数部分によって並べ替えられます。

```
select x, y
from table1
order by int(x);
```

同様に、Y が文字変数である場合、次の出力は Y の値の 2 番目の文字によって並べ替えられます。

```
select x, y
from table1
order by substring(y from 2 for 1);
```

数値リテラル(および数値リテラルの関数)のみ、または文字リテラル(および文字リテラルの関数)のみを含む式は無視されますので、注意してください。

UPDATE ステートメント

テーブルまたはビューの、既存の行の列の値を変更します。

制限事項: UPDATE は、UPDATE 処理をサポートしないエンジンによってアクセスされたテーブルには使用できません。

参照項目: “例 3: PROC SQL テーブルのデータの更新” (271 ページ)

構文

UPDATE *table-name* | *sas/access-view* | *proc-sql-view* <AS *alias*>
  SET *column-1*=*sql-expression-1* <, *column-2*=*sql-expression-2*, …>
    <SET *column-1*=*sql-expression-1* <, *column-1*=*sql-expression-2*, …>>
  <WHERE *sql-expression*>;

必須引数

table-name
PROC SQL テーブルを指定します。*table-name* としては、1 レベルの名前、2 レベルの *libref.table* の名前または単一引用符で囲まれた物理パス名が可能です。

sas/access-view
SAS/ACCESS ビューを指定します。

proc-sql-view
PROC SQL ビューを指定します。*proc-sql-view* には、1 レベル名、2 レベルの *libref.view* の名前、または単一引用符で囲んだ物理パス名が可能です。

alias
table-name、*sas/access-view* または *proc-sql-view* にエイリアスを割り当てます。

column
table-name、*sas/access-view* または *proc-sql-view* の列を指定します。

sql-expression
“sql-expression” (347 ページ)を参照してください。

制限事項 SET 句の式では、論理演算(AND、OR または NOT)は使用できません。

詳細

いくつかの制限付きで、テーブルの 1 つ以上の行をビューを介して更新できます。“PROC SQL ビューと SAS/ACCESS ビューの更新” (174 ページ)を参照してください。

- 変更対象外の列の値は、CASE 式を使用する一部のクエリの場合を除いて、変更されません。CASE 式の説明については、“CASE 式” (314 ページ)を参照してください。
- 列の定義または属性を追加、削除または変更するには、“ALTER TABLE ステートメント” (231 ページ)で説明されている ALTER TABLE ステートメントを使用します。
- SET 句では、等号の左側の列参照を、等号の右側の式の一部に含めることができます。たとえば、次の式を使用して、従業員に 1,000 ドルの休日手当を支給できます。

```
set salary=salary + 1000
```

- WHERE 句を省略すると、すべての行が更新されます。WHERE 句を使用すると、その条件を満たす行のみが更新されます。
- インデックスに使用されている列を更新すると、新しい値にはその列に定義されたインデックスが継承されます。

VALIDATE ステートメント

クエリ式を実行せずに、クエリ式の構文と意味の正確さをチェックします。

構文

VALIDATE *query-expression*;

必須引数

query-expression
“query-expression” (339 ページ)を参照してください。

詳細

- VALIDATE ステートメントは、クエリが有効であることを示すメッセージを SAS ログに書き込みます。エラーが存在する場合、VALIDATE はエラーメッセージを SAS ログに書き込みます。
- VALIDATE ステートメントは、マクロ機能を使用するアプリケーションに含めることもできます。そのようなアプリケーションで使用した場合、VALIDAT は、クエリ式の有効性を示す値を返します。この値は、SQLRC(SQL のリターンコードの省略形)マクロ変数によって返されます。たとえば、SELECT ステートメントが有効である場合、SQLRC マクロ変数は 0 の値を返します。詳細については、“PROC SQL 自動マクロ変数の使用” (163 ページ)を参照してください。

例: SQL プロシジャ

例 1: テーブルを作成し、データを挿入する

要素: CREATE TABLE ステートメント
column-modifier
INSERT ステートメント
VALUES 句
SELECT 句
FROM 句

表名: Proclib.Paylist

この例では、Proclib.Paylist テーブルを作成し、それにデータを挿入します。

プログラム

```
libname proclib 'SAS-library';

proc sql;
   create table proclib.paylist
       (IdNum char(4),
        Gender char(1),
        Jobcode char(3),
        Salary num,
        Birth num informat=date7.
                  format=date7.,
        Hired num informat=date7.
                  format=date7.);

insert into proclib.paylist
    values('1639','F','TA1',42260,'26JUN70'd,'28JAN91'd)
    values('1065','M','ME3',38090,'26JAN54'd,'07JAN92'd)
    values('1400','M','ME1',29769.'05NOV67'd,'16OCT90'd)

values('1561','M',null,36514,'30NOV63'd,'07OCT87'd)
    values('1221','F','FA3',.,'22SEP63'd,'04OCT94'd);

title 'Proclib.Paylist Table';

select *
   from proclib.paylist;

proc printto; run;
```

プログラムの説明

Proclib ライブラリを宣言します。これらの例では、Proclib ライブラリは作成されたテーブルの格納に使用されます。

```
libname proclib 'SAS-library';
```

Proclib.Paylist テーブルを作成します。 CREATE TABLE ステートメントによって、6 つの空の列を持つ Proclib.Paylist を作成します。それぞれの列定義は、列が文字であるか数値であるかを示しています。かっこ内の数字は、列の幅を指定しています。INFORMAT=と FORMAT=によって、Birth 列と Hired 列に入力形式と出力形式を割り当てています。

```
proc sql;
   create table proclib.paylist
       (IdNum char(4),
        Gender char(1),
        Jobcode char(3),
        Salary num,
        Birth num informat=date7.
                  format=date7.,
        Hired num informat=date7.
                  format=date7.);
```

Proclib.Paylist テーブルに値を挿入します。 INSERT ステートメントによって、VALUES 句の位置に従い、Proclib.Paylist にデータ値を挿入します。そのため、最初の VALUES 句の値 *1639* は最初の列に、*F* は 2 番目の列に、というように挿入ます。SAS の日付は、整数で格納され、その 0 の値は 1960 年 1 月 1 日を表します。内部の日付の値を使用する 1 つの方法は、次のように、日付に *d* という接尾語を付加することです。

```
insert into proclib.paylist
    values('1639','F','TA1',42260,'26JUN70'd,'28JAN91'd)
    values('1065','M','ME3',38090,'26JAN54'd,'07JAN92'd)
    values('1400','M','ME1',29769.'05NOV67'd,'16OCT90'd)
```

データに欠損値を含めます。 次の NULL 値は、文字列 Jobcode の欠損値を表しています。ピリオドは、数値列 Salary の欠損値を表しています。

```
values('1561','M',null,36514,'30NOV63'd,'07OCT87'd)
    values('1221','F','FA3',.,'22SEP63'd,'04OCT94'd);
```

タイトルを指定します。

```
title 'Proclib.Paylist Table';
```

Proclib.Paylist テーブル全体を表示します。 SELECT 句によって、Proclib.Paylist から列を選択します。アスタリスク(*)は、すべての列を選択します。FROM 句では、選択対象のテーブルとして、Proclib.Paylist を指定しています。

```
select *
   from proclib.paylist;

proc printto; run;
```

出力:テーブルにデータを挿入

アウトプット 7.2 *Proclib.Paylist テーブル*

PROCLIB.PAYLIST Table

IdNum	Gender	Jobcode	Salary	Birth	Hired
1639	F	TA1	42260	26JUN70	28JAN91
1065	M	ME3	38090	26JAN54	07JAN92
1400	M	ME1	29769	05NOV67	16OCT90
1561	M		36514	30NOV63	07OCT87
1221	F	FA3	.	22SEP63	04OCT94

例 2: テーブルをクエリの結果から作成する

要素: CREATE TABLE ステートメント
AS*query expression*
SELECT 句
columnalias
FORMAT=*column-modifier*
object-item

他の要素: データセットオプション
OBS=

表名: Proclib.Payroll
Proclib.Bonus

詳細

この例では、算術演算式使用して列を構築し、クエリの結果から Proclib.Bonus テーブルを作成します。

```
proc sql outobs=10;
   title 'Proclib.Payroll';
   title2 'First 10 Rows Only';
   select * from proclib.payroll;
   title;
```

図 7.2 Proclib.Payroll のクエリ結果

PROCLIB.PAYROLL
First 10 Rows Only

IdNumber	Gender	Jobcode	Salary	Birth	Hired
1919	M	TA2	34376	12SEP60	04JUN87
1653	F	ME2	35108	15OCT64	09AUG90
1400	M	ME1	29769	05NOV67	16OCT90
1350	F	FA3	32886	31AUG65	29JUL90
1401	M	TA3	38822	13DEC50	17NOV85
1499	M	ME3	43025	26APR54	07JUN80
1101	M	SCP	18723	06JUN62	01OCT90
1333	M	PT2	88606	30MAR61	10FEB81
1402	M	TA2	32615	17JAN63	02DEC90
1479	F	TA3	38785	22DEC68	05OCT89

プログラム

```
libname proclib 'SAS-library';

proc sql;
   create table proclib.bonus as

 select IdNumber, Salary format=dollar8.,
          salary*.025 as Bonus format=dollar8.
      from proclib.payroll;

title 'Bonus Information';

select *
      from proclib.bonus(obs=10);
```

プログラムの説明

Proclib ライブラリを宣言します。これらの例では、Proclib ライブラリは作成されたテーブルの格納に使用されます。

```
libname proclib 'SAS-library';
```

Proclib.Bonus テーブルを作成します。CREATE TABLE ステートメントによって、その後のクエリの結果から Proclib.Bonus テーブルを作成します。

```
proc sql;
   create table proclib.bonus as
```

含める列を選択します。SELECT 句によって、新しいテーブルに含める 3 つの列 (IdNumber、Salary および Bonus)を指定します。FORMAT=によって、Salary に

DOLLAR8.出力形式を割り当てています。Bonus 列は、SQL 式(*salary*.025*)を使用して作成されます。

```
select IdNumber, Salary format=dollar8.,
         salary*.025 as Bonus format=dollar8.
      from proclib.payroll;
```

タイトルを指定します。

```
title 'Bonus Information';
```

Proclib.Bonus テーブルの最初の 10 行を表示します。 SELECT 句によって、Proclib.Bonus から列を選択します。アスタリスク(*)は、すべての列を選択します。FROM 句によって、選択対象のテーブルとして Proclib.Bonus を指定しています。OBS=データセットオプションによって、出力する行数を 10 行に制限しています。

```
select *
      from proclib.bonus(obs=10);
```

出力:クエリからのテーブルの作成

アウトプット 7.3 *Proclib.Bonus テーブル*

BONUS Information

IdNumber	Salary	Bonus
1919	$34,376	$859
1653	$35,108	$878
1400	$29,769	$744
1350	$32,886	$822
1401	$38,822	$971
1499	$43,025	$1,076
1101	$18,723	$468
1333	$88,606	$2,215
1402	$32,615	$815
1479	$38,785	$970

例 3: PROC SQL テーブルのデータの更新

要素: ALTER TABLE ステートメント
- DROP 句
- MODIFY 句

UPDATE ステートメント
- SET 句

CASE 式

表名: Employees

この例では、Employees テーブルのデータ値を更新し、列を削除します。

Employees テーブルを作成するプログラム

```
proc sql;
   title 'Employees Table';
   select * from Employees;
```

プログラムの説明

Employees テーブル全体を表示します。 SELECT 句によって、更新前のテーブルを表示します。アスタリスク(*)によって、表示するためにすべての列を選択しています。FROM 句によって、選択対象のテーブルとして Employees を指定しています。

```
proc sql;
   title 'Employees Table';
   select * from Employees;
```

出力:Employees テーブルの作成

アウトプット 7.4 *Employees テーブル*

Employees Table

IdNum	LName	FName	JobCode	Salary	Phone
1876	CHIN	JACK	TA1	42400	212/588-5634
1114	GREENWALD	JANICE	ME3	38000	212/588-1092
1556	PENNINGTON	MICHAEL	ME1	29860	718/383-5681
1354	PARKER	MARY	FA3	65800	914/455-2337
1130	WOOD	DEBORAH	PT2	36514	212/587-0013

Employees テーブルを更新するプログラム

```
proc sql;
update employees
      set salary=salary*
      case when jobcode like '__1' then 1.04
           else 1.025
      end;

alter table employees
      modify salary num format=dollar8.
      drop phone;

title 'Updated Employees Table';

select * from employees;
```

プログラムの説明

Salary 列の値を更新します。 UPDATE ステートメントによって、Employees の値を更新します。SET 句では、ジョブコードの最後が 1 の場合は Salary 列のデータに 1.04 を掛け、それ以外のジョブコードの場合は 1.025 を掛けることを指定しています。(2 つのアンダーラインは任意の文字を表しています。)CASE 式は、SET 句を完成するために、行ごとに値を返します。

```
proc sql;
update employees
      set salary=salary*
      case when jobcode like '__1' then 1.04
           else 1.025
      end;
```

Salary 列の出力形式を変更し、Phone 列を削除します。 ALTER TABLE ステートメントによって、変更対象のテーブルとして Employees を指定します。MODIFY 句によって、Salary 列の出力形式を永続的に変更します。DROP 句によって、Phone 列を永続的に削除します。

```
alter table employees
      modify salary num format=dollar8.
      drop phone;
```

タイトルを指定します。

```
title 'Updated Employees Table';
```

更新された Employees テーブル全体を表示します。 SELECT 句によって、更新後の Employees テーブルを表示します。アスタリスク(*)は、すべての列を選択します。

```
select * from employees;
```

出力:PROC SQL テーブルのデータの更新

アウトプット 7.5 *更新された Employees テーブル*

Updated Employees Table

IdNum	LName	FName	JobCode	Salary
1876	CHIN	JACK	TA1	$44,096
1114	GREENWALD	JANICE	ME3	$38,950
1556	PENNINGTON	MICHAEL	ME1	$31,054
1354	PARKER	MARY	FA3	$67,445
1130	WOOD	DEBORAH	PT2	$37,427

例 4: 2 つのテーブルを結合する

要素: FROM 句
テーブルのエイリアス

内部結合

結合テーブルの構成要素

PROC SQL ステートメントのオプション
    NUMBER

WHERE 句
    IN 条件

表名: Proclib.Staff

Proclib.Payroll

詳細

この例では、2 つのテーブルに共通するデータに関する詳細情報を取得するために、それらのテーブルを結合します。

```
proc sql outobs=10;
   title 'Proclib.Staff';
   title2 'First 10 Rows Only';
   select * from proclib.staff;
   title;
```

図 7.3 *Proclib.Staff テーブル*

PROCLIB.STAFF
First 10 Rows Only

idnum	lname	fname	city	state	hphone
1919	ADAMS	GERALD	STAMFORD	CT	203/781-1255
1653	ALIBRANDI	MARIA	BRIDGEPORT	CT	203/675-7715
1400	ALHERTANI	ABDULLAH	NEW YORK	NY	212/586-0808
1350	ALVAREZ	MERCEDES	NEW YORK	NY	718/383-1549
1401	ALVAREZ	CARLOS	PATERSON	NJ	201/732-8787
1499	BAREFOOT	JOSEPH	PRINCETON	NJ	201/812-5665
1101	BAUCOM	WALTER	NEW YORK	NY	212/586-8060
1333	BANADYGA	JUSTIN	STAMFORD	CT	203/781-1777
1402	BLALOCK	RALPH	NEW YORK	NY	718/384-2849
1479	BALLETTI	MARIE	NEW YORK	NY	718/384-8816

```
proc sql outobs=10;
   title 'Proclib.Payroll';
   title2 'First 10 Rows Only';
   select * from proclib.payroll;
   title;
```

図 7.4 Proclib.Payroll テーブル

PROCLIB.PAYROLL
First 10 Rows Only

IdNumber	Gender	Jobcode	Salary	Birth	Hired
1919	M	TA2	34376	12SEP60	04JUN87
1653	F	ME2	35108	15OCT64	09AUG90
1400	M	ME1	29769	05NOV67	16OCT90
1350	F	FA3	32886	31AUG65	29JUL90
1401	M	TA3	38822	13DEC50	17NOV85
1499	M	ME3	43025	26APR54	07JUN80
1101	M	SCP	18723	06JUN62	01OCT90
1333	M	PT2	88606	30MAR61	10FEB81
1402	M	TA2	32615	17JAN63	02DEC90
1479	F	TA3	38785	22DEC68	05OCT89

プログラム

```
libname proclib 'SAS-library';

proc sql number;

title 'Information for Certain Employees Only';

select Lname, Fname, City, State,
          IdNumber, Salary, Jobcode

from proclib.staff, proclib.payroll

where idnumber=idnum and idnum in
             ('1919', '1400', '1350', '1333');
```

プログラムの説明

Proclib ライブラリを宣言します。これらの例では、Proclib ライブラリは作成されたテーブルの格納に使用されます。

```
libname proclib 'SAS-library';
```

PROC SQL の出力に行番号を追加します。NUMBER によって、行番号を含む列を追加します。

```
proc sql number;
```

タイトルを指定します。

```
title 'Information for Certain Employees Only';
```

表示する列を選択します。SELECT 句を使用して、出力に表示する列を選択します。

```
select Lname, Fname, City, State,
          IdNumber, Salary, Jobcode
```

データを取得するテーブルを指定します。 FROM 句では、選択対象のテーブルを記述しています。

```
from proclib.staff, proclib.payroll
```

結合条件を指定して、クエリをサブセット化します。 WHERE 句では、各テーブルの ID 番号でテーブルを結合することを指定しています。さらに、WHERE 句では、IN 条件を使用してクエリをサブセット化しています。これにより、4 人の従業員のみの行が返されます。

```
where idnumber=idnum and idnum in
             ('1919', '1400', '1350', '1333');
```

出力:2 つのテーブルを結合する

アウトプット 7.6 特定の従業員のみの情報

Information for Certain Employees Only

Row	lname	fname	city	state	IdNumber	Salary	Jobcode
1	ADAMS	GERALD	STAMFORD	CT	1919	34376	TA2
2	ALHERTANI	ABDULLAH	NEW YORK	NY	1400	29769	ME1
3	ALVAREZ	MERCEDES	NEW YORK	NY	1350	32886	FA3
4	BANADYGA	JUSTIN	STAMFORD	CT	1333	88606	PT2

例 5: 2 つのテーブルを組み合わせる

要素: DELETE ステートメント
IS 条件
RESET ステートメントのオプション
DOUBLE
UNION セット演算子

表名: Proclib.Newpay
Proclib.Paylist
Proclib.Paylist2

入力テーブル

この例では、Proclib.Paylist と Proclib.Paylist2 という 2 つのテーブルを連結して、新しいテーブル Proclib.Newpay を作成します。

```
proc sql;
title 'Proclib.Paylist Table';
   select * from proclib.paylist;
```

図 7.5 *Proclib.Paylist テーブル*

PROCLIB.PAYLIST Table

IdNum	Gender	Jobcode	Salary	Birth	Hired
1639	F	TA1	42260	26JUN70	28JAN91
1065	M	ME3	38090	26JAN54	07JAN92
1400	M	ME1	29769	05NOV67	16OCT90
1561	M		36514	30NOV63	07OCT87
1221	F	FA3	.	22SEP63	04OCT94

```
proc sql;
title 'Proclib.Paylist2 Table';
  select * from proclib.Paylist2;
title;
```

図 7.6 *Proclib.Paylist2 テーブル*

PROCLIB.PAYLIST2 Table

IdNum	Gender	Jobcode	Salary	Birth	Hired
1919	M	TA2	34376	12SEP66	04JUN87
1653	F	ME2	31896	15OCT64	09AUG92
1350	F	FA3	36886	31AUG55	29JUL91
1401	M	TA3	38822	13DEC55	17NOV93
1499	M	ME1	23025	26APR74	07JUN92

プログラム

```
libname proclib 'SAS-library';

proc sql;
   create table proclib.newpay as
      select * from proclib.paylist
      union
      select * from proclib.paylist2;

delete
      from proclib.newpay
      where jobcode is missing or salary is missing;

reset double;

title 'Personnel Data';

select *
      from proclib.newpay;
```

プログラムの説明

Proclib ライブラリを宣言します。これらの例では、Proclib ライブラリは作成されたテーブルの格納に使用されます。

```
libname proclib 'SAS-library';
```

Proclib.Newpay テーブルを作成します。SELECT 句によって、FROM 句に記述されたテーブルから、すべての列を選択します。UNION セット演算子によって、2 つの SELECT 句から生成されたクエリ結果を連結します。

```
proc sql;
   create table proclib.newpay as
      select * from proclib.paylist
      union
      select * from proclib.paylist2;
```

Jobcode または Salary の値が欠損している行を削除します。DELETE ステートメントによって、WHERE 式を満たす行を Proclib.Newpay から削除します。IS 条件によって、Jobcode 列または Salary 列に欠損値を含む行を指定しています。

```
delete
      from proclib.newpay
      where jobcode is missing or salary is missing;
```

PROC SQL 環境をリセットし、出力の行間を 2 行に設定します。RESET によって、PROC SQL を停止して再開することなくプロシジャ環境を変更します。DOUBLE オプションによって、出力の行間を 1 行おきにします。(DOUBLE オプションは、ODS の出力に対しては無効です。)

```
reset double;
```

タイトルを指定します。

```
title 'Personnel Data';
```

Proclib.Newpay テーブル全体を表示します。SELECT 句によって、新しく作成したテーブル Proclib.Newpay から、すべての列を選択します。

```
select *
      from proclib.newpay;
```

出力:2 つのテーブルを組み合わせる

アウトプット 7.7 *Proclib.Newpay テーブル*

Personnel Data

IdNum	Gender	Jobcode	Salary	Birth	Hired
1065	M	ME3	38090	26JAN54	07JAN92
1350	F	FA3	36886	31AUG55	29JUL91
1400	M	ME1	29769	05NOV67	16OCT90
1401	M	TA3	38822	13DEC55	17NOV93
1499	M	ME1	23025	26APR74	07JUN92
1639	F	TA1	42260	26JUN70	28JAN91
1653	F	ME2	31896	15OCT64	09AUG92
1919	M	TA2	34376	12SEP66	04JUN87

例 6: DICTIONARY テーブルからレポートを作成する

要素: DESCRIBE TABLE ステートメント
DICTIONARY.*table-name* 構成要素

表名: DICTIONARY.Members

この例では、DICTIONARY テーブルを使用して、SAS ライブラリ内の SAS ファイルのリストを表示します。照会する DICTIONARY テーブルの列名がわからない場合、そのテーブルを指定して DESCRIBE TABLE ステートメントを使用します。

プログラム

```
libname proclib 'SAS-library';

proc sql;
   describe table dictionary.members;

title 'SAS Files in the Proclib Library';

select memname, memtype
      from dictionary.members
      where libname='PROCLIB';
```

プログラムの説明

Proclib ライブラリを宣言します。これらの例では、Proclib ライブラリは作成されたテーブルの格納に使用されます。

```
libname proclib 'SAS-library';
```

DICTIONARY.Members テーブルの列名を表示します。DESCRIBE TABLE によって、DICTIONARY.Members の列名を SAS ログに書き込みます。

```
proc sql;
   describe table dictionary.members;
```

タイトルを指定します。

```
title 'SAS Files in the Proclib Library';
```

Proclib ライブラリ内のファイルのリストを表示します。SELECT 句によって、MEMNAME 列と MEMTYPE 列を選択します。FROM 句では、選択対象のテーブルとして DICTIONARY.Members を選択しています。WHERE 句では、*Proclib* のライブラリ参照名を LIBNAME 列に持つ行のみを含むように、出力をサブセット化しています。

```
select memname, memtype
      from dictionary.members
      where libname='PROCLIB';
```

ログ

ログ 7.1 *DICTIONARY.Members テーブルのログの作成*

```
277 options nodate pageno=1 source linesize=80 pagesize=60; 278 279 proc sql;
280 describe table dictionary.members; NOTE:SQL table DICTIONARY.Members was
created like: create table DICTIONARY.Members ( libname char(8) label='Library
Name', memname char(32) label='Member Name', memtype char(8) label='Member
Type', engine char(8) label='Engine Name', index char(32) label='Indexes', path
char(1024) label='Path Name' ); 281     title 'SAS Files in the Proclib
Library'; 282 283     select memname, memtype 284        from dictionary.members
285        where libname='PROCLIB';
```

出力:Proclib ライブラリ内の SAS ファイル

アウトプット 7.8 *Proclib ライブラリ*

SAS Files in the PROCLIB Library

Member Name	Member Type
BONUS	DATA
NEWPAY	DATA
PAYLIST	DATA
PAYLIST2	DATA
PAYROLL	DATA
STAFF	DATA

例 7: 外部結合を実行する

要素: 結合テーブルの構成要素
左外部結合
SELECT 句
COALESCE 関数
WHERE 句
CONTAINS 条件

表名: Proclib.Payroll
Proclib.Payroll2

詳細

この例では、Proclib.Payroll テーブルと Proclib.Payroll2 テーブルの左外部結合について説明します。

```
proc sql outobs=10;
   title 'Proclib.Payroll';
   title2 'First 10 Rows Only';
   select * from proclib.payroll
   order by idnumber;
   title;
```

図 7.7 Proclib.Payroll

PROCLIB.PAYROLL
First 10 Rows Only

IdNumber	Gender	Jobcode	Salary	Birth	Hired
1009	M	TA1	28880	02MAR59	26MAR92
1017	M	TA3	40858	28DEC57	16OCT81
1036	F	TA3	39392	19MAY65	23OCT84
1037	F	TA1	28558	10APR64	13SEP92
1038	F	TA1	26533	09NOV69	23NOV91
1050	M	ME2	35167	14JUL63	24AUG86
1065	M	ME2	35090	26JAN44	07JAN87
1076	M	PT1	66558	14OCT55	03OCT91
1094	M	FA1	22268	02APR70	17APR91
1100	M	BCK	25004	01DEC60	07MAY88

```
proc sql;
```

```
title 'Proclib.Payroll2';
select * from proclib.payroll2
 order by idnum;
title;
```

図 7.8 Proclib.Payroll2

PROCLIB.PAYROLL2

idnum	gender	jobcode	salary	birth	hired
1036	F	TA3	42465	19MAY65	23OCT84
1065	M	ME3	38090	26JAN44	07JAN87
1076	M	PT1	69742	14OCT55	03OCT91
1106	M	PT3	94039	06NOV57	16AUG84
1129	F	ME3	36758	08DEC61	17AUG91
1221	F	FA3	29896	22SEP67	04OCT91
1350	F	FA3	36098	31AUG65	29JUL90
1369	M	TA3	36598	28DEC61	13MAR87
1447	F	FA1	22123	07AUG72	29OCT92
1561	M	TA3	36514	30NOV63	07OCT87
1639	F	TA3	42260	26JUN57	28JAN84
1998	M	SCP	23100	10SEP70	02NOV92

ID 番号に基づいて外部結合を使用するプログラム

```
libname proclib 'SAS-library';

proc sql outobs=10;

title 'Most Current Jobcode and Salary Information';

select p.IdNumber, p.Jobcode, p.Salary,
          p2.jobcode label='New Jobcode',
          p2.salary label='New Salary' format=dollar8.

from proclib.payroll as p left join proclib.payroll2 as p2

on p.IdNumber=p2.idnum;
```

プログラムの説明

Proclib ライブラリを宣言します。これらの例では、Proclib ライブラリは作成されたテーブルの格納に使用されます。

```
libname proclib 'SAS-library';
```

出力の行数を制限します。OUTOBS=によって、出力を 10 行に制限しています。

```
proc sql outobs=10;
```

最初のクエリのタイトルを指定します。

```
title 'Most Current Jobcode and Salary Information';
```

列を選択します。 SELECT 句には、選択する列を記述します。一部の列名が両方のテーブルに存在するため、それらの列名にはテーブルのエイリアスの接頭語が付加されています。LABEL=と FORMAT=は、列修飾子です。

```
select p.IdNumber, p.Jobcode, p.Salary,
          p2.jobcode label='New Jobcode',
          p2.salary label='New Salary' format=dollar8.
```

結合のタイプを指定します。 FROM 句では、結合するテーブルを記述し、テーブルにエイリアスを割り当てています。LEFT JOIN キーワードは、結合のタイプを指定します。FROM 句内のテーブルの順序は重要です。Proclib.Payroll を最初に記述しているため、これが"左"のテーブルであると見なされます。Proclib.Payroll2 が、"右"のテーブルになります。

```
from proclib.payroll as p left join proclib.payroll2 as p2
```

結合条件を指定します。 ON 句では、各テーブルの ID 番号の値に基づいて結合を実行することを指定しています。

```
on p.IdNumber=p2.idnum;
```

出力:ID 番号に基づく外部結合

出力が示すように、左のテーブル(Proclib.Payroll)のすべての行が返されています。PROC SQL は、一致する値が Payroll2 の IdNum に存在しない左のテーブル(Payroll)の行に、欠損値を割り当てます。

アウトプット 7.9 *最新のジョブコードと給与情報*

Most Current Jobcode and Salary Information

IdNumber	Jobcode	Salary	New Jobcode	New Salary
1009	TA1	28880		.
1017	TA3	40858		.
1036	TA3	39392	TA3	$42,465
1037	TA1	28558		.
1038	TA1	26533		.
1050	ME2	35167		.
1065	ME2	35090	ME3	$38,090
1076	PT1	66558	PT1	$69,742
1094	FA1	22268		.
1100	BCK	25004		.

COALESCE と LEFT JOIN を使用するプログラム

```
proc sql outobs=10;

title 'Most Current Jobcode and Salary Information';

select p.idnumber, coalesce(p2.jobcode,p.jobcode)
        label='Current Jobcode',

coalesce(p2.salary,p.salary) label='Current Salary'
                  format=dollar8.

from proclib.payroll p left join proclib.payroll2 p2
      on p.IdNumber=p2.idnum;
```

プログラムの説明

```
proc sql outobs=10;
```

---

2 番目のクエリのタイトルを指定します。

```
title 'Most Current Jobcode and Salary Information';
```

列を選択し、Jobcode 列を合体します。 SELECT 句には、選択する列を記述します。COALESCE は、同じ名前の列を重ね合わせます。COALESCE は、行ごとに、P2.JobCode または P.JobCode のいずれかの最初の非欠損値を返します。最初の引数が P2.JobCode であるため、P2.JobCode に非欠損値が存在する場合、COALESCE はその値を返します。したがって、出力には、全従業員の最新のジョブコード情報が含まれます。LABEL=によって、列ラベルを割り当てています。

```
select p.idnumber, coalesce(p2.jobcode,p.jobcode)
        label='Current Jobcode',
```

Salary 列の合体 COALESCE は、行ごとに、P2.Salary または P.Salary のいずれかの最初の非欠損値を返します。最初の引数が P2.Salary であるため、P2.Salary に非欠損値が存在する場合、COALESCE はその値を返します。したがって、出力には、全従業員の最新の給与情報が含まれます。

```
coalesce(p2.salary,p.salary) label='Current Salary'
                  format=dollar8.
```

結合のタイプと結合条件を指定します。 FROM 句では、結合するテーブルを記述し、テーブルにエイリアスを割り当てています。LEFT JOIN キーワードは、結合のタイプを指定します。ON 句では、各テーブルの ID 番号に基づいて結合することを指定しています。

```
from proclib.payroll p left join proclib.payroll2 p2
      on p.IdNumber=p2.idnum;
```

出力:COALESCE と LEFT JOIN

アウトプット 7.10 *最新のジョブコードと給与情報*

Most Current Jobcode and Salary Information

IdNumber	Current Jobcode	Current Salary
1009	TA1	$28,880
1017	TA3	$40,858
1036	TA3	$42,465
1037	TA1	$28,558
1038	TA1	$26,533
1050	ME2	$35,167
1065	ME3	$38,090
1076	PT1	$69,742
1094	FA1	$22,268
1100	BCK	$25,004

クエリをサブセット化するプログラム

```
proc sql;

title 'Most Current Information for Ticket Agents';
   select p.IdNumber,
          coalesce(p2.jobcode,p.jobcode) label='Current Jobcode',
          coalesce(p2.salary,p.salary) label='Current Salary'
      from proclib.payroll p left join proclib.payroll2 p2
      on p.IdNumber=p2.idnum
      where p2.jobcode contains 'TA';
```

プログラムの説明

クエリをサブセット化します。 WHERE 句では、TA の値が含まれた行のみを含むように、左結合をサブセット化しています。

```
proc sql;

title 'Most Current Information for Ticket Agents';
   select p.IdNumber,
          coalesce(p2.jobcode,p.jobcode) label='Current Jobcode',
          coalesce(p2.salary,p.salary) label='Current Salary'
      from proclib.payroll p left join proclib.payroll2 p2
      on p.IdNumber=p2.idnum
      where p2.jobcode contains 'TA';
```

出力:クエリのサブセット化

アウトプット 7.11 *TA の値を含むクエリ結果*

Most Current Information for Ticket Agents

IdNumber	Current Jobcode	Current Salary
1036	TA3	42465
1369	TA3	36598
1561	TA3	36514
1639	TA3	42260

例 8: ビューをクエリの結果から作成する

要素: CREATE VIEW ステートメント
GROUP BY 句
SELECT 句
　COUNT 関数
HAVING 句

他の要素: AVG 要約関数
データセットオプション
　PW=

表名: Proclib.Payroll
Proclib.Jobs

詳細

この例では、クエリ式の結果から PROC SQL ビュー(Proclib.Jobs)を作成します。

```
proc sql outobs=10;
   title 'Proclib.Payroll';
   title2 'First 10 Rows Only';
   select * from proclib.payroll
   order by idnumber;
   title;
```

図 7.9 Proclib.Payroll

PROCLIB.PAYROLL
First 10 Rows Only

IdNumber	Gender	Jobcode	Salary	Birth	Hired
1009	M	TA1	28880	02MAR59	26MAR92
1017	M	TA3	40858	28DEC57	16OCT81
1036	F	TA3	39392	19MAY65	23OCT84
1037	F	TA1	28558	10APR64	13SEP92
1038	F	TA1	26533	09NOV69	23NOV91
1050	M	ME2	35167	14JUL63	24AUG86
1065	M	ME2	35090	26JAN44	07JAN87
1076	M	PT1	66558	14OCT55	03OCT91
1094	M	FA1	22268	02APR70	17APR91
1100	M	BCK	25004	01DEC60	07MAY88

プログラム

```
libname proclib 'SAS-library';

proc sql;
   create view proclib.jobs(pw=red) as

select Jobcode,
             count(jobcode) as number label='Number',

avg(int((today()-birth)/365.25)) as avgage
                format=2. label='Average Age',
             avg(salary) as avgsal
                format=dollar8. label='Average Salary'

from payroll

group by jobcode
      having avgage ge 30;

title 'Current Summary Information for Each Job Category';
   title2 'Average Age Greater Than or Equal to 30';

select * from proclib.jobs(pw=red);

title2;
```

プログラムの説明

Proclib ライブラリを宣言します。これらの例では、Proclib ライブラリは作成されたテーブルの格納に使用されます。

```
libname proclib 'SAS-library';
```

Proclib.Jobs ビューを作成します。 CREATE VIEW によって、PROC SQL ビュー (Proclib.Jobs)を作成します。PW=データセットオプションでは、このビューによって生成されたデータに対して、パスワード保護を割り当てています。

```
proc sql;
   create view proclib.jobs(pw=red) as
```

列を選択します。 SELECT 句では、ビューに対して、Jobcode 列に、Number、AvgAge および AvgSal の 3 つの列(これらの値は積関数です)を合わせた 4 つの列を指定しています。COUNT は、データが Jobcode 別にグループ化されているため、ジョブコードごとに非欠損値の数を返します。LABEL=によって、列にラベルを割り当てています。

```
select Jobcode,
            count(jobcode) as number label='Number',
```

Avgage 列と Avgsal 列を計算します。 AVG 要約関数は、ジョブコードごとに平均年齢と平均給与を計算します。

```
avg(int((today()-birth)/365.25)) as avgage
               format=2. label='Average Age',
            avg(salary) as avgsal
               format=dollar8. label='Average Salary'
```

データを取得するテーブルを指定します。 FROM 句では、データを選択するテーブルとして Payroll を指定しています。PROC SQL は、CREATE VIEW ステートメントで Proclib が使用されているため、Payroll のライブラリ参照名が Proclib であると見なします。

```
from payroll
```

データをグループに編成し、出力に含めるグループを指定します。 GROUP BY 句では、Jobcode の値によってデータをグループ化しています。したがって、すべての要約統計量は、Jobcode の値によってグループ化された、行のグループごとに計算されます。HAVING 句は、グループ化されたデータをサブセット化し、30 歳以上の平均年令を含むジョブコードの行を返します。

```
group by jobcode
      having avgage ge 30;
```

タイトルを指定します。

```
title 'Current Summary Information for Each Job Category';
   title2 'Average Age Greater Than or Equal to 30';
```

Proclib.Jobs ビュー全体を表示します。 SELECT ステートメントによって、Proclib.Jobs から、すべての列を選択します。ビューがパスワードで保護されているため、PW=RED が必要です。

```
select * from proclib.jobs(pw=red);

title2;
```

出力:ビューをクエリの結果から作成する

アウトプット 7.12 *ジョブカテゴリごとの最新の要約情報*

Current Summary Information for Each Job Category
Average Age Greater Than or Equal to 30

Jobcode	Number	Average Age	Average Salary
BCK	9	45	$25,794
FA1	11	42	$23,039
FA2	16	46	$27,987
FA3	7	48	$32,934
ME1	8	43	$28,500
ME2	14	49	$35,577
ME3	7	51	$42,411
NA1	5	39	$42,032
NA2	3	51	$52,383
PT1	8	47	$67,908
PT2	10	52	$87,925
PT3	2	63	$10,505
SCP	7	46	$18,309
TA1	9	45	$27,721
TA2	20	46	$33,575
TA3	12	49	$39,680

例 9: 3 つのテーブルを結合する

要素: FROM 句
結合テーブルの構成要素
WHERE 句

表名: Proclib.Staff2
Proclib.Schedule2
Proclib.Superv2

詳細

この例では、3 つのテーブルを結合し、それらのテーブルの列を含むレポートを生成します。

例のコード 7.1 *Proclib.Staff2 Table*

```
data proclib.staff2;
input IdNum $4. @7 Lname $12. @20 Fname $8. @30 City $10.
      @42 State $2. @50 Hphone $12.;
   datalines;
1106  MARSHBURN    JASPER    STAMFORD    CT      203/781-1457
1430  DABROWSKI    SANDRA    BRIDGEPORT  CT      203/675-1647
1118  DENNIS       ROGER     NEW YORK    NY      718/383-1122
1126  KIMANI       ANNE      NEW YORK    NY      212/586-1229
1402  BLALOCK      RALPH     NEW YORK    NY      718/384-2849
1882  TUCKER       ALAN      NEW YORK    NY      718/384-0216
1479  BALLETTI     MARIE     NEW YORK    NY      718/384-8816
1420  ROUSE        JEREMY    PATERSON    NJ      201/732-9834
1403  BOWDEN       EARL      BRIDGEPORT  CT      203/675-3434
1616  FUENTAS      CARLA     NEW YORK    NY      718/384-3329
;
run;

proc sql;
   title 'Proclib.Staff2';
   select * from proclib.staff2;
   title;
```

図 7.10 *Proclib.Staff2*

PROCLIB.STAFF2

IdNum	Lname	Fname	City	State	Hphone
1106	MARSHBURN	JASPER	STAMFORD	CT	203/781-1457
1430	DABROWSKI	SANDRA	BRIDGEPORT	CT	203/675-1647
1118	DENNIS	ROGER	NEW YORK	NY	718/383-1122
1126	KIMANI	ANNE	NEW YORK	NY	212/586-1229
1402	BLALOCK	RALPH	NEW YORK	NY	718/384-2849
1882	TUCKER	ALAN	NEW YORK	NY	718/384-0216
1479	BALLETTI	MARIE	NEW YORK	NY	718/384-8816
1420	ROUSE	JEREMY	PATERSON	NJ	201/732-9834
1403	BOWDEN	EARL	BRIDGEPORT	CT	203/675-3434
1616	FUENTAS	CARLA	NEW YORK	NY	718/384-3329

例のコード 7.2 *Proclib.Schedule2 Table*

```
data proclib.schedule2;
   input flight $3. +5 date date7. +2 dest $3. +3 idnum $4.;
   format date date7.;
   informat date date7.;
   datalines;
132     01MAR94  BOS   1118
132     01MAR94  BOS   1402
219     02MAR94  PAR   1616
219     02MAR94  PAR   1478
622     03MAR94  LON   1430
622     03MAR94  LON   1882
271     04MAR94  NYC   1430
271     04MAR94  NYC   1118
579     05MAR94  RDU   1126
579     05MAR94  RDU   1106
;
run;

proc sql;
   title 'Proclib.Schedule2';
   select * from proclib.schedule2;
   title;
```

図 7.11 *Proclib.Schedule2*

PROCLIB.SCHEDULE2

flight	date	dest	idnum
132	01MAR94	BOS	1118
132	01MAR94	BOS	1402
219	02MAR94	PAR	1616
219	02MAR94	PAR	1478
622	03MAR94	LON	1430
622	03MAR94	LON	1882
271	04MAR94	NYC	1430
271	04MAR94	NYC	1118
579	05MAR94	RDU	1126
579	05MAR94	RDU	1106

例のコード 7.3 *Proclib.Superv2 Table*

```
data proclib.superv2;
   input supid $4. +8 state $2. +5  jobcat  $2.;
   label supid='Supervisor Id' jobcat='Job Category';
   datalines;
1417        NJ     NA
1352        NY     NA
```

```
1106        CT      PT
1442        NJ      PT
1118        NY      PT
1405        NJ      SC
1564        NY      SC
1639        CT      TA
1126        NY      TA
1882        NY      ME
;
run;

proc sql;
   title 'Proclib.Superv2';
   select * from proclib.superv2
   title;
```

図 7.12 *Proclib.Superv2*

PROCLIB.SUPERV2

Supervisor Id	state	Job Category
1417	NJ	NA
1352	NY	NA
1106	CT	PT
1442	NJ	PT
1118	NY	PT
1405	NJ	SC
1564	NY	SC
1639	CT	TA
1126	NY	TA
1882	NY	ME

