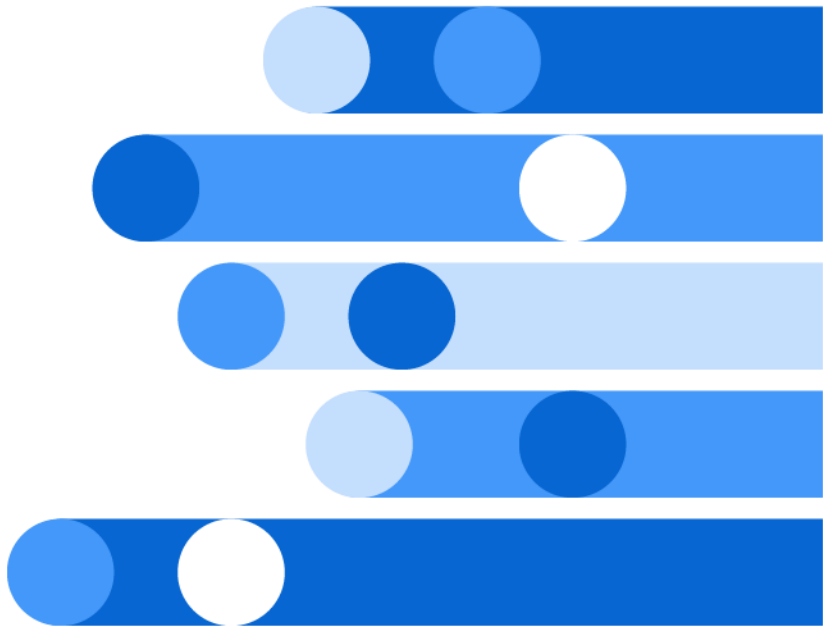
SAS®プログラマガイド: エッセンシャル

2024.01 - 2026.08*



* このドキュメントは、ソフトウェアの追加バージョンに適用される場合があります。このドキュメントを SAS Help Center で開き、バナーのバージョンをクリックすると、使用できるすべてのバージョンが表示されます。

SAS® ドキュメント
2026 年 9 月 14 日

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2020. *SAS® プログラマガイド: エッセンシャル*. Cary, NC: SAS Institute Inc.

SAS®プログラマガイド: エッセンシャル

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

August 2026

v_001-P1:lepg

目次

5 部 出力の作成 701

6 部 ファイルの管理 731

本書について

対象ユーザー

このドキュメントは、Base SAS プログラミング言語のすべてのユーザーに 適しています。新しいユーザーは例を実行して、各機能を簡単にデモンストレーションできます。経験豊富なユーザーは、詳細については関連する概念のトピックを参照してください。

構文とその他のリファレンス

このドキュメントの範囲には、ほとんどの SAS エンジンに共通の機能が含まれます。エンジンに固有の機能については、エンジンのドキュメントで説明されています。たとえば、一貫性制約やカタログなどの機能については、*SAS V9 LIBNAME エンジン: リファレンス*に記載されています。エンジンのドキュメントへのリンクについては、"SAS エンジンの概要" (272 ページ)を参照してください。

このドキュメントでは、Base SAS 構文の一般的な概念について説明します。次のリファレンス文書も参照してください。

- *Base SAS プロシジャガイド*
- *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 DATA ステップステートメント: リファレンス*
- *SAS グローバルステートメント: リファレンス*
- *SAS システムオプション: リファレンス*

1 部 はじめに

1

SAS 言語

SAS 言語プログラミングコンポーネント

SAS は、データへのアクセス、準備、分析、共有を可能にする高度な分析プログラミング環境です。SAS を使用すると、ほぼすべてのデータソースにアクセスし、データをマイニングして洞察を得ることができ、さまざまな形式で配布できる高品質のグラフィックやレポートを作成できます。結果を他のシステムにエクスポートすることもできます。SAS を使用すると、ローカルな SAS クライアント内、ターゲットデータベース内、ストリーミングサービス経由、高速メモリ内など、ニーズに最も適した場所でデータを分析できます。

これらは、SAS を使用するための重要な概念です。

SAS ファイル

SAS を使用する場合は、SAS によって作成および維持されるファイルだけでなく、SAS に関連しない他のシステムによって作成および維持されるファイルも使用します。SAS Compute Server によって作成および維持されるファイルは、*SAS ファイル*と呼ばれます。最も一般的に使用される SAS ファイルタイプは次のとおりです。

SAS データセット
: 1 つの単位として保存される変数とオブザベーションのセット。SAS データセットは、列と行で構成されるリレーショナルテーブルに似ています。ただし、SAS データセットには、データに加えて、変数とオブザベーションに関する説明情報も保存されます。詳細については、13 章, “SAS データセット” (299 ページ)を参照してください。

SAS ビュー
: カスタマイズされた動的なデータ表現を提供するために、他のファイルからデータ値を抽出する仮想データセット。ほとんどの場合、SAS データセットで使用できる機能は SAS ビューでも使用できます。詳細については、16 章, “SAS ビュー” (399 ページ)を参照してください。

SAS カタログ
: SAS ジョブで使用されるさまざまな種類の情報を保存するファイル。例には、データ値の読み取りと出力の指示、または SAS ユーザーインターフェイスのファンクションキー設定が含まれます。

  詳細については、“Catalogs” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。

SAS ストアドコンパイルプログラム
: 繰り返し使用するために作成および保存するコンパイル済みコードを含む SAS ファイルの一種。詳細については、“Stored, Compiled DATA Step Programs” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。

通常、SAS ライブラリを割り当てることによって SAS ファイルを操作します。詳細については、11 章, “SAS ライブラリ” (227 ページ)を参照してください。

CAS 構造

CAS で SAS を使用してデータを操作するには、CAS 構造に精通している必要があります。

CAS セッション
: CAS サーバー上の作業スペース。CAS で作業する前に、SAS Compute Server から CAS サーバー上で CAS セッションを確立する必要があります。

caslib
: CAS サーバーセッションでデータにアクセス、整理、および保護するためのメカニズム。各サーバーユーザーは個人用 caslib を持っています。CAS エンジンのユーザーは、CAS で処理するためにデータをこの個人用 caslib にロードできます。または、管理者が追加したグローバルスコープの caslib を使用することもできます。グローバルスコープの caslib は、複数のセッションで使用される一般的に使用されるデータソースへのアクセスを提供します。セッションが現在使用している caslib は、アクティブな caslib と呼ばれます。アクティブな caslib は、サーバー側アクセス用に選択された場所です。

CAS テーブル
: CAS にロードされた CAS セッションのインメモリデータ。CAS にロードすると、SAS データセットは CAS テーブルになります。

SAS データコネクタ
: caslib のデータアクセスの強化。Oracle Database、IBM DS2 などの一般的なデータソースのデータにアクセスするには、データコネクタを使用してデータソースと CAS 間のデータアクセスを有効にします。

外部ファイル

データの読み書きに使用するが、SAS にとって未知の構造にあるデータファイルは、*外部ファイル*と呼ばれます。外部ファイルには、CSV ファイル、Microsoft Excel ブックファイルなどが含まれます。外部ファイルには、分析するデータ、SAS プロシジャ出力、および SAS プログラミングステートメントを保存できます。

SAS を使用して SAS Compute Server 上の外部ファイルを操作する方法については、“外部ファイルの定義” (356 ページ)を参照してください。

CAS では、サポートされている外部ファイルタイプ(.CSV、イメージ、Microsoft Word 文書)にパスベースの caslib からアクセスできます。指定された CAS アクションは、出力として外部ファイルを生成できます。たとえば、CAS はインメモリテーブルを CSV、Parquet などの一般的なファイル形式で保存できます。詳細については、“caslib” (*SAS Cloud Analytic Servcies: 基本*)を参照してください。

データベース管理システムファイル

SAS は、多くの一般的なデータベース管理システム(DBMS)からデータを読み書きできます。

DBMS ファイルにアクセスするには、SAS/ACCESS ソフトウェアを DBMS 用に構成する必要があります。詳細については、15 章, “データベースと PC ファイル” (377 ページ)を参照してください。

CAS では、SAS は CAS にロードされたサードパーティ DBMS からのデータを操作できます。CAS から DBMS ファイルへのサーバー側アクセスを有効にするには、DBMS 用の SAS/ACCESS ソフトウェアを構成し、構成された動作環境を構成し、SAS データコネクタを使用する必要があります。詳細については、“SAS データコネクタの操作” (*SAS Cloud Analytic Services: ユーザーガイド*)を参照してください。

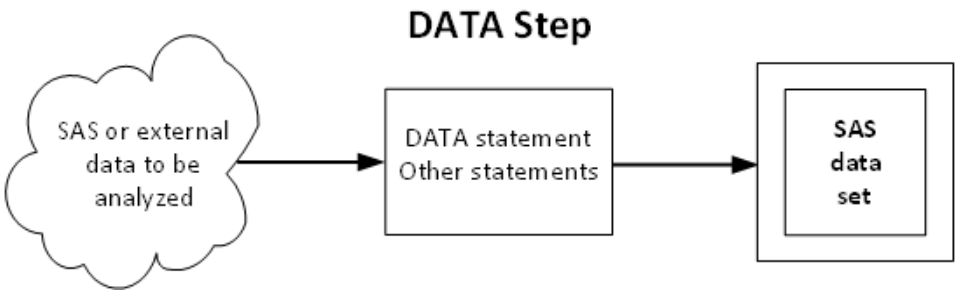
SAS 言語要素

SAS 言語は、他の多くのプログラミング言語と同じように、ステートメント、式、オプション、出力形式、関数、およびその他の SAS 言語要素で構成されています。SAS をユニークなものにしているのは、ほとんどの SAS 言語要素が、次の 2 つのグループの SAS ステートメントのいずれかで使用されるためです。

- DATA ステップ
- PROC ステップ

DATA ステップは、SAS データセットを作成または操作する一連のステートメントで構成されます。このステップは DATA ステートメントで始まるため、DATA ステップと呼ばれます。次の図は、DATA ステップが最も単純な形式で SAS データセットを作成する方法を示しています。

図 1.1　SAS DATA ステップの概要図



SAS ステートメントでデータを定義することで、SAS データセットを作成できます。または、入力ソースからデータを読み取ることで SAS データセットを作成できます。

CAS で DATA ステップを使用すると、DATA ステップはインメモリ CAS テーブルを操作します。したがって、DATA ステップを使用してデータを定義するのではなく、DATA ステップを使用して既存の SAS データセットを CAS サーバーにロードし、

CAS テーブルを操作します。(CASUTIL プロシジャを使用して、CAS サーバーにデータをロードすることもできます。)ロードプロセス中に、SAS データセットは CAS テーブルに変換されます。

さらに、SAS データセットと CAS テーブルの両方で DATA ステップを使用して次のタスクを実行できます。

- 新しい変数の値の計算
- データ内のエラーをチェックして修正する
- 既存のデータセットのサブセット化、マージ、更新により、新しい SAS データセットまたは CAS テーブルを作成する

DATA ステップでは、さまざまなステートメントがサポートされています。

CAS の DATA ステップは、SAS データセットで使用できる CAS テーブルの機能とほぼ同じ機能をサポートします。ただし、制限があります。DATA ステップが CAS でどのように実行されるかについては、*SAS Cloud Analytic Services: DATA ステッププログラミング*を参照してください。

PROC ステップは、通常は入力として SAS データセットを使用してプロシジャを呼び出して実行する SAS ステートメントのグループで構成されます。すべての PROC ステップは、名前が PROC という単語で始まり、プロシジャの目的を説明するキーワードが続くステートメントで始まります。PROC ステップを使用すると、データセット内のデータの並べ替え(PROC SORT)やデータセットの出力(PROC PRINT)などの単純なタスクを実行できます。さらに、SAS データセット内のデータを分析したり、フォーマットされたレポートやその他の結果を作成したり、SAS ファイルを管理したりすることができます。

プロシジャには、必須ステートメントとオプションのステートメントのセットが定義されています。

レポートを作成するには、CAS データを SAS データセットに転送して戻す必要があります。ただし、CAS エンジンで PROC SGPLOT を使用すると、CAS 内の大きなデータを要約し、その要約から統計グラフィックスを作成することができます。これを行う方法の例については、*SAS Viya プラットフォームプログラミング: 入門ガイド*の"単一の数値変数の要約"を参照してください。

グローバルステートメント

グローバルステートメントは、通常、DATA ステップと PROC ステップの外側で指定され、DATA ステップと PROC ステップの両方に情報を提供する SAS 言語要素です。グローバルステートメントの主な用途は、クライアントセッションのデータアクセスを定義することです。二次的な用途は、SAS またはそのサブシステムの 1 つのデフォルトの動作を変更するシステムオプションを設定することです。

SAS データセットでサポートされるグローバルステートメントの詳細については、*SAS グローバルステートメント: リファレンス*を参照してください。

CAS テーブルでサポートされるグローバルステートメントの詳細については、*SAS Cloud Analytic Services: ユーザーガイド*を参照してください。

SAS マクロ機能

マクロ機能は、SAS プログラムを拡張およびカスタマイズし、一般的なタスクを実行するために入力する必要があるコードの量を削減するための強力なプログラミングツールです。マクロを使用すると、SAS ステートメントとコマンドを自動的に生成したり、SAS ログにメッセージを書き込んだり、入力を受け入れたり、マクロ変数の値を作成および変更したりできます。マクロ機能を使用して実行できることの例については、*“マクロを使用した SAS コードの生成” (SAS マクロ言語: リファレンス)*および*“マクロ変数を使用した文字列の置換” (SAS マクロ言語: リファレンス)*を参照してください。

SAS マクロ変数の参照は、データを CAS にロードする CAS エンジンプログラムで許可されます。ただし、CAS にはマクロ機能がありません。マクロ変数は、CAS のアクションに送信される前に、SAS によって処理および解決されます。

SAS エンジン

SAS は、XLSX、XML、JSON などの形式で外部ファイルを読み取るためのライブラリエンジンを提供します。SAS/ACCESS エンジンにより、SAS を使用したデータベース管理システムへのアクセスが可能になります。

CAS エンジンにより、CAS テーブルへのアクセスが可能になります。SAS から使用できるエンジンの概要については、12 章, “SAS エンジン” (269 ページ)を参照してください。

SAS データセットの場合、出荷されるデフォルトの SAS エンジンは BASE です。SAS 9 および SAS Viya プラットフォームでは、BASE エンジンは V9 エンジンのエイリアスです。

追加言語

SAS では次の追加言語を使用できます。

表 1.1　SAS で使用できる追加言語

言語	ユーザーインターフェイス	説明
SAS SQL	PROC SQL	ANSI SQL:1992 コア規格の SAS 実装。SAS SQL の利点は、SAS 言語と完全に統合されていることです。SAS データ型を使用し、SAS システムオプションで管理でき、SAS データセットオプション、出力形式、および入力形式を完全にサポートします。すべての SAS エ

		ンジンで使用でき、SAS/ACCESS ソフトウェアと使用した場合、外部 DBMS への明示的な SQL パススルーをサポートします。 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。
SAS FedSQL	PROC FEDSQL	ANSI SQL:1999 コア規格の SAS 実装。FedSQL は、ANSI SQL 1999 データ型とコア機能および拡張機能を使用します。SAS FedSQL の利点は、SAS SQL よりも多くのデータ型をサポートし、ベンダーニュートラルな SQL ダイアレクトであることです。SAS/ACCESS ソフトウェアと併用すると、データソースに固有の SQL ダイアレクトでクエリをサブミットすることなく、さまざまなデータソースからデータにアクセスできます。FedSQL は SAS 言語と部分的に統合されています。特定の SAS エンジン、ほとんどの SAS 関数、およびほとんどの SAS 出力形式をサポートします。ほとんどの SAS システムオプションまたは SAS データセットオプションはサポートされていません。 FedSQL ステートメントはデフォルトでは SAS ライブラリで実行されますが、プロシジャオプションを指定することで CAS ライブラリで実行することもできます。 詳細については、*SAS FedSQL 言語リファレンス*を参照してください。
SAS DS2	PROC DS2	高度なデータ操作に適した SAS 独自のプログラミング言語。これは DATA ステップと交差しますが、追加のデータ型、ANSI SQL 型、プログラミング構造要素、ユーザー定義のメソッドとパッケージをサポートします。FedSQL と同様に、DS2 は部分的に SAS 言語と統合されています。特定の SAS エンジンとほとんどの SAS 関数をサポートします。SAS ストレージエンジンと併用すると、ほとんどの SAS 出力形式がサポートされます。ほとんどの SAS システムオプションおよび SAS データセットオプションはサポートされていません。 DS2 プログラムはデフォルトでは SAS ライブラリで実行されますが、プロシジャオプションを指定することで CAS ライブラリで実行することもできます。 詳細については、*SAS DS2 言語リファレンス*および *SAS DS2 プログラマガイド*を参照してください。

CASL	PROC CAS	SAS から CAS に CAS アクションをサブミットできるようにする SAS 独自の言語。CASL を使用すると、分析パイプラインの開発、CAS アクションの引数の作成、アクションから返された結果の操作、ユーザー定義のアクションと関数の作成が可能になります。詳細については、*SAS Cloud Analytic Services: CASL プログラマガイド*を参照してください。
Lua ステートメント	PROC LUA	SAS セッション内で Lua ステートメントを実行できるようになります。 詳細については、“LUA プロシジャ” (*Base SAS プロシジャガイド*)を参照してください。

プログラム入力

SAS クライアントプログラムでは、さまざまな入力データソースを使用できます。

SAS データセット
: SAS データセットまたは SAS ビューのいずれか。“SAS データセットの定義” (300 ページ)を参照してください。

生データ
: SAS データセットに読み取られていない未処理のデータ。次の 2 つのソースから生データを読み取ることができます。

  外部ファイル
  : フォーマットされたデータ(列に配置されたデータ)またはフリーフォーマットのデータ(列に配置されていないデータ)で構成されるレコードが含まれます。詳細については、“外部ファイルの定義” (356 ページ)を参照してください。

  インストリームデータ
  : ジョブストリーム内の生データ。詳細については、“生データの定義” (342 ページ)を参照してください。

SAS ライブラリ
: SAS ライブラリを使用すると、エンジンを使用して、ユニットとして参照される 1 つ以上のファイルにアクセスできます。ライブラリ割り当てで指定したエンジンに応じて、SAS データセット、選択した外部ファイル、サードパーティのデータベース管理システム(DBMS)、PC ファイルにアクセスできます。このエンジンは、ターゲットデータを SAS が認識できる形式に変換します。詳細については、“SAS ライブラリの定義” (228 ページ)を参照してください。

リモートデータ
: 外部ソースから入力データを読み取ることができます。SAS は、このデータを外部ファイルからのものであるかのように扱います。

アクセスに関する詳細については

Azure
: Microsoft Azure Data Lake Storage 内のデータにアクセスできるようにするアクセス方式を指定します。Azure アクセス方式は、SAS Viya プラットフォームでのみサポートされます。

SAS カタログ
: SAS カタログを外部ファイルとして参照できるようにするアクセス方式を指定します。

クリップボード
: ホストコンピューター上のクリップボードへのテキストデータの読み取りまたは書き込みを可能にするアクセス方式を指定します。

DATAURL
: DATAURL アクセス方式を使用してリモートファイルにアクセスできるようにするアクセス方式を指定します。

EMAIL
: SMTP (簡易メール転送プロトコル)電子メールインターフェイスを使用して、SAS からプログラムで電子メールを送信できるようにするアクセス方式を指定します。

FILESRV
: SAS Viya Files サービスを使用してユーザーコンテンツを保存および取得できるようにするアクセス方式を指定します。FILESRV アクセス方式は、SAS Viya プラットフォームでのみサポートされます。

FTP
: ファイル転送プロトコル(FTP)を使用して、FTP サーバーが実行されているネットワークに接続されているホストコンピューターからファイルの読み取りまたは書き込みを行うためのアクセス方式を指定します。

Hadoop
: 構成ファイルで場所が指定されている Hadoop 分散ファイルシステム (HDFS)上のファイルにアクセスできるようにするアクセス方式を指定します。

S3
: Amazon S3 ファイルにアクセスできるようにするアクセス方式を指定します。S3 アクセス方式は、SAS Viya プラットフォームでのみサポートされます。

TCP/IP ソケット
: TCP/IP ソケットの読み取りまたは書き込みを可能にするアクセス方式を指定します。

URL
: URL (Uniform Resource Locator)を使用して、URL サーバーが実行されているネットワークに接続されているホストコンピューターからファイルの読み取りおよび書き込みを行うためのアクセス方式を指定します。

WebDAV
: WebDAV プロトコルを使用して、WebDAV サーバーが実行されているネットワークに接続されているホストコンピューターからファイルの読み取りまたは書き込みを行うためのアクセス方式を指定します。

ZIP
zlib サービスを使用して ZIP ファイルにアクセスできるようにするアクセス方式を指定します。

DATA ステップ

DATA ステップの定義

- DATA ステップは入力データを処理します。
- DATA ステップでは、SAS データセット、生データ、SAS ライブラリ、リモートアクセス、割り当てステートメントからの入力を使用します。
- DATA ステップでは、値を計算し、処理する特定の入力レコードを選択し、条件付きロジックを使用できます。
- DATA ステップからの出力には、SAS データセットやレポートなど、いくつかのタイプがあります。データを SAS ログまたは外部データファイルに書き込むこともできます。

SAS DATA ステップと PROC ステップの処理

SAS 処理は、SAS 言語が入力データを読み取って変換し、要求された種類の出力を生成する方法です。

DATA ステップとプロシジャ(PROC)ステップは、SAS 言語の 2 つの重要なステップです。一般的に、DATA ステップはデータを作成および操作し、PROC ステップはデータを分析し、出力を生成し、SAS ファイルを管理します。これら 2 種類のステップは、単独または組み合わせて使用され、SAS プログラムの基礎を形成します。

- DATA ステップへの入力としてさまざまなタイプのデータを使用できます。
- DATA ステップには、データを処理するための命令を含む SAS ステートメントを含めます。
- SAS プログラムの各 DATA ステップがコンパイルまたは実行されると、SAS は処理メッセージとエラーメッセージを含むログを生成します。これらのメッセージは、SAS プログラムのデバッグに役立ちます。
- 通常、PROC ステップは SAS データセットの形式でデータを分析および処理しますが、SAS データセットの作成に使用される場合もあります。

画像

SAS が CAS サーバー上のデータを処理する方法については、*SAS Cloud Analytic Services: DATA ステッププログラミング*を参照してください。

CAS エンジンを使用する SAS プログラムは次の点で異なります。

- 入力データは、SAS データセットまたはインメモリ CAS テーブルです。データセットは 1 つのステップとして CAS にロードする必要があります。CAS テーブルは別のステップで処理する必要があります。DATA ステップは、SAS データセットとインメモリ CAS テーブルを同時に操作することはできません。
- DATA ステッププログラムは SAS クライアントでコンパイルされ、CAS サーバーで処理されます。

DATA ステップを使用する理由

SAS DATA ステップは、データの読み込み、新しいデータの作成、およびデータの操作を可能にする SAS 言語の基本的な構成要素です。コードブロックは、データの準備と操作を可能にする SAS 言語要素(関数、ステートメント、オプション)で構成されています。SAS DATA ステップの基本構造を次に示します。

```
DATA my_output_data;
 SET my_input_data;
   <other SAS language statements and functions that update or manipulate the input data>;
RUN;
```

ここでは、基本的な DATA ステップ処理について説明します。DATA ステップを使用すると、オブザベーションのフィルタリング、新しい変数の作成、データセットのマージ、データの要約など、さまざまなデータ操作タスクを実行できます。DATA ステップに関連するタスクの詳細については、次を参照してください。

- 指定された入力データを読み取ります。データは、既存の SAS データセット、外部ファイル(生データ)、またはインラインデータにすることができます。
- 入力データの各行に対して、DATA ステップで指定されたステートメントと関数を実行します。
- 指定された関数とステートメントに基づいて出力を作成します。出力は、SAS データセット、外部ファイル、または SAS でサポートされている別の出力タイプにすることができます。

要約すると、DATA ステップを使用して、次のタスクを実行できます。

- 既存の SAS データセットとテーブルのサブセット化、変更、結合、更新
- データの分析、操作、提示
- 新しい変数の値の計算
- レポートやその他の形式の出力の作成
- SAS ファイルの管理

DATA ステップを使用したデータの読み取りと作成

SAS DATA ステップを使用して、次の方法でデータを読み書きできます。

- DATA ステップを使用して生データを読み取る
- DATA ステップを使用して SAS データセットを読み取る
- DATA ステップを使用して SAS DATA ステップビューを読み取り、作成する
- DATA ステップを使用して DBMS および PC ファイルを作成し、読み取る
- DATA ステップステートメントを使用してデータを作成する
- DATA ステップを使用してストアドコンパイル DATA ステッププログラムを作成する

DATA ステップ出力

DATA ステップからの出力は、SAS データセット、またはプログラムログ、レポート、外部データファイルなどの外部ファイルにすることができます。別のデータセットを作成せずに、既存のファイルをその場で更新することもできます。次のタイプの DATA ステップ出力を作成できます。

SAS ログ
: 処理メッセージとプログラムエラーのリストが含まれています。SAS ログはデフォルトで生成されます。

SAS データセット
: データ部分とデータディスクリプター部分の 2 つの部分を含む SAS データセットです。

SAS ビュー
: ディスクリプター情報と他のファイルのデータを使用する SAS データセットです。SAS 使用すると、新しいデータセットを作成するためにディスク領域を使用せずに、さまざまなソースからのデータを動的に結合できます。SAS データセットには実際のデータ値が含まれています。ただし、SAS® Views®には、別の場所に保存されたデータへの参照のみが含まれます。SAS® Views®のメンバータイプは VIEW です。ほとんどの場合、SAS ビューは SAS データセットであるかのように使用できます。

外部データファイル
: DATA ステップ処理の結果が含まれます。これらのファイルはデータまたはテキストファイルです。データは、フォーマットされたレコードまたはフリーフォーマットのレコードです。

レポート
: DATA ステップ処理の結果が含まれます。通常は PROC ステップを使用してレポートを生成しますが、DATA ステップから次の 2 種類のレポートを生成できます。

  プロシジャ出力ファイル
  : DATA ステップ処理の出力結果が含まれており、通常は見出しと改ページが含まれます。

  HTML ファイル
  : Web 上に表示できる結果が含まれています。このタイプの出力は、Output Delivery System (ODS)を通じて生成されます。