プログラム

```
libname proclib 'SAS-library';

proc sql;
   title 'All Flights for Each Supervisor';
   select s.IdNum, Lname, City 'Hometown', Jobcat,
          Flight, Date

from proclib.schedule2 s, proclib.staff2 t, proclib.superv2 v

where s.idnum=t.idnum and t.idnum=v.supid;
```

プログラムの説明

Proclib ライブラリを宣言します。これらの例では、Proclib ライブラリは作成されたテーブルの格納に使用されます。

```
libname proclib 'SAS-library';
```

列を選択します。SELECT 句で、選択する列を指定します。IdNum は、2 つのテーブルに現われるため、テーブルのエイリアスの接頭語が付加されています。

```
proc sql;
   title 'All Flights for Each Supervisor';
   select s.IdNum, Lname, City 'Hometown', Jobcat,
          Flight, Date
```

結合に含めるテーブルを指定します。FROM 句では、結合する 3 つのテーブルを記述し、それらのテーブルにエイリアスを割り当てています。

```
from proclib.schedule2 s, proclib.staff2 t, proclib.superv2 v
```

結合条件を指定します。WHERE 句では、テーブルを結合する列を指定しています。Staff2 テーブルと Schedule2 テーブルにはそれぞれ IdNum 列が含まれています。この列の値が両方のテーブルで一致するときに、行が結合されます。Staff2 テーブルと Superv2 テーブルにはそれぞれ IdNum 列と SupId 列が含まれています。これらの列の値が両方のテーブルで一致するときに、行が結合されます。この 2 つの条件の組み合わせで 3 つのテーブルの結合が可能になります。

```
where s.idnum=t.idnum and t.idnum=v.supid;
```

出力:3 つのテーブルを結合する

アウトプット 7.13 *各管理者のすべての航空便*

All Flights for Each Supervisor

idnum	Lname	Hometown	Job Category	flight	date
1106	MARSHBURN	STAMFORD	PT	579	05MAR94
1118	DENNIS	NEW YORK	PT	132	01MAR94
1118	DENNIS	NEW YORK	PT	271	04MAR94
1126	KIMANI	NEW YORK	TA	579	05MAR94
1882	TUCKER	NEW YORK	ME	622	03MAR94

例 10: インラインビューをクエリする

要素: FROM 句
インラインビュー

表名: Proclib.Staff2
Proclib.Schedule2

Proclib.Superv2

この例では、“例 9: 3 つのテーブルを結合する” (289 ページ)で説明したクエリの作成方法とは別の方法を示します。これは、テーブルの 1 つをインラインビューの結果と結合することによって行います。この例では、インラインを使用して列名を変更する方法についても示します。

プログラム

```
libname proclib 'SAS-library';

proc sql;
   title 'All Flights for Each Supervisor';
   select three.*, v.jobcat

from (select lname, s.idnum, city, flight, date
               from proclib.schedule2 s, proclib.staff2 t
               where s.idnum=t.idnum)

as three (Surname, Emp_ID, Hometown,
                           FlightNumber, FlightDate),

proclib.superv2 v
      where three.Emp_ID=v.supid;
```

プログラムの説明

Proclib ライブラリを宣言します。これらの例では、Proclib ライブラリは作成されたテーブルの格納に使用されます。

```
libname proclib 'SAS-library';
```

列を選択します。SELECT 句では、インラインビュー(Three というエイリアスが割り当てられます)が返すすべての列に加えて、3 番目のテーブル(V というエイリアスが割り当てられます)の 1 つの列を選択しています。

```
proc sql;
   title 'All Flights for Each Supervisor';
   select three.*, v.jobcat
```

インラインクエリを指定します。FROM 句には、テーブルやビューの名前を含めるかわりに、3 つのテーブルのうちの 2 つを結合するクエリを含めています。インラインクエリでは、SELECT 句に、選択する列を記述しています。IdNum は、両方のテーブルに現れるため、テーブルのエイリアスの接頭語が付加されています。FROM 句では、結合対象の 2 つのテーブルを記述し、それらのテーブルにエイリアスを割り当てています。WHERE 句では、テーブルを結合する列を指定しています。Staff2 テーブルと Schedule2 テーブルにはそれぞれ IdNum 列が含まれています。この列の値が両方のテーブルで一致するときに、行が結合されます。

```
from (select lname, s.idnum, city, flight, date
               from proclib.schedule2 s, proclib.staff2 t
               where s.idnum=t.idnum)
```

クエリのエイリアスと列の名前を指定します。Three というエイリアスによって、インラインビューの結果を参照します。かっこ内の名前が、ビューの列の名前になります。

```
as three (Surname, Emp_ID, Hometown,
                           FlightNumber, FlightDate),
```

インラインビューの結果を、3 番目のテーブルと結合します。 WHERE 句によって、3 番目のテーブルとインラインビューを結合する列を指定します。なお、WHERE 句では、インラインビューで名前が変更された Emp_ID 列を指定しています。

```
proclib.superv2 v
      where three.Emp_ID=v.supid;
```

出力:インラインビューをクエリする

アウトプット 7.14 *各管理者のすべての航空便*

All Flights for Each Supervisor

Surname	Emp_ID	Hometown	FlightNumber	FlightDate	Job Category
MARSHBURN	1106	STAMFORD	579	05MAR94	PT
DENNIS	1118	NEW YORK	132	01MAR94	PT
DENNIS	1118	NEW YORK	271	04MAR94	PT
KIMANI	1126	NEW YORK	579	05MAR94	TA
TUCKER	1882	NEW YORK	622	03MAR94	ME

例 11: SOUNDS-LIKE 演算子を使用して値を取得する

要素: ORDER BY 句
SOUNDS-LIKE 演算子

表名: Proclib.Staff

この例では、WHERE 句の SOUNDS-LIKE 演算子の機能に基づいて行を返します。SOUNDS-LIKE 演算子は、発音が似ている語を識別する SOUNDEX アルゴリズムに基づいています。SOUNDEX アルゴリズムは、英語を前提にしているため、英語以外の言語にはあまり役立ちません。“SOUNDEX Function” (*SAS Functions and CALL Routines: Reference*)アルゴリズムの詳細については、*SAS 関数と CALL ルーチン: リファレンス*を参照してください。

詳細

```
proc sql outobs=10;
   title 'Proclib.Staff';
   title2 'First 10 Rows Only';
   select * from proclib.staff;
   title;
```

図 7.13 Proclib.Staff

PROCLIB.STAFF
First 10 Rows Only

idnum	lname	fname	city	state	hphone
1919	ADAMS	GERALD	STAMFORD	CT	203/781-1255
1653	ALIBRANDI	MARIA	BRIDGEPORT	CT	203/675-7715
1400	ALHERTANI	ABDULLAH	NEW YORK	NY	212/586-0808
1350	ALVAREZ	MERCEDES	NEW YORK	NY	718/383-1549
1401	ALVAREZ	CARLOS	PATERSON	NJ	201/732-8787
1499	BAREFOOT	JOSEPH	PRINCETON	NJ	201/812-5665
1101	BAUCOM	WALTER	NEW YORK	NY	212/586-8060
1333	BANADYGA	JUSTIN	STAMFORD	CT	203/781-1777
1402	BLALOCK	RALPH	NEW YORK	NY	718/384-2849
1479	BALLETTI	MARIE	NEW YORK	NY	718/384-8816

'Johnson'に似た発音の名前を選択するプログラム

```
libname proclib 'SAS-library';

proc sql;
   title "Employees Whose Last Name Sounds Like 'Johnson'";
   select idnum, upcase(lname), fname
      from proclib.staff

where lname=*"Johnson"
      order by 2;
```

プログラムの説明

Proclib ライブラリを宣言します。これらの例では、Proclib ライブラリは作成されたテーブルの格納に使用されます。

```
libname proclib 'SAS-library';
```

データの取得先の列とテーブルを選択します。SELECT 句によって、FROM 句のテーブル(Proclib.Staff)から、すべての列を選択します。

```
proc sql;
   title "Employees Whose Last Name Sounds Like 'Johnson'";
   select idnum, upcase(lname), fname
      from proclib.staff
```

クエリをサブセット化し、出力を並べ替えます。WHERE 句では、SOUNDS-LIKE 演算子を使用して、*Johnson* に似た発音の姓を持つ従業員によってテーブルをサブセット化しています。ORDER BY 句によって、2 番目の列で出力を並べ替えます。

```
where lname=*"Johnson"
```

```
   order by 2;
```

出力:'Johnson'に似た発音の名前

アウトプット 7.15 *Johnson 従業員テーブル*

Employees Whose Last Name Sounds Like 'Johnson'

idnum		fname
1411	JOHNSEN	JACK
1113	JOHNSON	LESLIE
1369	JONSON	ANTHONY

'Sanders'に似た発音の名前を選択するプログラム

SOUNDS-LIKE は役に立ちますが、条件を満たすと思われる行のうちの一部が返されない場合があります。Proclib.Staff には、SANDERS という姓を持つ従業員と、SANYERS という姓を持つ従業員が存在します。このアルゴリズムによって、SANYERS は検出されませんが、SANDERS と SANDERSON は検出されます。

```
proc sql;
title "Employees Whose Last Name Sounds Like 'Sanders'";
   select *
      from proclib.staff
      where lname=*"Sanders"
      order by 2;
```

出力:'Sanders'に似た発音の名前

アウトプット 7.16 *Sanders 従業員テーブル*

Employees Whose Last Name Sounds Like 'Sanders'

idnum	lname	fname	city	state	hphone
1561	SANDERS	RAYMOND	NEW YORK	NY	212/588-6615
1414	SANDERSON	NATHAN	BRIDGEPORT	CT	203/675-1715
1434	SANDERSON	EDITH	STAMFORD	CT	203/781-1333

例 12: 2 つのテーブルを結合して新しい値を計算する

要素: GROUP BY 句
HAVING 句

SELECT 句
ABS 関数
FORMAT=列修飾子
LABEL 列修飾子
MIN 要約関数
**演算子、累乗
SQRT 関数

表名: Stores
Houses

詳細

この例では、2 つのテーブルに対して一意だが、両方のテーブルに共通する列との関係がある値について比較して分析するために、これらのテーブルを結合します。

```
proc sql;
   title  'Stores Table';
   title2 'Coordinates of Stores';
   select * from stores;
   title  'Houses Table';
   title2 'Coordinates of Houses';
   select * from houses;
title;
```

これらのテーブルには、店舗と家の位置を表す X 座標と Y 座標が含まれています。

図 7.14 Stores テーブルと Houses テーブル

STORES Table
Coordinates of Stores

Store	x	y
store1	5	1
store2	5	3
store3	3	5
store4	7	5

HOUSES Table
Coordinates of Houses

House	x	y
house1	1	1
house2	3	3
house3	2	3
house4	7	7

プログラム

```
proc sql;
   title 'Each House and the Closest Store';
   select house, store label='Closest Store',
          sqrt((abs(s.x-h.x)**2)+(abs(h.y-s.y)**2)) as dist
              label='Distance' format=4.2
      from stores s, houses h

group by house
      having dist=min(dist);
```

プログラムの説明

クエリを指定します。 SELECT 句によって、3 つの列(HOUSE、STORE および DIST)を指定します。算術演算子では、平方ルート関数(SQRT)を使用して DIST の値を作成しています。これらの各行の値には、HOUSE から STORE までの距離が含まれます。二重アスタリスク(**)は、累乗を表します。LABEL=によって、STORE と DIST にラベルを割り当てます。

```
proc sql;
   title 'Each House and the Closest Store';
   select house, store label='Closest Store',
          sqrt((abs(s.x-h.x)**2)+(abs(h.y-s.y)**2)) as dist
              label='Distance' format=4.2
```

```
        from stores s, houses h
```

データをグループに編成し、クエリをサブセット化します。 データが家ごとにグループ化されるため、それぞれの家からすべての店舗までの最短距離が計算されます。HAVING 句では、各行を評価して、DIST の値と、その家から各店舗への最短距離とが同じかどうかを判定するように指定しています。

```
group by house
      having dist=min(dist);
```

出力:2 つのテーブルを結合して新しい値を計算する

house2 から最短距離にある店舗は、2 店舗あります。

アウトプット 7.17 *それぞれの家と、そこから最も近い店舗*

Each House and the Closest Store

House	Closest Store	Distance
house1	store1	4.00
house2	store2	2.00
house2	store3	2.00
house3	store3	2.24
house4	store4	2.00

例 13: 列内の値の使用可能な組み合わせをすべて作成する

要素: CASE 式
結合テーブルの構成要素
クロス結合
SELECT 句
DISTINCT キーワード

表名: Proclib.March
Flights

詳細

この例では、列の値のすべての可能な組み合わせを取得するために、テーブルとそのテーブル自身を結合します。

```
proc sql outobs=10;
   title 'Proclib.March';
   title2 'First 10 Rows Only';
   select * from proclib.march;

title;
```

図 7.15 Proclib.March

PROCLIB.MARCH
First 10 Rows Only

flight	date	depart	orig	dest	miles	boarded	capacity
114	01MAR08	7:10	LGA	LAX	2475	172	210
202	01MAR08	10:43	LGA	ORD	740	151	210
219	01MAR08	9:31	LGA	LON	3442	198	250
622	01MAR08	12:19	LGA	FRA	3857	207	250
132	01MAR08	15:35	LGA	YYZ	366	115	178
271	01MAR08	13:17	LGA	PAR	3635	138	250
302	01MAR08	20:22	LGA	WAS	229	105	180
114	02MAR08	7:10	LGA	LAX	2475	119	210
202	02MAR08	10:43	LGA	ORD	740	120	210
219	02MAR08	9:31	LGA	LON	3442	147	250

Flights テーブルを作成するプログラム

```
libname proclib 'SAS-library';

proc sql;
   create table flights as
      select distinct dest
         from proclib.march;

title 'Cities Serviced by the Airline';

select * from flights;
```

プログラムの説明

Proclib ライブラリを宣言します。これらの例では、Proclib ライブラリは作成されたテーブルの格納に使用されます。

```
libname proclib 'SAS-library';
```

Flights テーブルを作成します。CREATE TABLE ステートメントによって、クエリの出力から Flights テーブルを作成します。SELECT 句では、Dest の一意の値を選択しています。DISTINCT では、クエリから都市の値ごとにただ 1 つの行が返され、それが Flights テーブルに格納されることを指定しています。FROM 句では、選択対象のテーブルとして Proclib.March を指定しています。

```
proc sql;
   create table flights as
      select distinct dest
         from proclib.march;
```

タイトルを指定します。

```
title 'Cities Serviced by the Airline';
```

Flights テーブル全体を表示します。

```
select * from flights;
```

出力:Flights テーブルの作成

アウトプット 7.18 *Flights テーブル*

Cities Serviced by the Airline

dest
FRA
LAX
LON
ORD
PAR
WAS
YYZ

従来の結合を使用するプログラム

```
proc sql;

title 'All Possible Connections';

select f1.Dest, case
                      when f1.dest ne ' ' then 'to and from'
                    end,
          f2.Dest

from flights as f1, flights as f2

where f1.dest < f2.dest

order by f1.dest;
```

プログラムの説明

```
proc sql;
```

タイトルを指定します。

```
title 'All Possible Connections';
```

列を選択します。SELECT 句で、出力として 3 つの列を指定します。Dest に付加された接頭語は、Dest の値を取得するテーブルを指定するための、テーブルのエイリアスです。CASE 式によって、文字列"to and from"を含む列を作成します。

```
select f1.Dest, case
                      when f1.dest ne ' ' then 'to and from'
                    end,
```

```
        f2.Dest
```

結合のタイプを指定します。 FROM 句では、Flights とそれ自身を結合して、すべての可能な行の組み合わせを含むテーブル(デカルト積)を作成しています。このテーブルには、可能なルートごとに 2 つの行が含まれています。たとえば、PAR <-> WAS および WAS <-> PAR などです。

```
from flights as f1, flights as f2
```

結合条件を指定します。 WHERE 句では、F1.Dest の名前の順序が F2.Dest の名前よりも前になる行のみを選択することによって、内部テーブルをサブセット化しています。こうすることで、可能なルートごとに、だた 1 つの行が存在するようになります。

```
where f1.dest < f2.dest
```

出力の並べ替え ORDER BY によって、F1.Dest の値で結果を並べ替えます。

```
order by f1.dest;
```

出力:従来の結合

アウトプット 7.19 *すべての可能な接続*

All Possible Connections

dest		dest
FRA	to and from	YYZ
FRA	to and from	WAS
FRA	to and from	LAX
FRA	to and from	ORD
FRA	to and from	PAR
FRA	to and from	LON
LAX	to and from	LON
LAX	to and from	YYZ
LAX	to and from	WAS
LAX	to and from	PAR
LAX	to and from	ORD
LON	to and from	YYZ
LON	to and from	WAS
LON	to and from	PAR
LON	to and from	ORD
ORD	to and from	YYZ
ORD	to and from	WAS
ORD	to and from	PAR
PAR	to and from	YYZ
PAR	to and from	WAS
WAS	to and from	YYZ

クロス結合を使用するプログラム

```
proc sql;
   title 'All Possible Connections';
   select f1.Dest, case
                      when f1.dest ne ' ' then 'to and from'
                   end,
       f2.Dest
       from flights as f1 cross join flights as f2
       where f1.dest < f2.dest
       order by f1.dest;
```

プログラムの説明

クロス結合を指定します。 クロス結合はデカルト積結合と同じ機能であるため、クロス結合の構文は従来の結合の構文を代替できます。

```
proc sql;
   title 'All Possible Connections';
   select f1.Dest, case
                      when f1.dest ne ' ' then 'to and from'
                   end,
       f2.Dest
       from flights as f1 cross join flights as f2
       where f1.dest < f2.dest
       order by f1.dest;
```

出力:クロス結合

アウトプット 7.20 *すべての可能な接続*

All Possible Connections

dest		dest
FRA	to and from	YYZ
FRA	to and from	WAS
FRA	to and from	LAX
FRA	to and from	ORD
FRA	to and from	PAR
FRA	to and from	LON
LAX	to and from	LON
LAX	to and from	YYZ
LAX	to and from	WAS
LAX	to and from	PAR
LAX	to and from	ORD
LON	to and from	YYZ
LON	to and from	WAS
LON	to and from	PAR
LON	to and from	ORD
ORD	to and from	YYZ
ORD	to and from	WAS
ORD	to and from	PAR
PAR	to and from	YYZ
PAR	to and from	WAS
WAS	to and from	YYZ

例 14: ケース行とコントロール行の照合

要素: 結合テーブルの構成要素

表名: Match_11

Match

この例では、ケースコントロールスタディのデータを含むテーブルを使用します。それぞれの行には、ケースまたはコントロールの情報が含まれています。統計分析を実行するには、ケースとコントロールの組みごとに 1 つの行を持つテーブルが必要になりま

す。PROC SQL は、ケースとそれらに該当するコントロールを照合するために、テーブルとそのテーブル自身を結合します。行の照合が終わると、該当する行に対して差分処理を実行できます。

入力テーブル“Match_11” (434 ページ)には、ケースごとに 1 つの行と、コントロールごとに 1 つの行が含まれています。Pair には、ケースをそのコントロールに関連付ける番号が含まれています。コントロールの場合、Low は 0、ケースの場合、Low は 1 です。残りの列には、ケースとコントロールに関する情報が含まれています。

```
proc sql outobs=10;
    title 'Match_11 Table';
    title2 'First 10 Rows Only';
    select * from match_11;
```

図 7.16 *Match_11 テーブルの最初の 10 行*

MATCH_11 Table
First 10 Rows Only

Pair	Low	Age	Lwt	Race	Smoke	Ptd	Ht	UI	race1	race2
1	0	14	135	1	0	0	0	0	0	0
1	1	14	101	3	1	1	0	0	0	1
2	0	15	98	2	0	0	0	0	1	0
2	1	15	115	3	0	0	0	1	0	1
3	0	16	95	3	0	0	0	0	0	1
3	1	16	130	3	0	0	0	0	0	1
4	0	17	103	3	0	0	0	0	0	1
4	1	17	130	3	1	1	0	1	0	1
5	0	17	122	1	1	0	0	0	0	0
5	1	17	110	1	1	0	0	0	0	0

プログラム

```
proc sql;
   create table match as
      select
         one.Low,
         one.Pair,
         (one.lwt - two.lwt) as Lwt_d,
         (one.smoke - two.smoke) as Smoke_d,
         (one.ptd - two.ptd) as Ptd_d,
         (one.ht - two.ht) as Ht_d,
         (one.ui - two.ui) as UI_d

from match_11 one, match_11 two
      where (one.pair=two.pair and one.low>two.low);

   title 'Differences for Cases and Controls';
```

```
select *
      from match(obs=5);
```

プログラムの説明

Match テーブルを作成します。 SELECT 句によって、Match テーブルの列を指定します。SELECT 句の SQL 式によって、該当する列の差分を計算し、新しい行を作成します。

```
proc sql;
   create table match as
      select
         one.Low,
         one.Pair,
         (one.lwt - two.lwt) as Lwt_d,
         (one.smoke - two.smoke) as Smoke_d,
         (one.ptd - two.ptd) as Ptd_d,
         (one.ht - two.ht) as Ht_d,
         (one.ui - two.ui) as UI_d
```

結合のタイプと結合条件を指定します。 FROM 句では、Match_11 テーブルを 2 回記述しています。つまり、このテーブルは自分自身と結合されます。WHERE 句によって、それぞれの組みについて、ケースの値からコントロールの値を引いた差分を示す行のみが返されます。

```
from match_11 one, match_11 two
      where (one.pair=two.pair and one.low>two.low);
```

タイトルを指定します。

```
   title 'Differences for Cases and Controls';
```

Match テーブルの最初の 5 行を表示します。 SELECT 句によって、Match のすべての列を選択します。OBS=データセットオプションによって、出力の行数を 5 行に制限しています。

```
select *
      from match(obs=5);
```

出力:ケース行とコントロール行の照合

アウトプット 7.21 *ケースとコントロールの差分*

Differences for Cases and Controls

Low	Pair	Lwt_d	Smoke_d	Ptd_d	Ht_d	UI_d
1	1	-34	1	1	0	0
1	2	17	0	0	0	1
1	3	35	0	0	0	0
1	4	27	1	1	0	1
1	5	-12	0	0	0	0

例 15: SAS マクロを使用して欠損値をカウントする

要素: COUNT 関数

表名: Survey

この例では、SAS マクロを使用して列を作成します。SAS マクロについては、ここでは説明しません。SAS マクロの詳細については、*SAS マクロ言語: リファレンス*を参照してください。

"Survey" (446 ページ) には、食事と運動に関するアンケートのデータが含まれています。SAS では、欠損値に対して特殊表記を使用できます。EDUC 列の **.x** 表記は、回答者が無効な回答を返したことを示し、**.n** 表記は、回答者が質問に答えなかったことを示しています。ピリオドによる欠損値は、データ入力エラーを示しています。

プログラム

```
%macro countm(col);
    count(&col) "Valid Responses for &col",

nmiss(&col) "Missing or NOT VALID Responses for &col",

count(case
            when &col=.n  then "count me"
            end) "Coded as NO ANSWER for &col",
   count(case
            when &col=.x  then "count me"
            end) "Coded as NOT VALID answers for &col",
   count(case
            when &col=.  then "count me"
            end) "Data Entry Errors for &col"
%mend;

proc sql;
   title 'Counts for Each Type of Missing Response';
   select count(*)  "Total No. of Rows",
          %countm(educ)
```

```
        from survey;
```

プログラムの説明

欠損していない回答のカウント COUNTM マクロは、COUNT 関数を使用して、列に対するさまざまなカウントを実行します。それぞれの COUNT 関数は、CASE 式を使用してカウント対象の行を選択しています。最初の COUNT 関数は、引数として列のみを使用し、欠損していない行の数を返します。

```
%macro countm(col);
    count(&col) "Valid Responses for &col",
```

欠損しているか無効の回答をカウントします。 NMSS 関数は、いずれかのタイプの欠損値(.n、.x またはピリオド)を列に含む行の数を返します。

```
nmiss(&col) "Missing or NOT VALID Responses for &col",
```

さまざまな入力について、欠損しているか無効の回答の出現回数をカウントします。 最後の 3 つの COUNT 関数は、CASE 式を使用して、欠損値の 3 つの表記の出現回数をカウントしています。“count me”文字列によって、カウントする非欠損値を COUNT 関数に渡しています。

```
count(case
            when &col=.n  then "count me"
            end) "Coded as NO ANSWER for &col",
   count(case
            when &col=.x  then "count me"
            end) "Coded as NOT VALID answers for &col",
   count(case
            when &col=.  then "count me"
            end) "Data Entry Errors for &col"
%mend;
```

COUNTM マクロを使用して列を作成します。 SELECT 句によって、出力に含める列を指定します。COUNT(*)によって、テーブル内の行の総数が返されます。COUNTM マクロは、EDUC 列の値を使用して、マクロで定義されている列を作成します。

```
proc sql;
   title 'Counts for Each Type of Missing Response';
   select count(*)  "Total No. of Rows",
          %countm(educ)
      from survey;
```

出力:SAS マクロを使用して欠損値をカウントする

アウトプット 7.22 *欠損している回答の種類ごとのカウント*

Counts for Each Type of Missing Response

Total No. of Rows	Valid Responses for educ	Missing or NOT VALID Responses for educ	Coded as NO ANSWER for educ	Coded as NOT VALID answers for educ	Data Entry Errors for educ
8	2	6	1	3	2

8 章

SQL プロシジャの構成要素

概要 **311**

ディクショナリ **312**
BETWEEN 条件 312
BTRIM 関数 312
CALCULATED 313
CASE 式 314
COALESCE 関数 315
column-definition 316
column-modifier 317
column-name 319
CONNECTION TO 320
CONTAINS 条件 320
EXISTS 条件 321
IN 条件 321
IS 条件 322
joined-table 323
LIKE 条件 337
LOWER 関数 339
query-expression 339
sql-expression 347
SUBSTRING 関数 355
summary-function 356
table-expression 364
UPPER 関数 365

概要

このセクションでは、SQL プロシジャステートメントで使用される構成要素について説明します。構成要素は、PROC SQL 構文においてローマン体で表記される項目です。

ほとんどの構成要素は、ステートメント内の句に含まれます。たとえば、基本的な SELECT ステートメントには SELECT 句と FROM 句が含まれ、これらの句には 1 つ以上の構成要素が含まれます。構成要素には、他の構成要素を含めることもできます。

参照を容易にするために、構成要素はアルファベット順に記載されています。そのため、一部の用語は、定義される前に参照されています。インデックスまたは"関連項目"を参照して、他のステートメントまたは構成要素の説明を参照すると、役立つ場合があります。

ディクショナリ

BETWEEN 条件

列の値が値の範囲に含まれる場合に、その行を選択します。

構文

sql-expression <NOT> **BETWEEN** *sql-expression*
AND *sql-expression*

必須引数

sql-expression
“sql-expression” (347 ページ)を参照してください。

詳細

- 各 SQL 式は、それぞれ互換性のあるデータタイプである必要があります。それらは、すべて数値タイプであるか、すべて文字タイプである必要があります。
- BETWEEN 条件では、指定した 2 つの境界値が範囲として評価されます。このため、大きい値を先に指定してもかまいません。
- NOT 論理演算子を使用して、数値の範囲を除外できます。たとえば、最近獲得した顧客のデータを取得できるようにするために、1 から 15 までの顧客番号を除去できます。
- PROC SQL は、DATA ステップがサポートするのと同じ比較演算子をサポートします。たとえば、次のように指定します。

```
x between 1 and 3
x between 3 and 1
1<=x<=3
x>=1 and x<=3
```

BTRIM 関数

文字列の先頭、末尾またはその両方から、空白または指定した文字を削除します。

構文

BTRIM(<<*btrim-specification*> <'*btrim-character*' FROM>> *sql-expression*)

必須引数

sql-expression
文字列または文字変数へと展開される必要があります。

参照項目 “sql-expression” (347 ページ).

オプション引数

btrim-specification
次のいずれかを指定できます。

LEADING
文字列の先頭から、空白または指定した文字を削除します。

TRAILING
文字列の末尾から、空白または指定した文字を削除します。

BOTH
文字列の先頭と末尾の両方から、空白または指定した文字を削除します。

デフォルト BOTH

btrim-character
文字列から削除される 1 つの文字。デフォルトの文字は、空白です。

詳細

BTRIM 関数は、文字列を操作します。BTRIM は、LEADING、TRAILING または BOTH のいずれが指定されたかに応じて、文字列の先頭、末尾またはその両方から、btrim-character で指定された 1 つの文字を検出するとその文字を削除します。btrim-specification を指定しない場合、BOTH が使用されます。btrim-character を省略した場合、空白が削除されます。

注: SAS では、変数の長さよりも短い文字値の後には、空白が追加されます。長さが 10 で、**xxabcxx** という値を持つ文字変数 Z があるとします。SAS は、長さを 10 にするために、最後の x の後に 3 つの空白を追加して値を格納します。を使用してすべての x の文字を削除しようとしても、

```
btrim(both 'x' from z)
```

その結果は **abcxx** になります。これは、PROC SQL が、末尾の文字を x ではなく空白であると認識するためです。すべての x の文字を削除するには、を使用します。

```
btrim(both 'x' from btrim(z))
```

内側の BTRIM 関数によって末尾の空白が削除され、次にその値が外側の BTRIM 関数に渡されます。

CALCULATED

SELECT 句内ですでに計算されている列を参照します。

構文

CALCULATED *column-alias*

必須引数

column-alias
SELECT 句内の列に割り当てられた名前。

詳細

CALCULATED によって、同じ SELECT 句内または WHERE 句内の式の結果を使用できます。隣接するクエリ式で計算された列の参照に使用された場合にのみ有効です。

CASE 式

指定された条件を満たす結果の値を選択します。

例: “例 3: PROC SQL テーブルのデータの更新” (271 ページ)
“例 13: 列内の値の使用可能な組み合わせをすべて作成する” (300 ページ)

構文

CASE <*case-operand*>
WHEN *when-condition* **THEN** *result-expression*
<WHEN *when-condition* THEN *result-expression* …>
<ELSE *result-expression*>
END

必須引数

when-condition

- *case-operand* を指定した場合、*when-condition* は、*case-operand* をそのオペランドの 1 つとみなし、TRUE または FAlSE を決定する短縮された SQL 式です。
- *case-operand* を指定しない場合、*when-condition* は、TRUE または FALSE を決定する SQL 式です。

result-expression
値を決定する SQL 式。

参照項目 “sql-expression” (347 ページ).

オプション引数

case-operand
テーブル列に展開される有効な SQL 式。このテーブル列の値は、すべての *when-conditions* に対して比較されます。

参照項目 “sql-expression” (347 ページ).

詳細

CASE 式は、特定の条件が満たされた場合に値を選択します。テーブルまたはビューの各行ごとに条件を評価して 1 つの値を返します。照会または作成するテーブル内の一部の行だけに CASE 式を実行するときは、WHEN-THEN 句を使用します。THEN 式が実行されない場合の代替アクションはオプションの ELSE 式で指定します。

CASE オペランド(case-operand)を省略すると、WHEN 条件(when-condition)はブール値(TRUE または FALSE)として評価されます。WHEN 条件からゼロ以外の非欠損値が返された場合は、WHEN 句は TRUE になります。CASE オペランドを指定した場合

は、その値と WHEN 条件が等価かどうかが比較されます。CASE オペランドと WHEN 条件が等価の場合は、WHEN 句は TRUE になります。

実行する行の WHEN 条件が TRUE の場合は、THEN の後の結果式が実行されます。WHEN 条件が FALSE の場合は、後続の WHEN 条件が順番に評価されていきます。すべての WHEN 条件が FALSE の場合は、ELSE 式が実行され、その結果が CASE 式の結果になります。WHEN 条件がすべて FALSE で、ELSE 句が指定されていない場合は、CASE 式の結果は欠損値になります。

CASE 式を、SELECT 句の項目および SQL 式のどちらかのオペランドとして使用できます。

例

次の 2 つの PROC SQL ステップは、THEN 句で文字列を使用して文字型列を作成する、2 つの等価な CASE 式を示しています。2 番目の PROC SQL ステップの CASE 式は、同じ列に対してすべての比較を行う場合に役立つ、省略方法を示しています。

```
proc sql;
   select Name, case
                when Continent = 'North America' then 'Continental U.S.'
                when Continent = 'Oceania' then 'Pacific Islands'
                else 'None'
                end as Region
      from states;

proc sql;
   select Name, case Continent
                when 'North America' then 'Continental U.S.'
                when 'Oceania' then 'Pacific Islands'
                else 'None'
                end as Region
      from states;
```

注: この省略方法を使用する場合、すべての条件を等式で比較する必要があります。つまり、比較演算子などの、他の種類の演算子は使用できません。

COALESCE 関数

列のリストの最初の非欠損値を返します。

例: “例 7: 外部結合を実行する” (281 ページ)

構文

COALESCE (*column-name-1* <, *column-name-2*, …>)

必須引数

column-name
“column-name” (319 ページ)を参照してください。

詳細

COALESCE は、同じデータタイプの 1 つ以上の列名を受け取ります。COALESCE 関数は、記述された順序で各列の値をチェックし、最初の非欠損値を返します。1 つの列のみを記述した場合、COALESCE 関数はその列の値を返します。すべての引数のすべての値が欠損している場合、COALESCE 関数は 1 つの欠損値を返します。

一部の SQL DBMS では、COALESCE 関数は IFNULL 関数と呼ばれます。詳細については、“PROC SQL および ANSI 規格” (383 ページ)を参照してください。

注: クエリに多数の COALESCE 関数呼び出しが含まれる場合、かわりに自然結合を使用した方が効果的な場合があります。“自然結合” (332 ページ)を参照してください。

column-definition

PROC SQL のデータタイプと日付を定義します。

参照項目: “column-modifier” (317 ページ)

例: “例 1: テーブルを作成し、データを挿入する” (267 ページ)

構文

column data-type <column-modifier(s)>

必須引数

column
列名。

data-type
次のいずれかのデータタイプです。

CHARACTER|VARCHAR <(*width*)>
列幅が *width* の文字列を指定します。デフォルトの列幅は、8 文字です。

INTEGER|SMALLINT
整数列を指定します。

DECIMAL|NUMERIC|FLOAT <(*width*<, *ndec*>)>
列幅が *width* で、小数点の位置が *ndec* である、浮動小数点列を指定します。

REAL|DOUBLE PRECISION
浮動小数点列を指定します。

DATE
日付列を指定します。

オプション引数

column-modifier
“column-modifier” (317 ページ)を参照してください。

詳細

- SAS は、SQL ベースのデータベースがサポートするデータタイプの多くをサポートしますが、サポートされないデータタイプもあります。

- データタイプが数値(INTEGER、SMALLINT、DECIMAL、NUMERIC、FLOAT、REAL、DOUBLE PRECISION、DATE)の場合、SQL プロシジャは、それらすべてを SAS の NUMERIC データタイプにデフォルトで設定します。引数の width と ndec は無視されます。PROC SQL は、SAS で可能な最大精度を使用して、すべての数値列を作成します。使用するディスク領域が少ない数値列を作成する場合は、DATA ステップで LENGTH ステートメントを使用してください。width と ndec の引数に加えて、他の SQL ソフトウェアとの互換性を保つために、さまざまな数値データタイプ名が用意されています。
- データタイプが文字(CHARACTER および VARCHAR)の場合、SQL プロシジャは、それらを SAS の CHARACTER データタイプにデフォルトで設定します。width 引数は、そのまま適用されます。
- CHARACTER、INTEGER および DECIMAL の各データタイプは、それぞれ CHAR、INT および DEC に省略できます。
- DATE を使用して宣言された列は、日付入力形式または日付出力形式が設定された SAS 数値変数になります。任意の column-modifier を使用して、定義済みの列に適切な属性を設定できます。日付の詳細については、*SAS 出力形式と入力形式: リファレンス*を参照してください。
- Oracle データベースの VARCHAR2 データタイプ、または Greenplum データベースおよび Aster データベースの VARCHAR データタイプを使用する場合、列の値の後に空白を含めないでください。データベースによっては、VARCHAR2 データタイプと VARCHAR データタイプの値の後の空白は意味を持つと見なされます。そのため、結果が不正になる場合や、生成されるクエリの効率が悪くなる場合があります。

column-modifier

列属性を設定します。

参照項目: “column-definition” (316 ページ)
“SELECT 句” (252 ページ)

例: “例 1: テーブルを作成し、データを挿入する” (267 ページ)
“例 2: テーブルをクエリの結果から作成する” (269 ページ)

構文

column-modifier

必須引数

column-modifier
次のいずれかを指定できます。

INFORMAT=*informatw.d*
SAS がテーブルまたはビューのデータにアクセスするときに使用する、SAS 入力形式を指定します。永久入力形式の変更には ALTER ステートメントを使用します。入力形式は、SQL プロシジャのテーブル定義に格納されます。したがって、SQL プロシジャで作成したテーブルをほかの SAS プロシジャや DATA ステップで参照するときは、この入力形式の情報を使用することができます。

入力形式の詳細については、*SAS 出力形式と入力形式: リファレンス*を参照してください。

FORMAT=*formatw.d*
クエリ式による列の文字値と数値の表示方法を決める、SAS 出力形式を指定します。ALTER、CREATE TABLE または CREATE VIEW ステートメントで FORMAT=修飾子を使用した場合、SAS がそのテーブルまたはビューのデータを表示するときに使用する、永久出力形式を指定します。ある永久出力形式を、ALTER ステートメントを使用して別の出力形式に変更できます。

出力形式の詳細については、*SAS 出力形式と入力形式: リファレンス*を参照してください。

LABEL='*label*'
列ラベルを指定します。LABEL=修飾子を ALTER、CREATE TABLE または CREATE VIEW ステートメントで使用した場合、その列を表示するときに使用される永久ラベルを指定します。永久ラベルの変更には ALTER ステートメントを使用します。

ラベルの先頭文字として、a から z、A から Z、0 から 9、アンダーライン(_)、空白を使用できます。ラベルの先頭文字を、番号記号(#)などの他の文字にした場合、その文字は区切り文字として使用されます。その場合ラベルは、表示される際に、必ず次の行に改行されます。たとえば、次のように指定します。

```
select dropout label= '#Percentage of#Students
Who#Dropped Out' from educ(obs=5);
```

出力の先頭文字として特殊文字を表示する必要がある場合、その文字の前にスペースまたはスラッシュ(/)を挿入する必要があります。

column-modifier の LABEL=の部分を省略しても、ラベルを指定できます。その場合、次の例のように、必ずラベルを引用符で囲んでください。`select empname "Names of Employees" from sql.employees;`

ラベルにアポストロフィを表示する必要がある場合、SAS がアポストロフィをリテラルとして読み取るようにするために、アポストロフィを 2 回入力します。あるいは、単一引用符と二重引用符を交互に使用します(たとえば、“Date Rec'd”)。

LENGTH=*length*
列の長さを指定します。この列修飾子は、SELECT ステートメントのコンテキストでのみ有効です。

TRANSCODE=YES|NO
文字列に対して、値をトランスコードできるかどうかを指定します。トランスコードを抑制する場合は、TRANSCODE=NO を使用します。CREATE TABLE AS ステートメントを使用してテーブルを作成した場合、作成したテーブルの特定の文字列のトランスコード属性は、TRANSCODE=列修飾子を使用して変更しない限り、元のテーブルのトランスコード属性と同じになります。トランスコードの詳細については、*SAS 各国語サポート(NLS): リファレンスガイド*を参照してください。

デフォルト YES

制限事項 一部の SAS Workspace Server のクライアントでは、TRANSCODE=NO の引数はサポートされていません。SAS 9.2 では、この引数がサポートされない場合、TRANSCODE=NO が設定された列の値はアスタリスク(*)で置き換えられます(つまりマスクされます)。SAS 9.2 より前は、TRANSCODE=NO が設定された列の値はトランスコードされていました。

V6TAPE エンジンでは、トランスコードの抑制はサポートされていません。

操作 テーブル内のいずれかの文字変数に対して TRANSCODE=属性を NO に設定した場合、PROC CONTENTS は、データセット内の変数ごとの TRANSCODE=の値を含むトランスコード列を出力します。テーブル内のすべての変数を TRANSCODE=のデフォルトの値(YES)に設定した場合、トランスコード列は出力されません。

詳細

ラベルが設定された列を ORDER BY 句または GROUP BY 句で参照する場合、列名(列ラベルではない)、列のエイリアスまたは列の位置番号(整数)のいずれかを指定する必要があります(たとえば、`ORDER BY 2`)。ラベルの詳細については、*SAS ステートメント: リファレンス*の SAS ステートメントに関するセクションを参照してください。

column-name

選択する列を指定します。

参照項目: “column-modifier” (317 ページ)
“SELECT 句” (252 ページ)

構文

column-name

必須引数

column-name
次のいずれかを指定できます。

column
列の名前。

table-name.column
table-name テーブルの列の名前。

table-alias.column
table-alias が参照するテーブルの列の名前。

view-name.column
view-name ビューの列の名前。

view-alias.column
view-alias が参照するビューの列の名前。

詳細

列の名前が、現在のクエリ式に記述されているすべてのテーブルまたはビューの中で重複していない場合、列を名前のみで参照できます。クエリ式内の 2 つ以上のテーブルまたはビューに同じ列名が存在する場合、その列を含むテーブルへの参照を接頭語として列名に付加することによって、修飾した列名を使用する必要があります。次に例を示します。

```
SALARY       /* name of the column */
EMP.SALARY   /* EMP is the table or view name */
E.SALARY     /* E is an alias for the table
                or view that contains the
```

```
SALARY column */
```

CONNECTION TO

PROC SQL クエリまたは PROC SQL ビューでの DBMS データの取得と使用

ヒント: SELECT ステートメントの FROM 句で、FROM リストの一部として CONNECTION TO を使用できます。

参照項目: “パススルー機能を使用した DBMS への接続” (172 ページ)
SAS/ACCESS のマニュアル

構文

CONNECTION TO *dbms-name (dbms-query)*
CONNECTION TO *alias (dbms-query)*

必須引数

dbms-name
使用している DBMS を指定します。

dbms-query
DBMS に送信するクエリを指定します。クエリは、DBMS の動的 SQL を使用します。DBMS が理解できる SQL 構文であれば、PROC SQL では無効であっても、任意の SQL 構文を使用できます。たとえば、DBMS のクエリにセミコロンを含めることができます。

dbms-query を使用して結合できるテーブルの数は、DBMS によって決められます。それぞれの CONNECTION TO 構成要素は、1 個から 256 個(PROC SQL での結合の上限)までのテーブルとしてカウントされます。

DBMS のクエリの詳細については、*SAS/ACCESS for Relational Databases: Reference* を参照してください。

alias
CONNECT ステートメントでエイリアスを定義した場合、そのエイリアスを指定します。

CONTAINS 条件

文字列が列の値の一部であるかどうかをテストします。

別名: ?

制限事項: CONTAINS 条件は、文字オペランドでのみ使用されます。

例: “例 7: 外部結合を実行する” (281 ページ)

構文

sql-expression <NOT> **CONTAINS** *sql-expression*

必須引数

sql-expression
"sql-expression" (347 ページ)を参照してください。

EXISTS 条件

サブクエリが 1 つ以上の行を返すかどうかをテストします。

参照項目: "他の式(サブクエリ)" (351 ページ)

構文

<NOT> **EXISTS** (*query-expression*)

必須引数

query-expression
"query-expression" (339 ページ)を参照してください。

詳細

EXISTS 条件とは、右オペランドがサブクエリになっている演算子のことです。EXISTS 条件の結果は、サブクエリが少なくとも 1 つの行を決定した場合、TRUE になります。NOT EXISTS 条件の結果は、サブクエリがゼロ行と評価した場合、TRUE になります。たとえば、次のクエリは、サブクエリの条件に基づいて Proclib.Payrol をサブセット化しています。"例 2: テーブルをクエリの結果から作成する" (269 ページ)を参照してください。Staff.Idnum の値が Proclib.Staff で値 CT と同じ行にある場合、その値と一致する Proclib.Payroll 内の Idnum の値が出力に含まれます。"例 4: 2 つのテーブルを結合する" (273 ページ)を参照してください。したがって、クエリは、CT に住んでいるすべての従業員を Proclib.Payroll から返します。

```
proc sql;
  select *
    from proclib.payroll p
    where exists (select *
                    from proclib.staff s
                    where p.idnumber=s.idnum
                          and state='CT');
```

IN 条件

集合のメンバをテストします。

例: "例 4: 2 つのテーブルを結合する" (273 ページ)

構文

sql-expression <NOT> **IN** (*query-expression* | *constant-1*<, *constant-2*, …>)

必須引数

sql-expression
"sql-expression" (347 ページ)を参照してください。

query-expression
"query-expression" (339 ページ)を参照してください。

constant
固定値を示す数または引用符で囲まれた文字列(または他の特殊表記)です。定数は *literal* とも呼ばれます。

詳細

IN 条件は、左側の SQL 式が返す列の値が、右側の集合(定数またはクエリ式が返す値の集合)のメンバであるかどうかをテストします。IN 条件は、左側のオペランドの値が右側のオペランドによって定義された値の集合に含まれる場合、TRUE になります。

IS 条件

欠損値をテストします。

例: "例 5: 2 つのテーブルを組み合わせる" (276 ページ)

構文

sql-expression **IS** <NOT> **NULL** | **MISSING**

必須引数

sql-expression
"sql-expression" (347 ページ)を参照してください。

詳細

IS NULL と IS MISSING は、欠損値をテストする述語です。IS NULL と IS MISSING は、WHERE 式、ON 式および HAVING 式で使用されます。SQL 式の結果が欠損している場合、それぞれの述語は TRUE に決定されます。欠損していない場合は、FALSE に決定されます。

SAS では、数値の欠損値をピリオド(.)で格納され、文字の欠損値は空白で格納されます。SQL の一部のバージョンの欠損値とは異なり、SAS の欠損値は、必ず照合順序の先頭に現れます。したがって、ブール演算と比較演算では、次のような式は述語において TRUE に決定されます。

```
 3>null
-3>null
 0>null
```

欠損値を評価する SAS の方法は、ANSI 規格の SQL の方法とは異なります。規格に従った場合、これらの式は NULL になります。述語と演算子の詳細については、"sql-expression" (347 ページ)を参照してください。ANSI 規格の詳細については、付録 2, "PROC SQL および ANSI 規格" (383 ページ)を参照してください。

joined-table

テーブルを、そのテーブル自身あるいは他のテーブルまたはビューと結合します。

制限事項: 結合できる最大テーブル数は 256 個です。

参照項目: “FROM 句” (259 ページ)
“query-expression” (339 ページ)

例: “例 4: 2 つのテーブルを結合する” (273 ページ)
“例 7: 外部結合を実行する” (281 ページ)
“例 9: 3 つのテーブルを結合する” (289 ページ)
“例 13: 列内の値の使用可能な組み合わせをすべて作成する” (300 ページ)
“例 14: ケース行とコントロール行の照合” (306 ページ)

構文

table-name-1 <<AS> *alias-1*>, *table-name-2* <<AS> *alias-2*>
<, *table-name-3* <<AS> *alias-3*, …>>

table-name-1 <<AS> *alias-1*> <INNER> **JOIN** *table-name-2* <<AS> *alias-2*>
ON *sql-expression*

table-name-1 <<AS> *alias-1*> **LEFT JOIN** | **RIGHT JOIN** | **FULL JOIN**
table-name-2 <<AS> *alias-2*> **ON** *sql-expression*

table-name-1 <<AS> *alias-1*> **CROSS JOIN** *table-name-2* <<AS> *alias-2*>

table-name-1 <<AS> *alias-1*> **UNION JOIN** *table-name-2* <<AS> *alias-2*>

table-name-1 <<AS> *alias-1*> **NATURAL**
<INNER | FULL <OUTER> | LEFT <OUTER> | RIGHT <OUTER>> **JOIN** *table-name-2*
<<AS> *alias-2*>

必須引数

table-name
次のいずれかを指定できます。

- PROC SQL テーブルの名前。
- SAS ビューまたは PROC SQL ビューの名前。
- クエリ式。通常、FROM 句内のクエリ式は、インラインビューと呼ばれます。インラインビューの詳細については、“FROM 句” (259 ページ)を参照してください。
- CONNECTION TO 構成要素のフォームでの DBMS への接続。詳細については、“CONNECTION TO” (320 ページ)を参照してください。

table-name としては、1 レベルの名前、2 レベルの *libref.table* の名前または単一引用符で囲まれた物理パス名が可能です。

注: かっこを含める場合、必ず組みで含めてくださ。カンマで結合を(種類)のように囲むことは無効です。

sql-expression
“sql-expression” (347 ページ)を参照してください。

オプション引数

alias
table-name のエイリアスを指定します。AS キーワードは任意です。詳細については、“CALCULATED キーワードと列エイリアスの使用” (150 ページ)を参照してください。

詳細

結合の種類

- 内部結合。“内部結合” (325 ページ)を参照してください。
- 外部結合。“外部結合” (328 ページ)を参照してください。
- クロス結合。“クロス結合” (330 ページ)を参照してください。
- 和結合。“和結合” (331 ページ)を参照してください。
- 自然結合。“自然結合” (332 ページ)を参照してください。

テーブルの結合

FROM 句に複数のテーブル、ビューまたはクエリ式を記述した場合、それらが処理されて 1 つのテーブルが作成されます。結果のテーブルには、寄与している各テーブルのデータが含まれます。これらのクエリは、結合と呼ばれます。

概念的に説明すると、2 つのテーブルを指定した場合、テーブル A の各行がテーブル B のすべての行と組み合されて、内部テーブル(中間テーブル)が生成されます。中間テーブル(デカルト積)の行の数は、元の各テーブルの行数の積に等しくなります。中間テーブルは、他のクエリの入力になります。クエリでは、中間テーブルの行を WHERE 句によって一部を削除したり、要約関数によって要約したりできます。

結合の通常のタイプは、等結合です。等結合では、1番目のテーブルのある列の値が、2 番目のテーブルのある列の値と等しい必要があります。

テーブルの制限

SQL プロシジャでは、最大 256 個のテーブルを結合できます。結合でビューを使用する場合、ビューの元になるテーブルの数は、上限の 256 テーブルまでカウントされます。パススルー機能では、それぞれの CONNECTION TO 構成要素は 1 テーブルとしてカウントされます。

戻り行の指定

WHERE 句または ON 句には、条件(SQL 式)が含まれます。デカルト積の行は、それらの条件に基づいて、結果テーブルにおいて維持または削除されます。WHERE 句は、内部結合の行の選択に使用されます。ON 句は、内部結合または外部結合の行の選択に使用されます。

式は、“テーブルの結合” (324 ページ)で前述したそれぞれの中間テーブルの行ごとに評価されます。式の結果が TRUE(ゼロ以外の非欠損値)である場合、その行は一致していると見なされます。

注: クエリの結果をさらにサブセット化するために、ON 句の後に WHERE 句を記述できます。例については、“例 7: 外部結合を実行する” (281 ページ) を参照してください。

テーブルのエイリアス

結合において、あるテーブルの列を他の 1 つ以上のテーブルの列と区別するには、テーブルのエイリアスを使用します。同じ列名を含む複数のテーブルを結合する場合、

列名にテーブル名またはエイリアスの接頭語を付加する必要があります。テーブルのエイリアスの詳細については、“FROM 句” (259 ページ)を参照してください。

同じテーブル同士の結合

1 つのテーブルをそのテーブル自身と結合し、さらに多くの情報を生成できます。これらの結合は、再帰結合と呼ばれる場合もあります。これらの結合では、同じテーブルが FROM 句に 2 回記述されます。それぞれのテーブルのインスタンスには、テーブルのエイリアスを設定する必要があります。そうしないと、テーブルの 2 つのインスタンスの列への参照を区別できません。例については、“例 13: 列内の値の使用可能な組み合わせをすべて作成する” (300 ページ) および“例 14: ケース行とコントロール行の照合” (306 ページ)を参照してください。

内部結合

内部結合は、SQL 式で指定したとおりに、他のテーブル内に 1 つ以上の一致する行が存在する、あるテーブル内のすべての行を結果テーブルとして返します。内部結合は、同じクエリ式内で、最大で 256 個までのテーブルに対して実行できます。

カンマで区切られた table-name のリストを使用するか、INNER JOIN キーワードおよび ON キーワードを使用することによって、内部結合を実行できます。

次では、Lefttab テーブルおよび Righttab テーブルを使用して、このタイプの結合を説明します。

```
data lefttab;
   input Continent $ Export $ Country $;
   datalines;
NA   wheat Canada
EUR  corn  France
EUR  rice  Italy
AFR  oil   Egypt
;

data righttab;
  input Continent $ Export $ Country $;
   datalines;
NA   sugar USA
EUR  corn  Spain
EUR  beets Belgium
ASIA rice  Vietnam
;

proc sql;
   title 'Left Table - Lefttab';
   select * from lefttab;

   title 'Right Table - Righttab';
   select * from righttab;
```

アウトプット 8.1 *Lefttab テーブルと Righttab テーブル*

Left Table - Lefttab

Continent	Export	Country
NA	wheat	Canada
EUR	corn	France
EUR	rice	Italy
AFR	oil	Egypt

Right Table - Righttab

Continent	Export	Country
NA	sugar	USA
EUR	corn	Spain
EUR	beets	Belgium
ASIA	rice	Vietnam

次の例では、Lefttab テーブルと Righttab テーブルを結合して、2 つのテーブルのデカルト積を生成しています。デカルト積は、あるテーブルのすべての行を、別のテーブルのすべての行と結合した結果です。2 つのテーブルを結合し、WHERE 句または ON 句を使用してそれらをサブセット化しない場合、デカルト積が生成されます。

```
proc sql;
   title 'The Cartesian Product of';
   title2 'Lefttab and Righttab';
   select *
      from lefttab, righttab;
```

アウトプット 8.2 *Lefttab テーブルと Righttab テーブルのデカルト積*

The Cartesian Product of LEFTTAB and RIGHTTAB

Continent	Export	Country	Continent	Export	Country
NA	wheat	Canada	NA	sugar	USA
NA	wheat	Canada	EUR	corn	Spain
NA	wheat	Canada	EUR	beets	Belgium
NA	wheat	Canada	ASIA	rice	Vietnam
EUR	corn	France	NA	sugar	USA
EUR	corn	France	EUR	corn	Spain
EUR	corn	France	EUR	beets	Belgium
EUR	corn	France	ASIA	rice	Vietnam
EUR	rice	Italy	NA	sugar	USA
EUR	rice	Italy	EUR	corn	Spain
EUR	rice	Italy	EUR	beets	Belgium
EUR	rice	Italy	ASIA	rice	Vietnam
AFR	oil	Egypt	NA	sugar	USA
AFR	oil	Egypt	EUR	corn	Spain
AFR	oil	Egypt	EUR	beets	Belgium
AFR	oil	Egypt	ASIA	rice	Vietnam

FROM 句に Lefttab テーブルと Righttab テーブルの名前を記述することによって、それらを結合できます。次のクエリは、各テーブルの Continent の値が照合されるため、等結合を表しています。正しい列を選択できるようにするために、列名にはテーブルのエイリアスの接頭語が付加されています。

```
proc sql;
   title 'Inner Join';
   select *
      from lefttab as l, righttab as r
      where l.continent=r.continent;
```

アウトプット 8.3 *内部結合*

Inner Join

Continent	Export	Country	Continent	Export	Country
NA	wheat	Canada	NA	sugar	USA
EUR	corn	France	EUR	corn	Spain
EUR	corn	France	EUR	beets	Belgium
EUR	rice	Italy	EUR	corn	Spain
EUR	rice	Italy	EUR	beets	Belgium

次の PROC SQL ステップは、前述の PROC SQL ステップと等価であり、INNER JOIN キーワードおよび ON キーワードを使用して等結合を記述する方法を示しています。

```
proc sql;
   title 'Inner Join';
   select *
      from lefttab as l inner join
           righttab as r
      on l.continent=r.continent;
```

関連項目:

例

- “例 4: 2 つのテーブルを結合する” (273 ページ)
- “例 13: 列内の値の使用可能な組み合わせをすべて作成する” (300 ページ)
- “例 14: ケース行とコントロール行の照合” (306 ページ)

外部結合

外部結合は、結合において他のテーブルのどの行とも一致しなかった行が追加された内部結合です。外部結合には、左、右および完全の3種類があります。

LEFT JOIN キーワードと ON キーワードで指定する左外部結合には、2 つのテーブルのデカルト積のうちの SQL 式が TRUE となるすべての行に加えて、2 番目のテーブル(Righttab)のどの行とも一致しない 1 番目のテーブル(Lefttab)の行が含まれます。

```
proc sql;
   title 'Left Outer Join';
   select *
      from lefttab as l left join
           righttab as r
      on l.continent=r.continent;
```

アウトプット 8.4　左外部結合

Left Outer Join

Continent	Export	Country	Continent	Export	Country
AFR	oil	Egypt			
EUR	rice	Italy	EUR	beets	Belgium
EUR	corn	France	EUR	beets	Belgium
EUR	rice	Italy	EUR	corn	Spain
EUR	corn	France	EUR	corn	Spain
NA	wheat	Canada	NA	sugar	USA

RIGHT JOIN キーワードと ON キーワードで指定する右外部結合には、2 つのテーブルのデカルト積のうちの SQL 式が TRUE となるすべての行に加えて、1 番目のテーブル(Lefttab)のどの行とも一致しない 2 番目のテーブル(Righttab)の行が含まれます。

```
proc sql;
   title 'Right Outer Join';
   select *
      from lefttab as l right join
           righttab as r
      on l.continent=r.continent;
```

アウトプット 8.5　右外部結合

Right Outer Join

Continent	Export	Country	Continent	Export	Country
			ASIA	rice	Vietnam
EUR	rice	Italy	EUR	beets	Belgium
EUR	rice	Italy	EUR	corn	Spain
EUR	corn	France	EUR	beets	Belgium
EUR	corn	France	EUR	corn	Spain
NA	wheat	Canada	NA	sugar	USA

FULL JOIN キーワードと ON キーワードで指定する完全外部結合には、2 つのテーブルのデカルト積のうちの SQL 式が TRUE となるすべての行に加え、他のテーブルのどの行とも一致しないそれぞれのテーブルの行が含まれます。

```
proc sql;
   title 'Full Outer Join';
   select *
      from lefttab as l full join
           righttab as r
```

```
      on l.continent=r.continent;
```

アウトプット 8.6 *完全外部結合*

Full Outer Join

Continent	Export	Country	Continent	Export	Country
AFR	oil	Egypt			
			ASIA	rice	Vietnam
EUR	rice	Italy	EUR	beets	Belgium
EUR	rice	Italy	EUR	corn	Spain
EUR	corn	France	EUR	beets	Belgium
EUR	corn	France	EUR	corn	Spain
NA	wheat	Canada	NA	sugar	USA

関連項目:

“例 7: 外部結合を実行する” (281 ページ)

クロス結合

CROSS JOIN(交差結合)は、結合結果テーブルとして、2 つのテーブルの積を返します。

次のプログラムでは、例として LEFTTAB テーブルと RIGHTTAB テーブルを使用し、クロス結合を示しています。

```
proc sql;
   title 'Cross Join';
   select *
      from lefttab as l cross join
         righttab as r;
```

アウトプット 8.7　クロス結合

Cross Join

Continent	Export	Country	Continent	Export	Country
NA	wheat	Canada	NA	sugar	USA
NA	wheat	Canada	EUR	corn	Spain
NA	wheat	Canada	EUR	beets	Belgium
NA	wheat	Canada	ASIA	rice	Vietnam
EUR	corn	France	NA	sugar	USA
EUR	corn	France	EUR	corn	Spain
EUR	corn	France	EUR	beets	Belgium
EUR	corn	France	ASIA	rice	Vietnam
EUR	rice	Italy	NA	sugar	USA
EUR	rice	Italy	EUR	corn	Spain
EUR	rice	Italy	EUR	beets	Belgium
EUR	rice	Italy	ASIA	rice	Vietnam
AFR	oil	Egypt	NA	sugar	USA
AFR	oil	Egypt	EUR	corn	Spain
AFR	oil	Egypt	EUR	beets	Belgium
AFR	oil	Egypt	ASIA	rice	Vietnam

クロス結合は、機能的にデカルト積結合と異なりません。次のプログラムをサブミットしても、同じ結果が得られます。

```
proc sql;
   select *
      from lefttab, righttab;
```

クロス結合では、ON 句を使用しないでください。ON 句を使用すると、クロス結合が失敗します。ただし、WHERE 句を使用して出力をサブセット化することはできます。

和結合

UNION JOIN(和結合)は、両方のテーブルの列の和を返します。和結合は、各入力テーブルのそれぞれの列の値を持つすべての行を結果に含めます。あるテーブルに存在しない列については、結果として出力されるテーブルのそれらの行にヌル(欠損)値が設定されます。次の例は、和結合を示しています。

```
proc sql;
   title 'Union Join';
   select *
      from lefttab union join righttab;
```

アウトプット 8.8 和結合

Union Join

Continent	Export	Country	Continent	Export	Country
			NA	sugar	USA
			EUR	corn	Spain
			EUR	beets	Belgium
			ASIA	rice	Vietnam
NA	wheat	Canada			
EUR	corn	France			
EUR	rice	Italy			
AFR	oil	Egypt			

和結合の使用は、OUTER UNION セット演算子を使用してテーブルを連結することに類似しています。詳細については、“query-expression” (339 ページ)を参照してください。

和結合では、ON 句を使用しないでください。ON 句を使用すると、和結合が失敗します。

自然結合

NATURAL JOIN(自然結合)では、2 つのテーブルの同名同タイプの列に同じ値が入っている行を選択します。2 つの列が同じ名前で、タイプが異なる場合、エラーが発生します。自然結合を指定するときに結合仕様を省略すると、INNER が暗黙的に含まれます。類似する列が見つからない場合、クロス結合が実行されます。

下の例では、次の 2 つのテーブルを使用します。

```
data table1;
   input x y z;
   datalines;
1 2 3
2 1 8
6 5 4
2 5 6
;

data table2;
   input x b z;
   datalines;
1 5 3
3 5 4
2 7 8
6 0 4
;


proc sql;
```

```
      title 'Table1';
      select * from table1;
      title 'Table2';
      select * from table2;
   quit;
```

アウトプット 8.9 *自然結合によるテーブル*

table1

x	y	z
1	2	3
2	1	8
6	5	4
2	5	6

table2

x	b	z
1	5	3
3	5	4
2	7	8
6	0	4

次のプログラムは、自然内部結合を示しています。

```
proc sql;
   title 'Natural Inner Join';
   select *
   from table1 natural join table2;
```

アウトプット 8.10 *自然内部結合*

Natural Inner Join

x	z	b	y
1	3	5	2
2	8	7	1
6	4	0	5

次のプログラムは、自然左外部結合を示しています。

```
proc sql;
   title 'Natural Left Outer Join';
   select *
      from table1 natural left join table2;
```

アウトプット 8.11 *自然左外部結合*

Natural Left Outer Join

x	z	b	y
1	3	5	2
2	6	.	5
2	8	7	1
6	4	0	5

自然結合では、ON 句を使用しないでください。ON 句を使用すると、自然結合が失敗します。自然結合を使用すると、暗黙的に ON 句が含まれて、すべての類似する列が照合されます。

複数のテーブルを結合する

内部結合は、通常、2 つまたは 3 つのテーブルに対して実行されます。ただし、PROC SQL では、最大で 256 個までのテーブルに対して内部結合を実行できます。次のコード行に示すように、同じタイプまたは異なるタイプの複数の結合を組み合わせることができます。

```
a natural join b natural join c

a natural join b cross join c
```

また、次の例に示すように、かっこを使用して結合をまとめてグループ化し、結合が実行される順序を制御できます。

```
(a, b) left join c on a.X=c.Y

a left join (b full join c on b.Z=c.Z) on a.Y=b.Y
```

注: 可換性は、実行される結合のタイプによって変わります。

ここでは、テーブル間での関係の動作とその理由について説明するために、3 つのテーブルに対する結合を示します。

3 テーブル結合では、SQL 式は、次の 2 つの条件から成ります。1 つの条件は、1 番目のテーブルを 2 番目のテーブルに関連付けます。もう 1 つの条件は、2 番目のテーブルを 3 番目のテーブルに関連付けます。この例は、各ステージに分割できます。まず、2 テーブル結合を実行して一時テーブルを作成し、次にその一時テーブルを 3 番目のテーブルと結合できます。しかし、PROC SQL では、次の例で示すように、これらすべてのステップを 1 つのステップで実行できます。最終的なテーブルは、どちらの場合でも同じです。

この例は、3 つのテーブル(Comm、Price および Amount)の結合を示しています。各国の輸出総額を計算するには、輸出量(Amount テーブル)にそれぞれの単価(Price テーブル)を掛ける必要があり、各国が輸出する商品(Comm テーブル)を知る必要があります。

```
data comm;
   input Continent $ Export $ Country $ ;
   datalines;
NA   wheat Canada
EUR  corn  France
EUR  rice  Italy
AFR  oil   Egypt
;

data price;
   input Export $ Price;
   datalines;
rice 3.56
corn 3.45
oil  18
wheat 2.98
;

data amount;
   input Country $ Quantity;
   datalines;
Canada 16000
France  2400
Italy    500
Egypt  10000
;


proc sql;
   title 'Comm Table';
   select * from comm;
   title 'Price Table';
   select * from price;
   title 'Amount Table';
   select * from amount;
```

アウトプット 8.12 *3 つ以上のテーブルを結合するための入力*

COMM Table

Continent	Export	Country
NA	wheat	Canada
EUR	corn	France
EUR	rice	Italy
AFR	oil	Egypt

PRICE Table

Export	Price
rice	3.56
corn	3.45
oil	18
wheat	2.98

AMOUNT Table

Country	Quantity
Canada	16000
France	2400
Italy	500
Egypt	10000

```
proc sql;
title  'Total Export Revenue';
select c.Country, p.Export, p.Price,
      a.Quantity, a.quantity*p.price
      as Total
   from comm as c JOIN price as p
       on (c.export=p.export)
        JOIN amount as a
      on (c.country=a.country);
quit;
```

アウトプット 8.13 *3 テーブル結合*

Total Export Revenue

Country	Export	Price	Quantity	Total
Canada	wheat	2.98	16000	47680
France	corn	3.45	2400	8280
Italy	rice	3.56	500	1780
Egypt	oil	18	10000	180000

関連項目:

"例 9: 3 つのテーブルを結合する" (289 ページ)

結合とサブクエリの比較

多くの場合、サブクエリまたは結合のどちらを使用しても、同じ結果が得られます。ただし、外側のクエリとサブクエリが重複行を返さなければ、多くの場合、結合を使用したほうが効率的です。たとえば、次の 2 つのクエリは同じ結果を生成します。2 番目のクエリのほうが効率的です。

```
proc sql;
   select IDNumber, Birth
      from proclib.payroll
      where IDNumber in (select idnum
                         from proclib.staff
                         where lname like 'B%');

proc sql;
   select  p.IDNumber, p.Birth
     from proclib.payroll p, proclib.staff s
     where p.idnumber=s.idnum
           and s.lname like 'B%';
```

注: Proclib.Payroll は"例 2: テーブルをクエリの結果から作成する" (269 ページ)に示されています。

LIKE 条件

一致パターンをテストします。

構文

sql-expression <NOT> **LIKE** *sql-expression* <ESCAPE *character-expression*>

必須引数

sql-expression

"sql-expression" (347 ページ)を参照してください。

character-expression
1 つの文字を評価する SQL 式。*character-expression* のオペランドは、文字リテラルまたは文字列リテラルである必要があります。

注: ESCAPE 句を使用する場合、パターンマッチング指定は、引用符で囲まれた文字列または引用符で囲まれて連結された文字列である必要があります。パターンマッチング指定に列名を含めることはできません。

詳細

LIKE 条件は、文字列をパターンマッチング指定と比較することによって行を選択します。左のオペランドが右のオペランドで指定されたパターンに一致する場合、LIKE 条件が TRUE に決定され、一致する文字列が表示されます。通常はパターンマッチングで使用されるパーセント(%)文字およびアンダーライン(_)文字のリテラルインスタンスを検索するには、ESCAPE 句を使用します。

検索パターン

パターンは、次の 3 種類の文字から成ります。

アンダーライン(_)
任意の 1 つの文字と一致します。

パーセント記号(%)
任意の 0 個以上の文字の並びと一致します。

その他の文字
その文字と一致します。

これらのパターンは、照合する文字の前、後、または前後に記述できます。LIKE 条件では、大文字と小文字が区別されます。

次の例では、**Smith**、**Smooth**、**Smothers**、**Smart**、**Smuggle** の各値を使用します。

'Sm%'
Smith、**Smooth**、**Smothers**、**Smart**、**Smuggle** と一致します。

'%th'
Smith、**Smooth** に一致させます。

'S__gg%'
Smuggle と一致します。

'S_o'
3 文字の語に一致します。そのため、ここでは一致がありません。

'S_o%'
Smooth、**Smothers** と一致します。

'S%th'
Smith、**Smooth** に一致させます。

'Z'
1 つの大文字 **z** のみと一致します。そのため、ここでは一致がありません。

リテラル%と_の検索

LIKE 条件のコンテキストでは、%文字と_文字には特殊な意味があるため、入力文字列からこれらの文字リテラルを検索するには、ESCAPE 句を使用する必要があります。

次の例では、**app**、**a_%**、**a__**、**bbaa1**、**ba_1** の各値を使用します。

- `like 'a_%'`という条件は、app、a_%および a__と一致します。これは、検索パターン内のアンダーライン(_)が任意の 1 文字(アンダーラインを含む)と一致し、検索パターン内のパーセント(%)がゼロ個以上の文字('%'と'_'を含む)と一致するためです。
- `like 'a_^%' escape '^'`という条件は、a_%のみと一致します。これは、エスケープ文字(^)によってリテラル'%のパターン検索が指定されるためです。
- `like 'a_%' escape '_'`という条件は、どの値とも一致しません。これは、エスケープ文字(_)によって、'a'の後にリテラル'%'が続くパターンの検索が指定されていますが、そのパターンがどの値にも当てはまらないためです。

大文字小文字混在文字列の検索

大文字小文字混在文字列を検索するには、次のように、UPCASE 関数を使用してすべての名前を大文字に変換してから、LIKE 条件を入力します。

```
upcase(name) like 'SM%';
```

注: %文字を使用する場合、後に追加された空白の影響に注意してください。値を照合するには、TRIM 関数を使用して、後に追加された空白を削除する必要がある場合があります。

LOWER 関数

文字列を小文字に変換します。

参照項目: “UPPER 関数” (365 ページ)

構文

LOWER (*sql-expression*)

必須引数

sql-expression
“sql-expression” (347 ページ)を参照してください。

要件 *sql-expression* は文字列へと展開される必要があります。

詳細

LOWER 関数は、文字列を操作します。LOWER では、引数はすべて小文字に変更します。

注: LOWER 関数は、ANSI SQL 規格との互換性を保つために提供されています。SAS の LOWCASE 関数を使用することもできます。

query-expression

テーブルからデータを取得します。

参照項目: “インラインビュー” (260 ページ)
“他の式(サブクエリ)” (351 ページ)

“table-expression” (364 ページ)

構文

table-expression-1 <*set-operator table-expression-2*> <*set-operator table-expression-3* …>

必須引数

table-expression
“table-expression” (364 ページ)を参照してください。

オプション引数

set-operator
次のいずれかを指定できます。

INTERSECT <CORRESPONDING> <ALL>

OUTER UNION <CORRESPONDING>

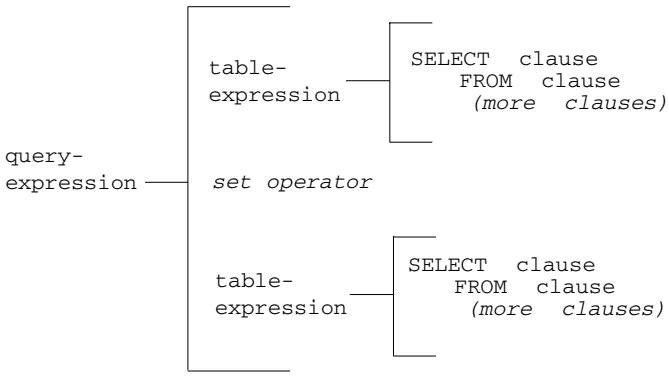
UNION <CORRESPONDING> <ALL>

EXCEPT <CORRESPONDING> <ALL>

詳細

クエリ式とテーブル式

クエリ式は、1 つ以上のテーブル式です。複数のテーブル式は、セット演算子によって結合されます。次の図は、テーブル式とクエリ式の間の関係を示しています。



セット演算子

PROC SQL では、次のセット演算子が提供されています。

OUTER UNION
クエリ結果を連結します。

UNION
両方のクエリ結果から、すべて一意の行を生成します。

EXCEPT
最初のクエリ結果のみに含まれる行を生成します。

INTERSECT
両方のクエリ結果に共通する行を生成します。

セット演算子を含むクエリ式は、次のように評価されます。

- それぞれのテーブル式が評価されて、内部で中間的な結果テーブルが生成されます。
- 次に、それぞれの中間的な結果テーブルは、セット演算子によって連結されるオペランドになり、式を形成します。
- たとえば、A UNION B などです。クエリ式に 3 つ以上のテーブル式が含まれる場合、最初の 2 つのテーブル式の結果は、次のセット演算子とオペランドのためのオペランドになります。たとえば、(A UNION B) EXCEPT C、((A UNION B) EXCEPT C) INTERSECT D などです。
- クエリ式を評価することによって、1 つの出力テーブルが生成されます。

式の中のセット演算子は、かっこによって無効にしない限り、次の優先順位の順序に従います。INTERSECT が最初に評価されます。OUTER UNION、UNION および EXCEPT の優先順位は同じです。

PROC SQL は、テーブル式で参照されるテーブルまたはビューの列の数が異なる場合でも、集合演算を実行します。このように動作する理由は、次のとおりです。SQL の ANSI 規格では、集合演算に関与するテーブルまたはビューの列の数が同じである必要があり、それらの列のデータタイプが一致している必要があります。連結されている 1 つ以上のテーブルまたはビューよりも列数が少ないテーブルまたはビューに対して集合演算を実行した場合、PROC SQL は、適切なデータタイプの欠損値を含む列を作成することによって、列が足りないテーブルまたはビューを拡張します。この一時的な変更によって、集合演算を正しく実行することができます。

CORRESPONDING (CORR) キーワード

CORRESPONDING キーワードは、セット演算子を指定する場合にのみ使用されます。CORR を指定すると、PROC SQL は、列の位置ではなく、名前によってテーブル式の列を照合します。名前が一致しない列は、OUTER UNION 演算子の場合を除き、結果テーブルから除外されます。“OUTER UNION” (341 ページ)を参照してください。

たとえば、2 つのテーブル式に対して集合演算を実行した場合、PROC SQL は、1 つのテーブル式で最初に指定された column-name(SELECT 句に記述された column-name)と、もう 1 つのテーブル式で最初に指定された column-name を照合します。CORR を省略した場合、PROC SQL は、列の位置によって列を照合します。

ALL キーワード

セット演算子は、出力テーブルから自動的に重複行を除去します。オプションの ALL キーワードを指定すると、重複行が維持され、実行ステップが 1 つ減ります。これによって、クエリ式のパフォーマンスが向上します。一意の行のみではなく、テーブル式の実行結果のすべての行を表示する場合に、このキーワードを使用します。ALL キーワードは、セット演算子も指定した場合にのみ使用します。

OUTER UNION

OUTER UNION の実行は、SET ステートメントを使用した SAS DATA ステップの実行によく似ています。OUTER UNION は、テーブル式の中間結果を連結します。したがって、1 番目のテーブル式によって生成されたすべての行に、2 番目のテーブル式によって生成されたすべての行が連結されて、クエリ式の結果テーブルに格納されます。同じ名前の列は、別の列として結果テーブルに含まれます。

たとえば、次のクエリ式は、ME1 テーブルと ME2 テーブルを連結しますが、類似する名前を持つ列を重ね合わせません。アウトプット 8.15 (343 ページ)に、結果を示します。

```
data me1;
   input IDnum $ Jobcode $ Salary Bonus;
```

```
   datalines;
1400   ME1         29769  587
1403   ME1         28072  342
1120   ME1         28619  986
1120   ME1         28619  986
;

data me2;
   input IDnum $ Jobcode $ Salary;
   datalines;
1653   ME2         35108
1782   ME2         35345
1244   ME2         36925
;


proc sql ;
   title 'ME1';
   select * from me1;
   title 'ME2';
   select * from me2;
```

アウトプット 8.14 *ME1 テーブルと ME2 テーブル*

ME1

IDnum	Jobcode	Salary	Bonus
1400	ME1	29769	587
1403	ME1	28072	342
1120	ME1	28619	986
1120	ME1	28619	986

ME2

IDnum	Jobcode	Salary
1653	ME2	35108
1782	ME2	35345
1244	ME2	36925

```
proc sql;
   title 'ME1 and ME2: OUTER UNION';
   select *
      from me1
   outer union
   select *
```

```
   from me2;
```

アウトプット 8.15 *ME1 テーブルと ME2 テーブルの外部結合*

ME1 and ME2: OUTER UNION

IDnum	Jobcode	Salary	Bonus	IDnum	Jobcode	Salary
1400	ME1	29769	587			.
1403	ME1	28072	342			.
1120	ME1	28619	986			.
1120	ME1	28619	986			.
		.	.	1653	ME2	35108
		.	.	1782	ME2	35345
		.	.	1244	ME2	36925

OUTER UNION セット演算子を使用したテーブルの連結は、和結合の実行に類似しています。詳細については、“和結合” (331 ページ)を参照してください。

同じ名前を持つ列を重ね合わせるには、CORRESPONDING キーワードを使用します。

```
proc sql;
   title 'ME1 and ME2: OUTER UNION CORRESPONDING';
   select *
      from me1
   outer union corr
   select *
      from me2;
```

アウトプット 8.16 *Outer Union Corresponding*

ME1 and ME2: OUTER UNION CORRESPONDING

IDnum	Jobcode	Salary	Bonus
1400	ME1	29769	587
1403	ME1	28072	342
1120	ME1	28619	986
1120	ME1	28619	986
1653	ME2	35108	.
1782	ME2	35345	.
1244	ME2	36925	.

連結によって得られたテーブルについて、次の点に注目してください。

- OUTER UNION CORRESPONDING は、一致しないすべての列を維持します。
- 同じ名前を持つ列については、1 番目のテーブル式の結果から値が欠損している場合、2 番目のテーブル式の同じ名前の列の値が挿入されます。
- 結果テーブルにすべての行を含めることが OUTER UNION 演算子のデフォルトのアクションであるため、この演算子では ALL キーワードは使用されません。そのため、ME1 テーブルの、IDnum が `1120` である 2 つの行が、出力に現れています。

UNION

UNION 演算子は、両方のテーブル式から得られた、すべて一意の行を含むテーブルを生成します。つまり、出力テーブルには、1 番目のテーブル式、2 番目のテーブル式、または両方のテーブル式から生成された行が含まれます。

列は、列名とは無関係に、テーブル内の位置によって追加されます。ただし、対応する列のデータタイプは一致している必要があります。そうでない場合、結合は実行されません。PROC SQL は、警告メッセージを発行して実行を停止します。

出力テーブルの列の名前は、1 番目のテーブル式の列に名前がない場合(式などの場合)を除き、1 番目のテーブル式の列の名前になります。1 番目のテーブル式の列に名前がない場合、出力テーブルの列名は、2 番目のテーブル式のそれぞれの列の名前になります。

次の例では、PROC SQL によって 2 つのテーブルを結合しています。

```
proc sql;
   title 'ME1 and ME2: UNION';
   select *
      from me1
   union
   select *
      from me2;
```

アウトプット 8.17 *ME1 テーブルと ME2 テーブルの結合*

ME1 and ME2: UNION

IDnum	Jobcode	Salary	Bonus
1120	ME1	28619	986
1244	ME2	36925	.
1400	ME1	29769	587
1403	ME1	28072	342
1653	ME2	35108	.
1782	ME2	35345	.

次の例では、ALL を指定して、ME1 の重複行を含めています。さらに、ALL は、PROC SQL に 1 つのパスのみを実行するよう指定することによって、順序を変更します。そのため、ME2 の値は、ME1 の値に単純に追加されます。

```
proc sql;
```

```
title 'ME1 and ME2: UNION ALL';
select *
   from me1
union all
select *
   from me2;
```

アウトプット 8.18 *UNION ALL*

ME1 and ME2: UNION ALL

IDnum	Jobcode	Salary	Bonus
1400	ME1	29769	587
1403	ME1	28072	342
1120	ME1	28619	986
1120	ME1	28619	986
1653	ME2	35108	.
1782	ME2	35345	.
1244	ME2	36925	.