これらの出力タイプの詳細については、27 章, “出力” (703 ページ)を参照してください。

ストアドコンパイル DATA ステッププログラム

ストアドコンパイル DATA ステッププログラムは、コンパイルされて SAS ライブラリに保存された DATA ステッププログラムを含む SAS ファイルです。ストアドコンパイルプログラムは、必要に応じて再コンパイルすることなく実行できます。ストアドコンパイル DATA ステッププログラムのメンバータイプは PROGRAM です。

ストアドコンパイルプログラムは、V9 エンジンおよび DATA ステップアプリケーションでのみ使用できます。詳細については、“Stored, Compiled DATA Step Programs” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。

DATA ステップの処理時間

DATA ステップの処理時間は 2 つの段階で発生します。1 つ目は起動(またはコンパイル)時間、2 つ目は実行時間です。

- コンパイル時間は、SAS コンパイラが SAS ソースコードをスキャンして実行可能プログラムに変換するのにかかる時間です。
- 実行時間は、SAS ファイル内の各オブザベーションの DATA ステップを SAS が実行するのにかかる時間です。

2 つのフェーズは同時に発生しません。つまり、DATA ステップが最初にコンパイルされ、次に実行されます。これらのフェーズの詳細については、“DATA ステップのデータの処理方法” (864 ページ)を参照してください。

これらの処理時間と、それが SAS プログラムの構造にどのように関係しているかを理解することは、パフォーマンスを向上させる方法を探すときに役立つ場合があります。一般に、DATA ステップで処理するステートメントが増えるほど、コンパイル時間も長くなります。また、多数のオブザベーションを処理する DATA ステップは、I/O 集約的であるため、実行時間が長くなる傾向があります。

たとえば、比較的少数のオブザベーションを処理するプログラムは、複雑性を減らし、反復や使用されないコードを削除するために再書き込みする必要がある場合があります。PROC FCMP で作成された DO ループとユーザー定義関数は、コンパイルする必要があるコードの量を減らすことでコンパイル時間を短縮できます。CPU 集約的プログラムを実行する際のパフォーマンスを向上させる方法の詳細については、“CPU パフォーマンスを最適化する手法” (680 ページ)を参照してください。

DATA ステップで使用される時間のほとんどが数百のオブザベーションの処理に費やされている場合は、I/O を最適化するために設計された他の手法の方が役立つ可能性があります。I/O 集約的プログラムを実行する際のパフォーマンスを向上させる方法の詳細については、“I/O を最適化する手法” (671 ページ)を参照してください。

いくつかの SAS システムオプションは、処理時間を最小限に抑え、パフォーマンスを最適化するのに役立つ情報を提供します。たとえば、FULLSTIMER オプションは、各 DATA ステップのパフォーマンス統計量を収集して表示するため、データ処理の

各ステップでどのリソースが使用されたかを判断できます。このオプションと最適化全般の詳細については、25 章, “システムパフォーマンスの最適化” (669 ページ)を参照してください。

次の例は、オブザベーション数が少ない非常に大規模な DATA ステップジョブのコンパイル時間を推定する方法を示しています。プログラムは、%PUT マクロステートメントとともに DATETIME 関数を使用して、コンパイルの開始時間を計算します。次に、_N_自動変数を使用して実行開始時間を調べます(SAS は、実行フェーズの開始時に常にこの変数を 1 に設定します)。2 つの時間の差を計算することにより、プログラムは DATA ステップの合計コンパイル時間を返します。

例のコード 1.1　コンパイル時間と実行時間を調べる

```
options nosource;
%put Starting compilation of DATA step: %QSYSFUNC(DATETIME(), DATETIME20.3);
%let startTime=%QSYSFUNC(DATETIME());

data a;
  if _N_ = 1 then do;
    endTime = datetime();
    put 'Starting execution of
        DATA step: ' endTime:DATETIME20.3;
    timeDiff=endTime-&startTime;
    put 'The Compile time for this DATA Step is
      approximately ' timeDiff:time20.6;
end;
 /* Lots of DATA step code */
run;
```

アウトプット 1.1　コンパイル時間と実行時間を調べるためのログ

```
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds


Starting compilation of DATA step: 29JUN12:16:17:54.725

Starting execution of DATA step: 29JUN12:16:17:54.755
The Compile time for this DATA Step is approximately 0:00:00.030000
NOTE: The data set WORK.A has 1 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.01 seconds
```

注: マクロステートメントとマクロ変数はコンパイル時に解決され、DATA ステップの実行にかかる時間には影響しません。詳細については、“入門ガイド: マクロ機能” (*SAS マクロ言語: リファレンス*)、および“SAS プログラムとマクロ処理” (*SAS マクロ言語: リファレンス*)を参照してください。

PROC ステップ

PROC ステップの定義

PROC ステップは、通常は入力として SAS データセットを使用してプロシジャを呼び出して実行する SAS ステートメントのグループで構成されます。プロシジャを使用すると、SAS® Views®データセット内のデータを分析したり、フォーマットされたレポートやその他の結果を作成したり、SAS ファイルを管理したりすることができます。最小限の労力で PROC を変更して、必要な出力を生成できます。PROC は、SAS データセットに関する情報の表示などの機能を実行することもできます。

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。

PROC ステップ出力

PROC ステップからの出力は、単変量記述統計量、度数表、クロス集計表、記述統計量から構成される表形式レポート、チャート、プロットなどを提供できます。出力は、更新されたデータセットの形式にすることもできます。

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。

SAS Viya プラットフォームでのプログラミング

SAS Viya プラットフォーム

SAS Viya プラットフォームは、SAS の次世代の高性能インメモリ分析を表します。SAS Viya プラットフォームには、SAS プログラムを実行するためのランタイム環境がいくつか用意されています。

SAS Compute Server
: 一般に SAS プログラミングランタイム環境(SPRE)とも呼ばれる SAS Compute Server を使用すると、ユーザーは SAS DATA ステップ、PROC ステップ、ストア

ドプロシジャなどの従来の SAS プログラムコードを実行できます。詳細については、SAS Compute Server (27 ページ)を参照してください。

注: SAS Viya の 2025.02 リリース以降、SAS Compute Server は、アクションベースの高度な分析プロシジャのインプロセスマルチスレッド実行をサポートするようになりました。これらの拡張機能の詳細については、“SAS Compute Server の拡張” (28 ページ)を参照してください。

SAS Cloud Analytic Services (CAS)
CAS サーバーは、SAS によるデータ管理と分析のためのクラウドベースのハイパフォーマンスなランタイム環境を提供します。CAS はハードウェアとソフトウェアの組み合わせを使用し、データの管理と分析は単一のマシン上で行われるか、または非常に大きなテーブルの場合は複数のマシンにまたがる分散サーバーとして行われます。CAS サーバーの詳細については、*SAS Cloud Analytic Services: 基本*を参照してください。

本書では、主に SAS Compute Server 上で実行される DATA ステップおよびプロシジャのプログラミングの概念とタスクに焦点を当てています。CAS サーバー上での DATA ステップの実行については、*SAS Cloud Analytic Services: DATA ステッププログラミング*を参照してください。

SAS DATA ステップおよび SAS プロシジャを使用する場合:

- ほとんどの既存の SAS プログラムは、変更を加えることなく SAS Viya プラットフォーム上で引き続き実行できます。
- SAS プログラムは、SAS データセット、外部ファイル、およびデータベース管理システムファイルとして保存されているデータを操作します。詳細については、次のリソースを参照してください。
  - 3 部, “データへのアクセス” (225 ページ)
  - 4 部, “データの操作” (411 ページ)
  - 6 部, “ファイルの管理” (731 ページ)

セキュリティ

SAS Viya プラットフォームが提供するセキュリティ機能は、データアクセスに影響を与える可能性があります。認証に加えて、SAS Viya プラットフォームには 2 つの権限システムがあります。1 つは SAS Cloud Analytic Services 用、もう 1 つは一般的な権限システムです。どちらのシステムも、許可されていないアクセスを暗黙的に禁止します。

SAS Viya プラットフォームの権限の詳細については、*SAS Viya プラットフォーム: 権限のオリエンテーション*を参照してください。

パスワードを割り当て、暗号化を要求することで、作成した SAS データセットを保護できます。

SAS パスワードとファイル暗号化の詳細については、“パスワード” (734 ページ)を参照してください。

SAS の LOCKDOWN セキュリティの詳細については、“LOCKDOWN” (19 ページ)を参照してください。

詳細情報の入手先

CAS で DATA ステップをいつどのように使用するかの詳細については、*SAS Cloud Analytic Services: DATA ステッププログラミング*を参照してください。

命名規則と構文規則については、本書の"構文"セクションを参照してください。

DATA ステップで実行できる操作の詳細については、本書の"データの操作"セクションを参照してください。

本書の"データへのアクセス"および"ファイルの管理"セクションは、SAS Compute Server を使用した保存されたデータの操作に関するものです。

本書の概念のほとんどは、V9 エンジンと SPD Engine の両方のストレージエンジンでの SAS の使用に適用されます。

- V9 エンジンを使用した SAS 言語機能の完全なガイドについては、*SAS V9 LIBNAME エンジン: リファレンス*のドキュメントも参照してください。
- V9 エンジンを使用した SAS 言語機能の完全なガイドについては、*SAS Scalable Performance Data Engine: リファレンス*のドキュメントも参照してください。

LOCKDOWN

LOCKDOWN とは

LOCKDOWN は、SAS サーバー上で実行されるセッションによってアクセス可能な特定のシステムファイルおよび SAS 言語機能へのアクセスを制御する SAS のセキュリティ機能です。LOCKDOWN の主な特徴は次のとおりです。

- ロックダウンは、すべての SAS Viya 配置でデフォルトで有効になっています。
- SAS 管理者は、LOCKDOWN システムオプション、 LOCKDOWN ステートメント、ロックダウン許可リストを使用して LOCKDOWN を管理します。
- LOCKDOWN が有効になっている場合、SAS は*ロックダウン状態*にあります。SAS がロックダウン状態にある場合、一部の SAS 言語機能はデフォルトで有効、制限、または無効になります。

- SAS がロックダウンされている場合、一部のファイルとファイルパスはデフォルトで有効または無効になっています。ファイルパスへのアクセスは、LOCKDOWN 許可リストを使用して SAS 管理者によって制御されます。

> 注意
> **安全でないコードを実行しないでください。** 安全でないコードを実行すると、環境がデータ損失、データ破損、セキュリティ違反、またはシステム不安定につながる可能性があります。組み込まれるすべてのコードが安全であり、テストされ、組織のポリシーおよび適用される法律に準拠していることを確認するのはユーザーの責任です。

LOCKDOWN のデフォルトの動作

SAS がロックダウンされている場合にデフォルトで使用可能なアクセス方式

次の表は、LOCKDOWN が有効になっている場合の SAS 言語機能のデフォルトの可用性を示しています。このテーブルには、LOCKDOWN の影響を受けない SAS 機能は含まれていません。LOCKDOWN を使用して SAS 機能を有効または無効にする方法については、"Enable or Disable Access Methods" (*SAS Viya Platform: Programming Run-Time Servers*)を参照してください。

✗は、SAS がロック状態にある場合に機能がデフォルトで無効になることを示します。

✓は、SAS がロックダウン状態にある場合に、機能がデフォルトで有効になることを示します。

表 1.2　SAS がロックダウンされている場合の SAS 言語要素のデフォルトの使用可能性

機能	デフォルト状態
SAS FILENAME ステートメントのオプション	
EMAIL アクセスメソッド	✓
FTP アクセス方式	✓
HADOOP アクセスメソッド	✓
HTTP/URL アクセス方式 [1]	✓
SOCKET アクセス方式	✗

機能	デフォルト状態
XSL アクセスメソッド [3]	✗
SAS プロシジャ	
PROC GROOVY[2]	✗
PROC HTTP	✗
PROC JAVAINFO	✓
PROC LUA IO および OS ライブラリ関数	✗
PROC PYTHON	✗
PROC R	✗
PROC SOAP	✗
PROC XSL[3]	✗
SAS 関数と CALL ルーチン	
ADDRLONG function	✗
CALL POKELONG routine	✗
CALL SYSTEM ルーチン [2]	✗
DCREATE 関数 [4]	✗
FILEEXIST 関数 [4]	✗
FILENAME 関数 [4]	✗
GIT functions [5]	✓
PEEKLONG 関数	✗
PEEKCLONG 関数	✗
RENAME 関数 [4]	✗

機能	デフォルト状態
その他の SAS 機能	
%SYSEXEC マクロステートメント [2]	✗
DATA ステップの Java オブジェクト [6]	✗
DS2 HTTP パッケージ	✗
PIPE オプション(FILENAME ステートメント)[2]	✗

1 HTTP および URL アクセス方式はエイリアスのペアです。

2 SAS がロックダウンされている場合、X コマンドと XCMD システムオプションは、LOCKDOWN によりデフォルトで無効になるため、この機能はデフォルトで無効になります。SAS 管理者は、XCMD を許可するように SAS Compute Server を構成することで、このオプションを有効にできます。

3 2024.12 以降では、SAS がロックダウンされている場合、XSL プロシジャはデフォルトで無効になります。SAS 管理者は、LOCKDOWN ステートメントで ENABLE_AMS=XSL を指定することで、このプロシジャを有効にできます。詳細については、次のドキュメントの Example 6 を参照してください: *SAS Viya Platform: Programming Run-Time Servers*。

4 この関数は、SAS がロックされている場合、デフォルトでは無効化されます。SAS 管理者は、ロックダウン許可リストに関数によって読み取られたパスを追加することで、この機能を有効にできます。

5 2025.07 以降、SAS 管理者は、VIYA_LOCKDOWN_USER_DISABLED_METHODS=GIT を指定して GIT 関数を無効にできます。これにより、ロックダウン許可リストから GIT アクセス方式が削除されます。詳細については、"Enable or Disable Access Methods" (*SAS Viya Platform: Programming Run-Time Servers*)を参照してください。

6 2025.08 以降、SAS がロックダウンされている場合、DATA ステップ Java オブジェクトはデフォルトで無効になります。SAS 管理者は、LOCKDOWN ステートメントで ENABLE_AMS=JAVAOBJ を指定することで、これらの方法を有効にできます。

注: これらの設定は SAS Viya プラットフォームにのみ適用されます。

ファイルアクセスとロックダウン許可リスト

SAS がロックダウンされると、すべてのローカルファイルとディレクトリへのアクセスはロックダウン許可リストを通じて検証されます。ロックダウン許可リストの主な特徴は次のとおりです。

- ロックダウン許可リストは、SAS のインストール時に自動的に作成され、SAS の実行に必要なリソース、アクセス方式、およびファイルの事前定義リストが含まれています。
- SAS 管理者は、PATH=オプションまたは ENABLE_AMS=オプションを LOCKDOWN ステートメントで指定することにより、追加のパス、アクセス方式、および機能をロックダウン許可リストに追加できます。
- 一部の SAS 関数は、ファイルパスを入力として読み取り、SAS がロックダウン状態にある場合は実行に失敗します。これらの関数が適切に実行されるようにするには、SAS 管理者はロックダウン許可リストにターゲットパスを追加する必要があります。

SAS Viya プラットフォームでのファイルシステムアクセスと 権限の詳細については、"File System and Storage Authorization" (*SAS Viya Platform Identity and Access Management: Fundamentals*)を参照してください。

表 1.3　LOCKDOWN が有効な場合にデフォルトで使用可能なファイルパス

パス	説明
FONTSLOC オプションパス	■ SAS によって提供されるフォントの場所を指定します。
JAVA_HOME	■ Java Runtime Environment がインストールされている場所を指定します。
MAPS オプションパス	■ SAS/GRAPH マップデータセットを含む SAS ライブラリの場所を指定します。
マウントされたボリュームパス	■ Kubernetes によって永続ボリュームがマウントされるパスを指定します。これらのボリュームは通常、SAS Viya の配置時に SAS 管理者によって永続ボリューム要求(PVC)として定義されます。 ■ SAS の起動時にパスが許可リストに自動的に追加され、LOCKDOWN が有効な場合に共有ストレージまたは永続ストレージへアクセスできるようになります。これらのパス内のすべてのサブディレクトリも 含まれます。 ■ 詳細については、"Persistent Storage for SAS Viya Software" (*SAS Viya Platform: Troubleshooting*)および "Persistent Storage Volumes, PersistentVolumeClaims, and Storage Classes" (*System Requirements for the SAS Viya Platform*)を参照してください。
SAS デフォルト作業ディレクトリ	■ SAS セッションが開始されるディレクトリまたは場所を指定します。 ■ *現在の作業ディレクトリ*とも呼ばれます。
SASROOT ディレクトリパス	■ ファイルシステム上で SAS がインストールされるディレクトリとディレクトリ構造を指定します。 ■ !SASROOT ディレクトリとも呼ばれます。 ■ 呼び出しポイント、構成ファイル、サンプルプログラム、カタログ、データセット、実行可能ファイルなど、SAS を使用するために必要なファイルが含まれています。SAS を使用するために、これらのディレクトリの構成を知る必要はありません。
SSLCALISTLOC、SSLCACERTDIR、SSLCERTLOC、SSLCRLLOC システムオプションパス	■ 信頼できる証明機関のパブリック証明書の場所を指定します。
TEXTURELOC オプションパス	■ ODS スタイルで使用されるテクスチャとイメージの場所を指定します。

パス	説明
	■ ディレクトリの物理名またはディレクトリへの URL 参照のいずれかを参照できます。
UTILLOC システムオプションパス	■ SAS がユーティリティファイルを 格納するディレクトリまたは場所を指定します。 ■ データセットの並べ替えや SQL プロシジャの処理など、特定の SAS プロセス中に作成される一時ファイルが含まれます。 ■ 必要に応じて管理者が変更できます。

2

SAS Viya プラットフォーム

プログラミングインターフェイス

SAS Data and AI Studio

SAS Data and AI Studio(2026.06 より前は SAS Studio という名前)は、SAS コードの開発に役立つさまざまな機能を含む Web ベースのアプリケーションです。コードを記述して結果を表示するための最新のプログラムエディターに加えて、ローカルマシン、SAS サーバー、またはクラウド上のファイルにアクセスすることもできます。一般的なアクション用に保存されたコードを活用し、ポイントアンドクリックタスクで生成コードを生成できます。SAS Data and AI Studio は、SAS Viya プラットフォームでの使用に特化して設計されています。

SAS® Data and AI Studio セッションのカスタマイズ

SAS Viya プラットフォームでは、SAS の起動前に適用されるシステムオプションと環境変数は、SAS Environment Manager を使用して管理者によって指定されます。SAS Viya プラットフォームは、構成インスタンスを使用して SAS サーバーと CAS サーバーを管理します。サーバー構成インスタンスを変更するには、管理者権限を持つユーザー ID が必要です。サーバー構成インスタンスの編集については、*SAS Viya*

*プラットフォーム: プログラミングランタイムサーバー*および *SAS Viya プラットフォーム: SAS Cloud Analytic Services* を参照してください。

SAS セッション内で強制されるシステムオプションは、エンドユーザーが SAS Data and AI Studio セッションで設定できます。SAS® Data and AI Studio セッションで設定されたシステムオプションは、セッションが終了すると期限切れになります。または、これらのシステムオプションを autoexec ファイルで自動割り当てに設定できます。SAS Data and AI Studio で **autoexec ファイル**を作成するには、**オプション**メニューから Autoexec ファイルを選択します。任意の SAS ステートメントを autoexec ファイルに含めることができます。たとえば、システムオプション、レポートタイトル、フットノートを設定したり、マクロやマクロ変数を autoexec ファイルを使用して自動的に作成したりできます。

システムオプションのリファレンスドキュメントには、特定のシステムオプションが有効な場所が指定されています。SAS Compute Server で使用できるシステムオプションについては、*SAS システムオプション: リファレンス*を参照してください。CAS で使用できるシステムオプションについては、*SAS Cloud Analytic Services: ユーザーガイド*を参照してください。

autoexec ファイルの設定方法については、*SAS Studio: ユーザーガイド*を参照してください。

その他の SAS Viya プラットフォームプログラミングインターフェイス

SAS Data and AI Studio(2026.06 より前は SAS Studio という名前)は SAS Viya プラットフォームのデフォルトのプログラミングインターフェイスですが、コードを開発してサブミットするための唯一のオプションではありません。Jupyter Notebook や R Studio などのオープンソースアプリケーションを使用できます。または、SAS 配置に SAS®9 メンテナンスリリース 5 以降が含まれている場合は、SAS Data および AI Studio(2026.06 より前は SAS Studio という名前)を含む SAS®9 アプリケーションを使用できます。SAS Enterprise Guide、または SAS ウィンドウ操作環境。

管理コマンドラインインターフェイス

SAS Viya プラットフォームには、管理コマンドラインインターフェイス(CLI)も含まれています。SAS Viya プラットフォームでは、CLI は SAS Viya REST サービスへのユーザーインターフェイスであり、コマンドラインにコマンドを入力し、システムから応答を受け取ります。CLI を使用すると、GUI を使用せずにプログラムで SAS Viya プラットフォームと直接対話できます。CLI の詳細については、*SAS Viya プラットフォーム: コマンドラインインターフェイスの使用*のコマンドラインインターフェイス: 概要を参照してください。

その他のリソース:

- SAS Viya でパラメーターを使用して SAS プログラムをバッチで実行する方法
- SAS® Data and AI Studio フローをバッチで実行する方法
- コマンドラインインターフェイス: プラグイン

関連項目

- *SAS Viya プラットフォームプログラミング: CAS 用 Java 入門ガイド*
- *SAS Viya プラットフォームプログラミング: CAS 用 Lua 入門ガイド*
- *SAS Viya プラットフォームプログラミング: CAS 用 Python 入門ガイド*
- *SAS® Viya® プラットフォームプログラミング: CAS 用 R 入門ガイド*

SAS Viya サーバー

SAS Viya では、分析の複雑さ、データのサイズ、およびコードの記述に使用される構文に応じて、プログラムがさまざまなサーバー環境で実行されます。SAS プログラムは、SAS Compute Server または CAS サーバーのいずれかで実行できます。SAS Viya プラットフォームサーバーの詳細については、“SAS Viya プラットフォーム” (17 ページ)を参照してください。

“SAS Compute Server の拡張” (28 ページ)により、SAS Viya でのプログラミングが容易になり、多くの CAS 対応プロシジャや CAS アクションを SAS Compute Server 上でインプロセスで実行できるようになります。

アーキテクチャの観点からの SAS Viya プラットフォームおよび SAS Viya プラットフォームサーバーの詳細については、*SAS® Viya® プラットフォーム: プログラミングランタイムサーバー*を参照してください。

SAS Compute Server

The SAS Viya platform provides the SAS Compute Server as the programming run-time server for processing data that is not performed on the CAS server.

The SAS Compute Server uses the SAS Viya platform Compute service to enable clients to submit SAS programs in the form of jobs for processing. For every job that is processed, the SAS Compute Server writes a logging message to the SAS log. If the job produces ODS results, output data sets, files, and so on, the output is associated with the job.

- The DATA step can run on the SAS Compute Server or on the CAS server. The DATA step determines whether it can be run on the CAS server based on what LIBREFS, statements, and features are included in the DATA step code. If the DATA step cannot run on the CAS server, the DATA step runs on the SAS Compute Server. For more information, see “例: CAS での DATA ステップの実行” (*SAS Cloud Analytic Services: DATA ステッププログラミング*).
- The data management and utility procedures and macros run on the SAS Compute Server. Many SAS procedures run on the CAS server. For more information, see SAS Procedures by Name. SAS procedures that do not run on the CAS server can transfer data from in-memory tables from the CAS server to the compute server for processing.

- FedSQL and DS2 run on the SAS Compute Server when they access SAS libraries. They run on the CAS server when instructed by the SESSREF= or SESSUID= procedure options.
- The SAS Compute Server supports SAS catalogs.
- High-performance SAS Viya platform procedures and other supporting procedures use a CAS LIBNAME engine libref to identify in-memory tables and then use CAS actions to perform data analysis and processing on the CAS server.
- Starting with the 2025.02 release of the SAS Viya platform, high-performance SAS Viya platform procedures and other supporting procedures use a SAS LIBNAME engine libref to perform data analysis and processing on the SAS Compute Server.