別の例については、“例 5: 2 つのテーブルを組み合わせる” (276 ページ)を参照してください。

EXCEPT

1 番目のテーブル式の EXCEPT 演算子は、2 番目のテーブル式の結果を除く、一意の行を含む出力テーブルを生成します。1 番目のテーブル式の中間結果に、2 番目のテーブル式の中間結果に含まれない行が少なくとも 1 つ出現した場合、1 番目のテーブル式のその行が結果テーブルに含まれます。

次の例の In_USA テーブルには、アメリカ内外の都市への航空便が含まれています。Out_USA テーブルには、アメリカ以外の都市への航空便のみが含まれています。

```
data in_usa;
   input Flight $ Dest $;
   datalines;
145 ORD
156 WAS
188 LAX
193 FRA
207 LON
;
data out_USA;
   input Flight $ Dest $;
   datalines;
193 FRA
207 LON
311 SJA
;
```

```
proc sql;
   title 'In_USA';
   select * from in_usa;
   title 'Out_USA';
   select * from out_usa;
```

アウトプット 8.19 *EXCEPT の例の入力テーブル*

IN_USA

Flight	Dest
145	ORD
156	WAS
188	LAX
193	FRA
207	LON

OUT_USA

Flight	Dest
193	FRA
207	LON
311	SJA

次の例では、Out_USA に含まれていない In_USA の行のみが返されます。

```
proc sql;
    title 'Flights from In_USA Only';
    select * from in_usa
    except
    select * from out_usa;
```

アウトプット 8.20 *In_USA のみの航空便*

Flights from IN_USA Only

Flight	Dest
145	ORD
156	WAS
188	LAX

INTERSECT

INTERSECT 演算子は、両方のテーブルに共通する行を含む出力テーブルを生成します。たとえば、次の例では、上で示した In_USA テーブルと Out_USA テーブルを使用して、両方のテーブルに含まれる行を返しています。

```
proc sql;
    title 'Flights from Both In_USA and Out_USA';
    select * from in_usa
    intersect
    select * from out_usa;
```

アウトプット 8.21 *In_USA と Out_USA の両方の航空便*

Flights from Both IN_USA and OUT_USA

Flight	Dest
193	FRA
207	LON

sql-expression

一連のオペランドと演算子から値を生成します。

構文

operand operator operand

必須引数

operand

次のいずれかを指定できます。

- *constant*。これは、固定値を示す数値または引用符で囲まれた文字列(または他の特殊表記)です。定数は *literal* とも呼ばれます。constant については、*SAS 関数と CALL ルーチン: リファレンス* で説明されています。
- column-name。これについては、“column-name” (319 ページ)で説明されています。
- CASE 式。これについては、“CASE 式” (314 ページ)で説明されています。
- サポートされている任意の SAS 関数。PROC SQL は、SAS DATA ステップで使用可能な多くの関数をサポートしています。サポートされない関数には、変数情報関数、データの配列を操作する関数、現在の行以外の行を操作する関数などがあります。ほかの SQL データベースでは、独自の関数セットがサポートされています。関数については、*SAS 関数と CALL ルーチン: リファレンス*で説明されています。
- PROC FCMP を使用して作成された、配列要素を操作する関数以外の任意の関数。

- ANSI SQL 関数の COALESCE, BTRIM、LOWER、UPPER、SUBSTRING。
- summary-function。これについては、“summary-function” (356 ページ)で説明されています。
- クエリ式。これについては、“query-expression” (339 ページ)で説明されています。
- USER リテラル。これは、プログラムをサブミットしたユーザーのユーザー ID を参照します。返されるユーザー ID はオペレーティングシステムに依存しますが、PROC SQL は、オペレーティング・システム上で&SYSJOBID マクロ変数に設定されているのと同じ値を使用します。

operator
“演算子と評価の順序” (349 ページ)を参照してください。

注: SAS 関数(要約関数を含む)は、SQL 式として単独で実行できます。たとえば、次のように指定します。

```
select min(x) from table;

select scan(y,4) from table;
```

詳細

SAS 関数

PROC SQL は、SAS DATA ステップで使用可能な多くの関数をサポートしています。サポートされない関数には、変数情報関数、データの配列を操作する関数などがあります。ほかの SQL データベースでは、独自の関数セットがサポートされています。たとえば、SCAN 関数は次のクエリで使用されます。

```
select style, scan(street,1) format=$15.
   from houses;
```

注: PROC SQL で使用される SAS DATA ステップ関数は、引数に関して ANSI SQL 規格ガイドラインおよび ISO SQL 規格ガイドラインに従う必要があります。SAS DATA ステップ関数には、空の引数を含めることはできません。一部の SAS 関数の引数は、PROC SQL での使用時にサポートされないことがあります。たとえば、DATA ステップの INPUT 関数は、“?”および “??”をサポートします。PROC SQL の INPUT 関数は“?”のみをサポートします。“??”を PROC SQL で使用すると、エラーが発生します。

PROC SQL は、ユーザーが記述した任意の関数をサポートします。ただし、“FCMP” (*Base SAS Procedures Guide*)を使用して作成された配列要素を操作する関数は除きます。

SAS 関数の詳細なドキュメントについては、*SAS 関数と CALL ルーチン: リファレンス* を参照してください。要約関数も、SAS 関数です。詳細については、“summary-function” (356 ページ)を参照してください。

USER リテラル

ビュー定義では、USER を指定できます。たとえば、ユーザーの部門のビューへのアクセスを制限するビューを作成できます。なお、USER リテラル値は大文字で格納されます。そのため、この値と比較する場合は、UPCASE 関数を使用することをお勧めします。

```
create view myemp as
   select * from dept12.employees
      where upcase(manager)=user;
```

このビューは、従業員情報を参照する管理者ごとに、異なる従業員情報の集合を生成します。

演算子と評価の順序

演算が評価される順序は、次の例外を除き、DATA ステップでの順序と同じです。PROC SQL では、NOT は AND 論理演算子および OR 論理演算子と共にグループ化されます。

SQL の一部のバージョンの欠損値とは異なり、SAS の欠損値は、必ず照合順序の先頭に現れます。したがって、ブール演算と比較演算では、次のような式は述語において TRUE に決定されます。

```
 3>null
-3>null
 0>null
```

かっこを使用して値をグループ化したり、数学式をネストしたりできます。かっこを使用すると式が読みやすくなり、演算子の評価順序を変更することもできます。かっこ付きの式の評価は、最も深いレベルのかっこから始まり、外側に向かって進みます。たとえば、SAS は、A+B*C を A+(B*C)として評価しますが、別の結果を得るために、かっこを追加して(A+B)*C として評価させることができます。

優先順位がより高い演算子が最初に実行されます。つまり、グループ 0 の演算子が評価されてから、グループ 5 の演算子が評価されます。次の表は、演算子と、優先順位グループを含む演算子の評価順序を示しています。

表 8.1 *演算子とその評価順序*

グループ	演算子	説明
0	( )	これで式を囲むと、最初に評価するよう強制します。
1	case-expression	指定した条件を満たす結果の値を選択します。
2	**	累乗します。
	単項の+、単項の-	正の数値または負の数値を示します。
3	*	積を演算します。
	/	除算します。
4	+	次の値を加算
	–	次の値を減算
5	||	連結します。
6	<NOT> BETWEEN 条件	“BETWEEN 条件” (312 ページ)を参照してください。
	<NOT> CONTAINS 条件	“CONTAINS 条件” (320 ページ)を参照してください。
	<NOT> EXISTS 条件	“EXISTS 条件” (321 ページ)を参照してください。
	<NOT> IN 条件	“IN 条件” (321 ページ)を参照してください。

グループ	演算子	説明
	IS <NOT>条件	“IS 条件” (322 ページ)を参照してください。
	<NOT> LIKE 条件	“LIKE 条件” (337 ページ)を参照してください。
7	=, eq	等しい。
	¬=, ^=, < >, ne	等しくない。
	>, gt	次の値より大きい
	<, lt	次の値より小さい
	>=, ge	次の値以上
	<=, le	次の値以下
	=*	発音が類似する(文字オペランドでのみ使用)。“例 11: SOUNDS-LIKE 演算子を使用して値を取得する” (295 ページ)を参照してください。
	eqt	切り詰められた文字列と等しい(文字オペランドでのみ使用)。“切り捨て文字列の比較演算子” (350 ページ)を参照してください。
	gtt	切り詰められた文字列より大きい
	ltt	切り詰められた文字列より小さい
	get	切り詰められた文字列と等しいかより大きい
	let	切り詰められた文字列と等しいかより小さい
	net	切り詰められた文字列と等しくない
8	¬, ^, NOT	論理 NOT を示します。
9	&, AND	論理 AND を示します。
10	|, OR	論理 OR を示します。

演算子の記号は、使用するオペレーティングシステムによって変わる場合があります。詳細については、“SAS Operators in Expressions” (*SAS Language Reference: Concepts*)を参照してください。

切り捨て文字列の比較演算子

PROC SQL は、切り詰められた文字列の比較演算子をサポートします。(表 8.1 (349 ページ)のグループ 7 を参照してください)。切り詰められた文字列の比較では、長い文字列を短い文字列の長さと同じになるように切り詰めることによって、文字列の長さを揃えてから比較が実行されます。たとえば、式`'TWOSTORY' eqt 'TWO'`の場合、文字列'TWOSTORY'が'TWO'に減らされてから比較が実行されるため、TRUE になりま

す。なお、切り詰めは内部で実行されます。いずれのオペランドも永続的には変更されません。

注: DATA ステップとは異なり、PROC SQL では、切り詰められた文字列を比較するコロン演算子(=:、>:、<=:など))はサポートされていません。アルファベット演算子(EQT、GTT、LET など)を使用してください。

他の式(サブクエリ)

クエリ式は、WHERE 句または HAVING 句で使用された場合、サブクエリと呼ばれます。サブクエリは、別のクエリ式の一部としてネストされたクエリ式です。サブクエリは、別のテーブルの値に基づいて、テーブルから 1 つ以上の行を選択します。

サブクエリは、それを含む句に応じて、1 つの値または複数の値を返すことができます。クエリ式で複数のサブクエリを使用した場合、最も内側のクエリが最初に評価され、次のその外側のクエリが評価されるというように、外側に向かって評価が進みます。

PROC SQL では、単純な列の値または定数を使用できる場所であれば、式のどの場所でもサブクエリを(かっこで囲んで)使用できます。この場合、サブクエリは、1 つの値、つまり 1 つの行の 1 つの列の値のみを返す必要があります。

1 つの値を返すサブクエリの例を次に示します。この PROC SQL ステップは、Proclib.Staff テーブルの情報に基づいて Proclib.Payroll テーブルをサブセット化します。(Proclib.Payroll は、“例 2: テーブルをクエリの結果から作成する” (269 ページ)に示されています。Proclib.Staff は、“例 4: 2 つのテーブルを結合する” (273 ページ)に示されています。)Proclib.Payroll には、従業員の識別番号(IdNumber)と給与(Salary)が含まれていますが、従業員の名前は含まれていません。Proclib.Payroll から 1 人の従業員の行のみを取得する場合、従業員の識別番号と名前(Lname と Fname)を含む Proclib.Staff テーブルを照会するサブクエリを使用できます。

```
proc sql;
    title 'Information for Earl Bowden';
    select *
       from proclib.payroll
       where idnumber=
             (select idnum
             from proclib.staff
             where upcase(lname)='BOWDEN');
```

アウトプット 8.22 *クエリの出力-1 つの値*

Information for Earl Bowden

IdNumber	Gender	Jobcode	Salary	Birth	Hired
1403	M	ME1	28072	28JAN69	21DEC91

サブクエリは、複数の値を返すことができます。次の例では、Proclib.Delay テーブルと Proclib.March テーブルを使用しています。これらのテーブルには、同じ航空便に関する情報が含まれており、共通して Flight 列が含まれています。次のサブクエリは、国際航空便に関する Proclib.Delay の Flight のすべての値を返します。このサブクエリの値によって、外側のクエリの WHERE 句が完成します。そのため、外側のクエリが実行されると、Proclib.March の国際航空便のみが出力されます。

```
proc sql outobs=5;
```

```
title 'International Flights from';
title2 'Proclib.March';
select Flight, Date, Dest, Boarded
   from proclib.march
   where flight in
         (select flight
          from proclib.delay
          where destype='International');
```

アウトプット 8.23 *クエリ出力-複数の値*

International Flights from PROCLIB.MARCH

flight	date	dest	boarded
219	01MAR08	LON	198
622	01MAR08	FRA	207
132	01MAR08	YYZ	115
271	01MAR08	PAR	138
219	02MAR08	LON	147

値を、サブクエリが返す値の集合と比較することが、役立つ場合があります。サブクエリが比較の右側のオペランドである場合、サブクエリの前で、ANY キーワードまたは ALL キーワードを指定できます。ALL を指定した場合、サブクエリが返すすべての値について比較が TRUE である場合にのみ、比較が TRUE になります。サブクエリが行を返さない場合、ALL 比較の結果は、外側のクエリの各行に対して TRUE になります。

ANY を指定した場合、サブクエリが返す値のいずれかについて比較が TRUE である場合に、比較が TRUE になります。サブクエリが行を返さない場合、ANY 比較の結果は、外側のクエリの各行に対して FALSE になります。

次の例では、**ME3** のうちの最高給与よりも多い収入を得ているすべての従業員を、Proclib.Payroll から選択しています。

```
proc sql;
title "Employees who Earn More than";
title2 "All ME's";
   select *
     from proclib.payroll
     where salary > all (select salary
                         from proclib.payroll
                         where jobcode='ME3');
```

アウトプット 8.24　ALL 比較を使用したクエリ出力

Employees who Earn More than All ME's

IdNumber	Gender	Jobcode	Salary	Birth	Hired
1333	M	PT2	88606	30MAR61	10FEB81
1739	M	PT1	66517	25DEC64	27JAN91
1428	F	PT1	68767	04APR60	16NOV91
1404	M	PT2	91376	24FEB53	01JAN80
1935	F	NA2	51081	28MAR54	16OCT81
1905	M	PT1	65111	16APR72	29MAY92
1407	M	PT1	68096	23MAR69	18MAR90
1410	M	PT2	84685	03MAY67	07NOV86
1439	F	PT1	70736	06MAR64	10SEP90
1545	M	PT1	66130	12AUG59	29MAY90
1106	M	PT2	89632	06NOV57	16AUG84
1442	F	PT2	84536	05SEP66	12APR88
1417	M	NA2	52270	27JUN64	07MAR89
1478	M	PT2	84203	09AUG59	24OCT90
1556	M	PT1	71349	22JUN64	11DEC91
1352	M	NA2	53798	02DEC60	16OCT86
1890	M	PT2	91908	20JUL51	25NOV79
1107	M	PT2	89977	09JUN54	10FEB79
1830	F	PT2	84471	27MAY57	29JAN83
1928	M	PT2	89858	16SEP54	13JUL90
1076	M	PT1	66558	14OCT55	03OCT91

注: ALL を使用した場合の効率に関する注意点については、“サブクエリと効率” (354 ページ)の最初の項目を参照してください。

サブクエリを他のクエリから視覚的に分離するために、サブクエリを任意の数のかっこの組みで囲むことができます。

相関するサブクエリ

相関するサブクエリでは、サブクエリの WHERE 式は、外側のクエリ内のテーブルの値を参照します。相関するサブクエリは、外側のクエリで行ごとに評価されます。相関するサブクエリでは、PROC SQL はサブクエリと外側のクエリを一緒に実行します。

次の例では、Proclib.Delay テーブルと Proclib.March テーブルを使用しています。DATA ステップは、Proclib.Delay を作成します。“Proclib.Delay” (436 ページ)を参照し

てください。Proclib.March は“例 13: 列内の値の使用可能な組み合わせをすべて作成する” (300 ページ)に示されています。Proclib.Delay には、Proclib.March と共通して Flight、Date、Orig、Dest の各列が含まれています。

```
proc sql outobs=5;
   title 'International Flights';
   select *
      from proclib.march
      where 'International' in
           (select destype
            from proclib.delay
            where march.Flight=delay.Flight);
```

サブクエリは、一度に 1 行について、サブクエリの WHERE 句に March.Flight のすべての値を代入することによって展開されます。たとえば、March.Flight が **219** の場合、サブクエリは次のように展開されます。

1. PROC SQL は、DELAY から Fligh が **219** となるすべての行を取得し、それらの行の DESTYPE の値を WHERE 句に渡します。
2. PROC SQL は、DESTYPE の値を使用して WHERE 句を完成します。

   ```
   where 'International' in
      ('International','International', ...)
   ```

3. WHERE 句は、**International** がリストに含まれるかどうかをチェックして判定します。このため、Flight の値が **219** である March のすべての行が出力に含まれます。

次の出力には、March の国際航空便のみの行が含まれています。

アウトプット 8.25 *相関するサブクエリ出力*

International Flights

flight	date	depart	orig	dest	miles	boarded	capacity
219	01MAR08	9:31	LGA	LON	3442	198	250
622	01MAR08	12:19	LGA	FRA	3857	207	250
132	01MAR08	15:35	LGA	YYZ	366	115	178
271	01MAR08	13:17	LGA	PAR	3635	138	250
219	02MAR08	9:31	LGA	LON	3442	147	250

サブクエリと効率

- サブクエリの前で ALL キーワードを使用するかわりに、サブクエリで MAX 関数を使用します。たとえば、次の 2 つのクエリは同じ結果を生成しますが、2 番目のクエリのほうが効率的です。

```
proc sql;
   select * from proclib.payroll
   where salary> all(select salary
                     from proclib.payroll
                     where jobcode='ME3');
```

```
proc sql;
   select * from proclib.payroll
   where salary> (select max(salary)
                  from proclib.payroll
                  where jobcode='ME3');
```

- サブクエリでは、可能であれば EXISTS のかわりに IN を使用します。たとえば、次の 2 つのクエリは同じ結果を生成しますが、通常は 2 番目のクエリのほうが効率的です。

```
proc sql;
   select *
      from proclib.payroll p
      where exists (select *
                    from staff s
                    where p.idnum=s.idnum
                       and state='CT');

proc sql;
   select *
      from proclib.payroll
      where idnum in (select idnum
                        from staff
                        where state='CT');
```

SUBSTRING 関数

文字式の一部を返します。

構文

SUBSTRING (*sql-expression* FROM *start* <FOR *length*>)

必須引数

sql-expression
“sql-expression” (347 ページ)を参照してください。

要件 *sql-expression* は文字列へと展開される必要があります。

start
は、位置を指定する数です(変数名でも列名でもありません)。位置は、文字列の左端からカウントします。この位置から部分文字列の抽出が開始されます。

オプション引数

length
は、抽出対象の部分文字列の長さを指定する数です(変数名でも列名でもありません)。

詳細

SUBSTRING 関数は、文字列を操作します。SUBSTRING は、指定した入力文字列の、start で指定した位置から始まる部分を返します。length を省略すると、

SUBSTRING 関数は、入力文字列の start から末尾までのすべての文字を返します。start と length の値は、変数ではなく数値である必要があります。正、負またはゼロの値を指定できます。

start が入力文字列の長さよりも大きい場合、SUBSTRING 関数は長さがゼロの文字列を返します。

start が 1 よりも小さい場合、SUBSTRING 関数は入力文字列の先頭から抽出を開始します。

length を指定した場合、start と length の合計が start を下回ることはできません。下回った場合、エラーが返されます。start と length の合計が入力文字列の長さよりも大きい場合、SUBSTRING 関数は、入力文字列の start から末尾までのすべての文字を返します。start と length の合計が 1 よりも小さい場合、SUBSTRING 関数は長さがゼロの文字列を返します。

注: SUBSTRING 関数は、ANSI SQL 規格との互換性を保つために提供されています。SAS 関数の SUBSTR を使用することもできます。

summary-function

統計的な要約計算を実行します。

制限事項: 要約関数を ON 句または WHERE 句に記述することはできません。

参照項目: “GROUP BY 句” (261 ページ)
“HAVING 句” (262 ページ)
“SELECT 句” (252 ページ)
“table-expression” (364 ページ)

例: “例 8: ビューをクエリの結果から作成する” (286 ページ)
“例 12: 2 つのテーブルを結合して新しい値を計算する” (297 ページ)
“例 15: SAS マクロを使用して欠損値をカウントする” (309 ページ)

構文

summary-function (<DISTINCT | ALL> *sql-expression*)

必須引数

summary-function
次のいずれかを指定できます。

AVG|MEAN
算術平均または値の平均

COUNT|FREQ|N
非欠損値の数

CSS
修正平方和

CV
変動係数(パーセント)

MAX
最大値

MEDIAN
中央値。サンプルデータ値を小さい順に並べた場合に、丁度中央に位置する値。

MIN
最小値

NMISS
欠損値の数

PRT
t 統計量 T に対する両側 *p* 値(自由度が*n* – 1であるスチューデント)です。

RANGE
値の範囲

STD
標準偏差

STDERR
平均の標準誤差

SUM
値の合計

SUMWGT
WEIGHT 変数値の合計 [1]

T
母集団の平均値がゼロに等しいという仮説を検定するためのスチューデントの *t* 値

USS
無修正平方和

VAR
分散

これらの統計量の説明と使用される式については、“SAS Elementary Statistics Procedures” (*Base SAS Procedures Guide*)を参照してください。

DISTINCT
SQL 式の一意の値のみを計算で使用することを指定します。

ALL
SQL 式のすべての値を計算で使用することを指定します。DISTINCT と ALL のどちらも指定しない場合、ALL が使用されます。

sql-expression
“sql-expression” (347 ページ)を参照してください。

詳細

データの要約

要約関数は、FROM 句に記述したテーブルまたはビュー全体の統計的要約、あるいは GROUP BY 句に指定したグループごとの統計的要約を生成します。GROUP BY を省略した場合、テーブルまたはビュー内のすべての行が 1 つのグループであると見なされます。これらの関数は、テーブル内の各行または各列のすべての値をまとめ、1 つの要約値または集計値を求めます。このため、これらの関数は、多くの場合、集計

[1] 現在、PROC SQL では、テーブルに対して WEIGHT 変数を指定する方法はありません。そのため、各行(またはオブザベーション)の重みは 1 です。

関数と呼ばれます。たとえば、ある列の合計(1 つの値)は、その列のすべての値を加算することによって得られます。

行のカウント

COUNT 関数は、行をカウントします。COUNT(*)は、グループまたはテーブルに含まれる行の総数を返します。COUNT の引数として列名を使用した場合、グループまたはテーブルに含まれ、その列が非欠損値である行の総数が返されます。列内の一意の値をカウントする場合は、COUNT(DISTINCT *column*)を指定します。

テーブル式の SELECT 句に 1 つ以上の要約関数が含まれ、そのテーブル式によって行が決定されなかった場合、要約関数の結果は欠損値になります。ゼロを返す例外を次に示します。

- COUNT(*)
- COUNT(<DISTINCT> *sql-expression*)
- NMISS(<DISTINCT> *sql-expression*)

例については、“例 8: ビューをクエリの結果から作成する” (286 ページ) および“例 15: SAS マクロを使用して欠損値をカウントする” (309 ページ)を参照してください。

引数の数に基づき統計量を計算する

要約関数で指定した引数の数は、計算の実行方法に影響を与えます。1 つの引数を指定した場合、その列の値が計算されます。複数の引数を指定した場合、記述した引数(つまり列)が、行ごとに計算されます。

注: SQL 集計関数内で複数の引数を使用した場合、その関数は SQL 集計(要約)関数とは見なされなくなります。類似する名前の Base SAS 関数が存在する場合、PROC SQL は、その Base SAS 関数を実行します。返される結果は、現在の行の値に基づきます。類似する名前の Base SAS 関数が存在しない場合、エラーが発生します。たとえば、AVG 関数に対して複数の引数を使用した場合、Base SAS には AVG 関数が存在しないため、エラーが発生します。

例として、次のテーブルに対する計算を考えます。

```
data summary;
   input X Y Z;
   datalines;
1 3 4
2 4 5
8 9 4
4 5 4
;

proc sql;
   title 'Summary Table';
   select * from summary;
```

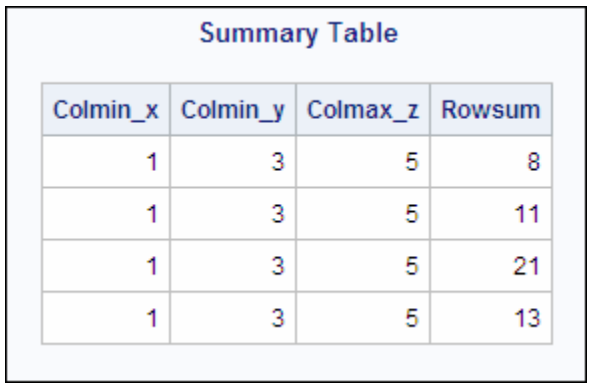
Summary Table

X	Y	Z
1	3	4
2	4	5
8	9	4
4	5	4

関数で 1 つの引数を使用した場合、その列に対してのみ計算が実行されます。複数の引数を使用した場合、指定した列の各行に対して計算が実行されます。次の PROC SQL ステップでは、MIN 関数と MAX 関数を使用して、列の最小値と最大値を返しています。SUM 関数は、行ごとに、引数で指定した列の合計を返します。

```
proc sql;
    select min(x) as Colmin_x,
           min(y) as Colmin_y,
           max(z) as Colmax_z,
           sum(x,y,z) as Rowsum
      from summary;
```

アウトプット 8.26　要約関数



Summary Table

Colmin_x	Colmin_y	Colmax_z	Rowsum
1	3	5	8
1	3	5	11
1	3	5	21
1	3	5	13

データの再マージ

SELECT 句または HAVING 句で要約関数を使用したときに、次のメッセージが SAS ログに表示される場合があります。

```
NOTE: The query requires remerging summary
      statistics back with the original
      data.
```

再マージ処理では、データのパススルーが 2 回実行されます。最初のパススルーでは、PROC SQL は次を実行します。

- 要約関数の値を計算して返します。次にその結果を使用して、要約関数が含まれる算術式を計算します。
- GROUP BY 句に従ってデータをグループ化します。

2 回目のパススルーでは、PROC SQL は、出力に表示する必要のあるその他の列および行を取得します。

注: データの再マージを使用するクエリを PROC SQL が処理しないように指定するには、PROC SQL の NOREMERGE オプションまたは NOSQLREMERGE システムオプションのいずれかを使用します。NOREMERGE オプションまたは NOSQLREMERGE システムオプションを設定した場合、再マージが試みられると、SAS ログにエラーが書き込まれます。詳細については、“REMERGE| NOREMERGE” (228 ページ) および“SQLREMERGE システムオプション” (378 ページ)を参照してください。

次の例では、Proclib.Payroll テーブルを使用して、データの再マージが必要な場合と不要な場合を示しています。“例 2: テーブルをクエリの結果から作成する” (269 ページ)を参照してください。

最初のクエリは、再マージを必要とします。1 回目のデータのパススルーによってデータを Jobcode ごとにグループ化し、グループごとに AVG 関数の値を決定します。しかし、PROC SQL は、IdNumber と Salary の値を取得するために、2 回目のパススルーを実行する必要があります。

```
proc sql outobs=10;
   title 'Salary Information';
   title2 '(First 10 Rows Only)';
   select  IdNumber, Jobcode, Salary,
           avg(salary) as AvgSalary
      from proclib.payroll
      group by jobcode;
```

アウトプット 8.27 *再マージを必要とする給与情報*

Salary Information
(First 10 Rows Only)

IdNumber	Jobcode	Salary	AvgSalary
1704	BCK	25465	25794.22
1677	BCK	26007	25794.22
1383	BCK	25823	25794.22
1845	BCK	25996	25794.22
1100	BCK	25004	25794.22
1663	BCK	26452	25794.22
1673	BCK	25477	25794.22
1389	BCK	25028	25794.22
1834	BCK	26896	25794.22
1132	FA1	22413	23039.36

ジョブコードごとの平均給与のみを返すように、前述のクエリを変更できます。次のクエリは、1 回目のデータのパススルーによって要約とグループ化が実行されるため、再マージを必要としません。そのため、2 回目のパススルーは不要です。

```
proc sql outobs=10;
   title 'Average Salary for Each Jobcode';
   select Jobcode, avg(salary) as AvgSalary
   from proclib.payroll
```

```
    group by jobcode;
```

アウトプット 8.28　再マージが不要な給与情報

Average Salary for Each Jobcode

Jobcode	AvgSalary
BCK	25794.22
FA1	23039.36
FA2	27986.88
FA3	32933.86
ME1	28500.25
ME2	35576.86
ME3	42410.71
NA1	42032.2
NA2	52383
PT1	67908

HAVING 句を使用すると、HAVING 式を決定するために、PROC SQL でデータの再マージが必要になる場合があります。

まず、HAVING 句を使用するが再マージを必要としないクエリを検討します。クエリは、Jobcode の値でデータをグループ化します。その結果には、Jobcode の値ごとに 1 行が含まれ、各 Jobcode についての従業員の要約情報が含まれています。1 回目のパススルーでは、要約関数が、**Number** 列、**Average Age** 列および **Average Salary** 列の値を返します。1 回目のパススルーでは、HAVING 句を決定するために PROC SQL が必要とするすべての値が返されます。そのため、再マージは不要です。

```
proc sql outobs=10;
title 'Summary Information for Each Jobcode';
title2 '(First 10 Rows Only)';
   select Jobcode,
          count(jobcode) as number
              label='Number',
          avg(int((today()-birth)/365.25))
              as avgage format=2.
              label='Average Age',
          avg(salary) as avgsal format=dollar8.
              label='Average Salary'
      from proclib.payroll
      group by jobcode
      having avgage ge 30;
```

アウトプット 8.29 再マージが不要なジョブコード情報

Summary Information for Each Jobcode
(First 10 Rows Only)

Jobcode	Number	Average Age	Average Salary
BCK	9	46	$25,794
FA1	11	42	$23,039
FA2	16	46	$27,987
FA3	7	48	$32,934
ME1	8	43	$28,500
ME2	14	49	$35,577
ME3	7	51	$42,411
NA1	5	39	$42,032
NA2	3	51	$52,383
PT1	8	47	$67,908

次のクエリでは、PROC SQL によってデータが再マージされています。これは、HAVING 句が SALARY 列を比較で使用しており、SALARY 列が GROUP BY 句に含まれていないためです。

```
proc sql outobs=10;
title 'Employees who Earn More than the';
title2 'Average for Their Jobcode';
title3 '(First 10 Rows Only)';
   select Jobcode, Salary,
          avg(salary) as AvgSalary
      from proclib.payroll
      group by jobcode
      having salary > AvgSalary;
```

アウトプット 8.30　再マージが必要なジョブコード情報

Employees who Earn More than the Average for Their Jobcode (First 10 Rows Only)

Jobcode	Salary	AvgSalary
BCK	26007	25794.22
BCK	25823	25794.22
BCK	25996	25794.22
BCK	26452	25794.22
BCK	26896	25794.22
FA1	23177	23039.36
FA1	23738	23039.36
FA1	23979	23039.36
FA1	23916	23039.36
FA1	23644	23039.36

次の場合、PROC SQL がデータを再マージすることに注意してください。

- 要約関数が返す値を、計算で使用する。たとえば、次のクエリは、X の値と、合計に対する X の割合(パーセンテージ)を行ごとに返します。1 回目のパススルーでは、PROC SQL は X の合計を計算し、2 回目のパススルーでは、PROC SQL は X の値ごとに合計に対する割合(パーセンテージ)を計算します。

```
data summary;
   input x;
   datalines;
32
86
49
49
;

proc sql;
   title 'Percentage of the Total';
   select X, (100*x/sum(X)) as Pct_Total
      from summary;
```

図 8.1 合計のパーセンテージ

Percentage of the Total

x	Pct_Total
32	14.81481
86	39.81481
49	22.68519
49	22.68519

- 要約関数が返す値を、GROUP BY 句で指定していない列の値と比較します。たとえば、次のクエリでは、Proclib.Payroll テーブルを使用しています。PROC SQL は、Salary 列が GROUP BY 句で指定されていないため、データを再マージします。

```
proc sql;
   select  jobcode,  salary,
           avg(salary) as avsal
      from proclib.payroll
      group by jobcode
      having salary > avsal;
```

- 入力テーブルの列を SELECT 句で指定し、GROUP BY 句で指定しません。このルールは、SELECT 句内の要約関数の引数で使用される列には適用されません。

  たとえば、次のクエリでは、SELECT 句に IdNumber が存在することにより、PROC SQL はデータを再マージします。これは、IdNumber が、1 回目のパススルーの際に、グループ化にも要約にも関係していないためです。PROC SQL が IdNumber の値を取得するには、2 回目のデータのパススルーを実行する必要があります。

```
proc sql;
   select IdNumber, jobcode,
          avg(salary) as avsal
      from proclib.payroll
      group by jobcode;
```

table-expression

クエリ式の一部または全体を定義します。

参照項目: “query-expression” (339 ページ)
“SELECT ステートメント” (252 ページ)

構文

SELECT <DISTINCT> *object-item-1* <, *object-item-2*, …>
<INTO :*macro-variable-specification-1* <, :*macro-variable-specification-2*, …>>
FROM *from-list*
<WHERE *sql-expression*>
<GROUP BY *group-by-item-1* <, *group-by-item-2*, …>>
<HAVING *sql-expression*>

詳細

テーブル式は、SELECT ステートメントです。これは、ほとんどの SQL プロシジャステートメントの基本的な構成要素です。セット演算子を使用して、複数のテーブル式の結果を結合できます。これによって、クエリ式が作成されます。クエリ式全体に対して、1 つの ORDER BY 句を使用します。クエリ式全体の最後にのみ、セミコロンを記述します。多くの場合、クエリ式は、単に 1 つの SELECT ステートメント(つまりテーブル式)です。

UPPER 関数

文字列を大文字に変換します。

参照項目: “LOWER 関数” (339 ページ)

構文

UPPER (*sql-expression*)

必須引数

sql-expression
“sql-expression” (347 ページ)を参照してください。

要件 *sql-expression* は文字列へと展開される必要があります。

詳細

UPPER 関数は、文字列を操作します。UPPER 関数は、指定された引数をすべて大文字に変換します。

3 部

付録

付録 1
SQL マクロ変数とシステムオプション *369*

付録 2
PROC SQL および ANSI 規格 *383*

付録 3
「SQL プロシジャの使用」で示されているコード例 *387*

付録 4
「SQL プロシジャリファレンス」で示されている例のデータセット *433*

付録 1

SQL マクロ変数とシステムオプション

ディクショナリ **369**
SQLCONSTDATETIME System Option 369
SQLGENERATION=システムオプション 370
SQLIPONEATTEMPT System Option 373
SQLMAPPUTTO=システムオプション 373
SQLREDUCEPUT=システムオプション 374
SQLREDUCEPUTOBS=システムオプション 376
SQLREDUCEPUTVALUES=システムオプション 377
SQLREMERGE システムオプション 378
SQLUNDOPOLICY=システムオプション 379
SYS_SQLSETLIMIT マクロ変数 381

ディクショナリ

SQLCONSTDATETIME System Option

SQL プロシジャがクエリを実行する前に、クエリ内の DATE 関数、TIME 関数、DATETIME 関数、TODAY 関数への参照を等価な定数値で置き換えるかどうかを指定します。

該当要素: 構成ファイル、SAS 呼び出し、OPTIONS ステートメント、SAS システムオプションウィンドウ

カテゴリ: ファイル:SAS ファイル
システム管理:SQL

PROC OPTIONS GROUP= SASFILES
SQL

注: このオプションは、サイト管理者は制限できます。詳細については、“Restricted Options” (*SAS System Options: Reference*)を参照してください。

構文

SQLCONSTDATETIME | **NOSQLCONSTDATETIME**

構文の説明

SQLCONSTDATETIME
SQL プロシジャが DATE 関数、TIME 関数、DATETIME 関数、TODAY 関数への参照を等価な数値定数値で置き換えるように指定します。

NOSQLCONSTDATETIME
SQL プロシジャが DATE 関数、TIME 関数、DATETIME 関数、TODAY 関数への参照を等価な数値定数値で置き換えないように指定します。

詳細

SQLCONSTDATETIME システムオプションを設定した場合、SQL プロシジャは、クエリ内の DATE 関数、TIME 関数、DATETIME 関数、TODAY 関数を一度評価し、クエリを通じてそれらの値を使用します。これらの値を一度計算すると、クエリで関数を複数回使用した場合、またはクエリが日付や時刻の境界近くで関数を実行した場合に結果の一貫性が保たれます。

NOSQLCONSTDATETIME システムオプションを設定すると、SQL プロシジャは、オブザベーションを処理するたびにクエリ内のこれらの関数を評価します。

SQLREDUCEPUT システムオプションと SQLCONSTDATETIME システムオプションの両方が指定された場合、SQL プロシジャは、PUT 関数値を決定するために、クエリを実行する前に DATE 関数、TIME 関数、DATETIME 関数、TODAY 関数を、それぞれの値で置き換えます。

```
select x from &lib..c where (put(bday, date9.) = put(today(), date9.));
```

注: SQLCONSTDATETIME システムオプションで指定した値は、PROC SQL ステートメントで CONSTDATETIME オプションが設定されない限り、すべての SQL プロシジャステートメントに対して有効です。CONSTDATETIME オプションの値は、SQLCONSTDATETIME システムオプションよりも優先されます。ただし、CONSTDATETIME オプションの値を変更しても、SQLCONSTDATETIME システムオプションの値は変更されません。

関連項目:

- “クエリパフォーマンスの向上” (144 ページ)

プロシジャステートメントオプション:

- CONSTDATETIME オプション (222 ページ)

システムオプション:

- “SQLREDUCEPUT=システムオプション” (374 ページ)

SQLGENERATION=システムオプション

SAS プロシジャでソースデータを in-database 処理する SQL を生成するかどうか、またいつ生成するのかを指定します。

該当要素: 構成ファイル、SAS 呼び出し、OPTIONS ステートメント、**SAS システムオプション**ウィンドウ

カテゴリ: システム管理: パフォーマンス

デフォルト: (NONE DBMS='ASTER DB2 GREENPLM NETEZZA ORACLE TERADATA')

制限事項: DBMS=値と EXCLUDEDB=値のエンジン名の最大長は 8 文字です。EXCLUDEPROC=値のプロシジャ名の最大長は 16 文字です。同じエンジンは 1 度しか指定できません。また、同じプロシジャは指定エンジンに対して 1 度しか指定できません。

データソース: Aster *n*Cluster、UNIX および PC ホストの DB2、Greenplum、Netezza、Oracle、Teradata

参照項目: SQLGENERATION= LIBNAME オプション(例を含む)、および *SAS In-Database Products: User's Guide* の"Running In-Database Procedures"

構文

SQLGENERATION=<(>NONE | DBMS <DBMS='*engine1 engine2…enginen* '>
<EXCLUDEDB=*engine* | '*engine1…enginen*'>
<EXCLUDEPROC="*engine='proc1…procn'*
enginen='proc1…procn' "><)>

SQLGENERATION=" "

構文の説明

NONE
in-database 処理に対応した SAS プロシジャが、in-database 処理の SQL を生成しないようにします。これが初期の状態です。

DBMS
in-database 処理に対応した SAS プロシジャに、サポート対象の SAS/ACCESS エンジンを介した DBMS テーブルの in-database 処理のための SQL の生成を許可します。これが初期の状態です。

DBMS='*engine1…enginen*'
1 つ以上の SAS/ACCESS エンジンを指定します。初期の状態が変更されます。

EXCLUDEDB=*engine* | '*engine1…enginen*'
SAS プロシジャが、1 つ以上の指定 SAS/ACCESS エンジンの in-database 処理の SQL を生成しないようにします。

EXCLUDEPROC="*engine='proc1…procn' enginen='proc1…procn'* "
データベース内で実行しないエンジン固有の SAS プロシジャを指定します。

" "
値を出荷時のデフォルトにリセットします。

詳細

PROC FREQ などのプロシジャでこのオプションを使用して、サポート対象の SAS/ACCESS エンジンによる DBMS テーブルの in-database 処理の SQL を生成するかを指定します。

NONE または DBMS、あるいはその両方を指定する必要があります。これらの引数のどちらかまたは両方を使用して、初期の状態を示します。

オプション値の最大長は 4096 です。また、このオプション値に複数のキーワードが含まれている場合はかっこが必要です。

すべてのプロシジャで、すべてのエンジンの種類の in-database 処理の SQL 生成がサポートされているわけではありません。サポートされていない設定を指定した場合、エラーメッセージにサポートされていない SQL 生成のレベルが示され、プロシジャがデフォルトにリセットされるため、SAS 内でソーステーブルレコードの読み取りと処理を行え

ます。この処理が不可能な場合は、プロシジャが終了し、必要に応じて SYSERR=が設定されます。

DATA=データセットと OUT=データセットに異なる SQLGENERATION=値を指定するには、データセットのそれぞれに異なる LIBNAME ステートメントを使用します。

SAS/ACCESS での優先順位の扱いを次に示します。

表 A1.1 *SQLGENERATION= LIBNAME およびシステムオプションの値の優先順位*

LIBNAME オプション	システムオプションでの PROC EXCLUDE の有無	エンジンの種類	システムオプションで指定したエンジン	結果値	ソース(オプション)
未設定 NONE DBMS	あり	データベースインターフェイス	NONE DBMS	NONE EXCLUDEDB	システム
NONE	なし			NONE	LIBNAME
DBMS				DBMS	
未設定			NONE	NONE	システム
NONE DBMS			DBMS	DBMS	LIBNAME
		このデータベースホストやデータベースバージョンに対する SQL を生成しない	NONE DBMS	NONE	
未設定		Base			システム
NONE DBMS					LIBNAME

例

製品出荷時のデフォルトを使用します。

```
options sqlgeneration='';
proc options option=sqlgeneration
run;
```

この例では、SAS プロシジャで、DB2 を除くすべてのデータベースの in-database 処理の SQL が生成されます。

```
options sqlgeneration='';
options sqlgeneration=(DBMS EXCLUDEDB='DB2');
proc options option=sqlgeneration;
run;
```

この例では、Teradata に対してのみ in-database 処理が発生します。その他のデータベース上で実行される SAS プロシジャでは、in-database 処理の SQL は生成されません。

```
options sqlgeneration='';
options SQLGENERATION=(NONE DBMS='Teradata');
```

```
proc options option=sqlgeneration;
run;
```

この例の場合、SAS プロシジャでは、Teradata と Oracle の in-database 処理の SQL が生成されます。ただし、Oracle の PROC1 と PROC2 に対しては SQL は生成されません。

```
options sqlgeneration='';
Options SQLGENERATION = (NONE DBMS='Teradata Oracle'
EXCLUDEPROC="oracle='proc1 proc2'");
proc options option=sqlgeneration;
run;
```

SQLIPONEATTEMPT System Option

暗黙的なパススルー要求が失敗した場合に、PROC SQL で SQL クエリによる処理の続行を許可するかどうかを指定します。

該当要素: 構成ファイル、SAS 呼び出し、OPTIONS ステートメント、SAS システムオプションウィンドウ

カテゴリ: システム管理:SQL

PROC OPTIONS GROUP= SQL

注: このオプションは、サイト管理者は制限できます。詳細については、“Restricted Options” (*SAS System Options: Reference*)を参照してください。

構文

SQLIPONEATTEMPT | **NOSQLIPONEATTEMPT**

構文の説明

SQLIPONEATTEMPT
クエリまたはクエリの一部の送信先の DBMS が暗黙的なパススルーを拒否した場合に、暗黙的なパススルー(IP)を行わないよう PROC SQL に指示します。

NOSQLIPONEATTEMPT
DBMS に送信されるクエリまたはクエリの一部を PROC SQL が拒否できる回数に上限を設定しません。これがデフォルトの設定です。

詳細

DBMS に送信されるクエリのいかなる部分も正常に処理できない場合、その PROC SQL ステートメントは失敗します。拒否が発生した場合、エラーが SAS ログに出力されます。

SQLMAPPUTTO=システムオプション

データベースの SAS_PUT()関数に PUT 関数をマッピングするかどうか、また、可能な場合は、どこに SAS_PUT()関数をマッピングするかを指定します。

該当要素: 構成ファイル、SAS 呼び出し、OPTIONS ステートメント

カテゴリ: ファイル: SAS ファイル

デフォルト: SAS_PUT

データソース: UNIX および PC ホストの DB2、Netezza、Teradata

参照項目: “SQL_FUNCTIONS= LIBNAME Option” (*SAS/ACCESS for Relational Databases: Reference*), *SAS In-Database Products: User’s Guide*

構文

SQLMAPPUTTO= NONE | SAS_PUT | (*database*.SAS_PUT)

構文の説明

NONE
PROC SQL に対して PUT マッピングをしないように指定します。

SAS_PUT
PUT 関数を SAS_PUT()関数にマッピングするように指定します。

***database*.SAS_PUT**
データベース名を指定します。

ヒント 出力形式定義と SAS_PUT()関数は、フォーマット対象のデータを含むデータベースと同じデータベースに存在する必要はありません。*database*.SAS_PUT 引数を使用すると、出力形式定義と SAS_PUT()関数がパブリッシュされているデータベースを指定できます。

ヒント データベース名には複数レベルの名前を指定できます。また、空白も含められます。

要件 データベース名を指定する場合は、引数全体をかっこで囲む必要があります。

詳細

出力形式をパブリッシュするマクロでは、PUT 関数の実装が SAS_PUT()という新しい関数としてデータベースに配置またはパブリッシュされます。また、出力形式をパブリッシュするマクロでは、PROC FORMAT を使用して作成するユーザー定義の出力形式と、SAS が提供する出力形式の両方がパブリッシュされます。SAS_PUT()関数では SAS 出力形式の使用がサポートされるので、データベースにサブミットされる SQL クエリで SAS 出力形式を使用して、SQL クエリ全体をデータベース内で処理できます。また、in-database プロシジャとともに使用することもできます。

このオプションは、SQLREDUCEPUT=、SQLREDUCEPUTOBS および SQLREDUCEPUTVALUES=システムオプションとあわせて使用できます。これらのオプションの詳細については、*SAS SQL プロシジャユーザーガイド*を参照してください。

SQLREDUCEPUT=システムオプション

SQL プロシジャの場合、クエリでの PUT 関数の最適化に使用されるエンジンタイプを指定します。PUT 関数は、論理的に等価な式で置き換えられます。

該当要素: 構成ファイル、SAS 呼び出し、OPTIONS ステートメント、SAS システムオプションウィンドウ

カテゴリ: ファイル:SAS ファイル

システム管理:SQL

システム管理:Performance

PROC OPTIONS GROUP=

SASFILES

SQL

PERFORMANCE

注: このオプションは、サイト管理者は制限できます。詳細については、“Restricted Options” (*SAS System Options: Reference*)を参照してください。

構文

SQLREDUCEPUT= ALL | NONE | DBMS | BASE

構文の説明

ALL
すべての PUT 関数の最適化を考慮することを指定します。これは、クエリがデータのアクセスにどのエンジンを使用するかにかかわりません。

NONE
どの PUT 関数も最適化しないことを指定します。

DBMS
SAS/ACCESS エンジンが実行するクエリ内のすべての PUT 関数について最適化を考慮することを指定します。これがデフォルトです。

要件 PUT 関数に渡す最初の引数は、テーブルから取得される変数である必要があります。テーブルには、SAS/ACCESS エンジンを使用してアクセスする必要があります。

BASE
SAS/ACCESS エンジンまたは Base SAS エンジンが実行するクエリ内のすべての PUT 関数について最適化を考慮することを指定します。

詳細

SQLREDUCEPUT=システムオプションを指定すると、SAS は PUT 関数を最適化してからクエリを実行します。クエリに WHERE 句も含まれる場合、WHERE 句の評価が簡略化されます。次の SELECT ステートメントは、SQLREDUCEPUT=オプションを **none** 以外の任意の値に設定した場合に最適化されるクエリの例です。

```
select x, y from &lib..b where (PUT(x, abc.) in ('yes', 'no'));
select x from &lib..a where (PUT(x, udfmt.) = trim(left('small')));
```

SQLREDUCEPUT=システムオプションと SQLCONSTDATETIME システムオプションの両方が指定された場合、PROC SQL は、PUT 関数値を決定するために、クエリを実行する前に DATE 関数、TIME 関数、DATETIME 関数、TODAY 関数をそれぞれの値で置き換えます。

次の 2 つの SELECT 句は、元のクエリと最適化されたクエリを示しています。

```
select x from &lib..c where (put(bday, date9.) = put(today(), date9.));
```

ここでは、SELECT 句が最適化されています。

```
select x from &lib..c where (x = '17MAR2011'D);
```

クエリに PUT 関数が含まれていない場合、そのクエリは最適化されません。

注: SQLREDUCEPUT=システムオプションで指定した値は、PROC SQL の REDUCEPUT=オプションが設定されない限り、すべての SQL プロシジャステートメントに対して有効です。REDUCEPUT=オプションの値は、SQLREDUCEPUT=システムオプションよりも優先されます。ただし、REDUCEPUT=オプションの値を変更しても、SQLREDUCEPUT=システムオプションの値は変更されません。

関連項目:

- “クエリパフォーマンスの向上” (144 ページ)

プロシジャステートメントオプション:

- REDUCEPUT=オプション (226 ページ)

システムオプション:

- “SQLCONSTDATETIME System Option” (369 ページ)
- “SQLREDUCEPUTOBS=システムオプション” (376 ページ)

SQLREDUCEPUTOBS=システムオプション

SQL プロシジャの場合、SQLREDUCEPUT=システムオプションを DBMS、BASE または ALL に設定した場合、PROC SQL がクエリ内の PUT 関数を最適化するためにテーブルに存在する必要のあるオブザベーションの最小数を指定します。

該当要素: 構成ファイル、SAS 呼び出し、OPTIONS ステートメント、SAS システムオプションウィンドウ

カテゴリ: ファイル:SAS ファイル
システム管理:SQL
システム管理:Performance

PROC OPTIONS GROUP= SASFILES
SQL
PERFORMANCE

操作: SQLREDUCEPUT=システムオプションを DBMS、BASE または ALL に設定した場合、PROC SQL が PUT 関数を最適化するには、SQLREDUCEPUTOBS=システムオプションと SQLREDUCEPUTVALUES=システムオプションの両方の条件が満たされる必要があります。

SQLREDUCEPUTOBS=システムオプションは、テーブルにオブザベーションの数を記録する DBMS に対してのみ動作します。使用している DBMS がオブザベーションの数を記録しなくても、テーブルの行カウントを作成すれば、SQLREDUCEPUTOBS=オプションは動作します。

注: このオプションは、サイト管理者は制限できます。詳細については、“Restricted Options” (*SAS System Options: Reference*)を参照してください。

構文

SQLREDUCEPUTOBS= *n*

構文の説明

n

PROC SQL がクエリ内の PUT 関数を最適化するためにテーブルに存在する必要のある、オブザベーションの最小数を指定します。

デフォルト　0。これは、PROC SQL が PUT 関数を最適化するために使用するテーブル内のオブザベーションの最小数がないことを示します。

範囲　0 から 2^{63}–1(約 9.2x10 の 18 乗)まで

要件　*n* は整数である必要があります。

詳細

テーブルの行数がわからないときに暗黙的なパススルーを許可するデータベースの場合、PROC SQL は、クエリ内の PUT 関数を最適化することを許可し、そのクエリはデータベースによって実行されます。SQLREDUCEPUT=システムオプションが DBMS、BASE または ALL に設定された場合、PROC SQL は、SQLREDUCEPUTVALUES=システムオプションと SQLREDUCEPUTOBS=システムオプションの両方の値を考慮して PUT 関数を最適化するかどうかを決定します。

暗黙的なパススルーを許可しないデータベースの場合、PROC SQL は PUT 関数を最適化せず、より多くのクエリが SAS によって実行されます。

関連項目:

- “クエリパフォーマンスの向上” (144 ページ)

システムオプション:

- “SQLREDUCEPUT=システムオプション” (374 ページ)
- “SQLREDUCEPUTVALUES=システムオプション” (377 ページ)

SQLREDUCEPUTVALUES=システムオプション

SQL プロシジャでは、SQLREDUCEPUT=システムオプションを DBMS、BASE または ALL に設定した場合、PROC SQL がクエリ内の PUT 関数を最適化するために PUT 関数式に存在できる SAS 出力形式値の最大数を指定します。

該当要素: 構成ファイル、SAS 呼び出し、OPTIONS ステートメント、SAS システムオプションウィンドウ

カテゴリ: ファイル:SAS ファイル
システム管理:SQL
システム管理:Performance

PROC OPTIONS GROUP= SASFILES
SQL
PERFORMANCE

操作: SQLREDUCEPUT=システムオプションを DBMS、BASE または ALL に設定した場合、PROC SQL が PUT 関数を最適化するには、SQLREDUCEPUTOBS=システムオプションと SQLREDUCEPUTVALUES=システムオプションの両方の条件が満たされる必要があります。

注: このオプションは、サイト管理者は制限できます。詳細については、“Restricted Options” (*SAS System Options: Reference*)を参照してください。

構文

SQLREDUCEPUTVALUES= *n*

構文の説明

n

PROC SQL がクエリ内の PUT 関数を最適化するために PUT 関数式に存在できる、SAS 出力形式値の最大数を指定します。

デフォルト	100
範囲	100 から 3,000 まで
要件	*n* は整数である必要があります。
操作	PUT 関数式内の SAS 出力形式値の数がこの値よりも多い場合、PROC SQL は PUT 関数を最適化しません。

詳細

出力形式(特に、ユーザー定義の出力形式)によっては、多くの出力形式値が含まれる場合があります。PUT 関数式に対応する数に応じて、結果として得られる式には、多くの出力形式値が記述される場合があります。出力形式値の数が多すぎる場合、クエリのパフォーマンスが低下する可能性があります。SQLREDUCEPUT=システムオプションが DBMS、BASE または ALL に設定された場合、PROC SQL は、SQLREDUCEPUTVALUES=システムオプションと SQLREDUCEPUTOBS=システムオプションの両方の値を考慮して PUT 関数を最適化するかどうかを決定します。

ヒント SQLREDUCEPUTVALUES=の値は、個別の最適化で使用されます。たとえば、WHERE 句に PUT 関数が存在し、GROUP BY 句に別の PUT 関数が存在する場合、SQLREDUCEPUTVALUES=の値は、それぞれの句に別々に適用されます。

関連項目:

- “クエリパフォーマンスの向上” (144 ページ)

システムオプション:

- “SQLREDUCEPUT=システムオプション” (374 ページ)
- “SQLREDUCEPUTOBS=システムオプション” (376 ページ)

SQLREMERGE システムオプション

再マージされるデータを使用するクエリを PROC SQL が処理できるかどうかを指定します。

該当要素: 構成ファイル、SAS 呼び出し、OPTIONS ステートメント、SAS システムオプションウィンドウ

カテゴリ: ファイル:SAS ファイル

システム管理:SQL

PROC OPTIONS GROUP= SASFILES
SQL

注: このオプションは、サイト管理者は制限できます。詳細については、“Restricted Options” (*SAS System Options: Reference*)を参照してください。

構文

SQLREMERGE | **NOSQLREMERGE**

構文の説明

SQLREMERGE
再マージされるデータを使用するクエリを PROC SQL が処理できるかことを指定します。

NOSQLREMERGE
再マージされるデータを使用するクエリを PROC SQL が処理できないことを指定します。

詳細

PROC SQL の再マージ機能は、テーブルの 2 回のパススルーを実行します。1 回目のパススルーで作成されたデータは、2 回目のパススルーでクエリを実行するために使用されます。NOSQLREMERGE システムオプションが指定されると、PROC SQL は再マージされたデータを処理できません。NOSQLREMERGE システムオプションを指定して再マージを実行しようとすると、エラーが SAS ログに書き込まれます。

関連項目:

- “クエリパフォーマンスの向上” (144 ページ)

プロシジャステートメントオプション:

- REMERGE オプション (228 ページ)
- “summary-function” (356 ページ)

SQLUNDOPOLICY=システムオプション

データの更新中にエラーが発生した場合に PROC SQL が更新済みデータをどう処理するかを指定します。UNDO_POLICY=を使用して、変更を永続化するかどうかを制御できます。

該当要素: 構成ファイル、SAS 呼び出し、オプションステートメント

カテゴリ: ファイル:SAS ファイル
システム管理:SQL

PROC OPTIONS GROUP= SASFILES
SQL

注: このオプションは、サイト管理者は制限できます。詳細については、“Restricted Options” (*SAS System Options: Reference*)を参照してください。

構文

SQLUNDOPOLICY=NONE | OPTIONAL | REQUIRED

構文の説明

NONE
すべての更新または挿入を維持します。

OPTIONAL
確実に破棄できるすべての更新または挿入を破棄します。

REQUIRED
エラー発生時までに実行されたすべての更新または挿入を破棄します。これがデフォルトです。

注意 **UNDO 操作によっては、確実に実行できるとは限りません。** 場合によっては、確実に UNDO 操作を実行できるとは限りません。変更を破棄できない場合、PROC SQL はエラーメッセージを発行し、ステートメントを実行しません。たとえば、プログラムが SAS/ACCESS ビューを使用する場合、または SAS データセットが SAS/SHARE サーバーを介してアクセスされ、CNTLLEV=RECORD データセットオプションを使用して開かれた場合、変更を確実に破棄することはできません。

注意 **UNDO 操作によっては、変更を破棄できない場合があります。** 複数のトランザクションが同じレコードに対して実行された場合、PROC SQL は変更を破棄できません。PROC SQL は、破棄するかわりにエラーメッセージを発行します。たとえば、あるトランザクションでレコードを挿入した際にエラーが発生し、別のトランザクションでその同じレコードが削除された場合、UNDO 操作はそのレコードの削除を破棄できないため、エラーメッセージが発行されます。

詳細

SQLUNDOPOLICY=システムオプションで指定した値は、PROC SQL の UNDO_POLICY=オプションが設定されない限り、すべての SQL プロシジャステートメントに対して有効です。UNDO_POLICY=オプションの値は、SQLUNDOPOLICY=システムオプションよりも優先されます。RESET ステートメントを使用して、UNDO_POLICY=オプションを設定またはリセットすることもできます。ただし、UNDO_POLICY=オプションの値を変更しても、SQLUNDOPOLICY=システムオプションの値は変更されません。SQL プロシジャの完了後、設定は SQLUNDOPOLICY=システムオプションの値に戻ります。

SAS Scalable Performance Data Engine を使用してデータセットを更新している場合、SQLUNDOPOLICY=NONE を設定することによって処理のパフォーマンスを大幅に向上できます。ただし、NONE がアプリケーションに適した設定であることを確認してください。

関連項目:

プロシジャステートメント

- UNDO_POLICY (230 ページ)

SYS_SQLSETLIMIT マクロ変数

SQL プロシジャでは、DBMS 処理中にハッシュ結合の最適化に使用される値の最大数を指定します。

構文

SYS_SQLSETLIMIT= *n*;

必須引数

n

処理するために DBMS に渡される IN 条件に含まれる値の最大数を指定します。

デフォルト 1024

制限事項 SYS_SQLSETLIMIT マクロ変数は、特定のハッシュ結合にのみ影響を与えます。

例

```
%let SYS_SQLSETLIMIT=250;
%let SYS_SQLSETLIMIT=1200;
```

詳細

ハッシュ結合

SQL プロシジャは、インデックス結合が除去されている場合に、パフォーマンスを最適化するためにハッシュ結合を使用する可能性があります。ハッシュ結合では、より小さいテーブルがハッシュテーブルとしてメモリ内で再構成されます。PROC SQL は、より大きなテーブルを順番にスキャンし、より小さいテーブルに対して行ごとにハッシュ検索を実行して、結果セットを作成します。ハッシュ結合を使用するかどうかは、メモリサイズ計算式によって決定されます。この計算式は、PROC SQL の UBUFSIZE オプションに基づきます。UBUFSIZE のデフォルト値は、64KB です。メモリに余裕のあるシステムでは、ハッシュ結合が使用される可能性を高めるために、UBUFSIZE を増やすことを検討してください。

付録 2

PROC SQL および ANSI 規格

準拠

PROC SQL は、ANSI(米国規格協会)によって設定された SQL 実装ガイドラインに準拠しています。ただし、現在の SQL の ANSI 規格に完全に準拠しているわけではありません。[2]

SAS の SQL リサーチプロジェクトは、SQL の照会言語としての表現機能を中心に研究してきました。その結果、SQL のデータベース機能の一部は、PROC SQL にはまだ実装されていません。

SQL プロシジャの拡張点

予約語

PROC SQL で予約されているキーワードはきわめて少数です。また、特定のコンテキストでのみキーワードが予約されます。ANSI 規格は、すべての SQL キーワードをすべてのコンテキストで予約しています。たとえば、ANSI 規格に従うと、GROUP BY キーワードが予約されているため、GROUP という名前の列を使用できません。

PROC SQL では、次の単語が予約されています。

- CASE キーワードは常に予約されています。このキーワードは(SQL2 の機能として)CASE 式で使用されるため、列名としては使用できません。

  テーブルに CASE という名前の列が存在し、それを PROC SQL ステップで指定する場合、SAS データセットオプションの RENAME=を使用して、クエリが存続する間、その列の名前を変更できます。CASE を二重引用符(“CASE”)で囲み、PROC SQL のオプションで DQUOTE=ANSI を設定できます。

- テーブルのエイリアスとして、AS、ON、FULL、JOIN、LEFT、FROM、WHEN、WHERE、ORDER、GROUP、RIGHT、INNER、OUTER、UNION、EXCEPT、HAVING、INTERSECT のいずれのキーワードも使用できません。これらのキーワードは、テーブル名の後に記述される句を表します。テーブルのエイリアスは任意です。そのため PROC SQL は、これらの単語のいずれかが対応する句を表しており、テーブルのエイリアスではないと仮定することによって、この曖昧さに対処します。これらのキーワードのいずれかをテーブルのエイリアスとして使用する場合、キーワードを二重引用符で囲み、PROC SQL のオプションで DQUOTE=ANSI を設定します。

- USER キーワードは、現在のユーザー ID 用に予約されています。CREATE TABLE ステートメントと共に SELECT ステートメントで USER を指定した場合、一時的な(_TEMA001 に似た)列名を持つ列がテーブルに作成されます。CREATE TABLE を使用せずに SELECT ステートメントで USER を指定した場合、列見出し

2 国際標準化機構(ISO):データベース SQL。ドキュメント ISO/IEC 9075:1992。ANSI(米国規格協会)のドキュメント ANSI X3.135-1992 としても提供されています。

のない列が出力に書き込まれます。いずれの場合も、列の値はオペレーティングシステムによって変わりますが、通常は、プログラムをサブミットするユーザーのユーザー ID または&SYSJOBID 自動マクロ変数の値になります。

テーブルに USER という名前の列が存在し、それを PROC SQL ステップで指定する場合、SAS データセットオプションの RENAME=を使用して、クエリが存続する間、その列の名前を変更できます。USER を二重引用符(“USER”)で囲み、PROC SQL のオプションで DQUOTE=ANSI を設定できます。

列の修飾子

PROC SQL は、SELECT ステートメントの式で、SAS の INFORMAT=修飾子、FORMAT=修飾子および LABEL=修飾子をサポートしています。これらの修飾子は、出力データを表示してラベルを付ける出力形式を制御します。

照合順序の変更

PROC SQL では、ORDER BY 句を指定したときに使用される、別の照合(並べ替え)順序を指定できます。SORTSEQ=オプションの詳細については、次を参照してください。“PROC SQL ステートメント” (220 ページ)

ビュー定義での ORDER BY 句

PROC SQL では、CREATE VIEW ステートメント内で ORDER BY 句を指定できます。ビューを照会するときに、そのビューに対するクエリに別の ORDER BY 句が含まれていなければ、ビューのデータは指定した順序に基づいて並べ替えられます。詳細については、“CREATE VIEW ステートメント” (242 ページ)を参照してください。

CONTAINS 条件

PROC SQL で CONTAINS 条件を指定すると、文字列が列の値に含まれているかどうかを検証できます。詳細については、“CONTAINS 条件” (320 ページ)を参照してください。

インラインビュー

FROM 句にネストされたクエリ式を記述できる機能は、ANSI 規格の要件です。PROC SQL では、ネストされたコードがサポートされます。

外部結合

一致する列と一致しない列の両方を結合式に含める機能は、ANSI 規格の要件です。PROC SQL では、この機能がサポートされます。

算術演算子

PROC SQL では、SAS の累乗演算子(**)がサポートされます。PROC SQL では、等しくないことを意味する<> 表記が使用されます。

直交式

PROC SQL では、比較式、ブール式および算術演算式を組み合わせることができます。たとえば、(X=3)*7 という式の値は、X=3 が TRUE の場合、TRUE が 1 として定義されているため、7 になります。X=3 が FALSE の場合、0 に決定されるため、この式全体の値は 0 になります。

PROC SQL では、任意の式でのサブクエリが許可されています。この機能は、ANSI 規格によって要求されています。したがって、WHERE 式内の比較演算子の左側で、サブクエリを記述できます。

PROC SQL では、ORDER BY 句と GROUP BY 句を使用した、任意のタイプの算術演算式(ただし、要約関数を含む算術演算式を除く)によるデータの並べ替えおよびグループ化が許可されています。また、SELECT ステートメントに記述した式によってグループ化するには、SELECT ステートメントでの式の位置を表す整数値を使用します。グループ化または並べ替えの基準として使用する式を選択する必要はありません。詳細については、次を参照してください。“ORDER BY 句” (263 ページ) および “GROUP BY 句” (261 ページ)

セット演算子

セット演算子である UNION、INTERSECT および EXCEPT は、ANSI 規格によって要求されています。PROC SQL では、これらの演算子に加えて、OUTER UNION 演算子が提供されます。

ANSI 規格は、操作対象のテーブルに、データタイプが一致する同じ数の列が存在することを要求しています。SQL プロシジャは、列の数が同じであるテーブルを操作しますが、列の数が異なるテーブルについても、クエリが正しく評価できるように仮想的な列を作成することよって操作します。詳細については、“query-expression” (339 ページ)を参照してください。

統計関数

PROC SQL では、SQL の ANSI 規格で要求されている要約関数よりもはるかに多くの要約関数がサポートされます。

PROC SQL では、要約関数の結果とテーブルの元のデータとの再マージがサポートされます。たとえば、合計に対する割合(パーセンテージ)の計算は、PROC SQL では `100*x/SUM(x)` によって実現できます。要約関数とデータの再マージの詳細については、次を参照してください。“summary-function” (356 ページ)

SAS DATA Step 関数

PROC SQL では、SAS DATA ステップで使用可能な多くの関数がサポートされます。サポートされない関数には、変数情報関数、データの配列を操作する関数などがあります。ほかの SQL データベースでは、独自の関数セットがサポートされています。

注意:

PROC SQL で使用される SAS DATA ステップ関数は、引数に関して ANSI SQL 規格ガイドラインおよび ISO SQL 規格ガイドラインに従う必要があります。SAS DATA ステップ関数には、空の引数を含めることはできません。

PROC FCMP 関数

PROC SQL は、ユーザーが記述した任意の関数をサポートします。ただし、“FCMP” (*Base SAS Procedures Guide*)を使用して作成された配列要素を操作する関数は除きます。

CALCULATED キーワード

CALCULATED キーワードによって、同じ SELECT 句内または WHERE 句内の式の結果を使用できます。詳細については、“CALCULATED” (313 ページ)および “CALCULATED キーワードと列エイリアスの使用” (150 ページ)を参照してください。

SQL プロシジャの省略

COMMIT ステートメント

COMMIT ステートメントはサポートされません。

ROLLBACK ステートメント

ROLLBACK ステートメントはサポートされません。PROC SQL の UNDO_POLICY= オプションまたは SQLUNDOPOLICY システムオプションによってロールバックに対応します。“PROC SQL ステートメント” (220 ページ) または“SQLUNDOPOLICY=システムオプション” (379 ページ)の UNDO_POLICY=オプションの説明を参照してください。

識別子と命名規則

SAS では、テーブル名、列名およびエイリアスは 32 文字までに制限されており、大文字と小文字の混在が許容されます。SAS の命名規則の詳細については、*Base SAS Utilities: リファレンス*を参照してください。SQL の ANSI 規格では、さらに長い名前が可能です。

ユーザー権限の付与

SQL の機能である GRANT ステートメント、PRIVILEGES キーワードおよび承認識別子はサポートされません。これらのかわりに、オペレーティングシステム固有のセキュリティ保護手段を使用することをお勧めします。

三値論理

ANSI 準拠 SQL は、三値論理を備えています。つまり、この論理には、NULL 値に関する比較を処理するための特殊ケースが含まれています。NULL 値と比較された値は、すべて NULL と評価されます。

PROC SQL では、SAS の規則に準拠して欠損値が処理されます。NULL の数値が NULL 以外の数値と比較された場合、NULL 値は NULL 以外のすべての値よりも小さいと判定されます。NULL の文字値が NULL 以外の文字と比較された場合、NULL の文字値は空白文字列として扱われます。

埋め込み SQL

現在は、DATA ステップや SAS/IML ソフトウェアなど、他の SAS プログラミング環境に PROC SQL ステートメントを埋め込むことはできません。

列のエイリアス

PROC SQL では、ANSI 規格でサポートされていないエイリアスの使用がサポートされています。詳細については、“列エイリアスの使用” (148 ページ)を参照してください。

付録 3

「SQL プロシジャの使用」で示されているコード例

この付録では、“SQL プロシジャの使用“セクション全体に含まれているコード例を示します。コードを SAS エディタにコピーアンドペーストする場合、HTML ページからコードをコピーするならば、そのコード内のスペーシングが保存されます。

SQL プロシジャについて

一部の出力テーブルでは、オブザベーション数を制限しています。テーブル全体を表示するには、SQL プロシジャの OUTOBS=オプションを削除します。

```
/*---------------------------------------------------------------------
   Output 1.1 Sample SQL Output

---------------------------------------------------------------------*/
proc sql;
   title 'Population of Large Countries Grouped by Continent';
   select Continent, sum(Population) as TotPop format=comma15.
      from sql.countries
      where Population gt 1000000
      group by Continent
      order by TotPop;
quit;

/*---------------------------------------------------------------------
   Output 1.2 Sample DATA Step Output

---------------------------------------------------------------------*/
title 'Large Countries Grouped by Continent';
proc summary data=sql.countries;
   where Population > 1000000;
   class Continent;
   var Population;
   output out=sumPop sum=TotPop;
run;

proc sort data=SumPop;
   by totPop;
run;

proc print data=SumPop noobs;
   var Continent TotPop;
   format TotPop comma15.;
   where _type_=1;
run;
```

```
/*-------------------------------------------------------------------
   Output 1.3 Countries (Partial Output)

-------------------------------------------------------------------*/
options nodate nonumber linesize=84 pagesize=60;
proc sql outobs=15;
 title 'Countries';
 select Name format=$19., Capital format=$15.,
        Population, Area, Continent format=$15., UNDate format=year4.
    from sql.countries;

/*-------------------------------------------------------------------
   Output 1.4 WorldCityCoords (Partial Output)

-------------------------------------------------------------------*/
proc sql outobs=15;
 title 'WorldCityCoords';
 select City format=$15., Country format=$12., Latitude, Longitude
    from sql.worldcitycoords;

/*-------------------------------------------------------------------
   Output 1.5 USCityCoords (Partial Output)

-------------------------------------------------------------------*/
proc sql outobs=15;
 title 'USCityCoords';
 select City format=$15., State format=$2., Latitude, Longitude
    from sql.uscitycoords;

/*-------------------------------------------------------------------
   Output 1.6 UnitedStates (Partial Output)

-------------------------------------------------------------------*/
proc sql outobs=15;
 title 'UnitedStates';
 select Name format=$17., Capital format=$15.,
        Population, Area, Continent format=$13., Statehood format=date9.
    from sql.unitedstates;

/*-------------------------------------------------------------------
   Output 1.7 PostalCodes (Partial Output)
.
-------------------------------------------------------------------*/
proc sql outobs=15;
title 'PostalCodes';
 select Name , Code
    from sql.postalcodes;

/*-------------------------------------------------------------------
   Output 1.8 WorldTemps (Partial Output)

-------------------------------------------------------------------*/
proc sql outobs=15;
title 'WorldTemps';
 select City, Country,avghigh, avglow
```

```
    from sql.worldtemps;

/*---------------------------------------------------------------------
   Output 1.9 OilProd (Partial Output)

---------------------------------------------------------------------*/
proc sql outobs=15;
title 'OilProd';
 select Country, BarrelsPerDay
    from sql.oilprod;

/*---------------------------------------------------------------------
   Output 1.10 OilRsrvs (Partial Output)

---------------------------------------------------------------------*/
proc sql outobs=15;
title 'OilRsrvs';
 select Country, Barrels
  from sql.oilrsrvs;

/*---------------------------------------------------------------------
   Output 1.11 Continents

---------------------------------------------------------------------*/
proc sql outobs=15;
title 'Continents';
 select Name format=$15., Area,
        Highpoint format =$15., Height, LowPoint format =$15., Depth
    from sql. continents;

/*---------------------------------------------------------------------
   Output 1.12 Features (Partial Output)

---------------------------------------------------------------------*/
proc sql outobs=15;
title 'Features';
 select Name format=$15., Type,Location format =$15.,Area,
        Height, Depth, Length
    from sql. features;
```

1 つのテーブルからのデータの取得

一部の出力テーブルでは、オブザベーション数を制限しています。テーブル全体を表示するには、SQL プロシジャの OUTOBS=オプションを削除します。

```
/*---------------------------------------------------------------------
   Output 2.1 Selecting All Columns in a Table

---------------------------------------------------------------------*/
proc sql outobs=12;
   title 'U.S. Cities with Their States and Coordinates';
   select *
      from sql.uscitycoords;

/*---------------------------------------------------------------------
   Output 2.2 Selecting One Column
```

```
-----------------------------------------------------------------*/
proc sql outobs=12;
   title 'Names of U.S. Cities';
   select City
      from sql.uscitycoords;

/*-----------------------------------------------------------------
   Output 2.3 Selecting Multiple Columns

-----------------------------------------------------------------*/
proc sql outobs=12;
   title 'U.S. Cities and Their States';
   select City, State
      from sql.uscitycoords;

/*-----------------------------------------------------------------
   Output 2.4 Selecting a Column with Duplicate Values

-----------------------------------------------------------------*/
proc sql outobs=12;
   title 'Continents of the United States';
   select Continent
      from sql.unitedstates;

/*-----------------------------------------------------------------
   Output 2.5 Eliminating Duplicate Values

-----------------------------------------------------------------*/
proc sql;
   title 'Continents of the United States';
   select distinct Continent
      from sql.unitedstates;

/*-----------------------------------------------------------------
   Log 2.1 Portion of Log to Determine the Structure of a Table

-----------------------------------------------------------------*/
proc sql;
   describe table sql.unitedstates;

/*-----------------------------------------------------------------
   Output 2.6 Adding Text to Output

-----------------------------------------------------------------*/
proc sql outobs=12;
   title 'U.S. Postal Codes';
   select 'Postal code for', Name, 'is', Code
      from sql.postalcodes;

/*-----------------------------------------------------------------
   Output 2.7 Suppressing Column Headings in Output

-----------------------------------------------------------------*/
proc sql outobs=12;
   title 'U.S. Postal Codes';
```

```
   select 'Postal code for', Name label='#', 'is', Code label='#'
      from sql.postalcodes;

/*---------------------------------------------------------------
   Output 2.8 Calculating Values

---------------------------------------------------------------*/
proc sql outobs=12;
   title 'Low Temperatures in Celsius';
   select City, (AvgLow - 32) * 5/9 format=4.1
      from sql.worldtemps;

/*---------------------------------------------------------------
   Output 2.9 Assigning a Column Alias to a Calculated Column

---------------------------------------------------------------*/
proc sql outobs=12;
   title 'Low Temperatures in Celsius';
   select City, (AvgLow - 32) * 5/9 as LowCelsius format=4.1
      from sql.worldtemps;

/*---------------------------------------------------------------
   Output 2.10 Referring to a Calculated Column by Alias

---------------------------------------------------------------*/
proc sql outobs=12;
   title 'Range of High and Low Temperatures in Celsius';
      select City, (AvgHigh - 32) * 5/9 as HighC format=5.1,
                   (AvgLow - 32) * 5/9 as LowC format=5.1,
                   (calculated HighC - calculated LowC)
                    as Range format=4.1
   from sql.worldtemps;

/*---------------------------------------------------------------
   Output 2.11 Using a Simple CASE Expression

---------------------------------------------------------------*/
proc sql outobs=12;
   title 'Climate Zones of World Cities';
   select City, Country, Latitude,
          case
             when Latitude gt 67 then 'North Frigid'
             when 67 ge Latitude ge 23 then 'North Temperate'
             when 23 gt Latitude gt -23 then 'Torrid'
             when -23 ge Latitude ge -67 then 'South Temperate'
             else 'South Frigid'
          end as ClimateZone
      from sql.worldcitycoords
      order by City;

/*---------------------------------------------------------------
   Output 2.12 Using a CASE Expression in the CASE-OPERAND Form

---------------------------------------------------------------*/
proc sql outobs=12;
   title 'Assigning Regions to Continents';
```

```
   select Name, Continent,
          case Continent
             when 'North America' then 'Continental U.S.'
             when 'Oceania' then 'Pacific Islands'
             else 'None'
          end as Region
      from sql.unitedstates;

/*---------------------------------------------------------------------
   Output 2.13 Using the COALESCE Function to Replace Missing Values

   Note: Either of the following two SQL statements will
         create Output 2.13.
---------------------------------------------------------------------*/
proc sql;
   title 'Continental Low Points';
   select Name, coalesce(LowPoint, 'Not Available') as LowPoint
      from sql.continents;

proc sql;
   title 'Continental Low Points';
   select Name, case
                   when LowPoint is missing then 'Not Available'
                   else Lowpoint
                end as LowPoint
      from sql.continents;
/*---------------------------------------------------------------------
   Output 2.14 Specifying Column Attributes

---------------------------------------------------------------------*/
proc sql outobs=12;
   title 'Areas of U.S. States in Square Miles';
   select Name label='State', Area format=comma10.
      from sql.unitedstates;

/*---------------------------------------------------------------------
   Output 2.15 Sorting by Column

---------------------------------------------------------------------*/
proc sql outobs=12;
   title 'Country Populations';
   select Name, Population format=comma10.
      from sql.countries
      order by Population;

/*---------------------------------------------------------------------
   Output 2.16 Sorting by Multiple Columns

---------------------------------------------------------------------*/
proc sql outobs=12;
   title 'Countries, Sorted by Continent and Name';
   select Name, Continent
      from sql.countries
      order by Continent, Name;

/*---------------------------------------------------------------------
```

```
   Output 2.17 Specifying a Sort Order

-----------------------------------------------------------------------*/
proc sql outobs=12;
   title 'World Topographical Features';
   select Name, Type
      from sql.features
      order by Type desc, Name;

/*-----------------------------------------------------------------------
   Output 2.18 Sorting by Calculated Column

-----------------------------------------------------------------------*/
proc sql outobs=12;
   title 'World Population Densities per Square Mile';
   select Name, Population format=comma12., Area format=comma8.,
          Population/Area as Density format=comma10.
      from sql.countries
      order by Density desc;

/*-----------------------------------------------------------------------
   Output 2.19 Sorting by Column Position

-----------------------------------------------------------------------*/
proc sql outobs=12;
   title 'World Population Densities per Square Mile';
   select Name, Population format=comma12., Area format=comma8.,
          Population/Area format=comma10. label='Density'
      from sql.countries
      order by 4 desc;

/*-----------------------------------------------------------------------
   Output 2.20 Sorting by Columns That Are Not Selected

-----------------------------------------------------------------------*/
proc sql outobs=12;
   title 'Countries, Sorted by Population';
   select Name, Continent
      from sql.countries
      order by Population;

/*-----------------------------------------------------------------------
   Output 2.21 Sorting Columns That Contain Missing Values

-----------------------------------------------------------------------*/
proc sql;
   title 'Continents, Sorted by Low Point';
   select Name, LowPoint
      from sql.continents
      order by LowPoint;

/*-----------------------------------------------------------------------
   Output 2.22 Using a Simple WHERE Clause

-----------------------------------------------------------------------*/
proc sql outobs=12;
```

```
   title 'Countries in Europe';
   select Name, Population format=comma10.
      from sql.countries
      where Continent = 'Europe';

/*---------------------------------------------------------------------
   Output 2.23 Retrieving Rows Based on a Comparison

---------------------------------------------------------------------*/
proc sql;
   title 'States with Populations over 5,000,000';
   select Name, Population format=comma10.
      from sql.unitedstates
      where Population gt 5000000
      order by Population desc;

/*---------------------------------------------------------------------
   Output 2.24 Retrieving Rows That Satisfy Multiple Conditions

---------------------------------------------------------------------*/
proc sql;
   title 'Countries in Africa with Populations over 20,000,000';
   select Name, Population format=comma10.
      from sql.countries
      where Continent = 'Africa' and Population gt 20000000
      order by Population desc;

/*---------------------------------------------------------------------
   Output 2.25 Using the IN Operator

---------------------------------------------------------------------*/
proc sql outobs=12;
   title 'World Mountains and Waterfalls';
   select Name, Type, Height format=comma10.
      from sql.features
      where Type in ('Mountain', 'Waterfall')
      order by Height;

/*---------------------------------------------------------------------
   Output 2.26 Using the IS MISSING Operator

---------------------------------------------------------------------*/
proc sql;
   title 'Countries with Missing Continents';
   select Name, Continent
      from sql.countries
      where Continent is missing;

/*---------------------------------------------------------------------
   Output 2.27 Using the BETWEEN-AND Operators

---------------------------------------------------------------------*/
proc sql outobs=12;
   title 'Equatorial Cities of the World';
   select City, Country, Latitude
      from sql.worldcitycoords
```

```
      where Latitude between -5 and 5;

/*---------------------------------------------------------------
   Output 2.28 Using the LIKE Operator

---------------------------------------------------------------*/
proc sql;
   title1 'Country Names that Begin with the Letter "Z"';
   title2 'or Are 5 Characters Long and End with the Letter "a"';
   select Name
      from sql.countries
      where Name like 'Z%' or Name like '____a';

/*---------------------------------------------------------------
   Output 2.29 Using a Truncated String Comparison Operator

---------------------------------------------------------------*/
proc sql;
   title '"New" U.S. States';
   select Name
      from sql.unitedstates
      where Name eqt 'New ';

/*---------------------------------------------------------------
   Output 2.30 Using a WHERE Clause with Missing Values (Incorrect Output)

---------------------------------------------------------------*/
/* incorrect output */

proc sql outobs=12;
   title 'World Features with a Depth of Less than 500 Feet';
   select Name, Depth
      from sql.features
      where Depth lt 500
      order by Depth;

/*---------------------------------------------------------------
   Output 2.31 Using a WHERE Clause with Missing Values (Corrected Output)

---------------------------------------------------------------*/
/* corrected output */

proc sql outobs=12;
   title 'World Features with a Depth of Less than 500 Feet';
   select Name, Depth
      from sql.features
      where Depth lt 500 and Depth is not missing
      order by Depth;

/*---------------------------------------------------------------
   Output 2.32 Using the MEAN Function with a WHERE Clause

---------------------------------------------------------------*/
proc sql outobs=12;
   title 'Mean Temperatures for World Cities';
   select City, Country, mean(AvgHigh, AvgLow)
```

```
          as MeanTemp
      from sql.worldtemps
      where calculated MeanTemp gt 75
      order by MeanTemp desc;

/*---------------------------------------------------------------------
   Output 2.33 Displaying Sums

---------------------------------------------------------------------*/
proc sql;
   title 'World Oil Reserves';
   select sum(Barrels) format=comma18. as TotalBarrels
      from sql.oilrsrvs;

/*---------------------------------------------------------------------
   Output 2.34 Using Aggregate Functions

---------------------------------------------------------------------*/
proc sql outobs=12;
   title 'Largest Country Populations';
   select Name, Population format=comma20.,
          max(Population) as MaxPopulation format=comma20.
      from sql.countries
      order by Population desc;

/*---------------------------------------------------------------------
   Output 2.35 Remerging Summary Statistics

---------------------------------------------------------------------*/
proc sql outobs=12;
   title 'Percentage of World Population in Countries';
   select Name, Population format=comma14.,
          (Population / sum(Population) * 100) as Percentage
          format=comma8.2
      from sql.countries
      order by Percentage desc;

/*---------------------------------------------------------------------
   Output 2.36 Using DISTINCT with the COUNT Function

---------------------------------------------------------------------*/
proc sql;
   title 'Number of Continents in the COUNTRIES Table';
   select count(distinct Continent) as Count
      from sql.countries;

/*---------------------------------------------------------------------
   Output 2.37 Effect of Not Using DISTINCT with the COUNT Function

---------------------------------------------------------------------*/
proc sql;
   title 'Countries for Which a Continent is Listed';
   select count(Continent) as Count
      from sql.countries;

/*---------------------------------------------------------------------
```

```
   Output 2.38 Using the COUNT FUnction to Count All Rows in a Table

-------------------------------------------------------------------*/
proc sql;
   title 'Number of Countries in the SQL.COUNTRIES Table';
   select count(*) as Number
      from sql.countries;

/*-------------------------------------------------------------------
   Output 2.39 Finding Errors Caused by Missing Values (Unexpected Output)

-------------------------------------------------------------------*/
/* unexpected output */

proc sql;
   title 'Average Length of Angel Falls, Amazon and Nile Rivers';
   select Name, Length, avg(Length) as AvgLength
      from sql.features
      where Name in ('Angel Falls', 'Amazon', 'Nile');

/*-------------------------------------------------------------------
   Output 2.40 Finding Errors Caused by Missing Values (Modified Output)

-------------------------------------------------------------------*/
/* modified output */

proc sql;
   title 'Average Length of Angel Falls, Amazon and Nile Rivers';
   select Name, Length, coalesce(Length, 0) as NewLength,
               avg(calculated NewLength) as AvgLength
      from sql.features
      where Name in ('Angel Falls', 'Amazon', 'Nile');

/*-------------------------------------------------------------------
   Output 2.41 Grouping by One Column

-------------------------------------------------------------------*/
proc sql;
   title 'Total Populations of World Continents';
   select Continent, sum(Population) format=comma14. as TotalPopulation
      from sql.countries
      where Continent is not missing
      group by Continent;

/*-------------------------------------------------------------------
   Output 2.42 Grouping without Aggregate Functions
   Log 2.2 Grouping without Aggregate Functions (Partial Log)

-------------------------------------------------------------------*/
proc sql outobs=12;
   title 'High and Low Temperatures';
   select City, Country, AvgHigh, AvgLow
      from sql.worldtemps
      group by Country;

/*-------------------------------------------------------------------
```

```
   Output 2.43 Grouping by Multiple Columns

-------------------------------------------------------------------*/
proc sql;
   title 'Total Square Miles of Deserts and Lakes';
   select Location, Type, sum(Area) as TotalArea format=comma16.
      from sql.features
      where type in ('Desert', 'Lake')
      group by Location, Type;

/*-------------------------------------------------------------------
   Output 2.44 Grouping with an ORDER BY Clause

-------------------------------------------------------------------*/
options nodate nonumber linesize=90 pagesize=60;
proc sql;
   title 'Total Square Miles of Deserts and Lakes';
   select Location, Type, sum(Area) as TotalArea format=comma16.
      from sql.features
      where type in ('Desert', 'Lake')
      group by Location, Type
      order by Location desc;

/*-------------------------------------------------------------------
   Output 2.45 Finding Grouping Errors Caused by Missing Values
               (Unexpected Output)

-------------------------------------------------------------------*/
options nodate nonumber linesize=90 pagesize=60;
/* unexpected output */

proc sql outobs=12;
   title 'Areas of World Continents';
   select Name format=$25.,
          Continent,
          sum(Area) format=comma12. as TotalArea
      from sql.countries
      group by Continent
      order by Continent, Name;

/*-------------------------------------------------------------------
   Output 2.46 Adjusting the Query to Avoid Errors Due to Missing Values
               (Modified Output)

-------------------------------------------------------------------*/
options nodate nonumber linesize=90 pagesize=60;
/* modified output */

proc sql outobs=12;
   title 'Areas of World Continents';
   select Name format=$25.,
          Continent,
          sum(Area) format=comma12. as TotalArea
      from sql.countries
      where Continent is not missing
      group by Continent
```

```
      order by Continent, Name;
/*---------------------------------------------------------------------
   Output 2.47 Using a Simple HAVING Clause

---------------------------------------------------------------------*/
proc sql;
   title 'Numbers of Islands, Oceans, and Seas';
   select Type, count(*) as Number
      from sql.features
      group by Type
      having Type in ('Island', 'Ocean', 'Sea')
      order by Type;

/*---------------------------------------------------------------------
   Output 2.48 Using HAVING with the COUNT Function

---------------------------------------------------------------------*/
proc sql;
   title 'Total Populations of Continents with More than 15 Countries';
   select Continent,
          sum(Population) as TotalPopulation format=comma16.,
          count(*) as Count
      from sql.countries
      group by Continent
      having count(*) gt 15
      order by Continent;

/*---------------------------------------------------------------------
   Log 2.3 Validating a Query (Partial Log)

---------------------------------------------------------------------*/
proc sql;
   validate
      select Name, Statehood
         from sql.unitedstates
         where Statehood lt '01Jan1800'd;

/*---------------------------------------------------------------------
   Log 2.4 Validating an Invalid Query (Partial Log)

---------------------------------------------------------------------*/
proc sql;
   validate
      select Name, Statehood
      from sql.unitedstates
      where lt '01Jan1800'd;
```