SAS Compute Server の拡張

SAS Compute Server は、SAS プログラムの主要な実行環境として SAS Viya プラットフォームとともに導入されました。

SAS Viya 2025.02 以降、SAS Viya 2025.02 以降、主要な拡張機能により、以前は CAS サーバーが必要であったプロシジャを SAS Compute Server で直接実行できるようになり、SAS プログラミングが簡素化されます。これらの向上により、多くの分析プロシジャにおける CAS への依存度が軽減されると同時に、必要に応じて CAS サーバー上で分析プロシジャを実行する機能が維持されます。さらに、拡張機能では SAS Compute Server 内の R とネイティブ Python プログラムのサポートが拡張されます。

注: このセクションで説明する拡張機能は、SAS Compute Server およびその他のサポートされている SAS Viya プログラミングランタイムサーバーに適用されます。これらの拡張機能は、SAS Viya 4 の SAS Analytics Pro (APro)オファリングには適用されません。SAS Viya 4 には、設計上これらの拡張機能は含まれていません。

主な機能

SAS Viya の SAS Compute Server の主な拡張機能は次のとおりです。

- **インプロセス実行**: 以前は CAS サーバーを必要としていたものも含め、高度な分析プロシジャをサポートし、SAS Compute Server 内で直接実行できるようにします。これによりオーバーヘッドが削減され、実行が簡素化されます。
- **マルチスレッドのサポート**: SAS Compute Server でアクションベースの高度な分析プロシジャをマルチスレッド実行できるようにし、パフォーマンスを向上させます。これらのプロシジャの実行は、データとプロセスが複数のノードに分散されていない単一マシン環境に制限されることに注意してください。アーキテクチャに関する考慮事項の詳細については、SAS Viya Stable 2025.02 による SAS プログラミングランタイムサーバーの拡張を参照してください。

- **多言語サポート**: SAS Viya へのスムーズな移行のために SAS 9 プロシジャ構文をサポートし、sasviya.ml ライブラリを通じてネイティブ Python サポートを導入します。多言語サポートの詳細については、次のリソースを参照してください。
  - *SAS Viya プラットフォーム: 外部言語との統合*
  - *Computer Vision Python API ガイド*
  - *機械学習 Python API ガイド*
  - *Python の例*
- **分析実行の拡張**: CAS テーブルへの変換を必要とせずに SAS データセットで高度な分析プロシジャをサポートし、すべての CAS アクションに SAS プロシジャが存在することを保証するため、ユーザーは CAS 言語構文を学習しなくても同じレベルの分析を達成できます。

> 重要　この拡張機能により、以前は基になる CAS アクションを実行するために CAS サーバーを必要としていたプロシジャが、CAS サーバーなしでこれらの分析アクションを実行できるようになりました。これらのプロシジャが SAS Compute Server 内で実行される場合、サポートされるのは UTF-8 文字エンコーディングのみです。

注: PROC CAS、CAS アクション、および CAS 言語(CASL)は、SAS Viya で引き続き完全にサポートされます。

高度なプロシジャ

SAS Compute Server の拡張機能は、複雑な CAS アクション構文を必要とせずに、バックグラウンドで CAS アクションをシームレスに実行することにより、SAS の分析能力を SAS プログラミングランタイム環境に提供します。次の SAS Viya プロシジャは、拡張された SAS Compute Server 環境で実行できるようになりました。

- Computer Vision プロシジャ
- Econometrics プロシジャ
- 数理最適化プロシジャ
- OPTNETWORK プロシジャ
- Machine Learning プロシジャ
- Text Analytics プロシジャ
- Visual Forecasting プロシジャ
- Visual Statistics プロシジャ

簡略化された構文

次の表の最初の 2 つの列は、従来の SAS プロシジャ構文を使用して SAS Compute Server で BINNING プロシジャを実行する場合の構文と、CAS サーバーで実行する場合の構文を比較しています。最後の列は、PROC BINNING をサポートする CAS アクションを実行するための SAS Viya での継続的なサポートを示しています。

表 2.1 SAS Viya 計算環境: 構文の比較

SAS Compute Server でのプログラム実行	CAS サーバーでのプログラム実行	CASL 構文を使用した CAS サーバーでのプログラム実行

data cars;
  set sashelp.cars;
run;
proc binning data=cars
  method=quantile
  numbin=5;
  input MSRP;
  output out=binned_cars;
run;
	cas casauto;
	
	libname mycas cas;
	
	data mycas.cars;
	  set sashelp.cars;
	run;
	
	proc binning data=mycas.cars
	  method=quantile
	  numbin=5;
	  input MSRP;
	  output out=mycas.binned_cars;
	run;
		cas casauto;
		libname mycas cas;
		
		data mycas.cars;
		  set sashelp.cars;
		run;
		
		proc cas;
		  dataPreprocess.binning /
		  table={name="cars",
		  caslib="casuser"}
		  inputs={"msrp", "invoice"}
		  method="quantile"
		  nbins={3, 4};
		quit;

注:

ここで説明する SAS Compute Server の拡張機能は、SAS Batch Server や SAS/CONNECT Server など、SAS Viya の他のプログラミングランタイムサーバーにも適用できます。SAS Viya プラットフォームランタイムサーバーの詳細については、*SAS Viya プラットフォーム: プログラミングランタイムサーバー*を参照してください。

SAS Cloud Analytic Services (CAS)

CAS サーバーは、SAS のハイパフォーマンス分析をサポートするクラウドベースのランタイム環境です。PROC CAS や CAS アクションなどの CAS 言語(CASL)で記述されたプログラムは、CAS サーバー上で実行されます。CAS サーバーと CAS サーバー処理の詳細については、*"SAS Cloud Analytic Services (CAS)" (SAS Viya Platform Programming: Getting Started)*を参照してください。

2 部

SAS 構文

3

単語と名前

SAS 単語

SAS 単語の定義

SAS プログラミング言語の*単語*は、SAS に意味を伝える文字の集合です。単語を独立して使用できる小さな単位に分割することはできません。1 つの単語には最大 32,767 バイトを含めることができます。

SAS が次のいずれかの状況に遭遇すると、単語は終了します。

- 新しい単語の始まり
- 名前または数字の後の空白
- *リテラル*単語の最後の引用符

単語またはトークンの種類

SAS 言語の各単語は、次の 4 つのカテゴリのいずれかに属します。

名前
: 文字またはアンダースコアで始まる一連の文字です。以降の文字には、文字、アンダースコア、および数字を含めることができます。名前には最大 32,767 バイトを含めることができます。ただし、ほとんどのコンテキストでは、*名前*は 32 バイトや 8 バイトなど、より短い最大長に制限されます。詳細については、表 3.1 (38 ページ)を参照してください。名前トークンの例をいくつか示します。

  data

  _new

  yearcutoff

  year_99

  descending

  _n_

リテラル
: 単一引用符または二重引用符で囲まれた 1-32,767 バイトで構成されます。リテラルの例をいくつか示します。

- 'Chicago'
- "1990-91"
- 'Amelia Earhart'
- 'Amelia Earhart''s plane'
- "Amelia Earhart's plane"
- "Report for the Third Quarter"

注: 周囲の引用符は文字列をリテラルとして識別しますが、SAS はこれらの引用符をリテラル文字列の一部として保存しません。

数値
一般に、完全に数字で構成され、オプションの小数点と先頭のプラスまたはマイナス記号が付きます。SAS は、指数 (E−)表記、16 進表記、*欠損値*記号、日付と時間リテラルの形式の数値を数値トークンとして認識します。数値の例をいくつか示します。

- 5683
- 2.35
- 0b0x
- -5
- 5.4E-1
- '24aug90'd

特殊文字
通常、文字、数字、アンダースコア、空白以外の単一のキーボード文字です。通常、各特殊文字は 1 つの単語ですが、**や<=などの一部の 2 文字演算子は 1 つの単語を形成します。空白は名前や数字の末尾に付けることはできますが、単語ではありません。特殊文字の例をいくつか示します。

- =
- ;
- '
- +
- @
- /

単語の配置と間隔

SAS ステートメント内の単語の間隔要件には次のものが含まれます。

- SAS ステートメントは行の任意の列で開始し、同じ行に複数のステートメントを記述することができます。

- ステートメントをある行で開始し、別の行で継続することはできますが、単語を 2 行に分割することはできません。
- 空白は、リテラルまたはリテラルの一部として引用符で囲まれていない限り、SAS ステートメント内の文字として扱われません。したがって、SAS ステートメント内の単一の空白を配置できる場所であれば、どこにでも複数の空白を配置できます。構文には影響しません。
- 単語またはトークンの境界を認識するためのルールにより、SAS プログラム内での単語またはトークン間の間隔の使用が決まります。SAS が演算子などの手がかりによって各トークンの先頭を判断できる場合は、空白を含める必要はありません。SAS が各トークンの先頭を判断できない場合は、空白を使用する必要があります。

SAS 名

SAS 名の定義

*SAS 名*は、次の SAS 言語要素のいずれかを表す名前 *トークン*です。

■ 配列	■ 出力形式	■ オプション
■ カタログエントリ	■ 入力形式	■ プロシジャ
■ コンポーネントオブジェクト	■ ライブラリ参照名またはファイル参照名	■ ステートメントラベル
■ データセット	■ マクロまたはマクロ変数	■ 変数

SAS には 2 種類の名前があります。

- *SAS 言語要素*名
- ユーザー指定の名前

ほとんどの SAS 名の規則

次のリストには、ほとんどの *SAS 名*を作成するときに使用する規則が含まれています。

注: SAS 変数名、データセット名、ビュー名、アイテムストア名については、他の言語要素よりも規則が柔軟です。変数名の規則およびメンバー名の規則を参照してください。

- *SAS 名*の長さは、SAS 名が割り当てられている要素によって異なります。多くの *SAS 名*は 32 バイトの長さになります。他のものは最大長が 8 バイトです。*SAS 名*とその最大長のリストについては、表 3.1 を参照してください。
- 最初の文字は英字(A、a、B、b、C、c、...、Z、z)またはアンダースコア(_)にする必要があります。後続の文字には、文字、数字(0、1、...、9)、またはアンダースコア。

  > 重要　ユーザー定義の出力形式名の末尾を数字にすることはできません。

- 大文字または小文字を使用できます。
- 空白は許可されません。
- アンダースコアを除く特殊文字は使用できません。*ファイル参照名*でのみ、ドル記号($)、番号記号(#)、およびアットマーク(@)を使用できます。
- SAS では、自動変数と変数リスト、SAS データセット、ライブラリ参照名用にいくつかの名前が予約されています。
  - 変数を作成するときは、特殊な SAS 自動変数の名前(_N_や_ERROR_など)や特殊な変数リスト名(_CHARACTER_、_NUMERIC_、_ALL_など)を使用しないでください。
  - *ライブラリ参照名*を SAS ライブラリに関連付ける場合は、次の *ライブラリ参照名*を使用しないでください。
    - Sashelp
    - Sasmsg
    - Sasuser
    - Work
  - SAS データセットを作成するときは、次の名前を使用しないでください。
    - _NULL_
    - _DATA_
    - _LAST_
- *ファイル参照名*を*外部ファイル*に割り当てるときは、ファイル名を SASCAT 使用しないでください。
- マクロ変数を作成するときは、SYS で始まる名前を使用しないでください。

名前の長さの規則

*SAS 名*の長さは、SAS 名が割り当てられている要素によって異なります。

表 3.1 言語要素別の SAS 名の最大長

言語要素名	バイト単位の最大長
配列名	32
CALL ルーチン名	16
カタログ名	32
コンポーネントオブジェクト名	32
データセット(テーブル)名	32
エンジン名	8
ファイル参照名	8
出力形式名、文字	31
出力形式名、数値	32
関数名	16
世代データセット名	28
インデックス名	32
入力形式名、文字	30
入力形式名、数値	31
ライブラリ参照名(ライブラリ名)	8
ライブラリメンバー名 [1]	32
マクロ変数名	32
マクロウィンドウ名	32
マクロ名	32
パスワード名	8
プロシジャ名(最初の 8 文字は一意である必要があり、"SAS"で始めることはできません)	16

1. SAS ライブラリメンバー名の例としては、データセット名、ビュー名、カタログ名などがあります。詳細については、“メンバーの種類” (*Base SAS プロシジャガイド*)を参照してください。

言語要素名	バイト単位の最大長
ステートメントラベル名	32
変数(DATA ステップ変数および SCL 変数を含む)名	32
変数ラベル名	256
ビュー名	32
ウィンドウ名	32

SAS 名の拡張

VALIDVARNAME=ANY または VALIDMEMNAME=EXTEND の場合、これらの SAS 名の長さはバイト単位で測定する必要があります。

システムオプション設定	バイト単位で測定される SAS 名	バイト単位の最大長
VALIDVARNAME=ANY	変数名	32
VALIDMEMNAME=EXTEND	SAS データセット名 SAS ビュー名 アイテムストア名	32

これらのシステムオプション値が設定されている場合、メンバー名に使用できる最大文字数は、1 文字を保存するために使用されるバイト数です。この値は、SAS セッションの SAS エンコーディング値によって設定されます。*各国語サポート*(NLS) 文字の使用を許可するには、VALIDVARNAME=ANY または VALIDMEMNAME=EXTEND を設定する必要があります。それ以外の場合は、半角文字のみが許可されます。

西欧言語の SAS エンコーディングは、1 文字を保存するために 1 バイトのストレージを使用します。したがって、西欧言語では、これらの SAS 名に 32 文字を使用できます。一部のアジア言語の SAS エンコーディングでは、1 文字を保存するために 1-2 バイトのストレージが使用されます。Unicode エンコーディングである UTF-8 は、1 つの文字に対して 1-4 バイトのストレージをサポートします。SAS エンコーディングが 1 文字を保存するために 4 バイトを使用する場合、これらの SAS 名の 1 つの最大長は 8 文字です。

すべての SAS エンコーディングは、文字 A–Z および a–z を半角文字としてサポートします。

SAS 名に使用できる最大文字数を確認するには、次の手順に従ってください。

1 次のいずれかの方法で SAS エンコーディングを見つけます。

- SAS セッションでは、OPTIONS プロシジャで ENCODING=システムオプションをサブミットします。

```
proc options option=encoding;
run;
```

2 "データのトランスコーディングに使用する SBCS、DBCS および Unicode エンコーディング値"の表で、SAS エンコーディングの 1 文字あたりの最大バイト数を見つけます。この表は、*SAS 各国語サポート(NLS): リファレンスガイド*に記載されています。

3 *SAS 名*の最大バイト数を表 3.1 から見つけます。この数値を 1 文字あたりのバイト数で割ります。結果は、*SAS 名*に使用できる最大文字数になります。

CAS エンジンの変数名とデータセット名

CAS の変数名とデータセット名については、"CAS エンジンの変数名とデータセット名" (*SAS Cloud Analytic Services: ユーザーガイド*)を参照してください。

関連項目

例

- "例: 空白を含む変数名の作成"
- "例: 単語の配置と間隔の管理"

ステートメント

- "OPTIONS ステートメント" (*SAS グローバルステートメント: リファレンス*)

システムオプション

- "VALIDMEMNAME= システムオプション" (*SAS システムオプション: リファレンス*)
- "VALIDVARNAME= システムオプション" (*SAS システムオプション: リファレンス*)

SAS Name Literals

SAS 名リテラルの定義

SAS *名リテラル*は、引用符で囲まれた文字列として表現され、その後に大文字または小文字の *n* が続くユーザー指定の名前です。

VALIDVARNAME システムオプションが V=7 に設定されている場合、SAS 変数名の最初の文字は、英字(A、a、B、b、C、...、Z、z、)またはアンダースコア(_)にする必要があります。後続の文字には、文字、数字(0、1、...、9)、またはアンダースコア。

名前リテラルを使用すると、名前の最初の文字として空白、各国文字、数字など、さらに多くの文字を使用できるようになります。ただし、名前の最初の文字として数字を使用するには、VALIDVARNAME=オプションを ANY に設定する必要があります。

```
options validvarname=any;
data test;
  "1ABC"n="hello";
run;
```

次のタイプの SAS 名で名前リテラルを使用できます。

- DBMS 列名
- DBMS テーブル
- アイテムストア
- SAS データセット
- SAS ビュー
- ステートメントラベル
- 変数名と値

名前リテラルに_、A–Z、a–z 以外の文字を使用するには、VALIDVARNAME=ANY または VALIDMEMNAME=EXTEND システムオプションを設定する必要があります。次の表に、*SAS 名*リテラルを使用するために設定する必要があるオプションを示します。

表 3.2　SAS 名リテラルシステムオプションの要件

SAS 名のタイプ	名前リテラルの要件
DBMS 列	VALIDVARNAME=ANY を設定します。
DBMS テーブル名	VALIDMEMNAME=EXTEND を設定します。

SAS 名のタイプ	名前リテラルの要件
アイテムストア	VALIDMEMNAME=EXTEND を設定します。
SAS データセット名	VALIDMEMNAME=EXTEND を設定します。
SAS ビュー	VALIDMEMNAME=EXTEND を設定します。
ステートメントラベル	VALIDVARNAME=ANY を設定します。
変数	VALIDVARNAME=ANY を設定します。

名前リテラルは、特殊文字を含む DBMS の列名やテーブル名を表現したり、SAS 名に各国文字を含めたりする場合に特に役立ちます。

重要な制限事項

- *名前リテラル*は、変数、ステートメントラベル、DBMS 列名とテーブル名、SAS データセット、SAS ビュー、アイテムストアにのみ使用できます。
- SAS データセット名、SAS ビュー名、またはアイテムストア名の名前リテラルに、VALIDMEMNAME=COMPAT の場合に許可されない文字が含まれている場合は、システムオプション VALIDMEMNAME=EXTEND を設定する必要があります。*“VALIDMEMNAME= システムオプション” (SAS システムオプション: リファレンス)*を参照してください。
- 変数、DBMS テーブル、または DBMS 列の名前リテラルに、VALIDVARNAME=V7 のときに許可されない文字が含まれている場合は、システムオプション VALIDVARNAME=ANY を設定する必要があります。*“VALIDVARNAME= システムオプション” (SAS システムオプション: リファレンス)*を参照してください。
- パーセント記号(%)またはアンパサンド(&)を使用する場合は、SAS マクロ機能との相互作用を避けるために、名前リテラルで単一引用符を使用する必要があります。
- DBMS テーブルまたは列の名前リテラルに SAS 規則で無効な文字が含まれている場合は、SAS/ACCESS LIBNAME ステートメントオプションの指定が必要になる場合があります。SAS/ACCESS LIBNAME ステートメントの詳細と例、および SAS 命名規則に準拠していない DBMS テーブル名と列名の使用については、*リレーショナルデータベース用 SAS/ACCESS: リファレンス*を参照してください。
- 引用符で囲まれた文字列では、SAS は先頭の空白を保持して使用しますが、SAS は末尾の空白を無視して削除します。
- 名前リテラルを指定する場合、終わり引用符と *n* の間の空白は無効です。
- VALIDVARNAME=ANY を設定した場合でも、V6 エンジンは空白を挟む名前をサポートしないことに注意してください。

BY グループでの名前リテラルの使用

BY グループ処理において、BY 変数として名前リテラルを指定し、対応する FIRST. または LAST.一次変数を参照する場合は、2 レベル変数名の FIRST.またはたは LAST. 部分を引用符で囲む必要があります。

名前リテラル使用時のエラーの回避

名前リテラルを使用する際のエラーを回避する方法については、“文字定数でよくあるエラーの回避”を参照してください。

関連項目

例

- “例: 空白を含む変数名の作成”
- “例: FIRST.および LAST.BY 変数を名前リテラルとして指定する”

ステートメント

- “BY ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “OPTIONS ステートメント” (*SAS グローバルステートメント: リファレンス*)

システムオプション

- “VALIDMEMNAME= システムオプション” (*SAS システムオプション: リファレンス*)
- “VALIDVARNAME= システムオプション” (*SAS システムオプション: リファレンス*)

変数名

変数名の規則

SAS 変数名の規則が拡張され、より多くの機能が提供されるようになりました。“VALIDVARNAME= システムオプション” (*SAS システムオプション: リファレンス*)の設定は、SAS セッションで作成する変数、および既存のデータセットから読み取る変数にどの規則を適用するかを決定します。

VALIDVARNAME=システムオプションには 3 つの設定(V7、UPCASE、および ANY)があり、それぞれ変数名の柔軟性が異なります。SAS セッションで VALIDVARNAME=システムオプションを指定しない場合、デフォルト値 V7 が SAS セッションに自動的に割り当てられます。

次の表は、VALIDVARNAME=システムオプションが V7 に設定されている場合の SAS 変数の命名規則の概要を示しています。これらは Base SAS のデフォルト設定です。SAS Visual Analytics や SAS Cloud Analytic Services などの一部の SAS アプリケーションでは、SAS 変数の命名を最も柔軟に行えるように、VALIDVARNAME=システムオプションがデフォルトで設定されています。これらの規則を表 3.5 にまとめます。

表 3.3　SAS 変数の命名に関するデフォルト規則の概要

変数名 (VALIDVARNAME=V7 を使用)
■ 長さは最大 32 バイトです。 ■ 特殊文字(アンダースコアを除く)、空白、または各国文字を含めることはできません。 ■ ラテンアルファベット(A–Z、a–z)またはアンダースコアで始める必要があります。 ■ 大文字と小文字を混在させることができます。SAS は、変数への最初の参照で使用されるのと同じ大文字小文字で変数名を保存および書き込みます。 SAS は内部ですべての変数名を大文字に変換することに注意してください。 たとえば、cat、Cat、および CAT という名前はすべて同じ変数を表します。

SAS 変数の拡張規則

次の表は、VALIDVARNAME=システムオプションが UPCASE に設定されている場合の SAS 変数の命名規則の概要を示しています。変数名はすべて大文字です。

表 3.4　SAS 変数の命名に関する拡張規則の概要

変数名 (VALIDVARNAME=UPCASE を使用)
■ 長さは最大 32 バイトです。 ■ 特殊文字(アンダースコアを除く)、空白、または各国文字を含めることはできません。 ■ ラテンアルファベット(A–Z、a–z)またはアンダースコアの文字で始める必要があります。 ■ 大文字と小文字を混在させることができます。SAS は、変数への最初の参照で使用されるのと同じ大文字小文字で変数名を保存および書き込みます。 SAS は内部ですべての変数名を大文字に変換することに注意してください。 たとえば、cat、Cat、および CAT という名前はすべて同じ変数を表します。

次の表は、最高レベルの柔軟性が許可される場合(つまり、VALIDVARNAME=システムオプションが ANY に設定されている場合)の、SAS 変数の命名規則の概要を示しています。

表 3.5　SAS 変数の命名に関する拡張規則の概要

変数名 (VALIDVARNAME=ANY を使用)
■ 長さは最大 32 バイトです。 ■ 次のような特殊文字を含めることができます(/ \ * ? " < > | : -)。特殊文字を含む名前は、名前リテラルとして指定する必要があります。 ■ 空白、各国文字、特殊文字、マルチバイト文字など、任意の文字で始めることができます。 ■ 先頭の空白は保持されますが、末尾の空白は無視されます。 ■ 大文字と小文字を混在させることができます。SAS は、変数への最初の参照で使用されるのと同じ大文字小文字で変数名を保存および書き込みます。 SAS はすべての変数名を大文字に変換することに注意してください。 たとえば、cat、Cat、および CAT という名前はすべて同じ変数を表します。 ■ すべての空白を含めることはできません。

変数名 (VALIDVARNAME=ANY を使用)

重要　SAS 全体で、32 バイト制限を超える変数名、または過剰な引用符が埋め込まれた変数名で名前リテラル構文を使用すると、予期しない結果が発生する可能性があります。VALIDVARNAME=ANY システムオプションの目的は、埋め込み空白や各国文字を許可するなど、他の DBMS 変数(列)命名規則との互換性を有効にすることです。

注: VALIDVARNAME=V7 で、文字、数字、アンダースコア以外の文字を使用する場合は、変数名を名前リテラルとして表現し、VALIDVARNAME=ANY を設定する必要があります。名前にパーセント記号(%)またはアンパサンド(&)が含まれている場合は、SAS マクロ機能との相互作用を避けるために、名前リテラルで単一引用符を使用する必要があります。SAS 名リテラルおよび 名前リテラル使用時のエラーの回避を参照してください。

関連項目

例

- “例: 空白を含む変数名の作成”
- “例: FIRST.および LAST.BY 変数を名前リテラルとして指定する”
- “例: 特殊文字を含む SAS データセット名の作成”

ステートメント

- “OPTIONS ステートメント” (*SAS グローバルステートメント: リファレンス*)

システムオプション

- “VALIDMEMNAME= システムオプション” (*SAS システムオプション: リファレンス*)
- “VALIDVARNAME= システムオプション” (*SAS システムオプション: リファレンス*)

メンバー名

メンバー名の規則

メンバー名には、SAS データセット名、ビュー名、およびアイテムストア名が含まれます。

SAS メンバー名の規則が拡張され、より多くの機能が提供されるようになりました。"VALIDMEMNAME= システムオプション" (*SAS システムオプション: リファレンス*) の設定により、SAS セッションで作成するメンバー名に適用する規則を決定します。VALIDMEMNAME=システムオプションには 3 つの設定があり、それぞれメンバー名の柔軟性が異なります。SAS セッションで VALIDMEMNAME=システムオプションを指定しない場合、デフォルトは COMPATIBLE です。

表 3.6 SAS メンバー名の命名に関するデフォルト規則の概要

メンバー名 (VALIDMEMNAME=COMPATIBLE を使用)
■ 名前の長さは最大 32 文字です。 ■ 名前は、ラテンアルファベット(A–Z、a–z)またはアンダースコアで始める必要があります。後続の文字には、ラテンアルファベット、数字、またはアンダースコアを使用できます。 ■ 名前には、空白やアンダースコアを除く特殊文字を含めることはできません。 ■ 名前には大文字と小文字を混在させることができます。SAS は内部でメンバー名を大文字に変換します。たとえば、customer、Customer、および CUSTOMER という名前はすべて同じメンバー名を表します。Linux 環境では大文字と小文字が区別されるため、オペレーティングシステムレベルの関連ファイルの名前は小文字で構成されることに注意してください。

SAS メンバー名の拡張規則

次の表は、最高レベルの柔軟性が許可されている場合(つまり、VALIDMEMNAME=システムオプションが EXTEND に設定されている場合)の、SAS メンバー名の命名規則の概要を示しています。

表 3.7 SAS メンバー名の命名に関する拡張規則の概要

変数名 (VALIDMEMNAME=EXTEND を使用)
■ 名前には各国文字を含めることができます。 ■ 名前には、/ \ * ? " < > |: -. 文字を除く特殊文字を含めることができます。 **注:** SPD エンジンでは、メンバー名のどこにも'.'(ピリオド) は使用できません。 ■ 名前には、少なくとも 1 つの文字(文字、数字、有効な特殊文字、各国文字)が含まれている必要があります。 ■ 名前の長さは最大 32 バイトです。 ■ null バイトは許可されません。名前は空白または '.'(ピリオド)で始めることはできません。

変数名 (VALIDMEMNAME=EXTEND を使用)

注: SPD Engine では、メンバー名の最初の文字として'$'を使用することはできません。

- 先頭と末尾の空白は、メンバーの作成時に削除されます。
- 名前には大文字と小文字を混在させることができます。SAS は内部でメンバー名を大文字に変換します。customer、Customer、および CUSTOMER という名前はすべて同じメンバー名を表すことに注意してください。外部的には、データセット名が大文字で作成されている場合でも、SAS は Linux 上のファイル名に小文字を使用します。

重要　SAS 全体で、32 バイト制限を超える変数名、または過剰な引用符が埋め込まれた変数名で名前リテラル構文を使用すると、予期しない結果が発生する可能性があります。VALIDVARNAME=ANY システムオプションの目的は、埋め込み空白や各国文字を許可するなど、他の DBMS 変数(列)命名規則との互換性を有効にすることです。

注: VALIDMEMNAME=システムオプションの値に関係なく、メンバー名の末尾に特殊文字#とそれに続く 3 桁の数字を付けることはできません。これは、世代データセットの命名規則と矛盾するためです。このようなメンバー名を使用するとエラーが発生します。

注: VALIDMEMNAME=EXTEND の場合、SAS データセット名、SAS データビュー名、およびアイテムストア名は、名前に空白スペース、特殊文字、または各国文字が含まれているときは、SAS 名リテラルとして書き込む必要があります。パーセント記号(%)またはアンパサンド(&)を使用する場合は、SAS マクロ機能との相互作用を避けるために、名前リテラルで単一引用符を使用する必要があります。詳細については、"SAS Name Literals"を参照してください。

注: Linux 動作環境の場合、SAS ファイルを物理名で直接参照する場合、最後に埋め込まれたピリオドは拡張子区切り文字になります。物理ファイル参照にピリオドを含む SAS メンバー名が含まれている場合は、ファイル拡張子を追加する必要があります。たとえば、データセット名 my.member を物理ファイルとして参照する場合、SET ステートメント set './saslib/my.member.sas7bdat';に示すように、ファイル拡張子 sas7bdat を参照に追加します。

関連項目

ステートメント

- "OPTIONS ステートメント" (*SAS グローバルステートメント: リファレンス*)

システムオプション

- "VALIDMEMNAME= システムオプション" (*SAS システムオプション: リファレンス*)
- "VALIDVARNAME= システムオプション" (*SAS システムオプション: リファレンス*)

データセット名

データセット名の定義

データセット名は、SAS データセットの作成時に付けるユーザー指定の名前です。DATA ステップで作成する出力データセットには、DATA ステートメントで名前が付けられます。プロシジャステップで作成する SAS データセットには、通常、プロシジャステートメントまたは OUTPUT ステートメントで名前が付けられます。

SAS データセットに名前を付けるときは、SAS メンバー名の命名規則に従ってください。詳細については、メンバー名の規則 (47 ページ)を参照してください。

DATA ステートメントで出力データセットの名前を指定しない場合、SAS はデフォルトのデータセット名を自動的に割り当てます。SET ステートメントで入力データセットの名前を指定しない場合、SAS は最後に作成されたデータセットを自動的に使用します。詳細については、“特殊データセット名” (52 ページ)を参照してください。

データセットの名前の指定が必要になる可能性のあるステートメントをいくつか示します。

- DATA= option
- MERGE ステートメント
- MODIFY ステートメント
- OPEN 関数
- SET ステートメント
- SQL プロシジャ
- UPDATE ステートメント

SAS ビュー名は、次のいずれかを使用して作成されます。

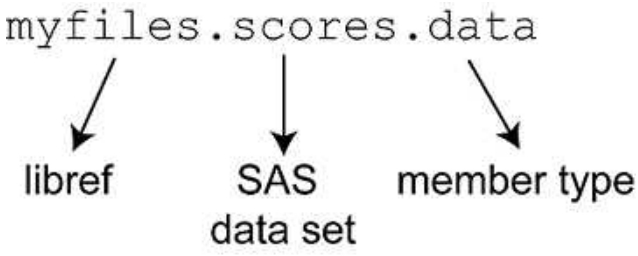
- “DATA ステートメント” (*SAS DATA ステップステートメント: リファレンス*)の VIEW=オプション
- ACCESS プロシジャ

注: 同じライブラリを共有する SAS データセットと SAS®Views®に同じ名前を付けることはできません。

データセット名の構造

レベル

すべての SAS データセットの完全名には 3 つのレベルがあります。最初の 2 つのレベルを割り当てると、SAS が 3 番目のレベルを提供します。SAS データセット名の形式は次のとおりです。



ライブラリ参照名(ライブラリ名)
SAS ライブラリの物理的な場所に関連付けられたユーザー指定の論理名です。

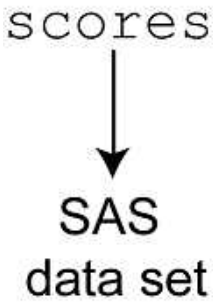
SAS データセット
ユーザー指定のデータセット名で、バージョン 7 以降の Base SAS エンジンでは最大 32 バイトの長さになります。以前の SAS バージョンでは、名前が 8 バイトに制限されていました。

メンバータイプ
SAS ファイルに保存される情報の種類を識別する、SAS によって割り当てられるシステム作成の名前です。たとえば、SAS データセットのタイプは DATA で、SAS DATA ステップビューのタイプは VIEW です。他のメンバータイプは、ACCESS、AUDIT、DMBD、CATALOG、FDB、INDEX、ITEMSTOR、MDDB、PROGRAM、UTILITY です。

プログラムステートメントで SAS データセットを参照する場合は、1 レベルまたは 2 レベルの名前を使用します。データセットが一時ライブラリ Work にある場合は、1 レベル名を使用できます。さらに、予約されたライブラリ参照名 User が割り当てられている場合、データセットが永久ライブラリ User にあるときは、1 レベル名を使用できます。データセットが、確立した他の永久ライブラリにある場合は、2 レベル名を使用します。

1 レベル名

1 レベル名はデータセット名のみで構成されます。ライブラリ参照名を省略し、次の形式で 1 レベル名を使用してデータセットを参照できます。



1 レベル名を持つデータセットは、次の 2 つの SAS ライブラリのいずれかに自動的に割り当てられます。

Work
: データセットを 1 レベル名で一時的に保存するために使用される SAS ライブラリ。一時的とは、SAS ジョブまたはセッションの終了時にライブラリの内容が削除されることを意味します。1 レベル名を持つデータセットは、デフォルトで Work ライブラリに保存されます。

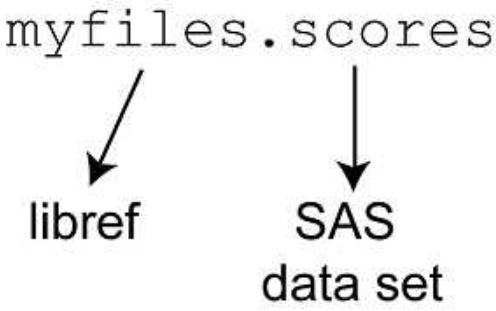
User
: データセットを 1 レベル名で永久保存するために使用される SAS ライブラリ。

最も一般的には、これらは一時ライブラリ Work に割り当てられ、SAS ジョブまたはセッションの終了時に削除されます。ライブラリ参照名 User を SAS ライブラリに関連付けている場合、または USER=システムオプションを使用して User ライブラリを設定した場合、1 レベル名を持つデータセットがそのライブラリに保存されます。

Work ライブラリと User ライブラリの使用方法の詳細については、“User ライブラリ”および “Work ライブラリ(一時)”を参照してください。

2 レベル名

2 レベル名は、ライブラリ参照名とデータセット名の両方で構成されます。永久 SAS ライブラリで SAS データセットを作成、読み取り、または書き込みするために最も一般的に使用される形式は、次に示すように 2 レベル名です。



新しい SAS データセットを作成するとき、ライブラリ参照名はそれが保存される場所を示します。既存のデータセットを参照すると、ライブラリ参照名は SAS にそのデータセットの場所を指示します。

特殊データセット名

予約語は、システム使用のみを目的として"予約"されている SAS のキーワードであり、データセットの名前付けには使用できません。次の特殊データセット名は、SAS の *予約語* です。

- _DATA_
- _LAST_
- _NULL_

_DATA_データセット

出力データセットの名前を指定しない場合、または DATA ステートメントで予約名 _DATA_ を指定しない場合は、SAS によって DATA1、DATA2、... という名前が自動的に割り当てられます。DATA9999 で、その後作成する各データ セットの名前は DATA1 から再開されます。

この機能は、DATA *n* 命名規則と呼ばれます。

たとえば、次のプログラムを実行すると、SAS は WORK ライブラリに 3 つのデータセットを作成し、DATA1、DATA2、および DATA3 という名前を付けます。

```
data _data_;
  x=1;
run;

data;
  y=2;
run;

data _data_;
  z=3;
run;
```

この機能を使用すると、データを上書きする危険を冒さずに、出力データセットの命名を自動化できます。名前は、WORK ライブラリに書き込まれる時点では一意であり、SAS セッション中に数値的に増加し続けます。

SAS DATA*n* 命名規則の使用方法を示すいくつかの実際的な例については、Splitting a data set into smaller data sets を参照してください。

_LAST_データセット

SET ステートメントで入力データセットの名前を指定しない場合、SAS は最後に作成されたデータセットを自動的に使用します。SAS は、予約名_LAST_を通じて、最後に作成されたデータセットを追跡します。入力データセットを指定せずに DATA

または PROC ステップを実行すると、SAS はデフォルトで_LAST_データセットを使用します。

_LAST_予約名を明示的に使用することもできます。たとえば、次の 3 つの PROC PRINT ステートメントは、同じ出力を生成するため同等です。

```
data A;
  x = 1;
run;

proc print data=A;
run;

proc print;
run;

proc print data=_last_;
run;
```

LAST=システムオプションを使用すると、データセットを_LAST_データセットとして指定できます。

SAS DATA*n* 命名規則の使用方法を示す実際的な例については、Splitting a data set into smaller data sets を参照してください。

参照

_NULL_データセット

DATA ステップを実行するが、SAS データセットを作成しない場合は、データセット名としてキーワード_NULL_を指定できます。次のステートメントは、データセットを作成しない DATA ステップを開始します。

```
data _null_;
```

_NULL_を使用すると、SAS は新しいデータセットを作成しているかのように DATA ステップを実行しますが、オブザベーションや変数は出力データセットに書き込まれません。レポート作成など、DATA ステップの出力を SAS データセットとして保存する必要がない機能に DATA ステップを使用している場合、このプロセスはコンピューターリソースをより効率的に使用できます。

SAS データセットと変数の命名規則

以下の表は、VALIDMEMNAME=システムオプションが COMPATIBLE に設定され、VALIDVARNAME=システムオプションが V7 に設定されている場合の、SAS データセットの命名規則の概要を示しています。これらは Base SAS のデフォルト設定です。SAS Visual Analytics などの一部の SAS アプリケーションでは、SAS 変数およびデータセットの命名を最も柔軟に行えるように、VALIDMEMNAME=および VALIDVARNAME=システムオプションがデフォルトで設定されています。これらの規則を表 3.8 にまとめます。

表 3.8　SAS データセットと変数の命名に関するデフォルト規則の概要

SAS データセット名、ビュー名、およびアイテムストア名 (VALIDMEMNAME=COMPAT を使用)	変数名 (VALIDVARNAME=V7 を使用)
■ 長さは最大 32 バイトです。 ■ 特殊文字(アンダースコアを除く)、空白、または各国文字を含めることはできません。 ■ ラテンアルファベット(A–Z、a–z)またはアンダースコアの文字で始める必要があります。 ■ 大文字と小文字を混在させることができます。SAS は内部でメンバー名を大文字に変換します。したがって、大文字と小文字の組み合わせが異なる同じメンバー名を使用して、異なるメンバーを表すことはできません。たとえば、cat、Cat、および CAT はすべて同じメンバー名を表します。ディスク上の名前の表示方法は、動作環境によって決まります。	■ 長さは最大 32 バイトです。 ■ 特殊文字(アンダースコアを除く)、空白、または各国文字を含めることはできません。 ■ ラテンアルファベット(A–Z、a–z)またはアンダースコアの文字で始める必要があります。 ■ 大文字と小文字を混在させることができます。SAS は、変数への最初の参照で使用されるのと同じ大文字小文字で変数名を保存および書き込みます。ただし、SAS が変数名を処理するときは、内部で変数名を大文字に変換します。したがって、大文字と小文字の組み合わせが異なる同じ変数名を使用して、異なる変数を表すことはできません。たとえば、cat、Cat、および CAT はすべて同じ変数を表します。

重要　Linux 動作環境では、SAS はすべて小文字で書かれたデータセット名のみを読み取ります。

SAS データセットと変数の命名に関する拡張規則

次の表は、VALIDMEMNAME=システムオプションが EXTEND に設定され、VALIDVARNAME=システムオプションが ANY に設定されている場合の、SAS データセットの命名規則の概要を示しています。

表 3.9　SAS データセットと変数の命名に関する拡張規則の概要

SAS データセット名、ビュー名、およびアイテムストア名 (VALIDMEMNAME=EXTEND を使用)	変数名 (VALIDVARNAME=ANY を使用)
■ 長さは最大 32 バイトです。 ■ / \ * ? " < > | : - を除く特殊文字を含めることができます。特殊文字を含む名前	■ 長さは最大 32 バイトです。 ■ 次のような特殊文字を含めることができます(/ \ * ? " < > | : -)。特殊文

SAS データセット名、ビュー名、およびアイテムストア名 (VALIDMEMNAME=EXTEND を使用)	変数名 (VALIDVARNAME=ANY を使用)
は、名前リテラルとして指定する必要があります。 ■ 空白またはピリオドで始めることはできません。 ■ 先頭と末尾の空白は無視されます。 ■ 大文字と小文字を混在させることができます。SAS は内部でメンバー名を大文字に変換します。したがって、大文字と小文字の組み合わせが異なる同じメンバー名を使用して、異なる変数を表すことはできません。たとえば、cat、Cat、および CAT はすべて同じメンバー名を表します。 [1]	字を含む名前は、名前リテラルとして指定する必要があります。 ■ 空白、各国文字、特殊文字、マルチバイト文字など、任意の文字で始めることができます。 ■ 先頭の空白は保持されますが、末尾の空白は無視されます。 ■ 大文字と小文字を混在させることができます。SAS は、変数への最初の参照で使用されるのと同じ大文字小文字で変数名を保存および書き込みます。ただし、SAS が変数名を処理するときは、内部で変数名を大文字に変換します。したがって、大文字と小文字の組み合わせが異なる同じ変数名を使用して、異なる変数を表すことはできません。たとえば、cat、Cat、および CAT はすべて同じ変数を表します。 ■ すべての空白を含めることはできません。

重要　Linux 動作環境では、SAS はすべて小文字で書かれたデータセット名のみを読み取ります。

関連項目

例

- “例: 特殊文字を含む SAS データセット名の作成”
- “例: 1 レベルデータセット名の作成”
- “例: データセットの命名に自動命名機能を使用する”
- “例: _LAST_=システムオプションを使用してデフォルト入力データセットを指定する”

関数

- OPEN 関数

1. Linux 動作環境では、SAS はすべて小文字で書かれたデータセット名のみを読み取ります。

ステートメント

- DATA ステートメント
- SET ステートメント
- MERGE ステートメント
- MODIFY ステートメント
- “OPTIONS ステートメント” (*SAS グローバルステートメント: リファレンス*)
- UPDATE ステートメント

システムオプション

- _LAST_=システムオプション
- USER=システムオプション
- “VALIDMEMNAME= システムオプション” (*SAS システムオプション: リファレンス*)
- “VALIDVARNAME= システムオプション” (*SAS システムオプション: リファレンス*)
- VIEW=オプション

例: SAS の単語と名前

例: 空白を含む変数名の作成

コード例

次の例は、空白を含む変数名の作成を示しています。

```
options validvarname=any;   /* 1 */
data mydata;
  'First Name'n='John';    /* 2 */
run;
```

1 VALIDVARNAME=システムオプションを OPTIONS ステートメントで指定します。VALIDVARNAME=ANY を使用すると、空白、特殊文字、各国文字、マルチバイト文字を含む変数名を作成できます。

2 変数名を*名前リテラル*として指定します。名前リテラルを使用すると、変数または SAS データセットを指定するときに *SAS 名*に特殊文字を使用できます。名前リテラルは、名前を引用符で囲み、その後に文字 *n* を続けることによって作成されます。

要点

- VALIDVARNAME=システムオプションは、SAS セッションで作成および処理できる変数にどの規則を適用するかを決定します。
- VALIDVARNAME=システムオプションがデフォルト設定(V7)に設定されている場合に有効な文字以外の文字を使用する場合は、変数名を名前リテラルとして表現し、VALIDVARNAME=ANY を設定する必要があります。
- 名前リテラルを指定する場合、終わり引用符と *n* の間の空白は無効です。

関連項目

- “SAS 名リテラルの定義”
- “OPTIONS ステートメント” (*SAS グローバルステートメント: リファレンス*)
- “ほとんどの SAS 名の規則”
- “変数名の規則”
- “VALIDVARNAME= システムオプション” (*SAS システムオプション: リファレンス*)

例: 特殊文字を含む SAS データセット名の作成

コード例

次の例は、特殊文字$を含む新しい SAS データセット名の作成を示しています。

```
options validmemname=extend;   /* 1 */
data 'my$data'n;               /* 2 */
  x=1;
  y=3;
  sum=x+y;
run;
```

1 VALIDMEMNAME=システムオプション OPTIONS ステートメントを指定します。VALIDMEMNAME=EXTEND を使用すると、/ \ * ? " < > | : -を除く特殊文字を含む SAS データセット名を作成できます。

2 データセット名を*名前リテラル*として指定します。名前リテラルを使用すると、SAS データセット名または変数を指定するときに、SAS 名に特殊文字を使用でき

ます。名前リテラルは、名前を引用符で囲み、その後に文字 *n* を続けることによって作成されます。

要点

- SAS *名リテラル*は、引用符で囲まれた*文字列*として表現され、その後に大文字または小文字の *n* が続くユーザー指定の名前です。
- 名前リテラルを使用すると、他の方法では許可されない文字(空白および各国文字を含む)を使用できるようになります。
- 特殊文字を含む SAS データセット名を指定する場合は、VALIDMEMNAME=EXTEND システムオプションを使用します。

関連項目

- “SAS 名リテラルの定義”
- “SAS データセットと変数の命名に関する拡張規則”
- “OPTIONS ステートメント” (*SAS グローバルステートメント: リファレンス*)
- “VALIDMEMNAME= システムオプション” (*SAS システムオプション: リファレンス*)

例: 単語の配置と間隔の管理

コード例

次の例では、SAS は次の単語の先頭を調べることですべての 単語の境界を判断できるため、空白は必要ありません。

```
total=x+y;
```

次の例では、等号は単語 total の終わりを示しています。もう 1 つの特殊文字であるプラス記号は、単語 x の終わりを示します。最後の特殊文字であるセミコロンは、y という単語の終わりを示します。この例では単語の末尾に空白は必要ありませんが、読みやすくするために空白を追加できます。

```
total = x + y;
```

次の例では、SAS はスペースがないと 個々の単語を認識できないため、空白が必要です。空白がない場合、セミコロンまでのステートメント全体が名前の規則に適合

します。したがって、ステートメントでは、個々の名前と番号を区別するために空白が必要です。

```
input group 15 room 20;
```

要点

- SAS プログラミング言語の単語は、SAS に意味を伝える文字の集合です。
- 単語を独立して使用できる小さな単位に分割することはできません。1 つの単語には最大 32,767 バイトを含めることができます。
- 名前は、文字またはアンダースコアで始まる一連の文字です。以降の文字には、文字、アンダースコア、および数字を含めることができます。名前には最大 32,767 バイトを含めることができます。ただし、ほとんどのコンテキストでは、名前は 32 バイトや 8 バイトなど、より短い最大長に制限されます。

関連項目

- “SAS 単語の定義” (34 ページ)
- “単語またはトークンの種類”
- “ほとんどの SAS 名の規則” (36 ページ)

例: FIRST.および LAST.BY 変数を名前リテラルとして指定する

コード例

次の例は、FIRST.および LAST 変数を名前リテラルとして指定する方法を示しています。BY 変数を名前リテラルとして指定する:

```
options validvarname=any;              /* 1 */
data sedanTypes;
  set cars;
  by 'Sedan Types'n;                   /* 2 */
  if 'first.Sedan Types'n then type=1;   /* 3 */
    else type=2;
run;
```

1 VALIDVARNAME=システムオプションを OPTIONS ステートメントで指定します。VALIDVARNAME=ANY を使用すると、空白、特殊文字、各国文字、マルチバイト文字を含む変数名を作成できます。

2 変数 Sedan Types には 2 つの単語の間に空白が含まれています。したがって、変数名を引用符で囲み、その後に文字 *n* を付けて変数を名前リテラルとして宣言します。

3 BY グループ処理において、BY 変数として名前リテラルを指定し、対応する FIRST.または LAST.一次変数を参照する場合は、2 レベル変数名の FIRST.または たは LAST.部分を引用符で囲んで使用します。

要点

- VALIDVARNAME=システムオプションは、SAS セッションで作成および処理できる変数にどの規則を適用するかを決定します。
- VALIDVARNAME=システムオプションがデフォルト設定(V7)に設定されている場合に有効な文字以外の文字を使用する場合は、変数名を名前リテラルとして表現し、VALIDVARNAME=ANY を設定します。
- 変数名に特殊文字がある場合は、FIRST.変数と LAST.変数を名前リテラルとして指定します。
- 名前リテラルを指定する場合、終わり引用符と *n* の間の空白は無効です。

関連項目

- “BY ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “OPTIONS ステートメント” (*SAS グローバルステートメント: リファレンス*)
- “VALIDMEMNAME= システムオプション” (*SAS システムオプション: リファレンス*)
- “VALIDVARNAME= システムオプション” (*SAS システムオプション: リファレンス*)
- “BY グループでの名前リテラルの使用”

例: 1 レベルデータセット名の作成

コード例

次の例は、SAS ステートメントでの 1 レベル名の使用を示しています。

```
options user='/u/temp';   /* 1 */
data test;              /* 2 */
  x=1;
run;
data samptest;          /* 3 */
  x=1;
  y=6;
  s=x*y+y;
run;
```

1 USER=システムオプションは、デフォルトの永久 SAS ライブラリの名前を指定します。USER=システムオプションを指定すると、1 レベル名で作成したデータセットは、指定したライブラリに永久保存されます。

2 DATA ステートメントは、SAS システムライブラリ User に永久データセット Test を作成します。データセット名の一部として User を参照する必要はないことに注意してください。

3 DATA ステートメントは、現在のディレクトリに永久データセット Samptest を作成します。

注: SAS が User ライブラリに Samptest を作成するために、DATA ステップの前に USER=システムオプションを参照したり、データセット名の一部として User を参照したりする必要はありません。

次のように SAS ログに書き込まれます。

アウトプット 3.1　SAS ログ: 1 レベルデータセット名

```
85   options user='/u/temp';
86   data test;
87   x=1;
88   run;

NOTE: The data set USER.TEST has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


89   data samptest;
90   x=1;
91   y=6;
92   s=x*y+y;
93   run;

NOTE: The data set USER.SAMPTEST has 1 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

要点

- 1 レベル名を持つデータセットには、Work ライブラリまたは User ライブラリのいずれかが自動的に割り当てられます。ライブラリ参照名を省略し、1 レベル名を使用してデータセットを参照できます。
- 最も一般的には、データセットは一時ライブラリ Work に割り当てられ、SAS ジョブまたはセッションの終了時に削除されます。
- ライブラリ参照名 User を SAS ライブラリに関連付けている場合、または USER=システムオプションを使用して User ライブラリを設定した場合、1 レベル名を持つデータセットがそのライブラリに保存されます。

関連項目

- DATA ステートメント
- 1 レベル名
- OPTIONS ステートメント
- データセット名の構造
- USER=システムオプション

例: 2 レベルデータセット名の作成

コード例

次の例は、*SAS ステートメント*での 2 レベル名の使用を示しています。

```
libname finance '/u/finance';   /* 1 */
data finance.balances;          /* 2 */
  rate=0.03;
  months=12;
  balance=1000;
  endbal=(rate*months)+balance;
run;
```

1 LIBNAME ステートメントは、SAS ライブラリを *ライブラリ参照名* Finance に関連付けます。後続の DATA または PROC ステップで *ライブラリ参照名*を参照して、永久ライブラリ内で SAS データセットを作成、読み取り、または書き込みできます。

2 DATA ステートメントは、Finance *ライブラリ参照名*を参照し、Finance ライブラリに新しい SAS データセットを作成し、新しいデータセットに Balances という名前を付けるように SAS に指示します。新しいデータセットには 4 つの変数と 1 つのオブザベーションが含まれています。

次のように SAS ログに書き込まれます。

アウトプット 3.2　SAS ログ: 2 レベルデータセット名

```
94   libname finance '/u/finance;
NOTE: Libref FINANCE was successfully assigned as follows:
      Engine:        V9
      Physical Name: /u/finance
95
96   data finance.balances;
97     rate=.03;
98     months=12;
99     balance=1000;
100    endbal=(rate*months)+balance;
101  run;

NOTE: The data set FINANCE.BALANCES has 1 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time           0.06 seconds
      cpu time            0.01 seconds
```

要点

- 2 レベルデータセット名は、永久 SAS ライブラリで SAS データセットを作成、読み取り、または書き込みするために最も一般的に使用される形式です。
- 新しい SAS データセットを作成するとき、ライブラリ参照名はそれが保存される場所を示します。既存のデータセットを参照すると、ライブラリ参照名は SAS にそのデータセットの場所を指示します。

関連項目

- DATA ステートメント
- LIBNAME ステートメント
- データセット名の構造
- 2 レベル名

例: データセットの命名に自動命名機能を使用する

コード例

次の例は、DATAn 自動命名機能の SAS DATA ステップでの使用を示しています。出力データセットの名前を指定しないか、DATA (または大文字と小文字が区別されないため data)として指定した場合、SAS は作成したデータセットに DATA1、DATA2 などとして DATA9999 までの名前を自動的に割り当てます。SAS は、名前のないすべてのデータセットに連続した名前を割り当てます。

この例には、3 つの名前なし出力データセットと 1 つの名前付きデータセットが含まれています。

```
data;
  setsashelp.class;
run;

data DATA;
  set sashelp.air;
run;

data sample;
  set sashelp.cars;
run;
```

```
data data;
  setsashelp.flags;
run;
```

次のように SAS ログに書き込まれます。

アウトプット 3.3　SAS ログ: DATA*n* 自動命名機能を使用する

```
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set WORK.DATA1 has 19 observations and 5 variables.
NOTE: There were 144 observations read from the data set SASHELP.AIR.
NOTE: The data set WORK.DATA2 has 144 observations and 2 variables.
NOTE: There were 428 observations read from the data set SASHELP.CARS.
NOTE: The data set WORK.SAMPLE has 428 observations and 15 variables.
NOTE: There were 220 observations read from the data set SASHELP.FLAGS.
NOTE: The data set WORK.DATA3 has 220 observations and 7 variables.
```

要点

- DATA ステートメントで SAS データセット名を指定しない場合、SAS は DATA1、DATA2 などの名前を持つデータセットを連続したデータセットに自動的に作成します。
- 自動的に名前が付けられたデータセットは、Work ライブラリまたは User ライブラリに保存されます。
- 自動的に名前が付けられたデータセットを永久ライブラリに保存しない場合、SAS セッションの終了時にデータセットが削除されます。

関連項目

- DATAn データセット
- DATA ステートメント
- SET ステートメント

例: _LAST_=システムオプションを使用してデフォルト入力データセットを指定する

コード例

次の例は、_LAST_=システムオプションを使用して、現在の SAS セッションのデフォルト入力データセットを指定する方法を示しています。

```
options _last_=mysas.class;   /* 1 */
libname mysas '/u/Users/';
data mysas.class;             /* 2 */
  set sashelp.class;
run;
data mysas.test;              /* 3 */
  set;
run;
```

1　_LAST_=システムオプションを使用すると、データセットを_LAST_データセットとして指定できます。指定した名前は、新しいデータセットを作成するまでデフォルトのデータセットとして使用されます。多数のプロシジャステップを含む SAS ジョブに既存の永久データセットを使用する場合は、_LAST_=システムオプションを使用できます。_LAST_=システムオプションを発行すると、各プロシジャステートメントで SAS データセット名を指定する必要がなくなります。

2　最初の DATA ステップでは、LAST_=システムオプションで参照される mysas.class という名前のデータセットを作成します。

3　2 番目の DATA ステップでは、mysas.test という名前の永久データセットを作成しますが、SET ステートメントではデータセットを指定しません。次に、SET ステートメントは_LAST_=システムオプションを実行し、入力データセットを mysas.class として設定します。

アウトプット 3.4　SAS ログ出力