複数のテーブルからのデータの取得

一部の出力テーブルでは、オブザベーション数を制限しています。テーブル全体を表示するには、SQL プロシジャの OUTOBS=オプションを削除します。

```
/*---------------------------------------------------------------------
   Output 3.1 Table One, Table Two
   Output 3.2 Cartesian Product of Table One and Table Two
```

```
----------------------------------------------------------------------*/
data one;
   input X Y $;
datalines;
1 2
2 3
;


data two;
   input X Z $;
datalines;
2 5
3 6
4 9
;
run;

proc sql;
  title 'Table One';
  select * from one;

  title 'Table Two';
  select * from two;

/*----------------------------------------------------------------------
   Output 3.3 Table One and Table Two Joined

----------------------------------------------------------------------*/
data one;
 input X Y $;
 datalines;
1 2
2 3
;


data two;
 input X Z $;
 datalines;
2 5
3 6
4 9
;
run;

proc sql;
   title 'Table One and Table Two';
   select * from one, two
      where one.x=two.x;

/*----------------------------------------------------------------------
   Output 3.4 Abbreviating Column Names by Using Table Aliases

----------------------------------------------------------------------*/
proc sql outobs=6;
```

```
   title 'Oil Production/Reserves of Countries';
   select * from sql.oilprod as p, sql.oilrsrvs as r
      where p.country = r.country;

/*---------------------------------------------------------------------
   Output 3.5  Orderging the Output of Joined Tables

   Note: Both of the following PROC SQL steps
         produce Output 3.6.
---------------------------------------------------------------------*/
proc sql outobs=6;
   title 'Oil Production/Reserves of Countries';
   select p.country, barrelsperday 'Production', barrels 'Reserves'
      from sql.oilprod p, sql.oilrsrvs r
      where p.country = r.country
      order by barrelsperday desc;

proc sql outobs=6;
   title 'Oil Production/Reserves of Countries';
   select p.country, barrelsperday 'Production', barrels 'Reserves'
      from sql.oilprod p inner join sql.oilrsrvs r
           on p.country = r.country
      order by barrelsperday desc;

/*---------------------------------------------------------------------
   Output 3.6 Using Comparison Operators to Join Tables
   Log 3.2 Comparison Query Log Message

---------------------------------------------------------------------*/
proc sql;
   title 'US Cities South of Cairo, Egypt';
   select us.City, us.State, us.Latitude, world.city, world.latitude
      from sql.worldcitycoords world, sql.uscitycoords us
      where world.city = 'Cairo' and
            us.latitude lt world.latitude;

/*---------------------------------------------------------------------
   Output 3.7 Joining Tables That Contain Null Values

---------------------------------------------------------------------*/
data one;
 input a $ b;
 datalines;
 a 1
 b 2
 c .
 d 4
 ;

data two;
 input a $ b;
 cards;
 a 1
 b 2
 c .
 d 4
```

```
 e .
 f .
 ;
  run;

proc sql;
   title 'One and Two Joined';
   select one.a 'One', one.b, two.a 'Two', two.b
      from one, two
      where one.b=two.b;

/*---------------------------------------------------------------------
   Output 3.8  Results of Adding IS NOT MISSING to Joining Tables That
               Contain Null Values

   Note: This example uses the same data sets as in Output 3.7.
---------------------------------------------------------------------*/
proc sql;
   select one.a 'One', one.b, two.a 'Two', two.b
      from one, two
      where one.b=two.b and
            one.b is not missing;

/*---------------------------------------------------------------------
   Output 3.9 Selecting Capital City Coordinates (incorrect output)

---------------------------------------------------------------------*/
proc sql;
   title 'Coordinates of Capital Cities';
   select Capital format=$12., Name format=$12.,
          City format=$12., Country format=$12.,
          Latitude, Longitude
      from sql.countries, sql.worldcitycoords
      where Capital like 'L%' and
                  Capital = City;

/*---------------------------------------------------------------------
   Output 3.10 Selecting Capital City Coordinates (correct output)

---------------------------------------------------------------------*/
proc sql;
   title 'Coordinates of Capital Cities';
   select Capital format=$12., Name format=$12.,
          City format=$12., Country format=$12.,
          latitude, longitude
      from sql.countries, sql.worldcitycoords
      where Capital like 'L%' and
            Capital = City and
            Name = Country;

/*---------------------------------------------------------------------
   Output 3.11 Selecting Data from More Than Two Tables

---------------------------------------------------------------------*/
title 'Coordinates of State Capitals';
proc sql outobs=10;
```

```
   select us.Capital format=$15., us.Name 'State' format=$15.,
          pc.Code, c.Latitude, c.Longitude
      from sql.unitedstates us, sql.postalcodes pc,
           sql.uscitycoords c
      where us.Capital = c.City and
            us.Name = pc.Name and
            pc.Code = c.State;

/*---------------------------------------------------------------------
   Output 3.12 Joining a Table to Itself (Self-Join)

---------------------------------------------------------------------*/
 proc sql;
    title "Cities' High Temps = Cities' Low Temps";
    select High.City format $12., High.Country format $12.,
           High.AvgHigh, ' | ',
           Low.City format $12., Low.Country format $12.,
           Low.AvgLow
       from sql.worldtemps High, sql.worldtemps Low
       where High.AvgHigh = Low.AvgLow and
             High.city ne Low.city and
             High.country ne Low.country;

/*---------------------------------------------------------------------
   Output 3.13 Left Join of Countries and WordCityCoords

---------------------------------------------------------------------*/
proc sql outobs=10;
   title 'Coordinates of Capital Cities';
   select Capital format=$20., Name 'Country' format=$20.,
          Latitude, Longitude
      from sql.countries a left join sql.worldcitycoords b
           on a.Capital = b.City and
              a.Name = b.Country
      order by Capital;

/*---------------------------------------------------------------------
   Output 3.14 Right Join of Countries and WorldCityCoords

---------------------------------------------------------------------*/
proc sql outobs=10;
   title 'Populations of Capitals Only';
   select City format=$20., Country 'Country' format=$20.,
          Population
      from sql.countries right join sql.worldcitycoords
           on Capital = City and
              Name = Country
      order by City;

/*---------------------------------------------------------------------
   Output 3.15 Full Outer Join of Countries and WorldCityCoords

---------------------------------------------------------------------*/
proc sql outobs=10;
   title 'Populations and/or Coordinates of World Cities';
   select City '#City#(WorldCityCoords)' format=$20.,
```

```
          Capital '#Capital#(Countries)' format=$20.,
          Population, Latitude, Longitude
      from sql.countries full join sql.worldcitycoords
           on Capital = City and
              Name = Country;

/*---------------------------------------------------------------------
   Output 3.16 Tables One and Two

---------------------------------------------------------------------*/
data one;
 input X Y $;
 datalines;
1 2
2 3
;


data two;
 input W Z $;
 datalines;
2 5
3 6
4 9
;
run;
proc sql;
  title 'Table One';
  select * from one;

  title 'Table Two';
  select * from two;

title;
quit;

/*---------------------------------------------------------------------
   Output 3.17 Cross Join

   Note: This example uses the same data sets as in Output 3.16.
---------------------------------------------------------------------*/
proc sql;
   title 'Table One and Table Two';
   select *
      from one cross join two;

/*---------------------------------------------------------------------
   Output 3.18 Union Join

   Note: This example uses the same data sets as in Output 3.16.
---------------------------------------------------------------------*/
proc sql;
   select *
      from one union join two;

/*---------------------------------------------------------------------
```

```
   Output 3.19 Natural Inner Join of OilProd and OilRsrvs

-------------------------------------------------------------------*/
libname sql 'SAS-library';

proc sql outobs=6;
   title 'Oil Production/Reserves of Countries';
   select country, barrelsperday 'Production', barrels 'Reserve'
      from sql.oilprod natural join sql.oilrsrvs
      order by barrelsperday desc;

/*-------------------------------------------------------------------
   Output 3.20 Using COALESCE in Full Outer Join of Countries and
               WorldCityCoords

-------------------------------------------------------------------*/
proc sql outobs=10;
   title 'Populations and/or Coordinates of World Cities';
   select coalesce(Capital, City,Name)format=$20. 'City',
          coalesce(Name, Country) format=$20. 'Country',
          Population, Latitude, Longitude
      from sql.countries full join sql.worldcitycoords
           on Capital = City and
           Name = Country;

/*-------------------------------------------------------------------
   Output 3.21 Merged Tables When All the Values Match

   Note: The two DATA steps produce the datasets used by both
         the merge and the join. The merge and the join produce
         identical output.
-------------------------------------------------------------------*/
data fltsuper;
input Flight Supervisor $;
datalines;
145   Kang
150   Miller
155   Evanko
;

data fltdest;
input Flight Destination $;
datalines;
145   Brussels
150   Paris
155   Honolulu
;
run;

data merged;
   merge FltSuper FltDest;
   by Flight;
run;

proc print data=merged noobs;
   title 'Table MERGED';
```

```
run;

proc sql;
   title 'Table Merged';
   select s.flight, Supervisor, Destination
      from fltsuper s, fltdest d
      where s.Flight=d.Flight;

/*---------------------------------------------------------------------
   Output 3.22 Merged Tables When Some of the Values Match

   Note: The two DATA steps produce the datasets used by both
         the merge and the join. The merge and the join produce
         identical output.
---------------------------------------------------------------------*/
data fltsuper;
input Flight Supervisor $;
datalines;
145   Kang
150   Miller
155   Evanko
157   Lei
;
data fltdest;
input Flight Destination $;
datalines;
145   Brussels
150   Paris
165   Seattle
;
run;

data merged;
   merge fltsuper fltdest;
   by flight;
run;
proc print data=merged noobs;
   title 'Table Merged';
run;

proc sql;
   select coalesce(s.Flight,d.Flight) as Flight, Supervisor, Destination
      from fltsuper s full join fltdest d
           on s.Flight=d.Flight;

/*---------------------------------------------------------------------
   Output 3.23 Match-Merge of the FltSuper and FltDest Tables

---------------------------------------------------------------------*/
data fltsuper;
input Flight Supervisor $;
datalines;
145   Kang
145   Rameriz
150   Miller
150   Picard
```

```
155   Evanko
157   Lei
;

data fltdest;
input Flight Destination $;
datalines;
145   Brussels
145   Edmonton
150   Paris
150   Madrid
165   Seattle
;
run;

data merged;
   merge fltsuper fltdest;
   by flight;
run;

proc print data=merged noobs;
   title 'Table Merged';
run;

/*---------------------------------------------------------------------
   Output 3.24 PROC SQL Join of the FltSuper and FltDest Tables

---------------------------------------------------------------------*/
data fltsuper;
input Flight Supervisor $;
datalines;
145   Kang
145   Rameriz
150   Miller
150   Picard
155   Evanko
157   Lei
;

data fltdest;
input Flight Destination $;
datalines;
145   Brussels
145   Edmonton
150   Paris
150   Madrid
165   Seattle
;
run;

proc sql;
   title 'Table Joined';
   select *
      from fltsuper s, fltdest d
      where s.Flight=d.Flight;
```

```
/*---------------------------------------------------------------------
   Output 3.25 Single-Value Subquery

---------------------------------------------------------------------*/
proc sql;
   title 'U.S. States with Population Greater than Belgium';
   select Name 'State' , population format=comma10.
      from sql.unitedstates
      where population gt
                  (select population from sql.countries
                      where name = "Belgium");

/*---------------------------------------------------------------------
   Output 3.26 Multiple-Value Subquery Using IN

---------------------------------------------------------------------*/
proc sql outobs=5;
   title 'Populations of Major Oil Producing Countries';
   select name 'Country', Population format=comma15.
      from sql.countries
      where Name in
            (select Country from sql.oilprod);

/*---------------------------------------------------------------------
   Output 3.27 Multiple-Value Subquery Using NOT IN

---------------------------------------------------------------------*/
proc sql outobs=5;
   title 'Populations of NonMajor Oil Producing Countries';
   select name 'Country', Population format=comma15.
      from sql.countries
      where Name not in
            (select Country from sql.oilprod);

/*---------------------------------------------------------------------
   Output 3.28 Correlated Subquery

---------------------------------------------------------------------*/
proc sql;
   title 'Oil Reserves of Countries in Africa';
   select * from sql.oilrsrvs o
      where 'Africa' =
           (select Continent from sql.countries c
               where c.Name = o.Country);

/*---------------------------------------------------------------------
   Output 3.29 Testing for the Existence of a Group of Values

---------------------------------------------------------------------*/
proc sql;
   title 'Oil Reserves of Countries in Africa';
   select * from sql.oilrsrvs o
      where exists
           (select Continent from sql.countries c
               where o.Country = c.Name and
                     Continent = 'Africa');
```

```
/*-------------------------------------------------------------------
   Output 3.30 Multiple Levels of Subquery Nesting

-------------------------------------------------------------------*/
proc sql;
  title 'Coordinates of African Cities with Major Oil Reserves';
  select * from sql.worldcitycoords
     where country in
       (select Country from sql.oilrsrvs o
           where o.Country in
             (select Name from sql.countries c
                 where c.Continent = 'Africa'));

/*-------------------------------------------------------------------
   Output 3.31 Combining a Join with a Subquery

-------------------------------------------------------------------*/
proc sql outobs=10;
   title 'Neighboring Cities';
   select a.City format=$10., a.State,
          a.Latitude 'Lat', a.Longitude 'Long',
          b.City format=$10., b.State,
          b.Latitude 'Lat', b.Longitude 'Long',
          sqrt(((b.latitude-a.latitude)**2) +
               ((b.longitude-a.longitude)**2)) as dist format=6.1
      from sql.uscitycoords a, sql.uscitycoords b
      where a.city ne b.city and
            calculated dist =
            (select min(sqrt(((d.latitude-c.latitude)**2) +
                             ((d.longitude-c.longitude)**2)))
                from sql.uscitycoords c, sql.uscitycoords d
                where c.city = a.city and
                      c.state = a.state and
                      d.city ne c.city)
      order by a.city;

/*-------------------------------------------------------------------
   Output 3.32 Tables Used in Set Operation Examples

-------------------------------------------------------------------*/
proc sql;
  create table sql.A
    (x int, y varchar(12));

proc sql;
  create table sql.B
    (x int, z varchar(12));

proc sql;
   insert into sql.A
      values (1,'one')
      values (2,'two')
      values (2,'two')
      values (3,'three')
;
```

```
proc sql;
   insert into sql.B
      values (1,'one')
      values (2,'two')
      values (4,'four');

quit;


proc sql;
  title 'Table A';
  select * from sql.a;
  title 'Table B';
  select * from sql.b;

/*---------------------------------------------------------------------
   Output 3.33 Producing Unique Rows from Both Queries (UNION)

   Note: This example uses the data sets that were created in Ouptut 3.32.
---------------------------------------------------------------------*/
proc sql;
   title 'A UNION B';
   select * from sql.a
   union
   select * from sql.b;

/*---------------------------------------------------------------------
   Output 3.34 Producing Rows from Both Queries (UNION ALL)

   Note: This example uses the data sets that were created in Ouptut 3.32.
---------------------------------------------------------------------*/
proc sql;
   title 'A UNION ALL B';
   select * from sql.a
   union all
   select * from sql.b;

/*---------------------------------------------------------------------
   Output 3.35 Producing Rows That Are in Only the First Query Result (EXCEPT)

   Note: This example uses the data sets that were created in Ouptut 3.32.
---------------------------------------------------------------------*/
proc sql;
   title 'A EXCEPT B';
   select * from sql.a
   except
   select * from sql.b;

/*---------------------------------------------------------------------
   Output 3.36 Producing Rows That Are in Only the First Query Result
               (EXCEPT ALL)

   Note: This example uses the data sets that were created in Ouptut 3.32.
---------------------------------------------------------------------*/
proc sql;
```

```
   title 'A EXCEPT ALL B';
   select * from sql.a
   except all
   select * from sql.b;

/*---------------------------------------------------------------
   Output 3.37 Producing Rows That Belong to Both Query Results (INTERSECT)

   Note: This example uses the data sets that were created in Ouptut 3.32.
---------------------------------------------------------------*/
proc sql;
   title 'A INTERSECT B';
   select * from sql.a
   intersect
   select * from sql.b;

/*---------------------------------------------------------------
   Output 3.38 Concatenating the Query Results (OUTER UNION)

   Note: This example uses the data sets that were created in Ouptut 3.32.
---------------------------------------------------------------*/
proc sql;
   title 'A OUTER UNION B';
   select * from sql.a
   outer union
   select * from sql.b;

/*---------------------------------------------------------------
   Output 3.39 Concatenating the Query Results (OUTER UNION CORR)

   Note: This example uses the data sets that were created in Ouptut 3.32.
---------------------------------------------------------------*/
proc sql;
   title 'A OUTER UNION CORR B';
   select * from sql.a
   outer union corr
   select * from sql.b;

/*---------------------------------------------------------------
   Output 3.40 Producing Rows from the First Query or the Second Query

   Note: This example uses the data sets that were created in Ouptut 3.32.
---------------------------------------------------------------*/
proc sql;
   title 'A EXCLUSIVE UNION B';
   (select * from sql.a
   except
   select * from sql.b)
   union
   (select * from sql.b
   except
   select * from sql.a);
```

テーブルとビューの作成および更新

一部の出力テーブルでは、オブザベーション数を制限しています。テーブル全体を表示するには、SQL プロシジャの OUTOBS=オプションを削除します。

```
/*---------------------------------------------------------------------
   Log 4.1 Table Created from Column Definitions

---------------------------------------------------------------------*/
proc sql;
   create table sql.newstates
          (state char(2),           /* 2&ndash;character column for         */
                                    /* state abbreviation                   */

           date num                 /* column for date of entry into the US */
                informat=date9.     /* with an informat                     */
                format=date9.,      /* and format of DATE9.                 */

           population num);         /* column for population                */

proc sql;
   describe table sql.newstates;

/*---------------------------------------------------------------------
   Output 4.1 Table Created from a Query Result

---------------------------------------------------------------------*/
proc sql outobs=10;
   title 'Densities of Countries';
   create table sql.densities as
      select Name 'Country' format $15.,
             Population format=comma10.0,
             Area as SquareMiles,
             Population/Area format=6.2 as Density
         from sql.countries;

   select * from sql.densities;

/*---------------------------------------------------------------------
   Log 4.2 SAS Log for DESCRIBE TABLE Statement for Densities

---------------------------------------------------------------------*/
proc sql;
   describe table sql.densities;

/*---------------------------------------------------------------------
   Log 4.3 SAS Log for DESCRIBE TABLE Statement for NewCountries

---------------------------------------------------------------------*/
/* Create the newcountries table.   */

proc sql;
   create table sql.newcountries
      like sql.countries;
```

```
   describe table sql.newcountries;

/*---------------------------------------------------------------------
   Output 4.2 Rows Inserted with the SET Clause

---------------------------------------------------------------------*/
proc sql;
   create table sql.newcountries
      like sql.countries;

/* Insert all of the rows from countries into newcountries based  */
/* on a population of 130000000.                                  */

proc sql;
   insert into sql.newcountries
   select * from sql.countries
      where population ge 130000000;

/* Insert 2 new rows in the newcountries table.     */
/* Print the table.                                 */

proc sql;
   insert into sql.newcountries
      title "World's Largest Countries";
      set name='Bangladesh',
          capital='Dhaka',
          population=126391060
      set name='Japan',
          capital='Tokyo',
          population=126352003;

   title "World's Largest Countries";
   select name format=$20.,
          capital format=$15.,
          population format=comma15.0
      from sql.newcountries;

/*---------------------------------------------------------------------
   Output 4.3 Rows Inserted with the VALUES Clause

   Note: This example use the NewCountries table that was created in
         Log 4.3.

---------------------------------------------------------------------*/
proc sql;
   insert into sql.newcountries
      values ('Pakistan', 'Islamabad', 123060000, ., ' ', .)
      values ('Nigeria', 'Lagos', 99062000, ., ' ', .);
   title "World's Largest Countries";
   select name format=$20.,
          capital format=$15.,
          population format=comma15.0
      from sql.newcountries;

/*---------------------------------------------------------------------
   Output 4.4 Rows Inserted with a Query
```

```
-------------------------------------------------------------------*/
proc sql;
   create table sql.newcountries
      like sql.countries;

proc sql;
   title "World's Largest Countries";
   insert into sql.newcountries
   select * from sql.countries
      where population ge 130000000;

   select name format=$20.,
          capital format=$15.,
          population format=comma15.0
      from sql.newcountries;

/*-----------------------------------------------------------------
   Output 4.5 A Lessor Number of Columns in Rows Inserted with a Query

-------------------------------------------------------------------*/
proc sql;
  create table sql.newcountries
  like sql.countries;

proc sql;
   title "World's Largest Countries";
   insert into sql.newcountries (Name,Population)
   select Name,Population from sql.countries
      where population ge 130000000;

   select name format=$20., population format=comma15.0
      from sql.newcountries;

/*-----------------------------------------------------------------
   Output 4.6 Updating a Column for All Rows

-------------------------------------------------------------------*/
proc sql;
   create table sql.newcountries like sql.countries;
   insert into sql.newcountries
   select * from sql.countries
      where population ge 130000000;
quit;

proc sql;
   update sql.newcountries
      set population=population*1.05;
   title "Updated Population Values";
   select name format=$20.,
          capital format=$15.,
          population format=comma15.0
      from sql.newcountries;

/*-----------------------------------------------------------------
   Output 4.7 Selectively Updating a Column
```

```
-------------------------------------------------------------------*/
proc sql;
   create table sql.newcountries like sql.countries;
   insert into sql.newcountries
   select * from sql.countries
      where population ge 130000000;
quit;

proc sql;
   update sql.newcountries
      set population=population*1.05
         where name like 'B%';

   update sql.newcountries
      set population=population*1.07
         where name in ('China', 'Russia');

   title "Selectively Updated Population Values";
   select name format=$20.,
          capital format=$15.,
          population format=comma15.0
      from sql.newcountries;

You can accomplish the same result with a CASE expression:
update sql.newcountries
   set population=population*
      case when name like 'B%' then 1.05
           when name in ('China', 'Russia') then 1.07
           else 1
      end;

/*-------------------------------------------------------------------
   Log 4.4 SAS Log for the DELETE Statement

-------------------------------------------------------------------*/0
/* Create and populate NewCountries */
proc sql;
   create table sql.newcountries like sql.countries;
   insert into sql.newcountries
   select * from sql.countries
      where population ge 130000000;
quit;

proc sql;
   delete
      from sql.newcountries
      where name like 'R%';

/*-------------------------------------------------------------------
   Output 4.8 Adding a New Column

-------------------------------------------------------------------*/

/* Create and populate NewCountries */
proc sql;
```

```
   create table sql.newcountries like sql.countries;
   insert into sql.newcountries
   select * from sql.countries
      where population ge 130000000;
quit;

proc sql;
   alter table sql.newcountries
      add density num label='Population Density' format=6.2;

   title "Population Density Table";
   select name format=$20.,
          capital format=$15.,
          population format=comma15.0,
          density
      from sql.newcountries;

/*---------------------------------------------------------------
   Output 4.9 Fillin in the New Column's Values

   Note: After the NewCountries table is created, the two SQL statements
         produce the same output.
---------------------------------------------------------------*/

/* Create and populate Newcountries */
proc sql;
   create table sql.newcountries like sql.countries;
   insert into sql.newcountries
   select * from sql.countries
      where population ge 130000000;
   alter table sql.newcountries
      add density num label='Population Density' format=6.2;
quit;

proc sql;
   update sql.newcountries
      set density=population/area;

   title "Population Density Table";
   select name format=$20.,
          capital format=$15.,
          population format=comma15.0,
          density
      from sql.newcountries;

proc sql;
   create table sql.newcountries as
   select *, population/area as density
             label='Population Density'
             format=6.2
      from sql.newcountries;

/*---------------------------------------------------------------
   Output 4.10 Modifying a Column Format

---------------------------------------------------------------*/
```

```
/* Create and populate NewCountries */
proc sql;
   create table sql.newcountries like sql.countries;
   insert into sql.newcountries
   select * from sql.countries
      where population ge 130000000;
quit;

proc sql;
   title "World's Largest Countries";
   alter table sql.newcountries
      modify population format=comma15.;
   select name, population from sql.newcountries;

/*---------------------------------------------------------------------
   Output 4.11  Changing a Column's Width

---------------------------------------------------------------------*/
proc sql;
   title "World's Largest Countries";
   alter table sql.newcountries
      modify name char(60) format=$60.;
   update sql.newcountries
      set name='The United Nations member country is '||name;

   select name from sql.newcountries;

/*---------------------------------------------------------------------
   Output 4.12 Using a PROC SQL Table in PROC MEANS

---------------------------------------------------------------------*/
proc means data=sql.countries mean maxdec=2;
   title "Mean Area for All Countries";
   var area;
run;

/*---------------------------------------------------------------------
   Output 4.13 SAS Log Showing Integrity Constraints

---------------------------------------------------------------------*/
proc sql;
   create table sql.mystates
      (state      char(15),
       population num,
       continent  char(15),

          /* contraint specifications */
       constraint prim_key    primary key(state),
       constraint population  check(population gt 0),
       constraint continent   check(continent in ('North America', 'Oceania')));

   create table sql.uspostal
      (name      char(15),
       code      char(2) not null,          /* constraint specified as    */
                                            /* a column attribute         */
```

```
        constraint for_key foreign key(name) /* links NAME to the          */
                   references sql.mystates   /* primary key in MYSTATES    */

                      on delete restrict     /* forbids deletions to STATE */
                                             /* unless there is no         */
                                             /* matching NAME value        */

                      on update set null);   /* allows updates to STATE,   */
                                             /* changes matching NAME      */
                                             /* values to missing          */

proc sql;
   describe table sql.mystates;
   describe table constraints sql.uspostal;

/*-----------------------------------------------------------------
   Output 4.14 An SQL Procedure View

-----------------------------------------------------------------*/
proc sql;
   title 'Current Population Information for Continents';
   create view sql.newcontinents as
   select continent,
          sum(population) as totpop  format=comma15. label='Total Population',
          sum(area) as totarea format=comma15. label='Total Area'
      from sql.countries
      group by continent;

   select * from sql.newcontinents;

/*-----------------------------------------------------------------
   Log 4.5 SAS Log from DESCRIBE VIEW Statement

-----------------------------------------------------------------*/
proc sql;
   describe view sql.newcontinents;

/*-----------------------------------------------------------------
   Output 4.15 Using an In-Line View

-----------------------------------------------------------------*/
proc sql;
   title 'Countries With Population GT Caribbean Countries';
   select w.Name, w.Population format=comma15., c.TotCarib
      from (select sum(population) as TotCarib format=comma15.
               from sql.countries
                  where continent = 'Central America and Caribbean') as c,
                    sql.countries as w
            where w.population gt c.TotCarib;
```

SQL プロシジャを使用したプログラミング

一部の出力テーブルでは、オブザベーション数を制限しています。テーブル全体を表示するには、SQL プロシジャの OUTOBS=オプションを削除します。

```
/*-------------------------------------------------------------------
   Log 5.1 Expanded SELECT * Statement

-------------------------------------------------------------------*/
proc sql feedback;
   select * from sql.countries;

/*-------------------------------------------------------------------
   Logt 5.2 Comparing Run Times of Two Queries

-------------------------------------------------------------------*/
proc sql stimer ;
   select us.name, us.population
      from sql.unitedstates as us, sql.countries as w
      where us.population gt w.population and
            w.name = 'Belgium';

   select Name, population
      from sql.unitedstates
      where population gt
                  (select population from sql.countries
                      where name = 'Belgium');
/*-------------------------------------------------------------------
   Output 5.1 Resetting PROC SQL Options with the RESET Statement

-------------------------------------------------------------------*/
proc sql noprint;
   title 'Countries with Population Under 20,000';
   select Name, Population from sql.countries;
reset print number;
   select Name, Population from sql.countries
      where population lt 20000;

/*-------------------------------------------------------------------
   Log 5.3 Definition of DICTIONARY.Tables

-------------------------------------------------------------------*/
proc sql;
   describe table dictionary.tables;

/*-------------------------------------------------------------------
   Log 5.4 Description of Sashelp.Vstabvw

-------------------------------------------------------------------*/
proc sql;
   describe view sashelp.vstabvw;

/*-------------------------------------------------------------------
   Output 5.2 Tables and Views Used in This Document

-------------------------------------------------------------------*/
proc sql;
   title 'All Tables and Views in the SQL Library';
   select libname, memname, memtype, nobs
      from dictionary.tables
      where libname='SQL';
```

```
/*---------------------------------------------------------------------
   Output 5.3 Using DICTIONARY.Columns to Locate Specific Columns

---------------------------------------------------------------------*/
proc sql;
   title 'All Tables that Contain the Country Column';
   select libname, memname, name
      from dictionary.columns
      where name='Country' and
            libname='SQL';

/*---------------------------------------------------------------------
   Log 5.5 Creating Macro Variables from the First Row of a Query Result

---------------------------------------------------------------------*/
proc sql noprint;
   select country, barrels
      into :country1, :barrels1
      from sql.oilrsrvs;

%put &country1 &barrels;

/*---------------------------------------------------------------------
   Output 5.4 Including a Macro Variable Reference in the Title

---------------------------------------------------------------------*/
proc sql outobs=12;
   reset noprint;
   select max(AvgHigh)
      into :maxtemp
      from sql.worldtemps
      where country = 'Canada';
reset print;
   title "The Highest Temperature in Canada: &maxtemp";
   select city, AvgHigh format 4.1
      from sql.worldtemps
      where country = 'Canada';

/*---------------------------------------------------------------------
   Log 5.6 Creating Multiple Macro Variables

---------------------------------------------------------------------*/
proc sql noprint;
   select name, Population
      into :country1 - :country4, :pop1 - :pop3
           from sql.countries;

%put &country1 &pop1;
%put &country2 &pop2;
%put &country3 &pop3;
%put &country4;

/*---------------------------------------------------------------------
   Log 5.7 Concetenating Values in Macro Variables
```

```
-------------------------------------------------------------------*/
proc sql noprint inobs=5;
   select Name
      into :countries separated by ', '
      from sql.countries;

%put &countries;

/*-----------------------------------------------------------------
   Log 5.8 Concatenating Values in Macro Variables

-------------------------------------------------------------------*/
proc sql noprint inobs=5;
   select Name
      into :countries separated by ',' NOTRIM
      from sql.countries;

%put &countries;

/*-----------------------------------------------------------------
   Log 5.9 Defining Macros to Create Tables
   Output 5.5 Result Table and Message Created with SAS Macro Language
              Interface

   Note: The %ADDREF() macros at the end of this code should not
         be included in the PROC SQL step. See the text.
-------------------------------------------------------------------*/
proc sql;
create table sql.referee
   (Name      char(15),
    Subject   char(15));

   /* define the macro */
%macro addref(name,subject);
%local count;

   /* are there three referees in the table? */
reset noprint;
   select count(*)
      into :count
      from sql.referee
      where subject="&subject";

%if &count ge 3 %then %do;
   reset print;
   title "ERROR: &name not inserted for subject - &subject..";
   title2 "        There are 3 referees already.";
   select * from sql.referee where subject="&subject";
   reset noprint;
   %end;

%else %do;
   insert into sql.referee(name,subject) values("&name","&subject");
   %put NOTE: &name has been added for subject - &subject..;
   %end;
```

```
%mend;

%addref(Fay,sailing);
%addref(Einstein,relativity);
%addref(Smythe,sailing);
%addref(Naish,sailing);
/*-------------------------------------------------------------------
   Log 5.10 Using the PROC SQL Automatic Macro Variables

-------------------------------------------------------------------*/
proc sql noprint;
   select * from sql.countries;

%put SQLOBS=* &sqlobs* SQLOOPS=*&sqloops* SQLRC=*&sqlrc*;

/*-------------------------------------------------------------------
   Output 5.6 USCityCoords Table Showing Repeating State Values

-------------------------------------------------------------------*/
proc sql outobs=10;
   title 'US Cities';
   select State, City, Latitude, Longitude
      from sql.uscitycoords
      order by state;

/*-------------------------------------------------------------------
   Output 5.7 PROC REPORT Output Showing the First Occurence Only of
              Each State Value

-------------------------------------------------------------------*/
proc sql noprint;
   create table sql.cityreport as
   select *
      from sql.uscitycoords
      order by state;

proc report data=sql.cityreport
            headline
            headskip;
   title 'Coordinates of U.S. Cities in Pacific Rim States';
   column state city ('Coordinates' latitude longitude);
   define state / order format=$2. width=5 'State';
   define city / order format=$15. width=15 'City';
   define latitude / display format=4. width=8 'Latitude';
   define longitude / display format=4. width=9 'Longitude';
   where state='AK' or
         state='HI' or
         state='WA' or
         state='OR' or
         state='CA';
run;

/*-------------------------------------------------------------------
   Output 5.8 Output from Querying a DBMS Table

   Note: This example assumes a Payroll table exists in an
```

```
         ORACLE database.
----------------------------------------------------------------------*/
libname mydblib oracle user=user-id
        password=password
        path=path-name schema=schema-name;

proc sql;
   select jobcode label='Jobcode',
          sum(salary) as total
          label='Total for Group'
          format=dollar11.2
      from mydblib.payroll
      group by jobcode;
quit;
/*----------------------------------------------------------------------
   Output 5.98 PRINT Procedure Output

   Note: This example assumes a Schedule table exists in an
         ORACLE database.
----------------------------------------------------------------------*/
libname mydblib oracle user=user-id password=password
proc sql;
   create view LON as
   select flight, dates, idnum
      from mydblib.schedule
      where dest='LON';
quit;

proc print data=work.LON noobs;
run;

/*----------------------------------------------------------------------
   Output 5.10 Pass-Through Facility Example Output

   Note: This example assumes a Staff table exists in an
         ORACLE database.
----------------------------------------------------------------------*/
proc sql outobs=15;
   connect to oracle as ora2 (user=user-id password=password);
   select * from connection to ora2 (select lname, fname, state from staff);
   disconnect from ora2;
quit;

/*----------------------------------------------------------------------
   Output 5.11 The Coordinates of U.S. Cities

   Note: See the text for information concerning filenames.
----------------------------------------------------------------------*/
ods html body='odsout.htm';
   proc sql outobs=12;
      title 'U.S. Cities with Their States and Coordinates';
      select *
         from sql.uscitycoords;
ods html close;
```