```
NOTE: There were 19 observations read from the data set MYSAS.CLASS.
 NOTE: The data set MYSAS.CLASS has 19 observations and 5 variables.
```

要点

- SAS は、予約名_LAST_を通じて、最後に作成された SAS データセットを追跡します。

- 入力データセットを指定せずに DATA または PROC ステップを実行すると、SAS はデフォルトで_LAST_データセットを使用します。一部の関数では、_LAST_デフォルトも使用されます
- _LAST_=システムオプションを使用すると、データセットを_LAST_データセットとして指定できます。指定した名前は、新しいデータセットを作成するまでデフォルトのデータセットとして使用されます。
- 多数の DATA ステップまたはプロシジャステップを含む SAS ジョブに既存の永久データセットを使用する場合は、_LAST_=システムオプションを使用できます。
- _LAST_=システムオプションを指定すると、各 DATA ステップまたはプロシジャステートメントで SAS データセット名を指定する必要がなくなります。

関連項目

- データセット名
- _LAST_データセット
- LIBNAME ステートメント
- OPTIONS ステートメント
- SET ステートメント

4

変数

SAS 変数の定義

次の定義では、SAS 変数と DATA ステップでのそれらの動作に関連する重要な用語について説明します。

SAS variable
: ユーザーが値の保存および操作のために SAS プログラム内で作成する名前付きデータコンテナーです。SAS では、変数は CHARACTER (CHAR)または DOUBLE (現在の数値データ型)のいずれかになります。各変数には、データの格納方法と表示方法を制御する名前、長さ、出力形式、ラベルなどの追加の属性"変数属性"もあります。SAS データ型の詳細については、"データ型" (72 ページ)を参照してください。

automatic DATA step variable
: DATA ステップの実行時に SAS が自動的に作成するシステム生成変数(_N_や_ERROR_など)。これらの変数はプログラムデータベクター(PDV)に存在しますが、出力データセットには書き込まれません。自動 DATA ステップ変数をプログラムのロジックで使用し、フローの制御、問題の検出、条件付き出力の生成を行うことができますが、新しい変数に明示的に割り当てられない限り、出力データセットには表示されません。詳細については、"例: 自動変数の使用" (135 ページ)を参照してください。

uninitialized SAS variable
: SAS に値が割り当てられる前に使用される変数。これは、変数が入力データセットになく、DATA ステップ内で定義されていない場合に発生します。この場合、ログに NOTE が書き込まれます。

numeric precision
: 数値変数に保存できる有効桁数。SAS では、数値変数は浮動小数点表現(通常は 64 ビット)を使用して保存され、最大 15 桁または 16 桁の精度が提供されます。数値精度詳細については、"数値精度" (100 ページ)を参照してください。

データ型

定義

データ型とは、変数が保存するデータ型を定義するすべての SAS 変数の属性です。データ型は、データを文字列、浮動小数点数、または日時として識別します。データ型によって、変数の値に割り当てるメモリの量も決まります。Base SAS では、次のデータ型が DATA ステップでサポートされています。

SAS データ型の概要

表 4.1　SAS Compute Server がサポートするデータ型

データ型	説明	ストレージの長さ	数値精度	数値範囲
文字	固定長の文字列を保存します。長さは 1 から 32,767 バイトまでで、変数を作成するときに指定する必要があります。短い値は、定義された長さになるように空白が右埋め込まれます。	範囲: 1 から 32,767 バイト	適用できません	適用できません
DOUBLE	64 ビット、倍精度、2 進浮動小数点数を保存します。わずかな精度の損失の可能性はありますが、精度の高い非常に大きな数や非常に小さい数を処理できます。	8 バイト	16 桁	$\pm 2.23\times10^{-308}$ - $\pm 1.80\times10^{308}$

注: SAS では日付と時間の値を処理するための関数と出力形式が提供されていますが、これらは個別のデータ型ではありません。これらは、数値の上に階層化されてセマンティックに解釈される出力形式です。フォーマットは、値の表示方法に影響を与えますが、SAS での値の保存方法や処理方法には影響しません。

また、日付、時刻、時刻を処理するための特殊な形式もサポートしています。

- 日付: 1960 年 1 月 1 日からの日数を表す数値としてカレンダー日付を保存します。
- 時間: 現在の日の午前 0 時からの秒数を表す数値として時間値を保存します。
- 日時: 1960 年 1 月 1 日の午前 0 時からの秒数を表す数値として日時値を保存します。

詳細については、"SAS 日付値、時間値、日時値について" (*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。

入力形式を使用した数値データの読み込みと書き込み

標準数値データを読み取る場合、特殊な命令を指定する必要はありません。他のタイプの値を読み取るには、入力形式の形式で特殊な命令を指定する必要があります。表 4.3 (74 ページ)は、さまざまなタイプのデータ値と、それらを読み取るために必要な特別なツールを示します。SAS 入力形式については、*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。

表 4.2 標準数値データの読み取り

データ	説明	入力形式の可否
23	入力右揃え	不要
23	入力非整列	不要
23	入力左揃え	不要
00023	先頭にゼロを含む入力	不要
23.0	小数点付き入力	不要
2.3E1	指数表記。2.30×10^{1}	不要
230E-1	指数表記。230×10^{-1}	不要
-23	負の数のマイナス記号	不要

表 4.3　文字または 2 進値を含む数値データの読み取り

データ	説明	対策
2 3	空白を含む	COMMA.入力形式、または BZ.入力形式
- 23	空白を含む	COMMA.入力形式、または BZ.入力形式
2,341	カンマ	COMMA.入力形式
(23)	かっこ	COMMA.入力形式
C4A2	16 進値	HEX.入力形式
1MAR90	日付値	DATE.入力形式

表 4.4　無効な数値データの読み取り

データ	説明	対策
23 -	数字の後のマイナス記号	数値の前にマイナス記号を付けるか、プログラムで処理してください。S370FZDTw.d 入力形式を使用できる可能性がありますが、正の値には末尾のプラス記号(+)が必要です。 TRAILSGN 入力形式も使用できます。
..	単一ピリオドの代わりに二重ピリオド	欠損値を単一のピリオドとするプログラムを記述するか、INPUT ステートメントで??修飾子を使用して、無効な入力値を欠損値とするプログラムを記述します。
J23	数ではない	文字値として読み取るか、生データを編集して有効な数値に変更します。

文字データ

次のいずれかに該当する場合、INPUT ステートメントで読み取られる値は文字値であると見なされます。

- INPUT ステートメントの変数名の後にドル記号($)が付く。
- 文字入力形式の使用。

- 以前に文字として定義されていた変数。たとえば、変数が以前に LENGTH ステートメント、RETAIN ステートメント、割り当てステートメント、または式で文字として定義されている場合、値は文字値であると見なされます。

文字変数に保存する入力データには、任意の文字を含めることができます。生データに先頭の空白とセミコロンが含まれている場合は、次の表のガイドラインに従ってください。

表 4.5 先頭の空白とセミコロンを含む入力ストリームデータと外部ファイルの読み取り

データの文字	使用方法	理由
先頭または末尾の空白の保持	フォーマットされた入力および$CHARw.入力形式	リスト入力は、値が変数に割り当てられる前に、文字値から先頭と末尾の空白を削除します。
入力ストリームデータのセミコロン	DATALINES4 または CARDS4 ステートメントと、データの終わりを示す 4 つのセミコロン (;;;;)	通常の DATALINES および CARDS ステートメントでは、データ内のセミコロンがデータの終わりを途中で通知します。
区切り文字、空白文字、または引用符で囲まれた文字列	INFILE ステートメントで、DLM=または DLMSTR=オプションを使用した DSD オプション	これらのオプションにより、SAS は、引用符で囲まれた文字列内に区切り記号を含む文字値を読み取ることができます。これらのオプションは、2 つの連続する区切り文字を欠損値として扱い、文字値から引用符を削除することもできます。

CAS および CASL でサポートされているデータ型

CAS サーバーと CAS 言語(CASL)でサポートされているデータ型については、次を参照してください。

- *"データ型" (SAS Cloud Analytic Services: ユーザーガイド)*
- *"CASL データ型" (SAS Cloud Analytic Services: CASL プログラマガイド)*
- *"データコネクタ" (SAS Cloud Analytic Services: ユーザーガイド)*

変数の作成と変更

変数を作成する方法

表 4.6　DATA ステップで変数を作成する方法

方法	例	説明
割り当てステートメント	data vars; a='Tom Hanks'; a='Rita Wilson'; b=100; put a; run; “例: 割り当てステートメントを使用した SAS 変数の作成” (109 ページ)	割り当てステートメントは、a という文字変数を作成します。 LENGTH ステートメントを使用しない場合、最初に検出されたリテラルの長さによって文字変数の長さが決まります。変数 a の PUT ステートメントの出力は、**Rita Wils** として切り捨てられます。
LENGTH ステートメント	data vars; length a $ 4 b $ 6 c $ 2; x=a||b||c; run; “例: LENGTH ステートメントを使用して SAS 変数を作成” (111 ページ)。	LENGTH ステートメントは、3 つの文字変数 a、b、および c を作成します。それぞれの長さは 4、6、および 2 バイトです。割り当てステートメントは変数 x を作成します。長さは **12** で、これは割り当て演算子の右側にあるすべての変数の長さの合計です。 変数 x の値は、変数 a、b、および c を連結したときに生成される値です。
リストされた INPUT ステートメント	data vars; input Gem $ Carats Color $; datalines; emerald 2 green ; 例: INPUT ステートメントを使用した変数の作成	INPUT ステートメントは、次の 2 つの文字変数 Gem と Color、および 1 つの数値(DOUBLE)変数 Carats を作成します。変数は、生データ内の位置に基づいて定義されます。 各変数の長さは 8 バイトです。 INPUT ステートメントを使用して文字変数を作成し、LENGTH ステートメントまたは INPUT ステートメントを使用して変数の長さを指定しない場合、変数の長さ

方法	例	説明
		は自動的に 8 バイトに設定されます。
フォーマットされた入力の INPUT ステートメント	data vars; input Gem $char10. Carats comma3.1 Color $char.; datalines; emerald 2 green ; 例: INPUT ステートメントを使用した変数の作成	INPUT ステートメントは、次の 2 つの文字変数を作成します。Gem は入力形式で指定された長さで作成され、Color は長さ指定なしで作成されます。数値変数 Carats は、長さが指定された数値入力形式を使用して作成されます。 変数の型は、INPUT ステートメントで指定された入力形式のカテゴリに基づいて定義されます。 数値変数 Carats の長さは 8 バイトです。文字変数 Gem の長さは 10 バイトです。文字変数 Color の長さは 8 バイトです。SAS がフォーマットされた入力を使用して作成された変数の長さを決定する方法は、フォーマットされた入力の INPUT ステートメントを使用した新しい変数の作成を参照してください。

注: 明示的または暗黙的に変数を作成する方法は他にもあります。たとえば、SET、MERGE、MODIFY、および UPDATE ステートメントでも変数を作成できます。

これらのカテゴリのリストについては、カテゴリ別入力形式を参照してください。

フォーマットされた INPUT ステートメントを使用した変数の作成

フォーマットされた INPUT ステートメントを使用して新しい変数を作成する場合、変数のデータ型に基づいて変数を入力形式に関連付けます。最初に変数を明示的に定義しない限り、SAS は入力形式を使用してそのデータ型と長さを決定します。

- 生の数値データを読み取るときに、INPUT ステートメントの変数で数値入力形式を使用できます。変数の長さは、INPUT ステートメントの入力形式で指定された幅と一致します。入力形式または DATA ステップの他の場所で長さを指定しない場合、SAS はデフォルトの長さ 8 バイトを割り当てます。
- 生の文字データを読み取るときに、INPUT ステートメントの変数で文字入力形式を使用できます。変数の長さは、INPUT ステートメントの入力形式で指定された

幅と一致します。入力形式または DATA ステップの他の場所で長さを指定しない場合、SAS はデフォルトの長さ 8 バイトを割り当てます。

- 生の日付または時間データを読み取るときに、INPUT ステートメントの変数で日時入力形式を使用できます。変数の長さは、INPUT ステートメントの入力形式で指定された幅と一致します。入力形式または DATA ステップの他の場所で長さを指定しない場合、SAS はデフォルトの長さ 8 バイトを割り当てます。

“変数の作成と変更” (76 ページ)を参照してください。

カテゴリ別の入力形式のリストを表示するには、“カテゴリ別の入力形式” (*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。

INPUT ステートメントと入力形式を使用して新しい変数を作成する方法の詳細については、“フォーマット入力” (351 ページ)を参照してください。

ATTRIB ステートメントを使用した変数の作成

変数がまだ存在せず、ATTRIB ステートメントで FORMAT=または INFORMAT=オプションを使用して初めて変数を作成した場合、SAS は、OPTION ステートメントで指定された出力形式または入力形式に基づいて、変数とその属性を定義します。

- DATA ステップで、FORMAT または INFORMAT ステートメントを使用して変数を初めて作成するときに、*数値*の出力形式または入力形式を変数に関連付けると、変数は数値型として作成され、デフォルトの長さは 8 です。
- DATA ステップにおいて、FORMAT ステートメントまたは INFORMAT ステートメントを使用して初めて作成される変数に、*文字*出力形式または入力形式を関連付けると、その変数は文字型として作成されます。長さは、出力形式または入力形式の一部として指定した出力形式または入力形式の幅と一致します。長さを指定しない場合、SAS はデフォルトの長さの 8 バイトを割り当てます。

これらの出力形式および入力形式のリストについては、カテゴリ別出力形式およびカテゴリ別入力形式を参照してください。

FORMAT ステートメントまたは INFORMAT ステートメントを使用した変数の作成

変数がまだ存在せず、FORMAT または INFORMAT ステートメントで初めて作成する場合、SAS は指定された出力形式または入力形式に基づいて変数とその属性を定義します。

- DATA ステップで、FORMAT または INFORMAT ステートメントを使用して変数を初めて作成するときに、*数値*の出力形式または入力形式を変数に関連付けると、変数は数値型として作成され、デフォルトの長さは 8 です。
- DATA ステップにおいて、FORMAT ステートメントまたは INFORMAT ステートメントを使用して初めて作成される変数に、*文字*出力形式または入力形式を関連付けると、その変数は文字型として作成されます。長さは、出力形式または入力

形式の一部として指定した出力形式または入力形式の幅と一致します。長さを指定しない場合、SAS はデフォルトの長さの 8 バイトを割り当てます。

これらのカテゴリのリストについては、カテゴリ別出力形式および カテゴリ別入力形式を参照してください。

FORMAT ステートメントを使用して新しい変数を作成(長さを指定しない) (79 ページ)において、FORMAT ステートメントは文字変数 Flavor および数値変数 Amount を作成します。これら 2 つの変数は FORMAT ステートメントで初めて使用されるため、SAS はそれらに割り当てられた出力形式のカテゴリに基づいてそれらの型を決定します。変数 Amount は数値型出力形式(COMMAwd)に関連付けられているため、SAS はこれをデフォルトの長さは 8 の数値型変数として定義します。

Flavor 変数は、$UPCASEw.出力形式が文字型出力形式のため、文字型変数として定義されています。FORMAT ステートメントを使用して文字変数を作成すると、SAS は最初に FORMAT ステートメントで指定した長さに基づいて長さを決定します。出力形式または DATA ステップの他の場所で長さを指定しない場合、SAS は変数にデフォルトの長さ 8 を与えます。この例では、出力形式に長さの指定が含まれていないため、変数 Flavor の長さは 8 バイトです。

例のコード 4.1　FORMAT ステートメントを使用して新しい変数を指定(長さを指定しない)

```
data lollipops;
  format Flavor $upcase. Amount comma.;
  Flavor='Cherry';
  Amount=10;
run;
proc contents data=lollipops; run;
```

次の例は、FORMAT ステートメントで両方の変数に長さが指定されていることを除いて同じです。

アウトプット 4.1　FORMAT ステートメントを使用して新しい変数を作成(長さを指定しない)

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format
2	Amount	Num	8	COMMA.
1	Flavor	Char	8	$UPCASE.

例のコード 4.2　長さを指定した FORMAT ステートメントを使用した新しい変数の指定(長さを指定)

```
data lollipops;
  format Flavor $upcase10. Amount comma10.;
  Flavor='Cherry';
  Amount=10;
run;
proc contents data=lollipops; run;
```

アウトプット 4.2　FORMAT ステートメントを使用して新しい変数を指定した場合の PROC CONTENTS 出力(長さを指定)

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format
2	Amount	Num	8	COMMA10.
1	Flavor	Char	10	$UPCASE10.

次の例例のコード 4.3 (80 ページ)では、変数は FORMAT ステートメントではなく割り当てステートメントで初めて表示されます。したがって、この場合は割り当てステートメントは変数を作成します。変数が割り当てステートメントの左側に初めて現れると、SAS は割り当てステートメントの右側の式に基づいてその型と長さを自動的に設定します。右側の式は 5 文字の文字列 **Cherry** であるため、SAS は 5 バイトの長さと文字型を変数 Flavor に割り当てます。SAS は、8 バイトの長さを数値変数 Amount に割り当てます。

例のコード 4.3　FORMAT ステートメントを使用して既存の変数の出力形式を変更する

```
data lollipops;
  Flavor='Cherry';
  Amount=10;
  format Flavor $upcase10. Amount comma10.;
run;
proc contents data=lollipops; run;
```

アウトプット 4.3　FORMAT ステートメントを使用して既存の変数の出力形式を変更するための PROC CONTENTS 出力

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format
2	Amount	Num	8	COMMA10.
1	Flavor	Char	6	$UPCASE10.

関連項目

- 表 4.10 (87 ページ) (SAS が割り当てステートメントを使用して変数属性を決定する方法の詳細が説明されています。)
- *SAS 出力形式と入力形式: リファレンス*

変数の変更

変数を変更する方法

次の方法を使用して、SAS 変数を変更できます。

- FORMAT または INFORMAT ステートメント
- LENGTH ステートメント
- ATTRIB ステートメント

表 4.7 変数を変更する方法

方法	例	説明
FORMAT または INFORMAT ステートメント	Insurance=100.28; Fee=20.00; format Insurance Fee dollar10.2; 例: FORMAT ステートメント	新しい変数を作成し、FORMAT または INFORMAT ステートメントを使用してその出力形式または入力形式を指定できます。[1] FORMAT ステートメントは 2 つの変数 Insurance および Fee を、dollar 10.2 の出力形式で作成します。
LENGTH ステートメント	length Name $20; 例: LENGTH ステートメント	文字変数の場合、変数を使用する最初のステートメントで可能な限り長い値を使用する必要があります。これは、同じ DATA ステップ内の後続の LENGTH ステートメントで長さを変更できないためです。 SAS の文字変数の最大長は 32,767 バイトです。数値変数の場合、後続の LENGTH ステートメントを使用して変数の長さを変更できます。 SAS が文字変数に値を割り当てる場合、ターゲット変数の長さに一致するように、値に空白を埋め込むか、値の右側を切り捨てます。
ATTRIB ステートメント	attrib Class format=$12.; 例: ATTRIB ステートメント	ATTRIB ステートメントを使用すると、既存の変数に対して次の変数属性を 1 つ以上指定できます。 - FORMAT= - INFORMAT= - LABEL=

方法	例	説明
		■ LENGTH= 変数がまだ存在しない場合は、FORMAT=、INFORMAT=、および LENGTH=属性の 1 つ以上を使用して、新しい変数を作成できます。 ATTRIB ステートメントは、Class という名前の、出力形式が設定された新しい変数を作成します。 **注:** LABEL ステートメントまたは ATTRIB ステートメントの LABEL=属性を単独で使用して、新しい変数を作成できません。ラベルは、既存の変数にのみ適用できます。

1 FORMAT ステートメントまたは INFORMAT ステートメントを使用して変数を作成する際に、SAS が変数属性を決定する方法については、"FORMAT ステートメントまたは INFORMAT ステートメントを使用した変数の作成"を参照してください。

変数の出力の制御

ステートメントまたはデータセットオプションの使用

DROP、KEEP、および RENAME ステートメント、または DROP=、KEEP=、および RENAME=データセットオプションは、DATA ステップでどの変数を処理または出力するかを制御します。これらのステートメントとデータセットオプションの 1 つまたは組み合わせを使用して、目的の結果を得ることができます。SAS が実行するアクションは、次のいずれかのアクションを実行するかどうかによって大きく異なります。

- ステートメントまたはデータセットオプション、あるいはその両方を使用します。
- 入力または出力データセットでデータセットオプションを指定します。

次の表は、DROP、KEEP、および RENAME ステートメントと、DROP=、KEEP=、および RENAME=データセットオプションの一般的な相違点をまとめたものです。

表 4.8 変数の削除、保持、および名前変更に関するステートメントとデータセットオプション

ステートメント	データセットオプション
出力データセットのみに適用	出力または入力データセットに適用
すべての出力データセットに影響を与える	個々のデータセットに影響を与える

ステートメント	データセットオプション
DATA ステップでのみ使用可能	DATA ステップおよび PROC ステップで使用可能
DATA ステップのどこにでも指定可能	適用する各データセット名の直後に指定

入力または出力データセットの使用

また、変数をプログラムデータベクトルまたは新しい SAS データセットに読み取る前に、変数を削除するか、保持するか、名前を変更するかを検討する必要があります。DROP、KEEP、または RENAME ステートメントを使用すると、変数が出力データセットに書き込まれるときに常にアクションが発生します。SAS データセットオプションでは、オプションを使用する場所によって、アクションが発生するタイミングが決まります。オプションが入力データセットで使用される場合、変数は、プログラムデータベクトルに読み取られる前に、削除、保持、または名前変更されます。出力データセットで使用する場合、データセットオプションは、変数が新しい SAS データセットに書き込まれるときに適用されます。(DATA ステップでは、入力データセットは、SET、MERGE、または UPDATE ステートメントで指定されたものです。出力データセットは、DATA ステートメントで指定されたものです。)決定を下すときは、次の事実を考慮してください。

- 変数が出力データセットに書き込まれず、処理を必要としない場合は、入力データセットオプションを使用して DATA ステップから変数を除外する方が効率的です。
- 変数を DATA ステップで処理する前に名前を変更する場合は、入力データセットで RENAME=データセットオプションを使用する必要があります。
- アクションが出力データセットに適用される場合は、出力データセットでステートメントまたはデータセットオプションのいずれかを使用できます。

次の表は、入力データセットと出力データセットに指定された場合のデータセットオプションとステートメントのアクションをまとめたものです。表の最後の列は、変数が DATA ステップで処理できるかどうかを示しています。変数の名前を変更する場合は、最後の列の情報を使用してください。

表 4.9　変数の削除、保持、および名前変更時の変数と変数名の状態

指定場所	データセットオプションまたはステートメント	目的	変数または変数名の状態
入力データセット	DROP= KEEP=	変数を処理に含めるか除外します	除外した場合、変数は DATA ステップで使用できません
	RENAME=	処理前に変数名を変更	プログラムステートメントと出力データセットオプシ

指定場所	データセットオプションまたはステートメント	目的	変数または変数名の状態
			ョンで新しい名前を使用し、他の入力データセットオプションで古い名前を使用します
出力データセット	DROP、KEEP	すべての出力データセットに書き込まれる変数を指定します	すべての変数が処理に使用できます
	RENAME	すべての出力データセットの変数名を変更	プログラムステートメントで古い名前を使用し、出力データセットオプションで新しい名前を使用します
	DROP= KEEP=	個々の出力データセットに書き込まれる変数を指定	すべての変数を処理できます
	RENAME=	個々の出力データセットの変数名を変更	プログラムステートメントおよびその他の出力データセットオプションで古い名前を使用します

適用の順序

プログラムでは、複数のデータセットオプション、またはデータセットオプションとステートメントの組み合わせを使用する必要がある場合があります。SAS は、次の順序で変数を削除、保持、および名前変更することを知っておくと役立ちます。

- まず、入力データセットのオプションは、SET、MERGE、および UPDATE ステートメント内で左から右に評価されます。DROP=および KEEP=オプションは、RENAME=オプションの前に適用されます。
- 次に、DROP ステートメントと KEEP ステートメントが適用され、RENAME ステートメントが続きます。
- 最後に、出力データセットのオプションは、DATA ステートメント内で左から右に評価されます。DROP=および KEEP=オプションは、RENAME=オプションの前に適用されます。

関連項目

例

- DROP=データセットオプションと DROP ステートメントの使用
- DROP=および RENAME=データセットオプションの使用

ステートメント

- DROP ステートメント

データセットオプション

- DROP=データセットオプション
- RENAME=データセットオプション

データ型の変換

文字から数値への自動変換

デフォルトでは、算術演算などの数値コンテキストで文字変数を参照すると、SAS は変数値を数値に変換しようとします。プラス記号などの数値オペランドを必要とする演算子で文字変数を使用すると、SAS は文字変数を数値に変換します。等号などの比較演算子を使用して文字変数と数値変数を比較すると、文字変数は数値に変換されます。

次の例では、文字変数 Rate が数値コンテキストの中にあります。これに数値変数 Hours を乗算して、Salary という名前の新しい変数を作成します。

```
Salary=Rate*Hours;
```

このステップを実行すると、SAS は自動的に Rate の文字値を数値に変換しようと試み、計算を実行できるようにします。この変換は、Rate の各文字値の一時的な数値を作成することによって完了します。この一時的な値が計算に使用されます。Rate の文字値は数値に置き換えられません。データが自動的に変換されるたびに、変換が行われたことを示すメッセージが SAS ログに出力されます。

例のコード 4.1　SAS ログ

```
NOTE: Character values have been converted to numeric values at the places given by:
(Line):(Column).
9248:8
```

注: 文字変数を数値に変換すると無効な数値が生成される場合、SAS は結果に欠損値を割り当て、エラーメッセージをログに出力し、自動変数_ERROR_の値を 1 に設定します。

明示的な文字から数値への変換

INPUT 関数を使用すると、文字データ値を数値に明示的に変換できます。入力形式を選択する場合は、値の形式を読み取ることができる数値入力形式を選択してください。次の例では、INPUT 関数を使用して、Rate を数値に明示的に変換できます。*Rate* の長さは 2 で、数値入力形式 2.は変数の値を読み取るために使用されます。