PROC SQL を使用した問題の解決

一部の出力テーブルでは、オブザベーション数を制限しています。テーブル全体を表示するには、SQL プロシジャの OUTOBS=オプションを削除します。

```
/*-------------------------------------------------------------------
   Output 6.1 Sample Input Table for Weighted Average

-------------------------------------------------------------------*/
data Sample;
   do i=1 to 10;
      Value=2983*ranuni(135);
      Weight=33*rannor(579);
      if mod(i,2)=0 then Gender='M';
         else Gender='F';
      output;
   end;
   drop i;


proc print data=Sample;
  title 'Sample Data for Weighted Average';
run;

/*-------------------------------------------------------------------
   Output 6.2 PROC SQL Output for Weighted Averages

   Note: This example uses the data set Sample that was created in Output 6.1.
-------------------------------------------------------------------*/
proc sql;
   title 'Weighted Averages from Sample Data';
   select Gender, sum(Value*Weight)/sum(Weight) as WeightedAverage
      from (select Gender, Value,
                   case
                      when Weight gt 0 then Weight
                      else 0
                   end as Weight
              from Sample)
      group by Gender;

/*-------------------------------------------------------------------
   Output 6.3  Sample Input Tables for Table Comparison

-------------------------------------------------------------------*/
data oldstaff;
   input id $ Last : $10. First $ Middle $ Phone $ Location $;
   datalines;
5463 Olsen Mary K. 661-0012 R2342
6574 Hogan Terence H. 661-3243 R4456
7896 Bridges Georgina W. 661-8897 S2988
4352 Anson Sanford . 661-4432 S3412
5674 Leach Archie G. 661-4328 S3533
7902 Wilson Fran R. 661-8332 R4454
0001 Singleton Adam O. 661-0980 R4457
9786 Thompson Jack . 661-6781 R2343
```

```
;
data newstaff;
   input id $ Last : $10. First $ Middle $ Phone $ Location $;
   datalines;
5463 Olsen Mary K. 661-0012 R2342
6574 Hogan Terence H. 661-3243 R4456
7896 Bridges Georgina W. 661-2231 S2987
4352 Anson Sanford . 661-4432 S3412
5674 Leach Archie G. 661-4328 S3533
7902 Wilson Fran R. 661-8332 R4454
0001 Singleton Adam O. 661-0980 R4457
9786 Thompson John C. 661-6781 R2343
2123 Chen Bill W. 661-8099 R4432
;

/*-------------------------------------------------------------------
   Output 6.4 Rows That Have Changed

   Note: This example uses the data sets that were created in Output 6.3.
-------------------------------------------------------------------*/
proc sql;
   title 'Updated Rows';
   select * from newstaff
   except
   select * from oldstaff;

/*-------------------------------------------------------------------
   Output 6.5 Sample Input Tables for Overlaying Missing Values

-------------------------------------------------------------------*/
data league1;
input @1 Fullname $20. @21 Bowler $4. @29 AvgScore 3.;
cards;
Alexander Delarge   4224    164
John T Chance       4425
Jack T Colton       4264
                    1412    141
Andrew Shepherd     4189    185
;

data league2;
input @1 FirstName $10. @12 LastName $15. @28 AMFNo $4. @38 AvgScore 3.;
cards;
Alex       Delarge         4224      156
Mickey     Raymond         1412
                           4264      174
Jack       Chance          4425
Patrick    O'Malley        4118      164
;

proc sql;
title 'Bowling Averages from League1';
select * from league1;
title 'Bowling Averages from League2';
select * from league2;
```

```
/*-------------------------------------------------------------------
   Output 6.6 PROC SQL Output for Overlaying Missing Values

   Note: This example uses the data sets that were created in Output 6.5.
-------------------------------------------------------------------*/
proc sql;
   title "Averages from Last Year's League When Possible";
   title2 "Supplemented when Available from Prior Year's League";
   select coalesce(lastyr.fullname,trim(prioryr.firstname)
                  ||' '||prioryr.lastname)as Name format=$26.,
          coalesce(lastyr.bowler,prioryr.amfno)as Bowler,
          coalesce(lastyr.avgscore,prioryr.avgscore)as Average format=8.
      from league1 as lastyr full join league2 as prioryr
           on lastyr.bowler=prioryr.amfno
      order by Bowler;

/*-------------------------------------------------------------------
   Output 6.7  Input Table for Computing Subtotal Percentages
               (Partial Output)
-------------------------------------------------------------------*/
data survey;
    input State $ Answer $ @@;
    datalines;
NY YES NY YES NY YES NY YES NY YES NY YES NY NO  NY NO  NY NO  NC YES
NC YES NC YES NC YES NC YES NC YES NC YES NC YES NC YES NC YES NC YES
NC YES NC YES NC YES NC YES NC YES NC YES NC YES NC YES NC YES NC NO
NC NO  NC NO  NC NO  NC NO  NC NO  NC NO  NC NO  NC NO  NC NO  NC NO
NC NO  NC NO  NC NO  NC NO  NC NO  NC NO  NC NO  NC NO  NC NO  NC NO
NC NO  NC NO  NC NO  PA YES PA YES PA YES PA YES PA YES PA YES PA YES
PA YES PA YES PA NO  PA NO  PA NO  PA NO  PA NO  PA NO  PA NO  PA NO
PA NO  PA NO  PA NO  PA NO  PA NO  PA NO  PA NO  PA NO  PA NO  PA NO
VA YES VA YES VA YES VA YES VA YES VA YES VA YES VA YES VA YES VA YES
VA YES VA YES VA YES VA YES VA YES VA YES VA YES VA YES VA YES VA NO
VA NO  VA NO  VA NO  VA NO  VA NO  VA NO  VA NO  VA NO  VA NO  VA NO
VA NO  VA NO  VA NO  VA NO  VA NO  VA NO


proc print data=Survey(obs=10);
  title 'Sample Data for Subtotal Percentages';
run;

/*-------------------------------------------------------------------
   Output 6.8 PROC SQL Output That Computes Percentages within Subtotals

   Note: This example uses the data set Survey that was created in Output 6.7.
-------------------------------------------------------------------*/
proc sql;
   title1 'Survey Responses';
   select survey.Answer, State, count(State) as Count,
          calculated Count/Subtotal as Percent format=percent8.2
   from survey,
        (select Answer, count(*) as Subtotal from survey
           group by Answer) as survey2
   where survey.Answer=survey2.Answer
   group by survey.Answer, State;
quit;
```

```
/*-------------------------------------------------------------------
   Output 6.9 Sample Input Table for Counting Duplicates

-------------------------------------------------------------------*/
data Duplicates;
   input LastName $ FirstName $ City $ State $;
   datalines;
Smith John Richmond Virginia
Johnson Mary Miami Florida
Smith John Richmond Virginia
Reed Sam Portland Oregon
Davis Karen Chicago Illinois
Davis Karen Chicago Illinois
Thompson Jennifer Houston Texas
Smith John Richmond Virginia
Johnson Mary Miami Florida
;

proc print data=Duplicates;
  title 'Sample Data for Counting Duplicates';
run;
/*-------------------------------------------------------------------
   Output 6.10 PROC SQL Output for Counting Duplicates

   Note: This example uses the data set Duplicates that was created in
         Output 6.9.
-------------------------------------------------------------------*/
proc sql;
   title 'Duplicate Rows in Duplicates Table';
   select *, count(*) as Count
      from Duplicates
      group by LastName, FirstName, City, State
      having count(*) > 1;

/*-------------------------------------------------------------------
   Output 6.11 Sample Input Table for Expanding a Hierarchy

-------------------------------------------------------------------*/

data Employees;
   input ID $ LastName $ FirstName $ Supervisor $;
   datalines;
1001 Smith John 1002
1002 Johnson Mary None
1003 Reed Sam None
1004 Davis Karen 1003
1005 Thompson Jennifer 1002
1006 Peterson George 1002
1007 Jones Sue 1003
1008 Murphy Janice 1003
1009 Garcia Joe 1002
;

proc print data=Employees;
  title 'Sample Data for Expanding a Hierarchy';
```

```
run;

/*---------------------------------------------------------------------
   Output 6.12 PROC SQL Output for Expanding a Hierarchy

   Note: This example uses the data set Employees that was created
         in Output 6.11.
---------------------------------------------------------------------*/
proc sql;
   title 'Expanded Employee and Supervisor Data';
   select A.ID label="Employee ID",
          trim(A.FirstName)||' '||A.LastName label="Employee Name",
          B.ID label="Supervisor ID",
          trim(B.FirstName)||' '||B.LastName label="Supervisor Name"
      from Employees A, Employees B
      where A.Supervisor=B.ID and A.Supervisor is not missing;

/*---------------------------------------------------------------------
   Output 6.13 Sample Input Table for Summarizing Data from Multiple Columns

---------------------------------------------------------------------*/
data Sales;
   input Salesperson $ January February March;
   datalines;
Smith 1000 650 800
Johnson 0 900 900
Reed 1200 700 850
Davis 1050 900 1000
Thompson 750 850 1000
Peterson 900 600 500
Jones 800 900 1200
Murphy 700 800 700
Garcia 400 1200 1150
;


proc print data=Sales;
  title 'Sample Data for Summarizing Data from Multiple Columns';
run;

/*---------------------------------------------------------------------
   Output 6.14 PROC SQL Output for Summarizing Data from Multiple Columns

   Note: This example uses the data set Sales that was created in Output 6.13.
---------------------------------------------------------------------*/
proc sql;
   title 'Total First Quarter Sales';
   select sum(January)  as JanTotal,
          sum(February) as FebTotal,
          sum(March)    as MarTotal,
          sum(calculated JanTotal, calculated FebTotal,
              calculated MarTotal) as GrandTotal format=dollar10.
      from Sales;

/*---------------------------------------------------------------------
   Output 6.15 Sample Input Table for Creating a Summary Report
```

```
-------------------------------------------------------------------*/
data sales;
  input Site $ Product $ Invoice $ InvoiceAmount InvoiceDate $;
  datalines;
V1009  VID010  V7679 598.5  980126
V1019  VID010  V7688 598.5  980126
V1032  VID005  V7771  1070  980309
V1043  VID014  V7780  1070  980309
V421   VID003  V7831  2000  980330
V421   VID010  V7832   750  980330
V570   VID003  V7762  2000  980302
V659   VID003  V7730  1000  980223
V783   VID003  V7815   750  980323
V985   VID003  V7733  2500  980223
V966   VID001  V5020  1167  980215
V98    VID003  V7750  2000  980223
;

proc sql;
  title 'Sample Data to Create Summary Sales Report';
  select * from sales;
quit;

/*-------------------------------------------------------------------
   Output 6.16 PROC SQL Output for a Summary Report

   Note: This example uses the data set Sales that was created in Output 6.15.
-------------------------------------------------------------------*/
proc sql;
   title 'First Quarter Sales by Product';
   select Product,
          sum(Jan) label='Jan',
          sum(Feb) label='Feb',
          sum(Mar) label='Mar'
      from (select Product,
                   case
                      when substr(InvoiceDate,3,2)='01' then
                         InvoiceAmount end as Jan,
                   case
                      when substr(InvoiceDate,3,2)='02' then
                         InvoiceAmount end as Feb,
                   case
                      when substr(InvoiceDate,3,2)='03' then
                         InvoiceAmount end as Mar
               from work.sales)
      group by Product;

/*-------------------------------------------------------------------
   Output 6.17 Sample Input Data for a Customized Sort

-------------------------------------------------------------------*/
data chores;
   input Project $ Hours Season $;
   datalines;
weeding 48 summer
```

```
pruning 12 winter
mowing 36 summer
mulching 17 fall
raking 24 fall
raking 16 spring
planting 8 spring
planting 8 fall
sweeping 3 winter
edging 16 summer
seeding 6 spring
tilling 12 spring
aerating 6 spring
feeding 7 summer
rolling 4 winter
;

proc sql;
title 'Garden Chores';
select * from chores;
quit;

/*-------------------------------------------------------------------
   Output 6.18 PROC SQL Output for a Customized Sort Sequence

   Note: This example uses the data set Chores that was created in Output 6.17.
-------------------------------------------------------------------*/
proc sql;
    title 'Garden Chores by Season in Logical Order';
    select Project, Hours, Season
       from (select Project, Hours, Season,
               case
                  when Season = 'spring' then 1
                  when Season = 'summer' then 2
                  when Season = 'fall' then 3
                  when Season = 'winter' then 4
                  else .
               end as Sorter
               from chores)
       order by Sorter;

/*-------------------------------------------------------------------
   Output 6.19 Sample Input Data to Conditionally Change a Table

-------------------------------------------------------------------*/
data incentives;
   input @1 Name $18. @20 Department $2. Payrate
            Gadgets Whatnots;
   datalines;
Lao Che            M2   8.00  10193  1105
Jack Colton        U2   6.00   9994  2710
Mickey Raymond     M1  12.00   6103  1930
Dean Proffit       M2  11.00   3000  1999
Antoinette Lily    E1  20.00   2203  4610
Sydney Wade        E2  15.00   4205  3010
Alan Traherne      U2   4.00   5020  3000
Elizabeth Bennett  E1  16.00  17003  3003
```

```
;

proc sql;
  title 'Sales Data for Incentives Program';
  select * from incentives;
quit;

/*-----------------------------------------------------------------
   Output 6.20 PROC SQL Output for Conditionally Updating a Table

   Note: This example uses the data set Incentives that was created in
         Output 6.19.
-----------------------------------------------------------------*/
proc sql;
   update incentives
   set payrate = case
                    when gadgets > 10000 then
                       payrate + 5.00
                    when gadgets > 5000 then
                       case
                          when department in ('E1', 'E2') then
                             payrate + 2.00
                          else payrate + 3.00
                       end
                    else payrate
                 end;
   update incentives
   set payrate = case
                    when whatnots > 2000 then
                       case
                          when department in ('E2', 'M2', 'U2') then
                             payrate + 1.00
                          else payrate + 0.50
                       end
                    else payrate
                 end;
   title 'Adjusted Payrates Based on Sales of Gadgets and Whatnots';
   select * from incentives;

/*-----------------------------------------------------------------
   Output 6.21 Table with Updated Population Data

-----------------------------------------------------------------*/
proc sql;
title 'Updated U.S. Population Data';
select state, population format=comma10. label='Population' from sql.newpop;

/*-----------------------------------------------------------------
   Output 6.22 Sql.UnitedStates with Updated Population Data (Partial Output)

-----------------------------------------------------------------*/
proc sql;
   title 'UnitedStates';
   update sql.unitedstates as u
      set population=(select population from sql.newpop as n
               where u.name=n.state)
```

```
           where u.name in (select state from sql.newpop);
   select Name format=$17., Capital format=$15.,
          Population, Area, Continent format=$13., Statehood format=date9.
      from sql.unitedstates;
   /* use this code to generate output so you don't
      overwrite the sql.unitedstates table */
options ls=84;
proc sql outobs=10;
   title 'UnitedStates';
   create table work.unitedstates as
   select * from sql.unitedstates;
   update work.unitedstates as u
      set population=(select population from sql.newpop as n
                where u.name=n.state)
           where u.name in (select state from sql.newpop);
    select Name format=$17., Capital format=$15.,
           Population, Area, Continent format=$13., Statehood format=date9.
       from work.unitedstates
;

/*-------------------------------------------------------------------
   Output 6.23 Features (Partial Output)

-------------------------------------------------------------------*/
libname sql 'SAS-library';

proc sql outobs=10;
title 'Features';
 select Name format=$15., Type,Location format =$15.,Area,
        Height, Depth, Length
 from sql.features;
/*-------------------------------------------------------------------
   Log 6.1 SAS Log After Creating a Separate Data Set for Each Unique
           Value of a Column

-------------------------------------------------------------------*/
proc sql noprint;
   select count(distinct type)
      into :n
      from sql.features;
   select distinct type
      into :type1 - :type%left(&n)
      from sql.features;
quit;

%macro makeds;
   %do i=1 %to &n;
      data &&type&i (drop=type);
         set sql.features;
         if type="&&type&i";
      run;
   %end;
%mend makeds;
%makeds;
```

付録 4

「SQL プロシジャリファレンス」で示されている例のデータセット

概要 433
Employees 434
Houses 434
Match_11 434
Proclib.Delay 436
Proclib.Houses 437
Proclib.March 437
Proclib.Paylist2 438
Proclib.Payroll 439
Proclib.Payroll2 442
Proclib.Schedule2 442
Proclib.Staff 442
Proclib.Staff2 445
Proclib.Superv2 446
Stores 446
Survey 446

概要

このセクションでは、このガイドの PROC SQL の例で使用されるテーブルを作成するための DATA ステップを示します。

注: データセットを SAS エディタにコピーアンドペーストすると、列の間のスペーシングが変化することがあります。列の間のスペーシングをチェックし、データが INPUT ステートメントによって読み取り可能であることを確認します。

Employees

```
data Employees;
   input IdNum $4. +2 LName $11. FName $11. JobCode $3.
         +1 Salary 5. +1 Phone $12.;
   datalines;
1876  CHIN       JACK       TA1 42400 212/588-5634
1114  GREENWALD  JANICE     ME3 38000 212/588-1092
1556  PENNINGTON MICHAEL    ME1 29860 718/383-5681
1354  PARKER     MARY       FA3 65800 914/455-2337
1130  WOOD       DEBORAH    PT2 36514 212/587-0013
;
```

Houses

```
data houses;
   input House $ x y;
   datalines;
house1 1 1
house2 3 3
house3 2 3
house4 7 7
;
```

Match_11

```
data match_11;
   input Pair Low Age Lwt Race Smoke Ptd Ht UI @@;
   select(race);
      when (1) do;
         race1=0;
         race2=0;
      end;
      when (2) do;
         race1=1;
         race2=0;
      end;
      when (3) do;
         race1=0;
         race2=1;
      end;
   end;
   datalines;
1  0 14 135 1 0 0 0 0     1  1 14 101 3 1 1 0 0
2  0 15  98 2 0 0 0 0     2  1 15 115 3 0 0 0 1
3  0 16  95 3 0 0 0 0     3  1 16 130 3 0 0 0 0
```

```
4   0 17 103 3 0 0 0 0     4   1 17 130 3 1 1 0 1
5   0 17 122 1 1 0 0 0     5   1 17 110 1 1 0 0 0
6   0 17 113 2 0 0 0 0     6   1 17 120 1 1 0 0 0
7   0 17 113 2 0 0 0 0     7   1 17 120 2 0 0 0 0
8   0 17 119 3 0 0 0 0     8   1 17 142 2 0 0 1 0
9   0 18 100 1 1 0 0 0     9   1 18 148 3 0 0 0 0
10  0 18  90 1 1 0 0 1     10  1 18 110 2 1 1 0 0
11  0 19 150 3 0 0 0 0     11  1 19  91 1 1 1 0 1
12  0 19 115 3 0 0 0 0     12  1 19 102 1 0 0 0 0
13  0 19 235 1 1 0 1 0     13  1 19 112 1 1 0 0 1
14  0 20 120 3 0 0 0 1     14  1 20 150 1 1 0 0 0
15  0 20 103 3 0 0 0 0     15  1 20 125 3 0 0 0 1
16  0 20 169 3 0 1 0 1     16  1 20 120 2 1 0 0 0
17  0 20 141 1 0 1 0 1     17  1 20  80 3 1 0 0 1
18  0 20 121 2 1 0 0 0     18  1 20 109 3 0 0 0 0
19  0 20 127 3 0 0 0 0     19  1 20 121 1 1 1 0 1
20  0 20 120 3 0 0 0 0     20  1 20 122 2 1 0 0 0
21  0 20 158 1 0 0 0 0     21  1 20 105 3 0 0 0 0
22  0 21 108 1 1 0 0 1     22  1 21 165 1 1 0 1 0
23  0 21 124 3 0 0 0 0     23  1 21 200 2 0 0 0 0
24  0 21 185 2 1 0 0 0     24  1 21 103 3 0 0 0 0
25  0 21 160 1 0 0 0 0     25  1 21 100 3 0 1 0 0
26  0 21 115 1 0 0 0 0     26  1 21 130 1 1 0 1 0
27  0 22  95 3 0 0 1 0     27  1 22 130 1 1 0 0 0
28  0 22 158 2 0 1 0 0     28  1 22 130 1 1 1 0 1
29  0 23 130 2 0 0 0 0     29  1 23  97 3 0 0 0 1
30  0 23 128 3 0 0 0 0     30  1 23 187 2 1 0 0 0
31  0 23 119 3 0 0 0 0     31  1 23 120 3 0 0 0 0
32  0 23 115 3 1 0 0 0     32  1 23 110 1 1 1 0 0
33  0 23 190 1 0 0 0 0     33  1 23  94 3 1 0 0 0
34  0 24  90 1 1 1 0 0     34  1 24 128 2 0 1 0 0
35  0 24 115 1 0 0 0 0     35  1 24 132 3 0 0 1 0
36  0 24 110 3 0 0 0 0     36  1 24 155 1 1 1 0 0
37  0 24 115 3 0 0 0 0     37  1 24 138 1 0 0 0 0
38  0 24 110 3 0 1 0 0     38  1 24 105 2 1 0 0 0
39  0 25 118 1 1 0 0 0     39  1 25 105 3 0 1 1 0
40  0 25 120 3 0 0 0 1     40  1 25  85 3 0 0 0 1
41  0 25 155 1 0 0 0 0     41  1 25 115 3 0 0 0 0
42  0 25 125 2 0 0 0 0     42  1 25  92 1 1 0 0 0
43  0 25 140 1 0 0 0 0     43  1 25  89 3 0 1 0 0
44  0 25 241 2 0 0 1 0     44  1 25 105 3 0 1 0 0
45  0 26 113 1 1 0 0 0     45  1 26 117 1 1 1 0 0
46  0 26 168 2 1 0 0 0     46  1 26  96 3 0 0 0 0
47  0 26 133 3 1 1 0 0     47  1 26 154 3 0 1 1 0
48  0 26 160 3 0 0 0 0     48  1 26 190 1 1 0 0 0
49  0 27 124 1 1 0 0 0     49  1 27 130 2 0 0 0 1
50  0 28 120 3 0 0 0 0     50  1 28 120 3 1 1 0 1
51  0 28 130 3 0 0 0 0     51  1 28  95 1 1 0 0 0
52  0 29 135 1 0 0 0 0     52  1 29 130 1 0 0 0 1
53  0 30  95 1 1 0 0 0     53  1 30 142 1 1 1 0 0
54  0 31 215 1 1 0 0 0     54  1 31 102 1 1 1 0 0
55  0 32 121 3 0 0 0 0     55  1 32 105 1 1 0 0 0
56  0 34 170 1 0 1 0 0     56  1 34 187 2 1 0 1 0
;
```

Proclib.Delay

```
data proclib.delay;
   input flight $3. +5 date date7. +2 orig $3. +3 dest $3. +3
         delaycat $15. +2 destype $15. +8 delay;
   informat date date7.;
   format date date7.;
   datalines;
114     01MAR08  LGA   LAX   1-10 Minutes     Domestic              8
202     01MAR08  LGA   ORD   No Delay         Domestic             -5
219     01MAR08  LGA   LON   11+ Minutes      International        18
622     01MAR08  LGA   FRA   No Delay         International        -5
132     01MAR08  LGA   YYZ   11+ Minutes      International        14
271     01MAR08  LGA   PAR   1-10 Minutes     International         5
302     01MAR08  LGA   WAS   No Delay         Domestic             -2
114     02MAR08  LGA   LAX   No Delay         Domestic              0
202     02MAR08  LGA   ORD   1-10 Minutes     Domestic              5
219     02MAR08  LGA   LON   11+ Minutes      International        18
622     02MAR08  LGA   FRA   No Delay         International         0
132     02MAR08  LGA   YYZ   1-10 Minutes     International         5
271     02MAR08  LGA   PAR   1-10 Minutes     International         4
302     02MAR08  LGA   WAS   No Delay         Domestic              0
114     03MAR08  LGA   LAX   No Delay         Domestic             -1
202     03MAR08  LGA   ORD   No Delay         Domestic             -1
219     03MAR08  LGA   LON   1-10 Minutes     International         4
622     03MAR08  LGA   FRA   No Delay         International        -2
132     03MAR08  LGA   YYZ   1-10 Minutes     International         6
271     03MAR08  LGA   PAR   1-10 Minutes     International         2
302     03MAR08  LGA   WAS   1-10 Minutes     Domestic              5
114     04MAR08  LGA   LAX   11+ Minutes      Domestic             15
202     04MAR08  LGA   ORD   No Delay         Domestic             -5
219     04MAR08  LGA   LON   1-10 Minutes     International         3
622     04MAR08  LGA   FRA   11+ Minutes      International        30
132     04MAR08  LGA   YYZ   No Delay         International        -5
271     04MAR08  LGA   PAR   1-10 Minutes     International         5
302     04MAR08  LGA   WAS   1-10 Minutes     Domestic              7
114     05MAR08  LGA   LAX   No Delay         Domestic             -2
202     05MAR08  LGA   ORD   1-10 Minutes     Domestic              2
219     05MAR08  LGA   LON   1-10 Minutes     International         3
622     05MAR08  LGA   FRA   No Delay         International        -6
132     05MAR08  LGA   YYZ   1-10 Minutes     International         3
271     05MAR08  LGA   PAR   1-10 Minutes     International         5
114     06MAR08  LGA   LAX   No Delay         Domestic             -1
202     06MAR08  LGA   ORD   No Delay         Domestic             -3
219     06MAR08  LGA   LON   11+ Minutes      International        27
132     06MAR08  LGA   YYZ   1-10 Minutes     International         7
302     06MAR08  LGA   WAS   1-10 Minutes     Domestic              1
114     07MAR08  LGA   LAX   No Delay         Domestic             -1
202     07MAR08  LGA   ORD   No Delay         Domestic             -2
219     07MAR08  LGA   LON   11+ Minutes      International        15
622     07MAR08  LGA   FRA   11+ Minutes      International        21
132     07MAR08  LGA   YYZ   No Delay         International        -2
```

```
271     07MAR08  LGA   PAR   1-10 Minutes     International          4
302     07MAR08  LGA   WAS   No Delay         Domestic               0
;
```

Proclib.Houses

このデータセットの内容は、“Houses” (434 ページ)データセットとは異なります。このデータセットは、“例: INTO 句” (255 ページ)のみを対象にしています。

```
libname proclib 'SAS-library';

data proclib.houses;
input Style $ 1-8 SqFeet 15-18;
datalines;
CONDO          900
CONDO         1000
RANCH         1200
RANCH         1400
SPLIT         1600
SPLIT         1800
TWOSTORY      2100
TWOSTORY      3000
TWOSTORY      1940
TWOSTORY      1860
;
```

Proclib.March

```
data proclib.march;
   input flight $3. +5 date date7. +3 depart time5. +2 orig $3.
         +3 dest $3.  +7 miles +6 boarded +6 capacity;
   format date date7. depart time5.;
   informat date date7. depart time5.;
   datalines;
114     01MAR08    7:10  LGA   LAX       2475       172       210
202     01MAR08   10:43  LGA   ORD        740       151       210
219     01MAR08    9:31  LGA   LON       3442       198       250
622     01MAR08   12:19  LGA   FRA       3857       207       250
132     01MAR08   15:35  LGA   YYZ        366       115       178
271     01MAR08   13:17  LGA   PAR       3635       138       250
302     01MAR08   20:22  LGA   WAS        229       105       180
114     02MAR08    7:10  LGA   LAX       2475       119       210
202     02MAR08   10:43  LGA   ORD        740       120       210
219     02MAR08    9:31  LGA   LON       3442       147       250
622     02MAR08   12:19  LGA   FRA       3857       176       250
132     02MAR08   15:35  LGA   YYZ        366       106       178
302     02MAR08   20:22  LGA   WAS        229        78       180
271     02MAR08   13:17  LGA   PAR       3635       104       250
114     03MAR08    7:10  LGA   LAX       2475       197       210
202     03MAR08   10:43  LGA   ORD        740       118       210
```

```
219     03MAR08    9:31  LGA   LON       3442       197       250
622     03MAR08   12:19  LGA   FRA       3857       180       250
132     03MAR08   15:35  LGA   YYZ        366        75       178
271     03MAR08   13:17  LGA   PAR       3635       147       250
302     03MAR08   20:22  LGA   WAS        229       123       180
114     04MAR08    7:10  LGA   LAX       2475       178       210
202     04MAR08   10:43  LGA   ORD        740       148       210
219     04MAR08    9:31  LGA   LON       3442       232       250
622     04MAR08   12:19  LGA   FRA       3857       137       250
132     04MAR08   15:35  LGA   YYZ        366       117       178
271     04MAR08   13:17  LGA   PAR       3635       146       250
302     04MAR08   20:22  LGA   WAS        229       115       180
114     05MAR08    7:10  LGA   LAX       2475       117       210
202     05MAR08   10:43  LGA   ORD        740       104       210
219     05MAR08    9:31  LGA   LON       3442       160       250
622     05MAR08   12:19  LGA   FRA       3857       185       250
132     05MAR08   15:35  LGA   YYZ        366       157       178
271     05MAR08   13:17  LGA   PAR       3635       177       250
114     06MAR08    7:10  LGA   LAX       2475       128       210
202     06MAR08   10:43  LGA   ORD        740       115       210
219     06MAR08    9:31  LGA   LON       3442       163       250
132     06MAR08   15:35  LGA   YYZ        366       150       178
302     06MAR08   20:22  LGA   WAS        229        66       180
114     07MAR08    7:10  LGA   LAX       2475       160       210
202     07MAR08   10:43  LGA   ORD        740       175       210
219     07MAR08    9:31  LGA   LON       3442       241       250
622     07MAR08   12:19  LGA   FRA       3857       210       250
132     07MAR08   15:35  LGA   YYZ        366       164       178
271     07MAR08   13:17  LGA   PAR       3635       155       250
302     07MAR08   20:22  LGA   WAS        229       135       180
;
```

Proclib.Paylist2

```
proc sql;
   create table proclib.paylist2
       (IdNum char(4),
        Gender char(1),
        Jobcode char(3),
        Salary num,
        Birth num informat=date7.
                  format=date7.,
        Hired num informat=date7.
                  format=date7.);

insert into proclib.paylist2
values('1919','M','TA2',34376,'12SEP66'd,'04JUN87'd)
values('1653','F','ME2',31896,'15OCT64'd,'09AUG92'd)
values('1350','F','FA3',36886,'31AUG55'd,'29JUL91'd)
values('1401','M','TA3',38822,'13DEC55'd,'17NOV93'd)
values('1499','M','ME1',23025,'26APR74'd,'07JUN92'd);
```

```
title 'PROCLIB.PAYLIST2 Table';
select * from proclib.paylist2;
```

Proclib.Payroll

このデータセットは、“例 3: PROC SQL テーブルのデータの更新” (271 ページ) で更新されます。更新されたデータは、その後の例で使用されます。

```
data proclib.payroll;
   input IdNumber $4. +3 Gender $1. +4 Jobcode $3. +9 Salary 5.
         +2 Birth date7. +2 Hired date7.;
   informat birth date7. hired date7.;
   format birth date7. hired date7.;
   datalines;
1919   M    TA2         34376  12SEP60  04JUN87
1653   F    ME2         35108  15OCT64  09AUG90
1400   M    ME1         29769  05NOV67  16OCT90
1350   F    FA3         32886  31AUG65  29JUL90
1401   M    TA3         38822  13DEC50  17NOV85
1499   M    ME3         43025  26APR54  07JUN80
1101   M    SCP         18723  06JUN62  01OCT90
1333   M    PT2         88606  30MAR61  10FEB81
1402   M    TA2         32615  17JAN63  02DEC90
1479   F    TA3         38785  22DEC68  05OCT89
1403   M    ME1         28072  28JAN69  21DEC91
1739   M    PT1         66517  25DEC64  27JAN91
1658   M    SCP         17943  08APR67  29FEB92
1428   F    PT1         68767  04APR60  16NOV91
1782   M    ME2         35345  04DEC70  22FEB92
1244   M    ME2         36925  31AUG63  17JAN88
1383   M    BCK         25823  25JAN68  20OCT92
1574   M    FA2         28572  27APR60  20DEC92
1789   M    SCP         18326  25JAN57  11APR78
1404   M    PT2         91376  24FEB53  01JAN80
1437   F    FA3         33104  20SEP60  31AUG84
1639   F    TA3         40260  26JUN57  28JAN84
1269   M    NA1         41690  03MAY72  28NOV92
1065   M    ME2         35090  26JAN44  07JAN87
1876   M    TA3         39675  20MAY58  27APR85
1037   F    TA1         28558  10APR64  13SEP92
1129   F    ME2         34929  08DEC61  17AUG91
1988   M    FA3         32217  30NOV59  18SEP84
1405   M    SCP         18056  05MAR66  26JAN92
1430   F    TA2         32925  28FEB62  27APR87
1983   F    FA3         33419  28FEB62  27APR87
1134   F    TA2         33462  05MAR69  21DEC88
1118   M    PT3        111379  16JAN44  18DEC80
1438   F    TA3         39223  15MAR65  18NOV87
1125   F    FA2         28888  08NOV68  11DEC87
1475   F    FA2         27787  15DEC61  13JUL90
1117   M    TA3         39771  05JUN63  13AUG92
1935   F    NA2         51081  28MAR54  16OCT81
1124   F    FA1         23177  10JUL58  01OCT90
```

1422	F	FA1	22454	04JUN64	06APR91
1616	F	TA2	34137	01MAR70	04JUN93
1406	M	ME2	35185	08MAR61	17FEB87
1120	M	ME1	28619	11SEP72	07OCT93
1094	M	FA1	22268	02APR70	17APR91
1389	M	BCK	25028	15JUL59	18AUG90
1905	M	PT1	65111	16APR72	29MAY92
1407	M	PT1	68096	23MAR69	18MAR90
1114	F	TA2	32928	18SEP69	27JUN87
1410	M	PT2	84685	03MAY67	07NOV86
1439	F	PT1	70736	06MAR64	10SEP90
1409	M	ME3	41551	19APR50	22OCT81
1408	M	TA2	34138	29MAR60	14OCT87
1121	M	ME1	29112	26SEP71	07DEC91
1991	F	TA1	27645	07MAY72	12DEC92
1102	M	TA2	34542	01OCT59	15APR91
1356	M	ME2	36869	26SEP57	22FEB83
1545	M	PT1	66130	12AUG59	29MAY90
1292	F	ME2	36691	28OCT64	02JUL89
1440	F	ME2	35757	27SEP62	09APR91
1368	M	FA2	27808	11JUN61	03NOV84
1369	M	TA2	33705	28DEC61	13MAR87
1411	M	FA2	27265	27MAY61	01DEC89
1113	F	FA1	22367	15JAN68	17OCT91
1704	M	BCK	25465	30AUG66	28JUN87
1900	M	ME2	35105	25MAY62	27OCT87
1126	F	TA3	40899	28MAY63	21NOV80
1677	M	BCK	26007	05NOV63	27MAR89
1441	F	FA2	27158	19NOV69	23MAR91
1421	M	TA2	33155	08JAN59	28FEB90
1119	M	TA1	26924	20JUN62	06SEP88
1834	M	BCK	26896	08FEB72	02JUL92
1777	M	PT3	109630	23SEP51	21JUN81
1663	M	BCK	26452	11JAN67	11AUG91
1106	M	PT2	89632	06NOV57	16AUG84
1103	F	FA1	23738	16FEB68	23JUL92
1477	M	FA2	28566	21MAR64	07MAR88
1476	F	TA2	34803	30MAY66	17MAR87
1379	M	ME3	42264	08AUG61	10JUN84
1104	M	SCP	17946	25APR63	10JUN91
1009	M	TA1	28880	02MAR59	26MAR92
1412	M	ME1	27799	18JUN56	05DEC91
1115	F	FA3	32699	22AUG60	29FEB80
1128	F	TA2	32777	23MAY65	20OCT90
1442	F	PT2	84536	05SEP66	12APR88
1417	M	NA2	52270	27JUN64	07MAR89
1478	M	PT2	84203	09AUG59	24OCT90
1673	M	BCK	25477	27FEB70	15JUL91
1839	F	NA1	43433	29NOV70	03JUL93
1347	M	TA3	40079	21SEP67	06SEP84
1423	F	ME2	35773	14MAY68	19AUG90
1200	F	ME1	27816	10JAN71	14AUG92
1970	F	FA1	22615	25SEP64	12MAR91
1521	M	ME3	41526	12APR63	13JUL88
1354	F	SCP	18335	29MAY71	16JUN92
1424	F	FA2	28978	04AUG69	11DEC89

```
1132   F    FA1         22413  30MAY72  22OCT93
1845   M    BCK         25996  20NOV59  22MAR80
1556   M    PT1         71349  22JUN64  11DEC91
1413   M    FA2         27435  16SEP65  02JAN90
1123   F    TA1         28407  31OCT72  05DEC92
1907   M    TA2         33329  15NOV60  06JUL87
1436   F    TA2         34475  11JUN64  12MAR87
1385   M    ME3         43900  16JAN62  01APR86
1432   F    ME2         35327  03NOV61  10FEB85
1111   M    NA1         40586  14JUL73  31OCT92
1116   F    FA1         22862  28SEP69  21MAR91
1352   M    NA2         53798  02DEC60  16OCT86
1555   F    FA2         27499  16MAR68  04JUL92
1038   F    TA1         26533  09NOV69  23NOV91
1420   M    ME3         43071  19FEB65  22JUL87
1561   M    TA2         34514  30NOV63  07OCT87
1434   F    FA2         28622  11JUL62  28OCT90
1414   M    FA1         23644  24MAR72  12APR92
1112   M    TA1         26905  29NOV64  07DEC92
1390   M    FA2         27761  19FEB65  23JUN91
1332   M    NA1         42178  17SEP70  04JUN91
1890   M    PT2         91908  20JUL51  25NOV79
1429   F    TA1         27939  28FEB60  07AUG92
1107   M    PT2         89977  09JUN54  10FEB79
1908   F    TA2         32995  10DEC69  23APR90
1830   F    PT2         84471  27MAY57  29JAN83
1882   M    ME3         41538  10JUL57  21NOV78
1050   M    ME2         35167  14JUL63  24AUG86
1425   F    FA1         23979  28DEC71  28FEB93
1928   M    PT2         89858  16SEP54  13JUL90
1480   F    TA3         39583  03SEP57  25MAR81
1100   M    BCK         25004  01DEC60  07MAY88
1995   F    ME1         28810  24AUG73  19SEP93
1135   F    FA2         27321  20SEP60  31MAR90
1415   M    FA2         28278  09MAR58  12FEB88
1076   M    PT1         66558  14OCT55  03OCT91
1426   F    TA2         32991  05DEC66  25JUN90
1564   F    SCP         18833  12APR62  01JUL92
1221   F    FA2         27896  22SEP67  04OCT91
1133   M    TA1         27701  13JUL66  12FEB92
1435   F    TA3         38808  12MAY59  08FEB80
1418   M    ME1         28005  29MAR57  06JAN92
1017   M    TA3         40858  28DEC57  16OCT81
1443   F    NA1         42274  17NOV68  29AUG91
1131   F    TA2         32575  26DEC71  19APR91
1427   F    TA2         34046  31OCT70  30JAN90
1036   F    TA3         39392  19MAY65  23OCT84
1130   F    FA1         23916  16MAY71  05JUN92
1127   F    TA2         33011  09NOV64  07DEC86
1433   F    FA3         32982  08JUL66  17JAN87
1431   F    FA3         33230  09JUN64  05APR88
1122   F    FA2         27956  01MAY63  27NOV88
1105   M    ME2         34805  01MAR62  13AUG90
;
```

Proclib.Payroll2

```
data proclib.payroll2;
   input idnum $4. +3 gender $1. +4 jobcode $3. +9 salary 5.
         +2 birth date7. +2 hired date7.;
   informat birth date7. hired date7.;
   format birth date7. hired date7.;
   datalines;
1639   F    TA3         42260  26JUN57  28JAN84
1065   M    ME3         38090  26JAN44  07JAN87
1561   M    TA3         36514  30NOV63  07OCT87
1221   F    FA3         29896  22SEP67  04OCT91
1447   F    FA1         22123  07AUG72  29OCT92
1998   M    SCP         23100  10SEP70  02NOV92
1036   F    TA3         42465  19MAY65  23OCT84
1106   M    PT3         94039  06NOV57  16AUG84
1129   F    ME3         36758  08DEC61  17AUG91
1350   F    FA3         36098  31AUG65  29JUL90
1369   M    TA3         36598  28DEC61  13MAR87
1076   M    PT1         69742  14OCT55  03OCT91
;
```

Proclib.Schedule2

```
data proclib.schedule2;
   input flight $3. +5 date date7. +2 dest $3. +3 idnum $4.;
   format date date7.;
   informat date date7.;
   datalines;
132     01MAR94  BOS   1118
132     01MAR94  BOS   1402
219     02MAR94  PAR   1616
219     02MAR94  PAR   1478
622     03MAR94  LON   1430
622     03MAR94  LON   1882
271     04MAR94  NYC   1430
271     04MAR94  NYC   1118
579     05MAR94  RDU   1126
579     05MAR94  RDU   1106
;
```

Proclib.Staff

```
data proclib.staff;
   input idnum $4. +3 lname $15. +2 fname $15. +2 city $15. +2
```

```
        state $2. +5 hphone $12.;
   datalines;
1919   ADAMS            GERALD          STAMFORD        CT     203/781-1255
1653   ALIBRANDI        MARIA           BRIDGEPORT      CT     203/675-7715
1400   ALHERTANI        ABDULLAH        NEW YORK        NY     212/586-0808
1350   ALVAREZ          MERCEDES        NEW YORK        NY     718/383-1549
1401   ALVAREZ          CARLOS          PATERSON        NJ     201/732-8787
1499   BAREFOOT         JOSEPH          PRINCETON       NJ     201/812-5665
1101   BAUCOM           WALTER          NEW YORK        NY     212/586-8060
1333   BANADYGA         JUSTIN          STAMFORD        CT     203/781-1777
1402   BLALOCK          RALPH           NEW YORK        NY     718/384-2849
1479   BALLETTI         MARIE           NEW YORK        NY     718/384-8816
1403   BOWDEN           EARL            BRIDGEPORT      CT     203/675-3434
1739   BRANCACCIO       JOSEPH          NEW YORK        NY     212/587-1247
1658   BREUHAUS         JEREMY          NEW YORK        NY     212/587-3622
1428   BRADY            CHRISTINE       STAMFORD        CT     203/781-1212
1782   BREWCZAK         JAKOB           STAMFORD        CT     203/781-0019
1244   BUCCI            ANTHONY         NEW YORK        NY     718/383-3334
1383   BURNETTE         THOMAS          NEW YORK        NY     718/384-3569
1574   CAHILL           MARSHALL        NEW YORK        NY     718/383-2338
1789   CARAWAY          DAVIS           NEW YORK        NY     212/587-9000
1404   COHEN            LEE             NEW YORK        NY     718/384-2946
1437   CARTER           DOROTHY         BRIDGEPORT      CT     203/675-4117
1639   CARTER-COHEN     KAREN           STAMFORD        CT     203/781-8839
1269   CASTON           FRANKLIN        STAMFORD        CT     203/781-3335
1065   COPAS            FREDERICO       NEW YORK        NY     718/384-5618
1876   CHIN             JACK            NEW YORK        NY     212/588-5634
1037   CHOW             JANE            STAMFORD        CT     203/781-8868
1129   COUNIHAN         BRENDA          NEW YORK        NY     718/383-2313
1988   COOPER           ANTHONY         NEW YORK        NY     212/587-1228
1405   DACKO            JASON           PATERSON        NJ     201/732-2323
1430   DABROWSKI        SANDRA          BRIDGEPORT      CT     203/675-1647
1983   DEAN             SHARON          NEW YORK        NY     718/384-1647
1134   DELGADO          MARIA           STAMFORD        CT     203/781-1528
1118   DENNIS           ROGER           NEW YORK        NY     718/383-1122
1438   DABBOUSSI        KAMILLA         STAMFORD        CT     203/781-2229
1125   DUNLAP           DONNA           NEW YORK        NY     718/383-2094
1475   ELGES            MARGARETE       NEW YORK        NY     718/383-2828
1117   EDGERTON         JOSHUA          NEW YORK        NY     212/588-1239
1935   FERNANDEZ        KATRINA         BRIDGEPORT      CT     203/675-2962
1124   FIELDS           DIANA           WHITE PLAINS    NY     914/455-2998
1422   FUJIHARA         KYOKO           PRINCETON       NJ     201/812-0902
1616   FUENTAS          CARLA           NEW YORK        NY     718/384-3329
1406   FOSTER           GERALD          BRIDGEPORT      CT     203/675-6363
1120   GARCIA           JACK            NEW YORK        NY     718/384-4930
1094   GOMEZ            ALAN            BRIDGEPORT      CT     203/675-7181
1389   GOLDSTEIN        LEVI            NEW YORK        NY     718/384-9326
1905   GRAHAM           ALVIN           NEW YORK        NY     212/586-8815
1407   GREGORSKI        DANIEL          MT. VERNON      NY     914/468-1616
1114   GREENWALD        JANICE          NEW YORK        NY     212/588-1092
1410   HARRIS           CHARLES         STAMFORD        CT     203/781-0937
1439   HASENHAUER       CHRISTINA       BRIDGEPORT      CT     203/675-4987
1409   HAVELKA          RAYMOND         STAMFORD        CT     203/781-9697
1408   HENDERSON        WILLIAM         PRINCETON       NJ     201/812-4789
1121   HERNANDEZ        ROBERTO         NEW YORK        NY     718/384-3313
1991   HOWARD           GRETCHEN        BRIDGEPORT      CT     203/675-0007
```