```
input (payrate,2.);
```

INPUT 関数を使用すると、SAS ログに変換メッセージが出力されません。

数値から文字への自動変換

数値データから文字データへの自動変換は、文字から数値への変換と非常によく似ています。数値データ値は、文字コンテキストで使用される場合は必ず文字値に変換されます。具体的には、SAS は BEST12.出力形式で数値を書き込みます。結果の文字値は右揃えになります。この変換は、値が割り当てられる前、または演算子や関数で使用される前に行われます。ただし、数値から文字への自動変換は予期しない結果を引き起こす可能性があります。たとえば、文字変数に先頭の空白が含まれている場合、変数に対して操作を実行したり、関数で変数を使用したりすると、エラーが発生したり、予期しない結果が返されたりする可能性があります。

数値から文字への自動変換により、変換が行われたことを示すメッセージが SAS ログに出力されます。

次の例では、新しい文字変数 Loc を作成します。それには、数値変数 Site の値、および文字変数 Dept の値が連結されて入力されます。新しい変数値には、Site の値がスラッシュ付きで、また Dept の値が含まれます。

```
Loc=Site||'/'||Dept;
```

このステートメントを DATA ステップでサブミットすると、数値変数 Site の値を文字値に変換します。その理由は、Site が文字コンテキストで使用されているためです。変数 Site は文字値を必要とする連結演算子とともに表示されます。

明示的な数値から文字への変換

PUT 関数を使用して、数値データ値を文字データ値に明示的に変換します。割り当てステートメント内の PUT 関数は、数値データ値を文字に変換します。次の例では、Site の数値を明示的に文字値に変換します。文字値に変換するには、割り当てス

テートメントで PUT 関数を使用します。Site はソース変数です。Site 長さは 2 なので、数値出力形式として 2.を選択します。

```
loc=put(site,2.)||'/'||Dept;
```

PUT 関数を使用すると、SAS ログに変換メッセージは出力されません。

関連項目

例

- “例: 文字変数を数値に変換”
- “例: 数値変数を文字に変換”
- “例: 自動的な型の変換”

関数

- “INPUT 関数” (*SAS 関数と CALL ルーチン: リファレンス*)
- PUT 関数

変数属性

定義

*SAS 変数*は、文字や数値を保存して使用するためにプログラム内で作成するコンテナーです。変数には次の属性があります。

表 4.10 変数属性

変数属性	定義	可能な値	デフォルト値
名前	変数を識別します。変数名は、SAS 変数の命名規則に従っている必要があります。詳細については、“変数名”を参照してください。	有効な SAS 変数名。	なし
型	変数を CHARACTER(CHAR)または DOUBLE (現在の数値データ型)として識別	■ DOUBLE ■ 文字	DOUBLE

変数属性	定義	可能な値	デフォルト値
	します。詳細については、“データ型”を参照してください。		
長さ	SAS データセット内の変数の各値を格納するために使用されるバイト数を参照します。LENGTH ステートメントを使用して、変数の格納長を設定できます。	DOUBLE 値の場合は 2 から 8 バイト。 CHARACTER 値の場合は 1〜32,767 バイト。 ■ DOUBLE 値 - 2 から 8 バイト ■ CHARACTER 値 - 1 から 32,767	■ DOUBLE 値 - 8 バイト ■ CHARACTER 値 - 8 バイト
出力形式	変数値を出力する場合に SAS が使用する命令を指します。出力形式が指定されていない場合、デフォルトの出力形式は数値変数の場合は BEST12、文字変数の場合は$w.です。FORMAT ステートメントまたは ATTRIB ステートメントで SAS 出力形式を変数に割り当てることができます。	SAS 出力形式および入力形式: リファレンスの出力形式のディクショナリを参照してください。	数値の場合は BEST12.、文字の場合は$w.
入力形式	変数値を読み取る際に SAS が使用する命令を指定します。入力形式が指定されていない場合、デフォルトの入力形式は数値変数の場合は w.d、文字変数の場合は $w.です。FORMAT ステートメントまたは ATTRIB ステートメントで、SAS 入力形式を変数に割り当てることができます。 INPUT ステートメントまたは INPUT 関数で入力形式を割り当てることもできます。	SAS 出力形式および入力形式: リファレンスの入力形式を参照してください。	数値の場合は wd、文字の場合は$w.
ラベル	最大 256 文字の長さの説明ラベルを参照します。一部の SAS プロシ	最大 256 文字です。	なし

変数属性	定義	可能な値	デフォルト値
	ジャで出力できる変数ラベルは、レポート作成に役立ちます。LABEL または ATTRIB ステートメントを使用して、変数にラベルを割り当てることができます。		
オブザベーションの位置	変数が DATA ステップで定義された順序によって決定されます。	1- n	なし
インデックスの種類	変数がデータセットのインデックスの一部であるかどうかを示します。	NONE — 変数は索引付けされていません。 SIMPLE — 変数は単一インデックスの一部です。 COMPOSITE — 変数は 1 つ以上の複合インデックスの一部です。 BOTH — 変数は単一インデックスと複合インデックスの両方の一部です。	なし
拡張属性(ユーザー定義)	DATASETS プロシジャで XATTR ADD VAR ステートメントを使用して作成されるユーザー定義属性です。	数字または文字	なし

注: SAS 9.1 以降、変数の最大数は 32,767 を超えることができます。最大数は、環境とファイルの属性によって異なります。たとえば、変数の最大数はすべての変数の合計の長さに依存し、最大ページサイズを超えることはできません。

関連項目

例

- 変数属性の表示
- ATTRIB ステートメントを使用した変数属性の変更

ステートメント

- ATTRIB ステートメント
- FORMAT ステートメント
- INFORMAT ステートメント

プロシジャ

- CONTENTS プロシジャ

自動 DATA ステップ変数

自動 DATA ステップ変数は、実行中に DATA ステップによって自動的に作成される SAS システム変数です。これらの変数はプログラムデータベクトルに追加されますが、出力データセットには表示されません。自動変数の値は、DATA ステップの反復から次の反復まで、欠損値に設定されるのではなく、保持されます。DATA ステップを実行すると、SAS は DATA ステップによって処理されるデータに関する情報と、エラー処理のリターンコードなどの処理情報を含む変数を作成します。

ERROR
: デフォルトは 0 ですが、エラーが発生すると常に 1 に設定されます。エラーの例には、入力データエラー、変換エラー、または 0 による除算や浮動小数点オーバーフローなどの演算エラーが含まれます。この変数の値を使用すると、データレコード内のエラーを特定し、SAS ログにエラーメッセージを出力するのに役立ちます。

IORC
: 自動変数_IORC_には、MODIFY ステートメントが実行しようとする各 I/O 操作のリターンコードが含まれています。_IORC_の値をテストする最良の方法は、SYSRC 自動呼び出しマクロによって提供されるニーモニックコードを使用することです。各ニーモニックコードは 1 つの条件を表します。ニーモニックは、DATA ステッププログラムの問題をテストするための簡単な方法を提供します。この自動変数は、MODIFY ステートメントを使用して既存の SAS データセット内のオブザベーション値を置換、削除、追加する場合に、DATA ステップで作成されます。

_N_
: 初期値は 1 に設定されます。DATA ステップが DATA ステートメントをループするたびに、変数_N_は 1 ずつ増加します。_N_の値は、DATA ステップが反復された回数を表します。

FIRST.*variable*
: SAS が BY 変数ごとに作成する変数です。SAS は、BY グループ内の最初のオブザベーションを処理する際に FIRST.*variable* を設定します。

  例 “例: FIRST.および LAST.BY 変数を名前リテラルとして指定する” (59 ページ)

LAST.*variable*
: SAS が BY 変数ごとに作成する変数です。SAS は、BY グループ内の最後のオブザベーションを処理する際に LAST.*variable* を設定します。

例 “例: FIRST.および LAST.BY 変数を名前リテラルとして指定する” (59 ページ)

自動変数の名前は、DATA ステップの実行時に作成されるシステム生成変数として予約されています。アプリケーションを作成する際に、アンダースコアで始まりアンダースコアで終わる名前を使用しないことを推奨します。

関連項目

例

- “例: _N_自動変数の使用”
- “例: IF/THEN ステートメントで_ERROR_自動変数と_INFILE_自動変数を使用”

ステートメント

- “_INFILE_=variable” (*SAS DATA ステップステートメント: リファレンス*)

自動マクロ変数

自動マクロ変数

変数リスト

変数リストの定義

*SAS 変数*リストは、*変数*名のリストを参照する省略された方法です。SAS では、次の変数リストを使用できます。

- 番号付き範囲リスト
- 名前の範囲リスト
- 名前の接頭辞リスト
- 特殊な *SAS 名*リスト

SAS は、コンパイラが DATA ステップ内で検出した順序でアクティブな変数を追跡します。名前変数リストを使用すると、変数リスト内の変数を同じ順序で参照します。

変数を定義する場合に変数リストを使用できます。SAS ステートメントで変数リストを使用、および DATA ステップでデータセットオプションを使用できます。変数リストは、既存のデータグループを簡単に参照できるため便利です。

注: RENAME=オプションでは、番号付き範囲リストのみサポートされます。

変数リストの種類

型	定義	例
番号付き範囲リスト	同じ名前の接頭辞が付いているが、最後の文字の数値が異なる変数で構成されるリスト。値は連続している必要があります。 **注:** 2147483647 より大きい数字で終わるデータセットまたは変数には、番号付き範囲リストを指定できません。	input var1-var6; これは、範囲が指定されていない次の番号付きリストと同等です。 input var1 var2 var3 var4 var5 var6;
名前の範囲リスト	名前の範囲リストは変数定義の順序に依存します。	x--b プログラムデータベクトル内で順序付けされたすべての列(x から b まで)を指定しています。
名前の接頭辞リスト	一部の SAS 関数およびステートメントでは、名前の接頭辞リストを使用して、指定された文字列で始まるすべての変数を参照できます。	sum(of Sales:) この文字列は、Sales_Jan、Sales_Feb、Sales_Mar などの Sales で始まるすべての変数の合計を計算するように SAS に指示しています。
特殊な SAS 名リスト	特別な SAS 名リストには次のものがあります。 _NUMERIC_ 現在の DATA ステップですでに定義されているすべての数値変数を指定します。 _CHARACTER_ 現在の DATA ステップですでに定義されているすべての文字変数を指定します。 _ALL_ 現在の DATA ステップですでに定義されているすべての変数を指定します。	_NUMERIC_ _CHARACTER_ _ALL_

関数と変数リストを使用した OF 演算子

定義

OF 演算子を使用すると、SAS 変数リストまたは SAS 配列を関数の引数として指定できます。OF 演算子で使用される関数の構文は次のとおりです。

FUNCTION (**OF** *variable-list*)

FUNCTION (<*argument* |OF *variable-list* |OF *array-name*[*]><..., <*argument* |OF *variable-list* |OF *array-name*[*]>>)

> 重要　WHERE 式は OF 演算子を含む関数では使用できません。

次の表は、OF 演算子で有効な SAS 変数リストの種類を示しています。

表 4.11　OF 演算子で使用される変数リスト

リストの種類	例	説明
名前の範囲リスト	*function-name*(OF *x*-character-*a*)	x から a までのすべての文字変数に対して関数を実行します。
名前の接頭辞リスト	*function-name*(OF x:)	特定の変数で始まるすべての変数に対して関数を実行します。この例では、変数"x1"、"x2"、および"x3"は文字"x"で始まります。
番号付き範囲リスト	*function-name*(OF x1 – x*n*)	x1 から x*n* までの範囲の変数値に対して関数を実行します。[1]
配列	*function-name*(OF *array-name*(*))	指定された配列に対して関数を実行します。[2]

1. 最後の文字が連続した数字で、その数字以外は同じ名前である一連の変数が含まれる必要があります。

リストの種類	例	説明
特殊な SAS 名リスト	*function-name*(OF _numeric_)	_numeric_variable に対して関数を実行します。これは、現在の DATA ステップですでに定義されているすべての数値変数です。

次の表に、OF 演算子で使用できる SAS 関数を示します。

表 4.12　OF 演算子を使用する SAS 関数

CAT	CV	KURTOSIS	NMISS	STD
CATS	GEOMEAN	MAX	ORDINAL	STDERR
CATT	GEOMEANZ	MEAN	RANGE	SUM
CATX	HARMEAN	MIN	RMS	USS
CSS	HARMEANZ	N	SKEWNESS	VAR

関連項目

例

- “例: 番号付き範囲変数リストの作成”
- “例: 変数名の範囲リストを作成する”
- “例: 変数リストで OF 演算子を使用”
- “例: OF 演算子を使用して複数の変数リストを作成”

ステートメント

- ARRAY ステートメント
- リストされた入力の INPUT ステートメント
- KEEP ステートメント
- PUT ステートメント
- VAR ステートメント

関数

- SUM 関数

2. array-name が一時配列の場合、制限があります。詳細については、“一時配列で OF 演算子を使用する” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

欠損値

欠損値の定義

欠損値は、現在のオブザベーションの変数にデータ値が保存されていないことを示す値です。欠損値には次の 3 種類があります。

- 数値
- 文字
- 特殊数値

生データで欠損値を表現する方法

次の表は、生データを SAS に読み取るときに各データ型の欠損値を表現する方法を示しています。

表 4.13　欠損値の表現

欠損値の種類	データにおける表現	説明
数値	.	単一の小数点
文字	' '	引用符で囲まれた空白スペース
特殊	*.a*	小数点の後に任意の 1 文字(a-z)が続く
特殊	._	小数点の後にアンダースコアが続く

特殊欠損値

定義

特殊欠損値は数値欠損値の一種で、文字 A-Z またはアンダースコアを使用して、欠損データのさまざまなカテゴリを表すことができます。

特殊欠損値に関するヒント

- SAS は、大文字または小文字のいずれかを受け入れます。値は大文字で表示および出力されます。
- 特殊数値欠損値をピリオドで開始しない場合、SAS はそれを変数名として識別します。したがって、SAS 式または割り当てステートメントで特殊数値欠損値を使用するには、値をピリオドで始め、その後に文字またはアンダースコアを付ける必要があります。ここではいくつかの例を示します。

  ```
  x=.d;
  ```

- SAS が特殊欠損値を出力する場合、文字またはアンダースコアのみを出力します。
- SAS に特殊欠損値として解釈させたい数値フィールドの文字がデータ値に含まれている場合、MISSING ステートメントを使用してそれらの文字を指定します。

欠損値の順序

数値変数

SAS 内では、数値変数の欠損値はすべての数値よりも小さいと見なされます。データセットを数値変数で並べ替えると、その変数の値が欠損しているオブザベーションが、並べ替えられたデータセットの先頭に表示されます。数値変数の場合、特殊欠損値を数値と比較したり、相互に比較したりできます。次の表は、数値の並べ替え順序を示しています。

表 4.14　数値の並べ替え順

並べ替え順	記号	説明
最小	._	特殊欠損数値(ピリオドの後にアンダースコアが続く)
	.	ピリオド
	.A-.Z	特殊欠損値 A (最小)から Z (最大)
	-n	負の数
	0	ゼロ
最大	+n	正の数

並べ替えると、欠損数値は*特殊*欠損数値の前に表示されます。たとえば、**.**は**.A** の前に表示されます。

注: 特殊数値の欠損値を指定する場合、小文字と大文字は同じものと見なされます。たとえば、.a と.A は同じ値を表します。

注: システムが ASCII または EBCDIC のどちらの照合順序を使用しているかに関係なく、欠損値の数値の並べ替え順序は同じです。

文字変数

文字変数の欠損値は、表示可能な文字値よりも小さくなります。したがって、BY ステートメントを使用してデータセットを文字変数で並べ替えると、欠損(空白)文字値が常に表示可能な文字を含むオブザベーションの前に表示されます。表示不能文字は、欠損(空白)文字値よりも小さくなる場合があります。したがって、データに表示できない文字が含まれている場合、欠損値が並べ替えられたデータセットの先頭に表示されない場合があります。表示不能文字の例としては、コンピューターのキャリッジコントロール文字や、自動的に文字に変換された数値データなどがあります。

変数値が SAS によって自動的に欠損に設定される場合

生データを読み取る場合

DATA ステップの各反復の開始時に、SAS は DATA ステップで作成した各変数の値を欠損値に設定します。ただし、次の例外があります。

- RETAIN ステートメントで指定された変数
- SUM ステートメントで作成された変数
- _TEMPORARY_配列のデータ要素
- FILE または INFILE ステートメントのオプションで作成された変数
- FGET 関数によって作成された変数
- ARRAY ステートメントで初期化されたデータ要素
- 自動変数

SAS データセットを読み取る場合

SET、MERGE、または UPDATE ステートメントで変数を読み取る場合、SAS は DATA ステップの最初の反復の前に値を欠損値に設定します。BY ステートメントを使用する場合、BY グループが変更されると、変数も欠損に設定されます。変数は、割り当てステートメントで値が変更されるか、SET、MERGE、または UPDATE ステートメントの後続の反復で読み取られるまで、その値を保持します。SET、MERGE、および UPDATE ステートメントのオプションによって作成された変数も、1 つの反復から次の反復まで値を保持します。

マッチマージ操作(BY ステートメントを使用)でデータセット内のすべての行が処理されると、前述のように、出力データセット内の変数の値が保持されます。つまり、データセット内のすべての行が処理された際に有効な BY 値に変更がない限り、出力データセット内の変数は、最終的なオブザベーションの値を保持します。BY ステートメントによって生成される自動変数である FIRST.variable と LAST.variable は、どちらも値を保持します。それらの初期値は 1 です。

BY 値が変更されると、データセットには置換値を提供するための追加のオブザベーションが含まれていないため、変数は欠損に設定され、欠損のままになります。1 対 1 マージ操作(BY ステートメントを使用しない)ですべての行が処理されると、SAS は出力データセット内の変数を欠損に設定します。

SAS によって欠損値が生成される場合

計算における欠損値のプロパゲーション

算術計算で欠損値を使用すると、結果は欠損に設定されます。そして、その結果を別の計算で使用すると、次の結果も欠損します。この処理を、欠損値のプロパゲーションと言います。SAS ログには、欠損値を含む算術式と、作成日時が出力されます。ただし、処理は続行されます。

欠損値のプロパゲーションを防ぐ

欠損値を算術式に反映させたくない場合は、SUM ステートメントや SUM 関数などの統計関数を使用して、計算から欠損値を省略することができます。

無効な操作

次のような無効な操作を実行しようとすると、SAS はログにメッセージを出力し、結果に欠損値を割り当てます。

- 0 による除算
- 0 による対数計算
- 式を使用して、浮動小数点数として表すには大きすぎる数値を生成する(オーバーフロー)

文字から数値への自動データ型変換

値が算術式に現れる場合、SAS は自動的に文字値を数値に変換します。SAS がこのタイプの自動データ型変換を実行すると、ログに注記が出力され、_ERROR_自動 DATA ステップ変数が 1 になります。

関連項目

例

- “例: 欠損値を自動的に置換”
- “例: 特別な欠損値の作成”

- “例: 欠損値のプロパゲーションの防止”

関数

- MISSING 関数

ステートメント

- MISSING ステートメント

数値精度

概要

2 進数であろうと 10 進数であろうと、どのような記数法でも、表現できる正確な数には限界があります。その結果、近似値を作成する必要があります。

たとえば、10 進記数法では分数 1/3 が無限に繰り返される数字(.333...)が含まれているため、有限の 10 進数値として完全に表すことはできません。コンピューターでは、精度が有限であるため、この数値を完全な精度で表すことはできません。数値精度とは、数値が近似または表現される精度です。

計算において、ソフトウェアアプリケーションは、有限精度とマシンハードウェアの制限による数値精度エラーの影響を特に受けやすくなっています。コンピューターは有限の記憶容量を持つ有限のマシンのため、完全な精度で無限の数値の集合を表現することはできません。

この問題は、コンピューターが人間とは異なる記数法を使用しているという事実によってさらに複雑になります。10 進の無限精度演算は人間の計算の標準ですが、コンピューターは、有限の 2 進表現の値と有限精度演算を使用します。この表現は、多くの計算に適していることが証明されています。ただし、問題によっては、ハードウェアが提供する精度よりも広い拡張精度が必要になる場合があります。その場合、表現と演算は主にソフトウェアで行われ、ハードウェア演算よりもかなり遅くなります。

さらに、コンピューターは人間中心のソフトウェアインターフェイスを介して 10 進数と 10 進数演算を使用できますが、すべての数値とデータは最終的にバイナリ形式に変換され、コンピューターによって内部で保存および処理されます。精度が影響を受け、丸め誤差が導入されるのは、これらの 2 つの記数法(10 進数から 2 進数)の間の変換です。

2 進数の切り捨て

表現が無限に繰り返される 10 進値があるように、表現が無限に繰り返される 2 進値もあります。ただし、10 進数で不正確な数値は、2 進数で不正確な数値と必ずしも同じではありません。

たとえば、10 進数値 1/10 には有限の 10 進数表現(0.1)がありますが、2 進数では無限に繰り返される表現があります。2 進では、値は次のように変換されます

0.0001100110011001100110011 ...

このパターン 0011 は無限に繰り返されます。その結果、値はコンピューターに保存される際に丸められます。

SAS で不正確な数値に対して計算と比較を実行すると、予期しない結果が生じる可能性があります。最も単純な計算でさえ、間違った結論につながる可能性があります。ハードウェアは、10 進法で明らかで期待されるものと常に一致するとは限りません。

たとえば、10 進演算では、式(3 x 0.1)は 0.3 に等しいことが期待されます。したがって、(3 x 0.1)と(0.3)の差は 0 でなければなりません。10 進値 0.1 と 0.3 は正確な 2 進表現を持たないため、この等式は 2 進算術演算では当てはまりません。SAS プログラムで 2 つの値の差を計算すると、結果は 0 ではありません。

2 進数に相当する、2 進数が無限に繰り返される 10 進数の小数が多数あるため、一般的な有理数の結果を 10 進数で解釈する場合は注意が必要です。どちらの記数法でも問題にならない有理数がいくつかあります。たとえば、1/2 は 10 進法と 2 進法の両方で有限表現できます。

SAS が数値を保存する方法

最大整数サイズ

別の方法で指定しない限り、SAS はすべての数値を 8 バイトのストレージに保存します。これは、値が 8 桁に制限されているという意味ではなく、値を保存するために 8 バイトが割り当てられているという意味です。前のセクションでは、分数(整数ではない数)を保存すると精度の問題が発生する可能性があることを学びました。ただし、整数を扱う場合、大きさまたは範囲の問題が発生することもあります。

どのコンピューターでも、整数の絶対値の大きさには制限があります。SAS では、この最大整数値は次の 2 つの要因によって異なります。

- 変数を保存するために明示的に指定したバイト数(LENGTH ステートメントを使用)
- SAS が実行されている動作環境

ストレージのバイト数を明示的に指定していない場合、SAS はデフォルトの長さの 8 バイトを使用し、最大整数は使用しているオペレーティングシステムにのみ依存します。

- Linux オペレーティングシステムにおける SAS 変数の最小長は 3 バイトで、最大長は 8 バイトです。
- 変数の長さが増えると、確実に表現できる整数のサイズも増えます。
- 可変長の場合、最大整数はホストによって異なります。
- 実数は常に 8 バイトのストレージ全体に保存します。LENGTH ステートメントを使用して変数の長さを短くすることでディスク容量を節約したい場合は、値が整数である変数に対してのみ可能です。変数の長さを調整する場合は、値が指定された長さで許可されている最大の整数以下であることを確認してください。

  たとえば、Linux 動作環境で、数値変数の値が常に-8192 and 8192 の間の整数であることが分かっている場合、長さ 3 を明確に指定して数値を保存できます。