1102	HERMANN	JOACHIM	WHITE PLAINS	NY	914/455-0976
1356	HOWARD	MICHAEL	NEW YORK	NY	212/586-8411
1545	HERRERO	CLYDE	STAMFORD	CT	203/781-1119
1292	HUNTER	HELEN	BRIDGEPORT	CT	203/675-4830
1440	JACKSON	LAURA	STAMFORD	CT	203/781-0088
1368	JEPSEN	RONALD	STAMFORD	CT	203/781-8413
1369	JONSON	ANTHONY	NEW YORK	NY	212/587-5385
1411	JOHNSEN	JACK	PATERSON	NJ	201/732-3678
1113	JOHNSON	LESLIE	NEW YORK	NY	718/383-3003
1704	JONES	NATHAN	NEW YORK	NY	718/384-0049
1900	KING	WILLIAM	NEW YORK	NY	718/383-3698
1126	KIMANI	ANNE	NEW YORK	NY	212/586-1229
1677	KRAMER	JACKSON	BRIDGEPORT	CT	203/675-7432
1441	LAWRENCE	KATHY	PRINCETON	NJ	201/812-3337
1421	LEE	RUSSELL	MT. VERNON	NY	914/468-9143
1119	LI	JEFF	NEW YORK	NY	212/586-2344
1834	LEBLANC	RUSSELL	NEW YORK	NY	718/384-0040
1777	LUFKIN	ROY	NEW YORK	NY	718/383-4413
1663	MARKS	JOHN	NEW YORK	NY	212/587-7742
1106	MARSHBURN	JASPER	STAMFORD	CT	203/781-1457
1103	MCDANIEL	RONDA	NEW YORK	NY	212/586-0013
1477	MEYERS	PRESTON	BRIDGEPORT	CT	203/675-8125
1476	MONROE	JOYCE	STAMFORD	CT	203/781-2837
1379	MORGAN	ALFRED	STAMFORD	CT	203/781-2216
1104	MORGAN	CHRISTOPHER	NEW YORK	NY	718/383-9740
1009	MORGAN	GEORGE	NEW YORK	NY	212/586-7753
1412	MURPHEY	JOHN	PRINCETON	NJ	201/812-4414
1115	MURPHY	ALICE	NEW YORK	NY	718/384-1982
1128	NELSON	FELICIA	BRIDGEPORT	CT	203/675-1166
1442	NEWKIRK	SANDRA	PRINCETON	NJ	201/812-3331
1417	NEWKIRK	WILLIAM	PATERSON	NJ	201/732-6611
1478	NEWTON	JAMES	NEW YORK	NY	212/587-5549
1673	NICHOLLS	HENRY	STAMFORD	CT	203/781-7770
1839	NORRIS	DIANE	NEW YORK	NY	718/384-1767
1347	O'NEAL	BRYAN	NEW YORK	NY	718/384-0230
1423	OSWALD	LESLIE	MT. VERNON	NY	914/468-9171
1200	OVERMAN	MICHELLE	STAMFORD	CT	203/781-1835
1970	PARKER	ANNE	NEW YORK	NY	718/383-3895
1521	PARKER	JAY	NEW YORK	NY	212/587-7603
1354	PARKER	MARY	WHITE PLAINS	NY	914/455-2337
1424	PATTERSON	RENEE	NEW YORK	NY	212/587-8991
1132	PEARCE	CAROL	NEW YORK	NY	718/384-1986
1845	PEARSON	JAMES	NEW YORK	NY	718/384-2311
1556	PENNINGTON	MICHAEL	NEW YORK	NY	718/383-5681
1413	PETERS	RANDALL	PRINCETON	NJ	201/812-2478
1123	PETERSON	SUZANNE	NEW YORK	NY	718/383-0077
1907	PHELPS	WILLIAM	STAMFORD	CT	203/781-1118
1436	PORTER	SUSAN	NEW YORK	NY	718/383-5777
1385	RAYNOR	MILTON	BRIDGEPORT	CT	203/675-2846
1432	REED	MARILYN	MT. VERNON	NY	914/468-5454
1111	RHODES	JEREMY	PRINCETON	NJ	201/812-1837
1116	RICHARDS	CASEY	NEW YORK	NY	212/587-1224
1352	RIVERS	SIMON	NEW YORK	NY	718/383-3345
1555	RODRIGUEZ	JULIA	BRIDGEPORT	CT	203/675-2401
1038	RODRIGUEZ	MARIA	BRIDGEPORT	CT	203/675-2048
1420	ROUSE	JEREMY	PATERSON	NJ	201/732-9834

```
1561   SANDERS          RAYMOND          NEW YORK         NY     212/588-6615
1434   SANDERSON        EDITH            STAMFORD         CT     203/781-1333
1414   SANDERSON        NATHAN           BRIDGEPORT       CT     203/675-1715
1112   SANYERS          RANDY            NEW YORK         NY     718/384-4895
1390   SMART            JONATHAN         NEW YORK         NY     718/383-1141
1332   STEPHENSON       ADAM             BRIDGEPORT       CT     203/675-1497
1890   STEPHENSON       ROBERT           NEW YORK         NY     718/384-9874
1429   THOMPSON         ALICE            STAMFORD         CT     203/781-3857
1107   THOMPSON         WAYNE            NEW YORK         NY     718/384-3785
1908   TRENTON          MELISSA          NEW YORK         NY     212/586-6262
1830   TRIPP            KATHY            BRIDGEPORT       CT     203/675-2479
1882   TUCKER           ALAN             NEW YORK         NY     718/384-0216
1050   TUTTLE           THOMAS           WHITE PLAINS     NY     914/455-2119
1425   UNDERWOOD        JENNY            STAMFORD         CT     203/781-0978
1928   UPCHURCH         LARRY            WHITE PLAINS     NY     914/455-5009
1480   UPDIKE           THERESA          NEW YORK         NY     212/587-8729
1100   VANDEUSEN        RICHARD          NEW YORK         NY     212/586-2531
1995   VARNER           ELIZABETH        NEW YORK         NY     718/384-7113
1135   VEGA             ANNA             NEW YORK         NY     718/384-5913
1415   VEGA             FRANKLIN         NEW YORK         NY     718/384-2823
1076   VENTER           RANDALL          NEW YORK         NY     718/383-2321
1426   VICK             THERESA          PRINCETON        NJ     201/812-2424
1564   WALTERS          ANNE             NEW YORK         NY     212/587-3257
1221   WALTERS          DIANE            NEW YORK         NY     718/384-1918
1133   WANG             CHIN             NEW YORK         NY     212/587-1956
1435   WARD             ELAINE           NEW YORK         NY     718/383-4987
1418   WATSON           BERNARD          NEW YORK         NY     718/383-1298
1017   WELCH            DARIUS           NEW YORK         NY     212/586-5535
1443   WELLS            AGNES            STAMFORD         CT     203/781-5546
1131   WELLS            NADINE           NEW YORK         NY     718/383-1045
1427   WHALEY           CAROLYN          MT. VERNON       NY     914/468-4528
1036   WONG             LESLIE           NEW YORK         NY     212/587-2570
1130   WOOD             DEBORAH          NEW YORK         NY     212/587-0013
1127   WOOD             SANDRA           NEW YORK         NY     212/587-2881
1433   YANCEY           ROBIN            PRINCETON        NJ     201/812-1874
1431   YOUNG            DEBORAH          STAMFORD         CT     203/781-2987
1122   YOUNG            JOANN            NEW YORK         NY     718/384-2021
1105   YOUNG            LAWRENCE         NEW YORK         NY     718/384-0008
;
```

Proclib.Staff2

```
data proclib.staff2;
input IdNum $4. @7 Lname $12. @20 Fname $8. @30 City $10.
      @42 State $2. @50 Hphone $12.;
   datalines;
1106  MARSHBURN    JASPER    STAMFORD    CT      203/781-1457
1430  DABROWSKI    SANDRA    BRIDGEPORT  CT      203/675-1647
1118  DENNIS       ROGER     NEW YORK    NY      718/383-1122
1126  KIMANI       ANNE      NEW YORK    NY      212/586-1229
1402  BLALOCK      RALPH     NEW YORK    NY      718/384-2849
1882  TUCKER       ALAN      NEW YORK    NY      718/384-0216
1479  BALLETTI     MARIE     NEW YORK    NY      718/384-8816
```

```
1420  ROUSE        JEREMY    PATERSON    NJ      201/732-9834
1403  BOWDEN       EARL      BRIDGEPORT  CT      203/675-3434
1616  FUENTAS      CARLA     NEW YORK    NY      718/384-3329
;
```

Proclib.Superv2

```
data proclib.superv2;
   input supid $4. +8 state $2. +5  jobcat  $2.;
   label supid='Supervisor Id' jobcat='Job Category';
   datalines;
1417        NJ     NA
1352        NY     NA
1106        CT     PT
1442        NJ     PT
1118        NY     PT
1405        NJ     SC
1564        NY     SC
1639        CT     TA
1126        NY     TA
1882        NY     ME
;
```

Stores

```
data stores;
  input Store $ x y;
  datalines;
store1 5 1
store2 5 3
store3 3 5
store4 7 5
;
```

Survey

```
data survey;
  input id $ diet $ exer $ hours xwk educ;
  datalines;
1001 yes yes 1 3 1
1002 no  yes 1 4 2
1003 no  no  . . .n
1004 yes yes 2 3 .x
1005 no  yes 2 3 .x
1006 yes yes 2 4 .x
1007 no  yes .5 3 .
1008 no  no  . .  .
;
```

推奨資料

このタイトルに関連した推奨される参考資料のリストを次に示します。

- *Base SAS プロシジャガイド*
- *SAS 言語リファレンス: 解説編*
- *SAS データセットオプション: リファレンス*
- *SAS 出力形式と入力形式: リファレンス*
- *SAS 関数と CALL ルーチン: リファレンス*
- *SAS ステートメント: リファレンス*
- *SAS マクロ言語: リファレンス*
- *SAS/ACCESS for Relational Databases: Reference*
- *SAS/GRAPH: Reference*

SAS Press から発行されている推奨ドキュメントには次のものがあります。

- *The Essential PROC SQL Handbook for SAS Users*
- *PROC SQL by Example:Using SQL within SAS*
- *PROC SQL:Beyond the Basics Using SAS*
- *Combining and Modifying SAS Data Sets:Examples*
- *SAS Guide to Report Writing:Examples*
- *Little SAS Book:A Primer*

SAS 刊行物の一覧については、sas.com/store/books から入手できます。必要な書籍についての質問は SAS 担当者までお寄せください:

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
電話: 1-800-727-0025
ファクシミリ: 1-919-677-4444
メール: sasbook@sas.com
Web アドレス: sas.com/store/books

用語集

DISTINCT
SQL プロシジャの出力から重複行を削除するキーワード。

PROC SQL ビュー
SQL プロシジャによって作成される SAS データセット。PROC SQL ビューにはデータは含まれません。かわりに、他のファイル(SAS データファイル、SAS/ACCESS ビュー、DATA ステップビュー、他の PROC SQL ビューなど)からのデータ値の読み取りを可能にする情報が格納されます。PROC SQL ビューの出力は、1 つ以上のファイルのサブセットまたはスーパーセットのいずれかにすることができます。

SAS 出力形式
SAS 言語要素のタイプの 1 つであり、特定のデータ型(数値、文字、日付、タイムスタンプ)にしたがってデータ値を表示または書き出す場合に使用されます。省略形は出力形式です。

SAS データセット
いずれかのネイティブ SAS ファイル形式の内容を含むファイル。SAS データセットには次の 2 つのタイプがあります。SAS データファイルと SAS データビューです。

SAS データビュー
SAS データセットのタイプの 1 つであり、他のファイルからデータ値を取り出す場合に使用されます。SAS データビューには、変数(列)のデータ型や長さなどの情報と、他の SAS データセットから、または SAS 以外のソフトウェアベンダーのファイル形式でデータを格納しているファイルからデータ値を取り出すのに必要となる情報のみが含まれています。省略形はデータビューです。

SAS データファイル
SAS データセットの一種。データ値と、データに関連付けられたディスクリプタ情報を含みます。ディスクリプタ情報には、変数のデータ型や長さ、データの作成に使用されたエンジンの名前などが含まれています。

SQL
構造化照会言語(Structured Query Language:SQL)を参照。

SQL パススルー機能
SQL クエリコードを、処理するために特定の DBMS に渡すことを可能にする技術。省略形はパススルー機能です。

WHERE 句
1 つ以上の WHERE 式の前に記述するキーワード(WHERE)。

WHERE 式
オブザベーションの選択基準を定義します。

一貫性制約
SAS データファイル内の変数に格納できるデータ値を制限する、データ検証ルール。一貫性制約はデータのデータの妥当性と一貫性の維持に役立ちます。

インデックス
SAS が SAS データセットのオブザベーションに高速かつ効率的にアクセスできるようにするための、SAS データセットの構成要素。SAS インデックスの目的は、WHERE 句処理を最適化し、BY グループ処理を促進することです。

インラインビュー
SQL プロシジャの FROM 句内でネストされたクエリ式。インラインビューは、外側のクエリによってデータの選択に使用されるテーブルを内部で生成します。インラインビューを使用すると、ビューを作成してそれを別のクエリで参照するかわりに、FROM 句内でインラインでビューを指定できるため、プログラムステップを省くことができます。インラインビューは、インラインビューを定義したクエリ(またはステートメント)でのみ参照できます。

外部結合
1 つのテーブル内のすべての行に加えて、他のテーブル内の一部またはすべての行を返す、2 つのテーブル間の結合。左外部結合または右外部結合は、1 つのテーブル(左外部結合では SQL ステートメントの左側のテーブル、右外部結合では右側のテーブル)内のすべての行に加えて、他のテーブル内の一致する行を返します。完全外部結合は、両方のテーブル内のすべての行を返します。

クエリ
1 つ以上のデータソースからの特定の情報を要求する一連の指示。

クエリ式
PROC SQL において、少なくとも 1 つのテーブルを参照する SELECT ステートメント。実行されると、このステートメントが実行される間だけ存在する一時テーブルを作成します。セット演算子を使用して複数のテーブル式の結果を結合し、クエリ式を作成できます。

グループ
1 つ以上の共通する列または変数について、同じ 1 つ以上の値を持つ一連の行またはオブザベーション。

クロス結合
結合対象のテーブルの積を返す結合の一種。クロス結合は、機能的にデカルト積と同じです。

計算される列
クエリにおいて、照会されているどのテーブルにも存在しないが、列式の結果として作成される列。

結合
2 つ以上のテーブルのデータを結合する演算。通常、結合は、SQL (構造化照会言語)コードまたはユーザーインターフェイスによって作成します。

結合条件
テーブルの結合方法を決定する一連のパラメータ。結合条件は通常、WHERE 式または SQL ON 句で指定します。

欠損値
変数に対する値の一種で、特定の行または列に対するデータが含まれていない変数に対する値です。デフォルトでは、SAS は欠損している数値を 1 つのピリオドで記述し、欠損している文字値を空白で記述します。

構造化照会言語(Structured Query Language:SQL)
リレーショナルデータベース管理システムにおいて、データベース管理システム内でのオブジェクトの作成および操作のために使用される、標準化された高度な照会言語。SAS では、SQL プロシジャを介して SQL を実装します。略称は、SQL です。

自然結合
各テーブルの 1 つ以上の列が同じ名前と同じデータタイプを持ち、それらの列に同じ値が含まれる場合に、テーブルからそれらの行を選択して返す結合の一種。

出力形式
SAS 出力形式を参照してください。

条件演算子
SQL プロシジャにおいて、検索対象の行を指定する WHERE 句の一部。

単一インデックス
1 つの変数のみの値を使用してオブザベーションを特定するインデックス。

データセット
SAS データセットを参照してください。

データビュー
SAS データビューを参照してください。

デカルト積
結合対象の各テーブルの各行を、他のすべての結合対象のテーブルの各行と組み合わせる結合の一種。

等結合
SQL プロシジャでの結合の一種。たとえば、2 つのテーブルを等結合で結合する場合、SQL 式において、最初のテーブルのある列の値は、2 番目のテーブルのその列の値と等しい必要があります。

内部結合
他のテーブル内に 1 つ以上の一致する行が存在する 1 つのテーブル内のすべての行を返す、2 つのテーブル間の結合。

パススルー機能
SQL パススルー機能を参照してください。

ビュー
後で使用できるように名前を付けて保存された仮想データセットの定義。ビューには、データは含まれません。別の場所に保存されているデータの説明や定義のみ含まれます。

複合インデックス
2 つ以上のキー変数の値を検証することによって、SAS データセットのオブザベーションの場所を特定するインデックス。

和結合
各入力テーブルのすべての行をそれぞれの値と共に返す結合の一種。あるテーブルに存在しない列については、結果として出力されるテーブルのそれらの行にヌル(欠損)値が設定されます。

列
テーブルの垂直コンポーネントそれぞれの列には、一意の名前が付けられ、特定のタイプのデータが含まれ、特定の属性があります。列は、SAS 用語の変数に相当します。

列のエイリアス
列に一時的に使用する別名。エイリアスは、列名を指定または変更するために、SQL プロシジャの SELECT 句にオプションで指定することもできます。エイリアスの長さは、1 ワードです。

列の演算式
評価結果として単一のデータ値を導き出す演算子とオペランドの組み合わせ。結果のデータ値は、文字値の場合も数値の場合もあります。

キーワード

1
1 つの値のサブクエリ 96

A
ALL キーワード 341
  セット演算子 145
ALTER TABLE ステートメント 231
ANSI 規格
  SQL プロシジャ 383

B
BETWEEN-AND 演算子
  行の取得 51
BETWEEN 条件 312
BTRIM 関数 312

C
CALCULATED 313
CASE-OPERAND フォーム
  条件付き値の割り当て 34
CASE 式 314
  条件付き値の割り当て 32
COALESCE 関数 315
  結合 91
  欠損値の置換 35
column-definition 構成要素 316
column-modifier 構成要素 317
column-name 構成要素 319
COMMIT ステートメント 385
CONNECTION TO 構成要素 320
CONNECT ステートメント 235
CONSTDATETIME オプション 222
CONTAINS 条件 320, 384
CORRESPONDING キーワード 341
COUNT(*) 関数 358
CREATE INDEX ステートメント 127, 236
CREATE TABLE ステートメント 237
CREATE VIEW ステートメント 242

D
DATA ステップ
  SQL プロシジャとの比較 5
  マッチマージ 92
DATA ステップビュー
  SQL プロシジャ 215
DATETIME 関数
  参照の置換 147
DATE 関数
  参照の置換 147
DBMS
  LIBNAME ステートメントを使用した接続 169
  PUT 関数と SAS 出力形式の配置 147
  SAS/ACCESS を使用したアクセス 168
  パススルー機能を使用した接続 172
DBMS クエリ 320
DBMS 接続
  DBMS ステートメントの送信 248
  SQL プロシジャ 235
  終了 247
  ビューでの格納 244
DBMS テーブル 4
  PROC SQL ビュー 171
  クエリ 170
DELETE ステートメント 245
DESCRIBE VIEW ステートメント 153
DESCRIBE ステートメント 246
DICTIONARY.COLUMNS 155
DICTIONARY.TABLES 154
DICTIONARY テーブル 151
  使用 154
  情報の取得 153
  パフォーマンス 156
  ビュー 151
  レポート 279
DISCONNECT ステートメント 247
DOUBLE オプション 222
DQUOTE=オプション 222
DROP ステートメント 247

E
ERRORSTOP オプション 223

EXCEPT 演算子 345
  クエリの組み合わせ 103, 105
EXECUTE ステートメント 248
EXEC オプション 223
EXISTS 条件 99, 321
EXITCODE option 223

F
FEEDBACK オプション 223
  SELECT*ステートメントの展開 142
FLOW オプション 224
FROM 句 20, 259

G
GROUP BY 句 21, 261

H
HAVING 句 21, 262
  グループ化されたデータのフィルタリング 69
  グループ化されたデータのフィルタリング, WHERE 句 70
  集計関数 70

I
in-database プロシジャ
  SQL の生成 370
INNER JOIN キーワード 78
INOBS=オプション 224
  行の処理の制限 141
INSERT ステートメント 249
INTERSECT 演算子 347
  クエリの組み合わせ 103, 106
INTO 句 254
IN 演算子
  行の取得 50
  複数の値のサブクエリ 97
IN 条件 321
IPASSTHRU オプション 224
IPONEATTEMPT オプション 224
IS MISSING 演算子
  行の取得 50
IS NOT MISSING 演算子
  内部結合 81
IS 条件 322

J
joined-table 構成要素 323

L
LIBNAME エンジン
  DBMS テーブルのクエリ 170
LIBNAME ステートメント
  DBMS への接続 169
  ビューに埋め込む 244
LIKE 演算子
  行の取得 52
LIKE 条件 337
  大文字小文字混在文字列の検索 339
  検索パターン 338
  リテラルの検索 338
LOOPS=オプション 224
  反復の制限 141
LOWER 関数 339

M
MEAN 関数
  WHERE 句 57
  データの要約 57

N
NOEXEC オプション
  構文チェック 141
NOT IN 演算子
  複数の値のサブクエリ 97
null 値 5
  内部結合 79
NUMBER オプション 225

O
ODS (Output Delivery System)
  SQL プロシジャ 175
ODS 出力先 175
ORDER BY 句 21, 263, 384
  クエリパフォーマンス 145
  指定しない 145
OUTER UNION セット演算子 341
OUTER UNION 演算子
  クエリ結果の連結 107
  クエリの組み合わせ 103
OUTOBS=オプション 226
  行の処理の制限 141

P
PRINT オプション 226
PROC SQLL テーブル
  列の選択 252
PROC SQL ステートメント 220
PROC SQL テーブル 214
  3 つのテーブルを結合する 289
  値の挿入 250

一貫性制約 235, 242
一貫性制約の変更 231
エイリアス 260, 324
同じテーブル同士の結合 323, 325
行なし 241
行のカウント 358
行の削除 245, 246
行の選択 252
行の追加 249
行の並べ替え 263
クエリ結果から作成する 269
クエリ式から作成する 242
組み合わせる 276
結合 273, 297, 323
更新 265, 266, 271
再帰テーブル参照 242
削除 247
作成 237, 267
ソーステーブル 259
データの取得 339
データの挿入 267
テーブル式 340
テーブルの定義 246
複数のテーブルを結合する 334
列名の変更 235
列のインデックス 235
列の初期値 234
列の属性の変更 235
列の変更 231
PROC SQL ビュー
関連項目: ビュー
DBMS 接続情報の保存 244
LIBNAME ステートメントの埋め込み 244
SQL プロシジャ 215
行の削除 245, 246
行の選択 252
行の挿入 251
行の追加 249
クエリ結果から作成する 286
クエリ式から作成する 242
更新 174, 244
削除 247
取得データの並べ替え 243
ソースビュー 259
テーブルの更新 266
ビューの定義 246, 384
ライブラリ参照名と保存ビュー 244
列の値の更新 265
列の選択 242, 252
PROMPT オプション 226
PUT 関数
DBMS 内への配置 147
SAS_PUT 関数へのマッピング 373
エンジンの種類に基づく減少 374
減少 146
最適化 146
出力形式値の数に基づく減少 377
テーブルのオブザベーション数に基づく減少 376

Q

query-expression 構成要素 339

R

REDUCEPUT=オプション 226
REDUCEPUTOBS=オプション 227
REDUCEPUTOBS オプション 146
REDUCEPUTVALUES=オプション 227
REDUCEPUTVALUES オプション 146
REDUCEPUT オプション 146
REMERGE オプション 228
REPORT プロシジャ
SQL 出力のフォーマット 166
RESET ステートメント 251
SQL プロシジャオプションのリセット 143
ROLLBACK ステートメント 386

S

SAS System の情報 151
SAS_PUT()関数
PUT 関数のマッピング 373
SAS/ACCESS
DBMS へのアクセス 168
LIBNAME ステートメント 169
SAS/ACCESS ビュー
SQL プロシジャ 215
更新 174
Sashelp ビュー 151
情報の取得 153
SAS データビュー
DICTIONARY テーブル 151
SQL プロシジャ 215
SAS データファイル
参照項目: テーブル
SELECT *ステートメント
FEEDBACK オプションを使用した拡張 142
SELECT 句 20, 252
SELECT ステートメント 20, 252
関連項目: WHERE 句
FROM 句 20
GROUP BY 句 21
HAVING 句 21
ORDER BY 句 21
SELECT 句 20
句の並べ替え 22
SET 句

行の挿入 116
set メンバ 321
SORTMSG オプション 228
SORTSEQ=オプション 229
SOUNDS-LIKE 演算子 295
SQL 3
ソースデータを in-database 処理するように生成 370
SQL (Structured Query Language: *参照項目:* SQL
SQL procedure
syntax 217
sql-expression 構成要素 347
USER 348
演算子と評価の順序 349
関数 348
切り捨て文字列の比較演算子 350
クエリ式 351
サブクエリと効率 354
相関するサブクエリ 353
SQL, 埋め込み 386
SQLCONSTDATETIME システムオプション 369
SQLEXITCODE マクロ変数 163
SQLGENERATION=システムオプション 370
SQLIPONEATTEMPT システムオプション 373
SQLMAPPUTTO=システムオプション 373
SQLOBS マクロ変数 163
SQLOOPS マクロ変数 141, 164
SQLRC マクロ変数 164
SQLREDUCEPUT=システムオプション 146, 374
SQLREDUCEPUTOBS=システムオプション 146, 376
SQLREDUCEPUTVALUES=システムオプション 146, 377
SQLREMERGE システムオプション 378
SQLUNDOPOLICY 379
SQLUNDOPOLICY=システムオプション 164, 379
SQLXMSG マクロ変数 165
SQLXRC マクロ変数 165
SQL 構成要素 311
BETWEEN 条件 312
BTRIM 関数 313
CALCULATED 313
CASE 式 314
COALESCE 関数 315
column-definition 316
column-modifier 317
column-name 319
CONNECTION TO 320
CONTAINS 条件 320
EXISTS 条件 321
IN 条件 321
IS 条件 322
joined-table 323
LIKE 条件 337
LOWER 関数 339
query-expression 339
sql-expression 347
SUBSTRING 関数 355
summary-function 356
table-expression 364
UPPER 関数 365
SQL プロシジャ 3, 214
関連項目: SQL 構成要素
2 つのテーブルを組み合わせる 276
2 つのテーブルを結合する 273, 297
3 つのテーブルを結合する 289
ANSI 規格と 383
DATA ステップとの比較 5
DICTIONARY テーブル 151
DICTIONARY テーブルからレポートを作成する 279
ODS 175
PROC SQL テーブル 214
PROC SQL テーブルの更新 271
値の取得 295
インデックス 237
インラインビューのクエリ 293
オプションのリセット 143, 251
外部結合 281
クエリ結果からテーブルを作成する 269
クエリ結果からビューを作成する 286
クエリの作成 140
クエリのデバッグ 140
ケース行とコントロール行の照合 306
欠損値 309, 322
コーディング規則 216
構文チェック 141
個々のステートメントを使用するタイミング 142
サポート対象の関数 385
三値論理 386
サンプルテーブル 7
識別子と命名規則 386
出力のフォーマット 166
照合順序 384
タスクテーブル 219, 220
他のプロシジャでのテーブルの使用 208
直交式 384
データセットオプション 157
データの種類と日付 316
テーブルを作成し、データを挿入する 267
統計関数 385

ビュー 215
マクロ機能 158
マクロ変数の作成 158
マクロ変数の設定 163
マクロを使用して欠損値をカウントする 309
元に戻すポリシ 379
ユーザー権限 386
用語 4, 214
予約語 383
累積時間 142
列の値の組み合わせ 300
列の修飾子 384
SQL プロシジャのパススルー機能 172
DBMS への接続 172
リターンコード 173
例 173
STIMER オプション 229
SQL プロシジャを使用するタイミング 142
STOPONTRUNC オプション 229
SUBSTRING 関数 355
SUBSTRING を戻す 355
summary-function 構成要素 356
引数の数に基づく統計量 358
行のカウント 358
データの再マージ 359
データの要約 357
SUM 関数
データの要約 58
SYS_SQLSETLIMIT マクロ変数 381

T

table-expression 構成要素 364
THREADS オプション 229
TIME 関数
参照の置換 147
TODAY 関数
参照の置換 147

U

UBUFSIZE=オプション 230
UNDO_POLICY=オプション 164, 230
UNION 演算子 344
クエリの組み合わせ 103, 104
UNIQUE キーワード 127, 236
UPDATE ステートメント 265
UPPER 関数 365
USER リテラル 348

V

VALIDATE ステートメント 266
構文チェック 141
VALUES 句
行の挿入 117

W

WARNRECURS オプション 231
WHERE 句 21, 260
MEAN 関数 57
行の条件付き取得 45
グループ化されたデータのフィルタリング, HAVING 句 70
欠損値 54
データの要約 57
WHERE 式
結合 145

あ

アスタリスク(*)表記 253
値のグループの存在 99
主キー 129
一時テーブル
インラインビュー 145
一貫性制約 129
PROC SQL テーブル 235, 242
参照 129
一般的な一貫性制約 129
インデックス 127
CREATE INDEX ステートメントを使用して作成する 127
SQL プロシジャ 237
UNIQUE キーワード 236
管理 237
クエリパフォーマンス 144
削除 128, 247
作成 127
作成のヒント 127
重複しない値 127
単一インデックス 237
複合 127
複合インデックス 237
変更列 235
列 236, 251
インラインビュー 135, 260, 384
一時テーブル 145
クエリ 293
埋め込み LIBNAME ステートメント 244
埋め込み SQL 386
エイリアス
計算列を参照する 31
テーブルのエイリアス 76
列のエイリアスを割り当てる 30
エラー
欠損値が原因 63
欠損値が原因のグループ化のエラー 67

更新エラー 122
演算子
値 347
切り捨て文字列の比較演算子 350
算術 384
セット演算子 340, 385
評価の順序 349
オブザベーション
関連項目: 行
SQL プロシジャ 214
オプションのリセット 143
重み付き平均 178

か
階層データ
テーブルでの展開 189
外部キー 129
外部結合 85, 281, 328, 384
一致しない行を含む 85, 86
完全外部結合 87
左外部結合 85
右外部結合 86
カウント
重複しない値 61
重複する行 187
すべての行 62
非欠損値 62
拡張 SELECT *ステートメント 142
関係 3
関係理論 3
関数
FCMP プロシジャ 385
sql-expression 348
SQL プロシジャ 385
完全外部結合 87
行 4
関連項目: 行の取得
1 回目または 2 回目のクエリからの作成 108
SET 句を使用して挿入する 116
SQL プロシジャ 214
VALUES 句を使用して行を挿入する 117
一致しない 85, 86
一致する 90
カウント 62, 358
行の処理の制限 141
クエリを使用して挿入する 118
結合 324
異なる式で行を更新する 121
削除 123, 245, 246
サブクエリを使用して戻す 321
重複 145
重複のカウント 187
重複の削除 25
すべて選択 87
すべての組み合わせを含む 88
すべて含む 89
選択 252, 312
挿入 116, 251
テーブルまたはビューへの追加 249
同一式ですべての行を更新する 120
並べ替え 263
複数の行から 1 行にデータを組み合わせる 59
行の取得 44
BETWEEN-AND 演算子 51
IN 演算子 50
IS MISSING 演算子 50
LIKE 演算子 52
切り捨て文字列の比較演算子 53
欠損値を含む WHERE 句 54
欠損値を含む列の特定 50
条件を満たす行 44
その他の条件演算子 49
単一の WHERE 句 45
比較に基づく 45
複数の条件を満たす 47
行の挿入 116
SET 句 116
VALUES 句 117
クエリ 118
切り捨て文字列の比較演算子 53, 350
クエリ 5
2 つのクエリの実行時間の比較 142
DBMS クエリ 320
DBMS テーブル 170
インデックス 144
インラインビュー 135
インラインビューと一時テーブル 145
インラインビューのクエリ 293
行の処理の制限 141
行の挿入 118
結果からテーブルを作成する 269
結果からビューを作成する 286
検証 71
作成 140
サブクエリ 96
重複する行とパフォーマンス 145
出力へのテキストの追加 27
ステップへの分割 145
セット演算子の ALL キーワード 145
セット演算子を使用した組み合わせ 103
データの再マージ 378
デバッグ 140
パフォーマンスの向上 144
反復の制限 141
クエリ結果 5
重複する行の削除 25
テーブルの作成 113

マクロ変数の作成 159
連結 107
クエリ式 340
ALL キーワード 341
CORRESPONDING キーワード 341
EXCEPT 345
INTERSECT 347
OUTER UNION 341
PROC SQL テーブルの作成 242
PROC SQL ビューの作成 243
UNION 344
構文の検証 266
サブクエリ 351
セット演算子 340
クエリの検証 71
クエリのデバッグ 140
グループ化されたデータのフィルタリング 69
HAVING 句と WHERE 句 70
HAVING 句と集計関数の併用 70
単一の HAVING 句の使用 69
クロス結合 88, 330
ケースコントロールスタディ 306
計算列 29
SQL 314
エイリアスによる参照 31
並べ替え 40
列のエイリアスを割り当てる 30
結合 74, 324
2 つのテーブルを結合する 273
3 つのテーブルを結合する 289
COALESCE 関数 91
WHERE 式 145
同じテーブル同士の結合 325
外部結合 85, 281, 328, 384
クロス結合 88, 330
結果のサイズの縮小 145
再帰結合 325
サブクエリとの比較 145, 337
サブクエリを使用し組み合わせる 101
自然結合 90, 332
種類 324
使用が必要な場合 102
テーブルの制限 324
デカルト積 75
等結合 324
特殊結合 88
内部結合 75, 325
複数のテーブルを結合する 334
マッチマージの比較 92
戻り行 324
和結合 89, 331
欠損値 5
SQL プロシジャ 309, 322
WHERE 句 54
エラーの検索 63
重ね合わせ 182
行の取得 50
グループ化のエラーの検索 67
データのグループ化 67
データの要約 62
マクロを使用したカウント 309
列の置換 35
列の並べ替え 44
欠損値の重ね合わせ 182
合計
表示 58
構文チェック 141

さ

再帰結合 84, 325
サブクエリ 96, 351
1 つの値 96
値のグループの存在のテスト 99
行を戻す 321
結合機能を使用した組み合わせ 101
結合との比較 145, 337
効率 354
使用が必要な場合 102
相関 353
相関するサブクエリ 98
複数の値 97
複数のネストされた水準 100
三値論理 386
算術演算子 384
参照一貫性制約 129
サンプルテーブル 7
自己結合 84
自然結合 90, 332
実行時間 142
自動マクロ変数 158, 163
集計関数 56
HAVING 句 70
結果からマクロ変数を作成する 159
重複しない値 61
使用 56
テーブル 56
重複しない値
カウント 61
集計関数 61
すべての行のカウント 62
非欠損値のカウント 62
列 25
出力
REPORT プロシジャを使用したフォーマット 166
テキストの追加 27
出力オブジェクト 175
出力形式
DBMS 内への配置 147
列 317

列の出力形式の変更 125
小計
百分率の計算 185
条件演算子
行の取得 49
照合順序
代替 384
セット演算子 340, 385
ALL キーワード 145
クエリの組み合わせ 103
ソースデータ
in-database 処理する SQL の生成 370
ソートシーケンス 43
相関するサブクエリ 98, 353

た

単一インデックス 237
直交式 384
データセットオプション
SQL プロシジャ 157
テーブルの作成 115
データのグループ化 64
1 列 64
グループ化されたデータのフィルタリング 69
グループ化と並べ替え 66
欠損値 67
欠損値が原因のエラーの検索 67
複数列 65
要約しない 64
データの再マージ 378
SQL プロシジャ 359
データのサブセット化 260, 263
データの並べ替え 37
グループ化と並べ替え 66
計算列 40
欠損値を含む列 44
選択されていない列 42
複数列 38
列 38
列の位置 41
データのマージ
SQL プロシジャ 359
データの要約 56, 357
SQL プロシジャ 357
WHERE 句 57
欠損値 62
合計の表示 58
集計関数 56
集計関数, 使用 56
重複しない値の集計関数 61
複数の行から 1 行にデータを組み合わせる 59
複数列 192
要約統計量の再マージ 59
データファイル
参照項目: テーブル
テーブル 4
関連項目: PROC SQL テーブル
DBMS テーブル 4
SAS の SQL テーブル 128
値の更新 120
一時テーブルとインラインビュー 145
一貫性制約 129
同じテーブル同士の結合 84
階層データの展開 189
行の挿入 116
行を使用せずに作成する 112
クエリ結果から作成する 113
更新エラー 122
構造 27
異なる式で行を更新する 121
コピー 115
削除 128
作成 112
作成, 既存のテーブルと同じテーブル 115
作成時に、ORDER BY 句を指定しない 145
サンプルテーブル 7
重複する行のカウント 187
条件付き更新 200
すべての列の選択 22
他のプロシジャでの SQL テーブル 208
データセットオプションを使用して作成する 115
デカルト積 75
同一式ですべての行を更新する 120
特定の列の選択 23
比較 180
別のテーブルの値を使用して更新する 203
マクロを使用して作成する 162
列の選択 22
列の定義から作成する 112
テーブル式 340
テーブルのエイリアス 260, 324
内部結合 76
列名の省略 76
テーブルの更新
値の更新 120
エラー 122
条件付き 200
別のテーブルの値を使用する 203
テーブルの定義 246
デカルト積 75, 324, 326
クロス結合 88
テキスト
出力への追加 27
統計関数 385

統計量
  引数の数に基づく 358
統計量の要約 56, 356
等結合 324

な

内部結合 75, 325
  INNER JOIN キーワードを使用し、作成する 78
  null 値 79
  再帰結合 84
  自己結合 84
  出力の順序 77
  テーブル内のリレーションシップの表示 84
  テーブルのエイリアス 76
  比較演算子 78
  複数のテーブルからのデータ 83
  複数列の結合 81
並べ替え順序 37, 39
  カスタマイズ 197
平均, 重み付き 178
入力形式
  列 317
  列の入力形式の変更 125
ネストするサブクエリ 100

は

パターン検索 306, 337, 338
パターンマッチ 306, 337
幅
  列の幅の変更 125
パフォーマンス
  クエリ 144
反復
  限定 141
比較演算子
  行の取得 45
  切り捨て文字列 53
  内部結合 78
左外部結合 85
百分率
  小計内の計算 185
ビュー 5, 131
  DBMS テーブル 171
  DICTIONARY テーブル 151
  PROC SQL ビューと SAS/ACCESS ビューの更新 174
  Sashelp ビュー 151, 153
  SAS データビュー 151
  SAS の PROC SQL ビュー 137
  SQL プロシジャ 215
  インライン 135, 260, 293, 384
  更新 133
  削除 135
  作成 132
  作成時に、ORDER BY 句を指定しない 145
  使用のヒント 136
  説明 132
  ライブラリ参照名を埋め込む 133
ビューで取得したデータの並べ替え 243
ビューの更新 133, 174
ビューの定義 246
  ORDER BY 句 384
ファイル
  参照項目: テーブル
フィールド
  参照項目: 列
複合インデックス 127, 237
複数の値のサブクエリ 97
複数列の結合 81
プロシジャステートメントを使用するタイミング 142
変数
  関連項目: 列
  SQL プロシジャ 214
ホスト変数の参照 158

ま

マージ
  再マージの無効化 148
  要約統計量の再マージ 59
マクロ
  欠損値のカウント 309
  テーブル作成の定義 162
マクロ機能
  SQL プロシジャ 158
マクロ変数 158
  SQL プロシジャで作成する 158
  SQL プロシジャによる設定 163
  SYS_SQLSETLIMIT 381
  値の連結 161
  クエリ結果から作成する 159
  作成と使用 205
  集計関数の結果から作成する 159
  複数作成 160
マッチマージ 92
  値の位置が重要な場合 94
  一部の値が一致する場合 94
  結合との比較 92
  すべての値が一致する場合 92
右外部結合 86
文字列
  大文字への変換 365
  小文字への変換 339
  トリミング 313
文字列の比較演算子
  切り捨て 350

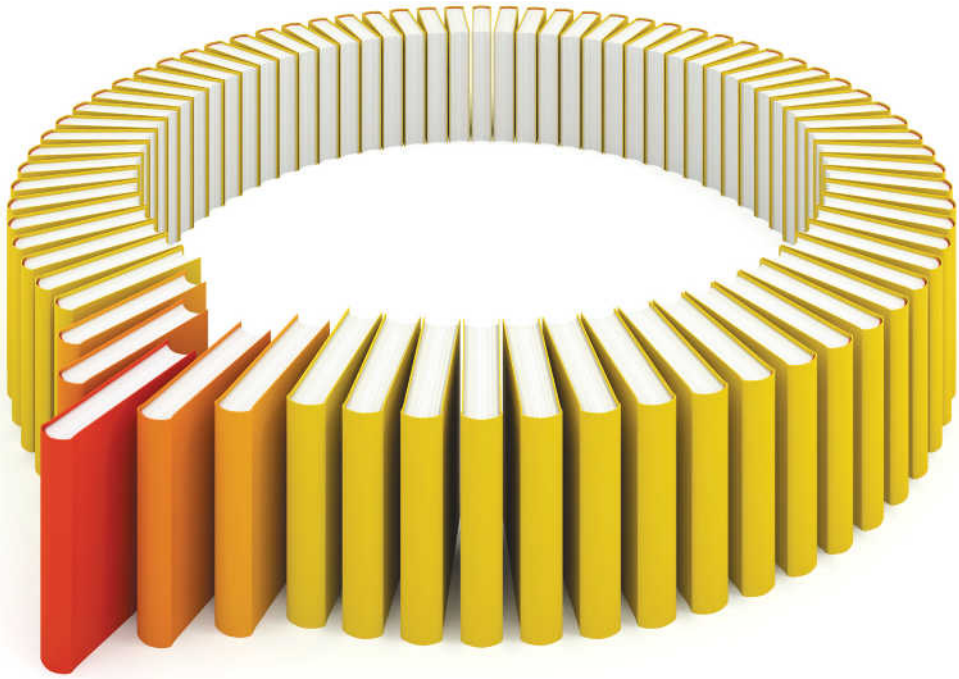
元に戻すポリシ
SQL プロシジャ 379

や

和結合 89, 331
ユーザー定義のマクロ変数 158
用語 4
要約関数
関連項目: 集計関数
データの再マージの無効化 148
要約統計量
再マージ 59
要約統計量の再マージ 59
再マージの無効化 148
要約レポート
作成 194

ら

ライブラリ参照名
ビューに埋め込む 133
保存ビュー 244
ラベル
列 317
列のラベルの変更 125
リターンコード
パススルー機能 173
レコード
参照項目: 行
列 4
1 つ列を基準にグループ化する 64
DICTIONARY.COLUMNS 155
SQL プロシジャ 214
値の格納 254
値の組み合わせ 300
値の計算 29
値の更新 265
値の挿入 250
値を戻す 315
インデックス 235, 236, 251
エイリアス 148
計算 313, 314
欠損値の置換 35
削除 126
作成 27
修飾子 384
重複しない値 25
出力形式の変更 125
条件付き値の割り当て 32
すべて選択 22
選択 22, 252, 319
選択していない列を基準に並べ替える 42
追加 123
特定の列の検索 155
特定の列の選択 23
長さ 317
名前の変更 125, 235, 252
並べ替え 38
並べ替え, 欠損値 44
入力形式の変更 125
幅の変更 125
複数の列を基準にグループ化する 65
複数の列を基準に並べ替える 38
複数列の結合 81
複数列のデータの要約 192
変更 123, 125, 231
ラベルの変更 125
リスト, 属性 27
列の位置を基準に並べ替える 41
列エイリアス
ANSI 規格の拡張機能 148
指定 148
列名の修飾 76
列名の省略 76
列名の変更 125
列のエイリアス 30
計算列に割り当てる 30
計算列を参照する 31
列の修飾子 384
列の属性 235, 317
指定 36
リスト 27
列の定義
テーブルの作成 112
列の名前
修飾 76
省略 76
列のヘッダー
表示しない 28
レポート
DICTONARY テーブル 279
要約レポートの作成 194
連結
クエリ結果 107
マクロ変数の値 161
ログ
SQL の定義の表示 246
論理演算子
行の取得 47



Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

support.sas.com/bookstore
for additional books and resources.