```
data myData;
 length num 3;
   num=8000;
run;
```

注意
実数を含む変数を保存するには、8 バイト全体を使用します。

浮動小数点表現

SAS は、整数であるか小数であるかにかかわらず、すべての数値を浮動小数点数として保存します。浮動小数点表現は、適切な数値精度で広範囲の値(非常に大きな数値または非常に小さな数値)をサポートします。

浮動小数点表現は*指数表記*に似ているため、すでによく知っているかもしれません。指数表記と浮動小数点表現の両方で、各数値は*仮数*、*基数*、*指数*または *m x b^n^*として表されます。

たとえば、次の整数値 987 は、基数 10 の指数表記で次のように表されます。

- *仮数*は、基数に掛けられる数値です。この例では、仮数は.987 です。
- *基数*は、べき乗される数値です。例では、基数は 10 です。
- *指数*は、基数のべき乗です。この例では、指数は 3 です。

指数表記と浮動小数点表現の大きな違いの 1 つは、指数表記では基数が通常 10 であることです。浮動小数点表現では、基数はオペレーティングシステムに応じて 2 または 16 です。

たとえば、同じ値、987(IEEE754)は、次のように 2 進浮動小数点表現で記述されます。

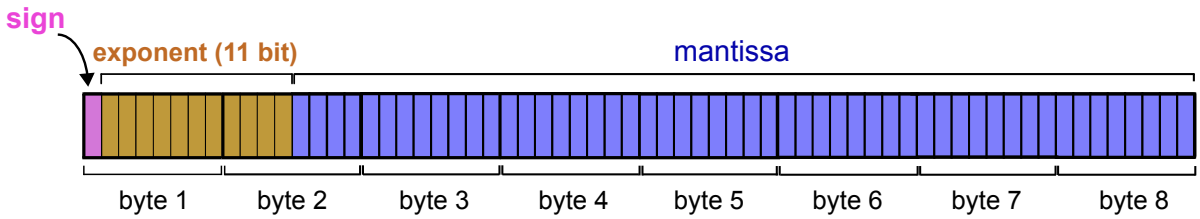
```
          sign    exponent      mantissa
987   =    0    100   0100   0111   0110
```

これも指数表記ですが、基数は 2 であることに注意してください。小さい値なので、丸めは必要ありません。

2 進浮動小数点数を保存するために、コンピューターは*交換形式*または*バイトレイアウト*と呼ばれる標準形式を使用します。バイトレイアウトは、浮動小数点数の部分が標準化された方法で表現されるように、ビット文字列を左から右にグループ化および順序付けする標準的な方法です。浮動小数点値の各部分(符号、指数、仮数)には、文字列内の特定のビット数と、文字列内の特定の位置が割り当てられます。これにより、浮動小数点データを効率的かつコンパクトな形式で交換できます。

図 4.1 (103 ページ)は、倍精度 2 進浮動小数点数のバイトレイアウトを示します。このレイアウトでは、最初のビットを使用して数値の符号をエンコードし、次の 11 ビットを使用して指数をエンコードし、最後の 52 ビットを使用して仮数をエンコードします。符号ビットが 1 の場合、数値は負になり、符号ビットが 0 の場合、数値は正になります。

図 4.1　倍精度 2 進浮動小数点数のバイトレイアウト



ホストコンピューターが異なれば、浮動小数点表現の形式と仕様も異なる場合があります。SAS が実行されるすべてのプラットフォームは、8 バイトの浮動小数点表現を使用します。

精度と大きさ

(丸めなしで)正確に表現できる最大の整数値は、基数と指数に割り当てられたビット数によって決まります。精度は、仮数に割り当てられたビット数によって決まります。オペレーティングシステムが桁を切り捨てたり丸めたりするかどうかは、表現のエラーに影響します。

SAS は、仮数ビット数を削減する LENGTH ステートメントを使用して、切り捨てられた浮動小数点数を保存します。IEEE 規格は、Linux オペレーティングシステムで使用されます。

表 4.15　IBM および IEEE 規格の浮動小数点形式

仕様	IBM メインフレーム	IEEE 規格	影響
基数	16	2	大きさ
指数ビット	7	11	大きさ
仮数ビット	56	52	精度
丸めまたは切り捨て	切り捨て	丸め	精度
指数のバイアス	64	1023	

次の箇条書きは、上の表をより詳細に説明しています。

- **基数 16** – 数字 0-9、および文字 A-F(値 10-15 を表します)を使用します。

  たとえば、10 進数値 3000 を 16 進数に変換するには、基数 16 の記数法を使用します。

基数 16						
16^7	…	16^4	16^3	16^2	16^1	16^0
268,435,456	…	65,536	4096	256	16	1

$$3000 = (\mathbf{B} \times 16^2) + (\mathbf{B} \times 16^1) + (\mathbf{8} \times 16^0)$$
$$= (\mathbf{11} \times 256) + (\mathbf{11} \times 16) + (\mathbf{8} \times 1)$$

  したがって、値 3000 は 16 進数では BB8 のように表されます。

- **基数 2** – 数字の 0 と 1 を使用します。

  たとえば、10 進数値 184 を 2 進数に変換するには、基数 2 の記数法を使用します。

基数 2						
2^7	…	2^4	2^3	2^2	2^1	2^0
128	…	16	8	4	2	1

$$184 = (\mathbf{1} \times 2^7) + (\mathbf{0} \times 2^6) + (\mathbf{1} \times 2^5) + (\mathbf{1} \times 2^4) + (\mathbf{1} \times 2^3) + (\mathbf{0} \times 2^2) + (\mathbf{0} \times 2^1) + (\mathbf{0} \times 2^0)$$
$$= 128 + 0 + 32 + 16 + 8 + 0 + 0 + 0$$

  したがって、値 184 は 2 進数では 10111000 のように表されます。

- **指数ビット** – 指数を保存するために予約されているビット数で、保存できる数値の大きさを決定します。指数ビットの数は、オペレーティングシステムによって異なります。IEEE 準拠のシステムでは、指数により多くのビットを使用するため、より大きな数値を生成します。
- **仮数ビット** – 数値の精度を決定する仮数を保存するために予約されているビット数。
- **丸めまたは切り捨て** – 2 桁以上の数の取り扱いに使用するために選択される変換方法です。仮数部には 16 進数 2 文字の余地しかないため、3 桁以上を処理する方法について考慮する必要があります。方法の 1 つは、保存できる長さで値を切り捨てることです。

  別の方法として、保存できない数字に基づいて値を丸める方法があります。これは、IEEE 準拠のシステムで行われます。どちらの方法も値の正確な表現にはならないため、このジレンマを処理する正しい方法も間違った方法もありません。

  SAS では、LENGTH ステートメントは、仮数ビットの数を切り捨てることによって機能します。切り捨てられた長さの影響の詳細については、“値を比較する場合の TRUNC 関数の使用”を参照してください。
- **バイアス** – 0 を表すことによって正と負の両方の指数を表現できるようにするための方法です。バイアスを使用しない場合は、指数に追加の符号ビットを割り当てる必要があります。たとえば、システムがバイアス 64 を使用する場合、値 66 の指数部は+2 の指数を表し、61 の指数部は-3 の指数を表します。

IEEE 規格を使用した浮動小数点表現

浮動小数点演算の IEEE 規格は、IEEE(Institute of Electrical and Electronic Engineers: 米国電気電子学会)によって作成された浮動小数点計算の技術規格です。この規格は、コンピューターが数値を浮動小数点表現で保存する方法を定義しています。浮動小数点数の IEEE 規格は、Linux で使用されています。

IEEE 準拠のプラットフォームは同じ一連の仕様を使用していますが、コンパイラの違いや数学ライブラリの違いにより、プラットフォーム間で結果が異なる場合があります。また、IEEE 規格では規格の実装方法の差異がある程度許容されているため、同じ規格に準拠していても、異なるプラットフォームでは実行方法が異なる場合があります。各オペレーティングシステムが計算を実行するために使用する基になる命令が多少異なるため、ホストによって結果が異なる場合があります。

計算の実行に標準的な方法はありません。すべてのオペレーティングシステムは、可能な限り正確に数値を計算しようとします。浮動小数点表現のコンポーネントが異なるオペレーティングシステム間でわずかに異なる結果が得られることは珍しくありません。

倍精度浮動小数点数の IEEE 規格では、数値は基数 2、バイアス 1023 の 11 ビット指数で構成されることが規定されています。これは、値が IBM メインフレームの表現よりもはるかに大きいことを意味します。この結果、仮数のビットが 3 少ない値になります。

IEEE 規格で表される 1 の値は次のとおりです。

```
3F F0 00 00 00 00 00 00
```

プロセッサは拡大実精度で計算を実行します。これは、基本形式(仮数部に 52 ビット、指数部に 11 ビット)で数値を保存するために使用される 64 ビットの代わりに、16 ビットが追加されることを意味します。つまり、仮数部に 12 ビット、仮数部に 4 ビットです。SAS の数値変数の最大幅は 8 バイトであるため、数値が 80 ビット(10 バイト)で保存されることはありません。これは単に、プロセッサが数値を 64 ビットメモリスロットに戻す前に、80 ビットを使用して数値を表すことを意味します。中間計算は 80 ビットで実行される可能性があり、これは最終的な計算結果の一部に影響します。

精度に関するエラーのトラブルシューティング

計算上の考慮事項

精度の高さにかかわらず、一部の数値は正確に表現できません。ほとんどの有理数(たとえば、.1)は、基数 2 や基数 16 では正確に表現できません。これは、浮動小数点表現で小数を保存するのは困難な場合が多いためです。

たとえば、33333 を 99999 に追加した場合、理論的な解は 1.33333 ですが、実際にはこの解を出すことは不可能です。値の計算が続くと、合計はさらに不正確になります。

たとえば、次の DATA ステップを考えます。

```
data _null_;
  do i=-1 to 1 by .1;
    put i=;
    if i=0 then put 'AT ZERO';
  end;
run;
```

DATA ステップの AT ZERO メッセージは、不正確な数値の累積により、正確な値 0 が検出されないため出力されません。計算結果は 0 に近づきますが、完全に 0 になることはありません。したがって、他の不正確な値を使用して数学演算を実行すると、不正確さがさらに悪化する可能性があります。

値を比較する場合の LENGTH ステートメントの使用

LENGTH ステートメントを使用すると、変数値の保存に使用されるバイト数を制御できます。ただし、誤差や重大なデータ損失を回避するために、LENGTH ステートメントは注意深く使用する必要があります。

数字 1234567890 について考えてみましょう。これは、基数 10 の浮動小数点表記.1234567890 の 10 乗です。精度が 5 桁しかない場合、数値は 123460000 に切り上げられます。これは、使用される 10 のべき乗に関係なく当てはまることに注意してください(.12346, 12.346, .0000012346 等)。

長さを定義するときは注意してください。.たとえば、IBM メインフレームで長さが 2 バイトの変数を作成するとします。システムは指数と符号を保存するために 1 バイトを使用し、仮数を保存するために 1 バイトを使用するので、完全に確実に保存できる最大の整数は 255 になります。ただし、一部のより大きな整数は 16 の倍数であるため、保存できます。

たとえば、256 から 272 までの数値の 8 バイト表現は表のようになります。

表 4.16　256 から 272 までの数値の 8 バイト表現の表

値	符号または指数	仮数 1	仮数 2-7	考慮事項
256	43	10	000000000000	末尾はゼロ。16 の倍数
257	43	10	100000000000	追加のバイトが必要
258	43	10	200000000000	
259	43	10	300000000000	
…	…	…	…	…
271	43	10	F00000000000	
272	43	11	000000000000	末尾はゼロ。16 の倍数

値を比較する場合の TRUNC 関数の使用

TRUNC 関数は、数値を要求された長さに切り捨ててから、数値を完全な長さに展開します。切り捨てとその後の展開は、完全な長さより短く数値を保存してからその数値を読み取る場合と同じ影響を及ぼします。たとえば、変数

```
x = 1/3
```

を長さ 3 で保存する場合、次の比較は真になりません。

```
if x = 1/3 then ...;
```

ただし、TRUNC 関数を追加すると、次のように比較が真になります。

```
if x=trunc(1/3,3) then ...;
```

この関数の詳細については、“TRUNC 関数” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

倍精度浮動小数点数と単精度浮動小数点数の比較

外部プログラムによって作成されたデータを SAS データセットに読み取る必要がある場合があります。データが浮動小数点表現の場合、RBw.d 入力形式を使用してデータを読み取ることができます。ただし、例外もあります。RBw.d 入力形式は、w 値が倍精度浮動小数点数のサイズ(このセクションで説明するすべてのオペレーティングシステムでは 8)より小さい場合、倍精度浮動小数点数を切り捨てる可能性があります。したがって、RB8.入力形式は完全な 8 バイト浮動小数点に対応します。RB4 入力形式は、4 バイトに切り捨てられた 8 バイトの浮動小数点に対応し、DATA ステップの LENGTH 4 とまったく同じです。

4 バイトに切り捨てられた 8 バイトの浮動小数点は、C 言語のプログラムの浮動小数点と同じではない可能性があります。C 言語では、8 バイトの浮動小数点数を double と呼びます。Fortran では、これは REAL*8 です。IBM PL/I では、これは FLOAT BINARY(53)です。4 バイトの浮動小数点数は、C 言語では float、Fortran では REAL*4、IBM PL/I では FLOAT BINARY(21)と呼びます。

IEEE 規格を使用するオペレーティングシステムでは、これは当てはまりません。単精度浮動小数点数では、指数に異なるビット数が使用され、異なるバイアスが使用されるます。そのため、RB4.入力形式を使用して値を読み取っても期待どおりの結果が得られません。

オペレーティングシステム間でのデータ転送

浮動小数点表記で表された非常に大きいまたは非常に小さい数値を含むデータを転送する場合、精度と大きさの問題が発生する可能性があります。“IEEE 規格を使用した浮動小数点表現”は、基数、指数、仮数の最大桁数を示しています。動作環境が異なると保存できる最大値が異なるため、あるコンピューターから別のコンピューターに浮動小数点データを転送する際に問題が発生する可能性があります。

注意
コンピューター間でデータを転送すると、数値の精度に影響を与える可能性があります。

この精度と大きさの違いは、ある動作環境から浮動小数点表現が異なる別の動作環境に移行する場合の要因となります。

別の解決策としては、オペレーティングシステム間でデータを転送する際の数値精度の問題を回避するおそらく最も安全な方法は、データ内の数値を整数に変換することです。

オペレーティングシステム間のデータの移動の詳細については、*SAS ファイルの移動とアクセス*を参照してください。

関連項目

例

- “例: SAS における不正確な値の比較”
- “例: 10 進値を浮動小数点表現に変換する”
- “例: 10 進数値を 16 進浮動小数点表現に変換”
- “例: 計算エラーを避けるために値を丸める”
- “例: 値を比較する場合の LENGTH ステートメントの使用”
- “例: 不正確な表現を持つ値を比較”
- “例: フォーマットを使用した精度エラーの確認”
- “例: 数値を正確に保存するための必要なバイト数の決定”

関数

- “ROUND 関数” (*SAS 関数と CALL ルーチン: リファレンス*)
- “TRUNC 関数” (*SAS 関数と CALL ルーチン: リファレンス*)

ステートメント

- “LENGTH ステートメント” (*SAS DATA ステップステートメント: リファレンス*)

出力形式および入力形式

- “HEXw. 出力形式” (*SAS 出力形式と入力形式: リファレンス*)
- “出力形式のディクショナリ” (*SAS 出力形式と入力形式: リファレンス*)

例: SAS 変数の管理

例: 割り当てステートメントを使用した SAS 変数の作成

コード例

この例では、割り当てステートメントを使用して数値変数と文字変数を作成する方法を示します。数値変数は式の右側の値を整数として評価し、変数型は暗黙的に数値データ型に設定されます。文字変数を作成するには、値を引用符で囲みます。変

数の長さは、文字値の長さに設定されます。文字変数 Patient の長さは 12 に設定されます。

```
data newvar;
  Insurance=100;
  Patient="John Whitman"
run;
```

要点

- 割り当てステートメントの左側で初めて変数を使用することで、新しい変数を作成し値を割り当てることができます。
- 変数は、割り当てステートメントの右側の式と同じ型と長さを取得します。

関連項目

- 割り当てステートメント
- 変数を作成する方法

例: INPUT ステートメントを使用した変数の作成

コード例

この例では、単純なリスト入力を使用して、gems という名前の SAS データセットを作成し、提供されたデータに基づいて 4 つの変数を定義する方法を示します。

```
data gems;
  input Name $ Color $ Carats Owner $;
  datalines;
emerald green 1 smith
sapphire blue 2 johnson
ruby red 1 clark
;
run;
```

INPUT ステートメントは 4 つの変数 Name、Color、Carats、および Owner を読み取ります。Name、Color、および Owner$が指定された文字変数、Carats は数値変数です。

要点

- 生データを SAS に読み取る場合、INPUT ステートメントを使用して、生データ内の位置に基づいて変数を定義できます。
- INPUT ステートメントで次のいずれかの方法を使用して、生データの構成に関する情報を提供できます。
  - カラム入力
  - リスト入力(単純または変更)
  - フォーマットされた入力
  - 名前付き入力

関連項目

- リストされた INPUT ステートメント
- 割り当てステートメント

例: LENGTH ステートメントを使用して SAS 変数を作成

コード例

次の例では、LENGTH ステートメントを使用して新しい SAS 文字変数(CHAR)を作成します。

```
data sales;
  length Salesperson $25;
  Salesperson='Jonathon Mark Walker';
run;
```

この例では、LENGTH ステートメントを使用して新しい SAS 数値変数を DOUBLE として作成します。

```
data round;
  length y 6;
  x=6.495833;
  y=round(x);
  put 'x rounded is ' y;
run;
```

要点

- 文字変数の場合、変数を使用する最初のステートメントで可能な限り長い値を使用する必要があります。これは、同じ DATA ステップ内の後続の LENGTH ステートメントで長さを変更できないためです。SAS の文字変数の最大長は 32,767 バイトです。
- 数値変数の場合、後続の LENGTH ステートメントを使用して変数の長さを変更できます。
- SAS は、文字変数に値を割り当てるとき、ターゲット変数の長さと一致するように、必要に応じて値に空白を埋め込むか、右側の値を切り捨てます。

関連項目

- LENGTH ステートメント
- 割り当てステートメント

例: FORMAT および INFORMAT ステートメントを使用した SAS 変数の変更

コード例

この例では、外部データファイルから生データを読み取り、単純なリスト INPUT とフォーマットされた入力の両方を使用して出力データセットに新しい変数を作成する方法を示します。INPUT ステートメントは、単純なリスト入力(長さは指定されていません)を使用して、変数 Name をデフォルトの長さ 8 の文字変数として作成します。INPUT ステートメントは、同じく長さ 8 の数値変数 Carats も作成します。INPUT ステートメントの最後の変数 Color は INPUT ステートメントで INFORMAT を指定しており、フォーマットされた入力を使用して読み取られるため、SAS は変数を長さ 10 の文字変数として定義します。

```
data gems;
  input Name $ Carats comma3.1 Color $char10.;
  informat Name $char10.;
  format Carats comma3.1;
datalines;
emerald    15 green
aquamarine 20 blue
```

```
;
proc print data=gems; run;
proc contents data=gems; run;

data gems;
  input Name $char10. Carats comma3.1 Color $char.;
datalines;
emerald    15 green
aquamarine 20 blue
;
proc print data=gems; run;
proc contents data=gems; run;
```

アウトプット 4.4　PROC PRINT 結果

Obs	Sale_Price	Date
1	$49.99	January 10, 2019
2	$29.99	January 14, 2019
3	$32.59	January 16, 2019

要点

- 変数がまだ存在せず、フォーマットされた INPUT ステートメントで初めて作成した場合、SAS は INPUT ステートメントで指定された入力形式のカテゴリに基づいて、変数とその属性を定義します。
  - フォーマットされた入力を使用して DATA ステップで変数を初めて作成される変数に、*数値*入力形式を関連付けると、変数はデフォルトの長さが 8 の数値型として作成されます。
  - フォーマットされた入力を使用して DATA ステップで変数を初めて作成される変数に、*文字*入力形式を関連付けると、変数が文字型として作成されます。長さは、INPUT ステートメントの入力形式で指定された幅と一致します。入力形式または DATA ステップの他の場所で長さを指定しない場合、SAS はデフォルトの長さ 8 バイトを割り当てます。
- FORMAT または INFORMAT ステートメントを使用して、変数を変更し、その形式または入力形式を指定できます。

関連項目

- FORMAT ステートメン
- INFORMAT ステートメント
- 変数を作成する方法

例: ATTRIB ステートメントを使用した SAS 変数の変更

コード例

次の DATA ステップでは、Flavor という名前の変数を lollipops という名前のデータセット内に作成します。

```
data lollipops;
  attrib Flavor format=$10.;
  Flavor="Cherry";
run;
```

要点

- ATTRIB ステートメントを使用すると、既存の変数に対して次の変数属性を 1 つ以上指定できます。
  - FORMAT=
  - INFORMAT=
  - LABEL=
  - LENGTH=
- 変数がまだ存在しない場合は、FORMAT=、INFORMAT=、および LENGTH=属性の 1 つ以上を使用して、新しい変数を作成できます。
- LABEL ステートメントまたは ATTRIB ステートメントの LABEL=属性を単独で使用して、新しい変数を作成できません。ラベルは、既存の変数にのみ適用できます。

関連項目

- ATTRIB ステートメント
- 変数を作成する方法

例: 変数属性の表示

コード例

この例では、CONTENTS プロシジャを使用して Sashelp.Cars の変数名、型、および属性を表示する方法を示します。

CONTENTS プロシジャは、*データセット*の内容に関する次のような要約情報を生成します。

- データセット内のオブザベーションの数
- データセット内の変数の数
- 各ページのオブザベーションの最大数とファイルサイズ
- 変数の名前、型、属性(形式、入力形式、ラベルを含む)
- データが変数によって並べ替えられている場合は、並べ替え情報も表示されます。

```
proc contents data=sashelp.cars;
run;
```

CONTENTS プロシジャの出力はエンジンに依存します。

CONTENTS プロシジャによって 生成される変数と属性テーブルのアルファベット順リストは、**#**、**Variable**、**Type**、**Len**、**Format** および **Label** になります。

#
: データセットの列内の変数の元の順序です。PROC CONTENTS は､変数をデータセット内での出現順ではなく、名前のアルファベット順に出力します。

変数
: 変数名を識別します。これは、出力に表示される名前とは異なる場合があります。

型
: 変数が数値(Num)または文字(Char)であるかを示します。

長さ
: 変数の幅を表します。

出力形式
: 変数が**結果**ウィンドウに出力されるときに使用される、割り当てられた形式です。

入力形式
: SAS に読み込むときの変数の元の形式です。

ラベル
: 変数の名前が**出力**ウィンドウに出力されるときに使用される､割り当てられた変数ラベルです。変数にラベルがない場合、この列は変数列と同じになります。

要点

- PROC CONTENTS は構造を表し、データ値ではなくデータセットの変数属性を表示します。
- このプロシジャは、ファイルからデータをインポートし、変数が正しく読み取られていること、および変数型とフォーマットが適切であることを確認する場合に特に便利です。

関連項目

- CONTENTS プロシジャ
- 変数属性

例: 変数属性の変更

コード例

この例では、ATTRIB ステートメントを使用して変数属性を変更する方法を示します。ATTRIB ステートメントは、出力形式、入力形式、ラベル、および長さを 1 つ以上の変数に関連付けます。

```
data flightmiles;
 attrib
   Atlanta label='ATL' length=8 format=comma8.
   Chicago label='ORD' length=8 format=comma8.
   Denver label='DEN' length=8 format=comma8.
   Houston label='IAH' length=8 format=comma8.
   LosAngeles label='LAX' length=8 format=comma8.
   Miami label='MIA' length=8 format=comma8.
   NewYork label='JFK' length=8 format=comma8.
   SanFrancisco label='SFO' length=8 format=comma8.
   Seattle label='SEA' length=8 format=comma8.
   WashingtonDC label='DCA' length=8; format=comma8.
;
  set sashelp.mileages;
run;
title1 'Flying Miles Between Ten US Cities';
proc print data=flightmiles label;
run;
```

アウトプット 4.5　ATTRIB ステートメントの使用 — ラベルと出力形式

Flying Miles Between Ten US Cities

Obs	ATL	ORD	DEN	IAH	LAX	MIA	JFK	SFO	SEA	DCA	City
1	0	.	.	.	.	.	.	.	.	.	Atlanta
2	587	0	.	.	.	.	.	.	.	.	Chicago
3	1,212	920	0	.	.	.	.	.	.	.	Denver
4	701	940	879	0	.	.	.	.	.	.	Houston
5	1,936	1,745	831	1,374	0	.	.	.	.	.	Los Angeles
6	604	1,188	1,726	968	2,339	0	.	.	.	.	Miami
7	748	713	1,631	1,420	2,451	1,092	0	.	.	.	New York
8	2,139	1,858	949	1,645	347	2,594	2,571	0	.	.	San Francisco
9	2,182	1,737	1,021	1,891	959	2,734	2,408	678	0	.	Seattle
10	543	597	1,494	1,220	2,300	923	205	2,442	2,329	0	Washington D.C.

PROC CONTENTS の手順を追加すると、ATTRIB ステートメントが変数属性をどのように変更したのかを確認できます。各変数は、出力形式(オレンジ色の囲み)、長さ(茶色の囲み)、およびラベル(赤色の囲み)です。

アウトプット 4.6　部分出力: PROC CONTENTS

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
1	Atlanta	Num	8	COMMA8.	ATL
2	Chicago	Num	8	COMMA8.	ORD
11	City	Char	15		
3	Denver	Num	8	COMMA8.	DEN
4	Houston	Num	8	COMMA8.	IAH
5	LosAngeles	Num	8	COMMA8.	LAX
6	Miami	Num	8	COMMA8.	MIA
7	NewYork	Num	8	COMMA8.	JFK
8	SanFrancisco	Num	8	COMMA8.	SFO
9	Seattle	Num	8	COMMA8.	SEA
10	WashingtonDC	Num	8	COMMA8.	DCA

要点

- 文字変数の場合、変数を使用する最初のステートメントで可能な限り長い値を使用する必要があります。これは、同じ DATA ステップ内の後続の LENGTH ステートメ

ントで長さを変更できないためです。SAS の文字変数の最大長は 32,767 バイトです。

- DATA ステップで ATTRIB ステートメントを使用すると、変数を含む SAS データセットのディスクリプター情報を変更することにより、属性と変数が永続的に関連付けられます。
- 引数を指定して ATTRIB ステートメントを使用すると、変数に関連付けられた属性を変更できます。

関連項目

- ATTRIB ステートメント
- 変数属性

例: マクロ変数の作成と変数名の変更

コード例

この例では、列名の辞書から変数名を取得してマクロ変数を作成します。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。

```
data q1;                                /* 1 */
  xvar4=4;
  xvar2=2;
  xvar1=1;
  xvar3=3;
  xvar5=5;
run;

proc sql noprint;                       /* 2 */
  select trim(name)||'= new_'||trim(name)
  into :varlist separated by ' '
  from DICTIONARY.COLUMNS
  WHERE LIBNAME EQ "WORK" and MEMNAME EQ "Q1";
quit;
%put &varlist;                          /* 3 */

proc datasets library=work nolist;      /* 4 */
  modify q1;
  rename &varlist;
quit;

proc contents data=q1;                  /* 5 */
run;
```

1 DATA ステートメントは、一時データセット q1 を作成します。

2 SQL プロシジャは、DICTIONARY.COLUMNS の変数名を取得します。LIBNAME および MEMNAME の値が既存のデータセットと一致することを確認します。XVAR の適切な変数の接頭辞として NEW を付けます。

3 %PUT ステートメントは、マクロ変数 VARLIST を作成します。

4 DATASETS プロシジャは、変数名を変更します。ライブラリ参照名とメンバー名を確認し、データセットと一致させます。

5 CONTENTS プロシジャは、名前変更を確認します。

アウトプット 4.7　PROC CONTENTS 出力: 変数の名前変更

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
3	new_xvar1	Num	8
2	new_xvar2	Num	8
4	new_xvar3	Num	8
1	new_xvar4	Num	8
5	new_xvar5	Num	8

要点

- DATASETS プロシジャを使用すると、変数名を変更できます。
- SQL プロシジャを使用すると、マクロ変数を作成できます。

関連項目

- CONTENTS プロシジャ
- DATA ステートメント
- DATASETS プロシジャ
- %PUT ステートメント
- SQL プロシジャ

例: 変数の出力の制御

例: 出力用の変数の選択

コード例

この例では、DROP ステートメントと DROP=データセットオプションを使用して、新しい SAS データセットへの変数の出力を制御します。DROP ステートメントは新しいデータセット newcars から **MSRP** 変数を削除します。DROP=データセットオプションは、元のデータセットから 7 つの変数を削除しますが、sashelp.cars からの読み込み時にはこれらの変数を新しいデータセットに提供しません。

sashelp.cars には、428 のオブザベーションと 15 の変数があります。新しいデータセット newcars には、17 個のオブザベーションと 7 個の変数が含まれています。

```
data newcars;
  set sashelp.cars(drop=origin enginesize cylinders horsepower weight wheelbase length);
  if MSRP>75000 then output;
  drop msrp;
run;
proc print data=newcars;
run;
```

アウトプット 4.8　DROP=および DROP ステートメントの出力

Obs	Make	Model	Type	DriveTrain	Invoice	MPG_City	MPG_Highway
1	Acura	NSX coupe 2dr manual S	Sports	Rear	$79,978	17	24
2	Audi	RS 6 4dr	Sports	Front	$76,417	15	22
3	Cadillac	XLR convertible 2dr	Sports	Rear	$70,546	17	25
4	Dodge	Viper SRT-10 convertible 2dr	Sports	Rear	$74,451	12	20
5	Jaguar	XKR coupe 2dr	Sports	Rear	$74,676	16	23
6	Jaguar	XKR convertible 2dr	Sports	Rear	$79,226	16	23
7	Mercedes-Benz	G500	SUV	All	$71,540	13	14
8	Mercedes-Benz	CL500 2dr	Sedan	Rear	$88,324	16	24
9	Mercedes-Benz	CL600 2dr	Sedan	Rear	$119,600	13	19
10	Mercedes-Benz	S500 4dr	Sedan	All	$80,939	16	24
11	Mercedes-Benz	SL500 convertible 2dr	Sports	Rear	$84,325	16	23
12	Mercedes-Benz	SL55 AMG 2dr	Sports	Rear	$113,388	14	21
13	Mercedes-Benz	SL600 convertible 2dr	Sports	Rear	$117,854	13	19
14	Porsche	911 Carrera convertible 2dr (coupe)	Sports	Rear	$69,229	18	26
15	Porsche	911 Carrera 4S coupe 2dr (convert)	Sports	All	$72,206	17	24
16	Porsche	911 Targa coupe 2dr	Sports	Rear	$67,128	18	26
17	Porsche	911 GT2 2dr	Sports	Rear	$173,560	17	24

要点

- DROP ステートメントまたは DROP=データセットオプションは、DATA ステップ中にどの変数が処理されるかを制御します。
- DROP、KEEP、または RENAME ステートメントを使用すると、変数が出力データセットに書き込まれるときに常にアクションが発生します。
- SAS データセットオプションでは、オプションを使用する場所によって、アクションが発生するタイミングが決まります。オプションが入力データセットで使用される場合、変数は、プログラムデータベクトルに読み取られる前に、削除、保持、または名前変更されます。
- 出力データセットで使用する場合、データセットオプションは、変数が新しい SAS データセットに書き込まれるときに適用されます。

関連項目

- DROP ステートメント
- DROP=データセットオプション

- ステートメントまたはデータセットオプションの使用

例: 処理する変数の選択

コード例

この例では、DROP=および RENAME=データセットオプションと INPUT 関数を使用して変数 Day を数字から文字へ変換します。新しい変数 Weekday を出力データセットに書き込むことができるように、処理前に変数名 Day を Weekday に変更します。変数 Day は出力データセットから削除され、新しい名前 Weekday がプログラムステートメントで使用されることに注意してください。

```
data fails (drop=Process);
  length Day 8;
  set sashelp.failure(rename=(Day=Weekday));
  Day=input(Weekday,8.);
run;
proc print data=fails;
run;
```

要点

- DROP=データセットオプションは、DATA ステップ中にどの変数が処理または出力されるかを制御します。
- SAS データセットオプションでは、オプションを使用する場所によって、アクションが発生するタイミングが決まります。オプションが入力データセットで使用される場合、変数は、プログラムデータベクトルに読み取られる前に、削除、保持、または名前変更されます。出力データセットで使用する場合、データセットオプションは、変数が新しい SAS データセットに書き込まれるときに適用されます。

関連項目

- RENAME=データセットオプション
- DROP=データセットオプション
- ステートメントまたはデータセットオプションの使用

例: 変数の並べ替えと整列

例: ATTRIB ステートメントを使用した変数の並べ替え

コード例

次の例では、データセット sashelp.class は、変数 Name、Sex、Age、Height、および Weight を含んでいます(この順序で)。ATTRIB ステートメントは SET ステートメントの前に指定されるため、変数 Sex は出力データセットの最初の位置に移動します。

```
data class3;
  attrib Sex length=$8;
  set sashelp.class;
  if Sex='M' then Sex='Male';
    else Sex='Female';
run;
proc print data=class3;
run;
```

ATTRIB ステートメントが SET ステートメントの前にあるため、出力には変数 Sex が最初にリストされます。

アウトプット 4.9　ATTRIB ステートメントを使用した変数の並べ替え

Obs	Sex	Name	Age	Height	Weight
1	Male	Alfred	14	69.0	112.5
2	Female	Alice	13	56.5	84.0
3	Female	Barbara	13	65.3	98.0
4	Female	Carol	14	62.8	102.5
5	Male	Henry	14	63.5	102.5
6	Male	James	12	57.3	83.0
7	Female	Jane	12	59.8	84.5
8	Female	Janet	15	62.5	112.5
9	Male	Jeffrey	13	62.5	84.0
10	Male	John	12	59.0	99.5
11	Female	Joyce	11	51.3	50.5
12	Female	Judy	14	64.3	90.0
13	Female	Louise	12	56.3	77.0
14	Female	Mary	15	66.5	112.0
15	Male	Philip	16	72.0	150.0
16	Male	Robert	12	64.8	128.0
17	Male	Ronald	15	67.0	133.0
18	Male	Thomas	11	57.5	85.0
19	Male	William	15	66.5	112.0

要点

- ATTRIB ステートメントを使用すると、SAS 出力に変数が表示される順序を制御できます。
- 変数の順序を変更するには、SET、MERGE、または UPDATE ステートメントの前に ATTRIB ステートメントを使用します。
- ATTRIB ステートメントにリストされていない変数は、元の位置を保持します。

関連項目

- ARRAY ステートメント
- ATTRIB ステートメント
- FORMAT ステートメン
- INFORMAT ステートメント
- LENGTH ステートメント
- RETAIN ステートメント
- CONTENTS プロシジャ

例: LENGTH ステートメントを使用した変数の並べ替え

コード例

次の例では、データセット sashelp.class は、変数 Name、Sex、Age、Height、および Weight を含んでいます(この順序で)。LENGTH ステートメントは SET ステートメントの前に指定されるため、変数 Height は出力データセットの最初の位置に移動します。

```
data class1;
  length Height 3;
  set sashelp.class;
run;
proc print data=class1;
run;
```

LENGTH ステートメントが SET ステートメントの前にあるため、出力には変数 Height が最初にリストされます。

アウトプット 4.10　LENGTH ステートメントを使用した変数の並べ替え

Obs	Height	Name	Sex	Age	Weight
1	69.0000	Alfred	M	14	112.5
2	56.5000	Alice	F	13	84.0
3	65.2969	Barbara	F	13	98.0
4	62.7969	Carol	F	14	102.5
5	63.5000	Henry	M	14	102.5
6	57.2969	James	M	12	83.0
7	59.7969	Jane	F	12	84.5
8	62.5000	Janet	F	15	112.5
9	62.5000	Jeffrey	M	13	84.0
10	59.0000	John	M	12	99.5
11	51.2969	Joyce	F	11	50.5
12	64.2969	Judy	F	14	90.0
13	56.2969	Louise	F	12	77.0
14	66.5000	Mary	F	15	112.0
15	72.0000	Philip	M	16	150.0
16	64.7969	Robert	M	12	128.0
17	67.0000	Ronald	M	15	133.0
18	57.5000	Thomas	M	11	85.0
19	66.5000	William	M	15	112.0

ヒント　CONTENTS プロシジャを Class1 データセットに使用して、各変数の長さを表示できます。

要点

- LENGTH ステートメントを使用すると、SAS 出力に変数が表示される順序を制御できます。
- 変数の順序を変更するには、SET、MERGE、または UPDATE ステートメントの前に LENGTH ステートメントを使用します。
- LENGTH ステートメントにリストされていない変数は、元の位置を保持します。

関連項目

- ARRAY ステートメント
- ATTRIB ステートメント
- FORMAT ステートメン
- INFORMAT ステートメント
- LENGTH ステートメント
- RETAIN ステートメント
- CONTENTS プロシジャ

例: RETAIN ステートメントを使用した変数の並べ替え

コード例

次の例では、RETAIN ステートメントにより、変数 Weight および Age は、出力データセットの最初にリストされます。データセット sashelp.class は、変数 Name、Sex、Age、Height、および Weight を含んでいます(この順序で)。

```
data class2;
  retain Weight Age;
  set Sashelp.Class;
run;
proc print data=class2;
run;
```

RETAIN ステートメントが SET ステートメントの前にあるため、出力には変数 Weight および Age が最初にリストされます。

アウトプット 4.11　RETAIN ステートメントを使用した変数の並べ替え

Obs	Weight	Age	Name	Sex	Height
1	112.5	14	Alfred	M	69.0
2	84.0	13	Alice	F	56.5
3	98.0	13	Barbara	F	65.3
4	102.5	14	Carol	F	62.8
5	102.5	14	Henry	M	63.5
6	83.0	12	James	M	57.3
7	84.5	12	Jane	F	59.8
8	112.5	15	Janet	F	62.5
9	84.0	13	Jeffrey	M	62.5
10	99.5	12	John	M	59.0
11	50.5	11	Joyce	F	51.3
12	90.0	14	Judy	F	64.3
13	77.0	12	Louise	F	56.3
14	112.0	15	Mary	F	66.5
15	150.0	16	Philip	M	72.0
16	128.0	12	Robert	M	64.8
17	133.0	15	Ronald	M	67.0
18	85.0	11	Thomas	M	57.5
19	112.0	15	William	M	66.5

要点

- RETAIN ステートメントは、他の変数属性の指定が必要ないという理由により、変数の順序を変更するためによく使用されます。
- RETAIN ステートメントは、データセットから読み込む既存の変数の値の保持には影響しません。
- 位置が関連する変数のみ指定する必要があります。RETAIN ステートメントに指定されていない変数は、元の位置を保持します。
- 変数の順序を変更するには、SET、MERGE、または UPDATE ステートメントの前に RETAIN ステートメントを使用します。

関連項目

- ARRAY ステートメント

- ATTRIB ステートメント
- FORMAT ステートメン
- INFORMAT ステートメント
- LENGTH ステートメント
- RETAIN ステートメント
- CONTENTS プロシジャ

例: Format ステートメントを使用した変数の並べ替え

コード例

次の例では、最初の DATA ステップでデータセット investment を作成します。ここで変数 Material、Item、Investment、および Profit が定義されています。

FORMAT ステートメントにより、変数 Item が出力データセットの最初にリストされます。データセット investment は、変数 Material、Item、Investment、および Profit を含んでいます(この順序で)。

```
data investment;
  input Material $1-7 Item $9-15 Investment Profit;
  datalines;
    cotton  shirts 2256354 83952175
    silk    ties   498678  2349615
    silk    suits  9482146 69839563
    leather belts  7693    14893
    leather shoes  7936712 22964
;
run;
data invest01;
  format Item Material $upcase9. Investment Profit dollar15.2;
  set investment;
run;
proc print data=invest01;
run;
```

次の出力には、赤い枠で囲まれたセクションがあり、変数 Item が最初にリストされます。オレンジのボックス内セクションは、FORMAT ステートメントの変換を示しています。Item および Material 変数は大文字に変更され、**Investment** と **Profit** にドル記号、カンマ、小数点以下 2 桁が含まれます。

アウトプット 4.12　Format ステートメントを使用した変数の並べ替え

Obs	Item	Material	Investment	Profit
1	SHIRTS	COTTON	$2,256,354.00	$83,952,175.00
2	TIES	SILK	$498,678.00	$2,349,615.00
3	SUITS	SILK	$9,482,146.00	$69,839,563.00
4	BELTS	LEATHER	$7,693.00	$14,893.00
5	SHOES	LEATHER	$7,936,712.00	$22,964.00

要点

- Format ステートメントを使用すると、SAS 出力に変数が表示される順序を制御できます。
- 変数の順序を変更するには、SET、MERGE、または UPDATE ステートメントの前に Format ステートメントを使用します。
- 1 つの FORMAT ステートメントで、同じフォーマットを複数の変数に関連付けたり、異なるフォーマットを異なる変数に関連付けたりできます。
- FORMAT ステートメントを使用して変数の順序を変更する場合、フォーマットを変数に関連付ける必要はありません。
- INFORMAT ステートメントを使用して変数の順序を変更することもできます。

関連項目

- ARRAY ステートメント
- ATTRIB ステートメント
- FORMAT ステートメン
- INFORMAT ステートメント
- LENGTH ステートメント
- RETAIN ステートメント
- CONTENTS プロシジャ

例: 変数型を変換する

例: 文字変数を数値に変換

コード例

この例では、文字データ値を数値に明示的に変換する方法を示します。INPUT 関数は、変数 rate を明示的に数値に変換し、rate の長さは 2 で、数値入力形式 2 は変数の値を読み取るために使用されます。データセットを出力し、新しい変数 pay_chk の数値を確認できます。

```
data work.weeksal;
  set work.payscale;
  pay_chk=input(rate,2.)*Hours;
run;
proc print data=work.weeksal;
run;
```

注: INPUT 関数を使用すると、SAS ログに変換メッセージが出力されません。

アウトプット 4.13　PROC PRINT 結果

Obs	EmployeeID	Rate	Hours	Dept	Site	PayChk
1	16102	12	40	OS	1	480
2	12414	10	35	OS	1	350
3	10175	15	40	TS	1	600
4	10477	16	40	TS	2	640
5	10775	10	36	OS	2	360
6	10771	10	32	OS	1	320
7	10739	10	20	OS	2	200
8	17344	15	40	TS	1	600
9	16885	18	40	TS	2	720

要点

- 明示的な変換は、データ型の制御に役立ち、変換エラーの発生を回避します。

関連項目

- “INPUT 関数” (*SAS 関数と CALL ルーチン: リファレンス*)
- 明示的な文字から数値への変換

例: 数値変数を文字に変換

コード例

この例では、PUT 関数を使用して数値から文字への変換を明示的に行う方法を示します。割り当てステートメントで PUT 関数を使用します。Site はソース変数です。Site 長さは 2 なので、数値出力形式として 2.を選択します。DATA ステップは、Loc という名前の新しい変数を、割り当てステートメントからデータセットに追加します。PROC PRINT を使用して、変数 Loc を表示します。

```
data work.dept;
  set work.payscale;
  Loc=catx('/',put(Site,2.),Dept);
run;
proc print data=work.dept;
run;
```

アウトプット 4.14　PROC PRINT 結果

Obs	EmployeeID	Rate	Hours	Dept	Site	Loc
1	16102	12	40	OS	1	1/OS
2	12414	10	35	OS	1	1/OS
3	10175	15	40	TS	1	1/TS
4	10477	16	40	TS	2	2/TS
5	10775	10	36	OS	2	2/OS
6	10771	10	32	OS	1	1/OS
7	10739	10	20	OS	2	2/OS
8	17344	15	40	TS	1	1/TS
9	16885	18	40	TS	2	2/TS

注: PUT 関数を使用すると、SAS ログに変換メッセージは出力されません。

要点

- 数値の前にあるかっこまたはマイナス記号(空白を挟まない)は、負の値を示します。
- 先頭の 0 と入力フィールド内の値の配置は、変数に割り当てられる値には影響しません。先頭の 0、先頭および末尾の空白は値とともに保存されません。一部の言語とは異なり、SAS はデフォルトでは末尾の空白をゼロとして読み取りません。末尾の空白をゼロとして読み取るには、BZ.入力形式を使用します。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。
- 数値データには先頭と末尾に空白を含めることができますが、埋め込み空白を含めることはできません(COMMA.または BZ.入力形式で読み取られる場合を除く)。
- 明示的な小数点を含まない入力行から 10 進数値を読み取るには、カラム入力で 10 進数パラメーターを使用するか、フォーマットされた入力で入力形式を使用して、小数点が属する場所を指定します。入力データ内の明示的な小数点は、INPUT ステートメントの小数点指定をオーバーライドします。

関連項目

- "INPUT 関数" (*SAS 関数と CALL ルーチン: リファレンス*)
- 明示的な数値から文字への変換

例: 自動的な型の変換

コード例

デフォルトでは、算術演算などの数値コンテキストで文字変数を参照すると、SAS は変数値を数値に変換しようとします。例では、Rate は文字変数ですが、数値コンテキストで使用されます。したがって、SAS は算術演算を完了できるように、自動的に Rate の値を数値に変換します。

数値データから文字データへの自動変換は、文字から数値への変換と非常によく似ています。数値データ値は、文字コンテキストで使用される場合は必ず、SAS によって暗黙的に文字値に変換されます。

この例では、Site は文字コンテキストで使用されるため、SAS は Site の数値を文字値に自動的に変換します。さらに、変数 Site は文字値を必要とする連結演算子とともに表示されます。

```
data work.autosal;
  set work.payscale;
```

```
  PayChk=Rate*Hours;
  Loc=Site||'/'||Dept;
run;
proc print data=work.autosal;
run;
```

データが自動的に変換されるたびに、変換が行われたことを示すメッセージが SAS ログに出力されます。

アウトプット 4.15　SAS ログ

```
NOTE: Character values have been converted to numeric values at the places given
by: (Line):(Column).
       75:10
NOTE: Numeric values have been converted to character values at the places given
by: (Line):(Column).
       76:7
NOTE: There were 9 observations read from the data set WORK.PAYSCALE.
NOTE: The data set WORK.AUTOSAL has 9 observations and 7 variables.
```

要点

- 自動変換は次の状況で発生します。
  - 文字値は、事前に定義された数値変数に割り当てられます。
  - 文字値は、演算子式で使用されます。
  - 文字値は、比較演算子を使用して数値と比較されます。
  - 文字値は、数値引数を必要とする関数が指定されています。
  - 数値データ値は、文字コンテキストで使用される場合は必ず文字値に変換されます。
- 自動変換については次のことが当てはまります。
  - *w*.入力形式を使用します。w は変換される文字値の幅を表します。
  - 標準の数値表記法(オプションの小数点、先行する符号、または指数表記を含む数字)に準拠していない文字値から数値欠損値が生成されます。

関連項目

- 文字から数値への自動変換
- 数値から文字への自動変換

例: 自動変数の使用

例: _N_自動変数の使用

コード例

次の例は、PUT 関数で_N_自動変数を使用する方法を示しています。

```
data test;
  input x $ y;
  put 'This is row ' _n_;
datalines;
  a 1
  b 2
  c 3
  x 24
  z 26
;
run;
proc print data=test;
run;
```

次のように SAS ログに書き込まれます。

アウトプット 4.16　SAS ログ: _N_自動変数

```
This is row 1
This is row 2
This is row 3
This is row 4
This is row 5
```

以下は、PRINT プロシジャからの出力です。

アウトプット 4.17　自動変数_N_ PRINT 出力

Obs	x	y
1	a	1
2	b	2
3	c	3
4	x	24
5	z	26

要点

- DATA ステップが DATA ステートメントをループするたびに、変数_N_は 1 ずつ増加します。
- _N_の値は、DATA ステップが反復された回数を表します。

関連項目

- 自動変数

例: IF/THEN ステートメントで_ERROR_自動変数と_INFILE_自動変数を使用

コード例

次の例では、ERROR と INFILE=を使用して、DATA ステップの各反復中にエラーが発生した入力レコードのコンテンツを SAS ログに書き込む方法を示します。

```
data testerr;
  input x $ y;
    if _error_=1 then put 'Error before row ' _n_ 'which contains ' _infile_;
    put _infile_;
datalines;
  a 1
  *^*#
  b 2
  c3
;
run;
```

次のように SAS ログに書き込まれます。

例のコード 4.2　SAS ログ: _ERROR_および_INFILE_

```
NOTE: Invalid data for y in line 65 2-2.

Error before row 2 which contains
 b 2
```

要点

- 自動変数は、DATA ステップまたは DATA ステップステートメントによって自動的に作成されます。これらの変数はプログラムデータベクトルに追加されますが、作成中のデータセットに出力として送信されません。
- _INFILE_は、INFILE ステートメントを使用するときに DATA ステップによって自動的に作成される自動文字変数です。これには、ファイルまたは DATALINES ステートメントのデータで読み込まれた現在の入力レコード(行)の値が含まれます。
- _ERROR_の値を使用すると、データレコード内のエラーを特定し、エラーメッセージを SAS ログに出力できます。

関連項目

- 自動変数
- “_INFILE_=variable” (*SAS DATA ステップステートメント: リファレンス*)

例: 変数リストの作成

例: 番号付き範囲変数リストの作成

コード例

この例では、番号付き範囲変数リストを作成する方法を示します。ユーザー指定の名前のルールに違反せず、番号が連続している限り、任意の番号で開始し、任意の番号で終了することができます。INPUT ステートメントを使用して、番号付き範囲リストを書き込みます。ARRAY ステートメントで番号付き範囲リストを使用することもできます。

```
data temperatures;
  input Day1-Day7;
  datalines;
     74 75 82 84 85 86 89
 ;
run;
proc print data=temperatures noobs;
```

```
  title "Average Daily Low Temperature";
run;
data tempCelsius(drop=i);
  set temperatures;
  array celsius{7} Day1-Day7;
  do i=1 to 7;
    celsius{i}=(celsius{i}-32)*5/9;
  end;
run;
proc print data=tempCelsius;
  title "Average Daily Low Temperature in Celsius";
run;
```

次の図は、温度の PROC PRINT 出力を示しています。赤で強調表示されたボックスは、番号付き範囲変数リスト Day1–Day7 を示します。

アウトプット 4.18　PROC PRINT 出力: 温度

Average Daily Low Temperature

Day1	Day2	Day3	Day4	Day5	Day6	Day7
74	75	82	84	85	86	89

Average Daily Low Temperature in Celsius

Day1	Day2	Day3	Day4	Day5	Day6	Day7
23.3333	23.8889	27.7778	28.8889	29.4444	30	31.6667

要点

- 番号付き範囲変数リストは、同じ名前の接頭辞が付いているが、最後の文字の数値が異なる変数で構成されるリストです。
- 数値は連続した数字である必要があります。
- 番号付き範囲リストでは、名前に同じ接頭辞が付いていれば、作成された変数を任意の順序で参照できます。

関連項目

- ARRAY ステートメント
- リストされた INPUT ステートメント
- 変数リストの種類

例: 変数名の範囲リストを作成する

コード例

この例では、KEEP ステートメントで指定された名前範囲リストは、nAtBat および nOuts を含むその間のすべての数値変数を保持します。ARRAY ステートメントで指定された名前範囲リストは、nAtbat および nRuns の間のすべての変数を読み込みます。この場合、変数 nHits および nHome も含みます。

注: 範囲リスト内のすべての変数は、大文字と小文字が同じである必要があります。

PRINT プロシジャの VAR ステートメントで指定された名前範囲リストは、Name および nBB を含むその間のすべての変数を PROC PRINT 出力で出力することを指定します。

```
data changeStats(where=(YrMajor>18));
 set sashelp.baseball;
 keep Name nAtBat-numeric-nOuts YrMajor;
 array stats(4) nAtBat--nRuns;
  do i=1 to 4;
   stats{i} = stats{i}*10;
  end;
run;
proc print data=changeStats;
 var Name--nBB;
run;
```

Obs	Name	nAtBat	nHits	nHome	nRuns	nRBI	nBB
1	Baker, Dusty	2420	580	40	250	19	27
2	Nettles, Graig	3540	770	160	360	55	41
3	Rose, Pete	2370	520	0	150	25	30
4	Jackson, Reggie	4190	1010	180	650	58	92
5	Perez, Tony	2000	510	20	140	29	25
6	Simmons, Ted	1270	320	40	140	25	12

要点

- 配列を宣言する前に変数を定義している限り、ARRAY 宣言で名前範囲リストを使用できます。変数は、同じ DATA ステップまたは前の DATA ステップで定義できます。

- 名前範囲リストでは、変数間の範囲を指定するために 2 つのハイフン(--)を使用し、番号付き範囲リストでは範囲を指定するために 1 つのハイフンを使用することに注意してください。

関連項目

- ARRAY ステートメント
- KEEP ステートメント
- VAR ステートメント
- 変数リストの種類

例: 変数リストで OF 演算子を使用

コード例

次の例では、OF 演算子を使用する場合と使用しない場合の両方で、引数が番号付き範囲リストとして渡されます。

```
data _null_;
  x1=30; x2=20; x3=10;
  T=sum(x1-x3);        /* #1 */
  T2=sum(OF x1-x3);    /* #2 */
  put T=;              /* #3 */
  put T2=;             /* #4 */
run;
```

1 最初の SUM 関数は、T=20 を返します。

2 2 番目の SUM 関数は、T2=60 を返します。

3 x1 および x3 間の差異をログに書き込みます。

4 変数 x1、x2、および x3 の値の合計をログに書き込みます。

要点

- OF 演算子を使用すると、SAS 変数リストまたは SAS 配列を関数の引数として指定できます。
- OF 演算子は、引数が番号付き範囲リストの形式である関数で使用する場合に重要です。たとえば、番号付き範囲(x1–xn)の形式の引数は、リストの前に OF 演算子がある場合にのみ値の範囲として読み込まれます。

- 同じリストの前に OF 演算子がなく、それが SUM 関数で使用される場合、(-)文字は減算記号として扱われます。この関数は、値の範囲の合計ではなく、変数間の差を返します。

関連項目

- SUM 関数
- PUT ステートメント
- 一時配列での OF 演算子の使用

例: OF 演算子を使用して複数の変数リストを作成

コード例

次の例は、リスト全体の前に OF 演算子があり、リストが空白またはカンマで区切られている範囲変数リストを示しています。

```
data _null_;
  T=sum(OF x1-x3 y1-y3 z1-z3);
run;
```

または

```
data _null_;
  T=sum(OF x1–x3, OF y1–y3, OF z1–z3);
run;
```

要点

- OF 演算子を使用すると、SAS 変数リストまたは SAS 配列を関数の引数として指定できます。
- OF 演算子は、複数の範囲指定されたリストが関数の引数として使用される場合に、ある変数リストを別の変数リストから区別するためにも使用されます。

関連項目

- 変数リストの種類

- 関数と変数リストを使用した OF 演算子
- 一時配列での OF 演算子の使用

例: 変数値の欠損の管理

例: 欠損値を自動的に置換

コード例

SAS は、変数に割り当てられた値を検出すると、欠損値を置き換えます。したがって、プログラムステートメントを使用して新しい変数を作成する場合、次の DATA ステップに示すように、割り当てステートメントで値を割り当てるまで、各オブザベーションの値は欠損します。

```
data new;
  input x;
  if x=1 then y=2;
  datalines;
    4
    1
    3
    1
;
run;
proc print data=new;
run;
```

y が作成され、x=1 の場合に **2** に設定されます。それ以外の場合は、y は欠損に設定されます。

アウトプット 4.19　欠損値の DATA ステップ出力

Obs	x	y
1	4	.
2	1	2
3	3	.
4	1	2

要点

- SAS は、変数に割り当てられた値を検出すると、欠損値を置き換えます。
- DATA ステップの各反復の開始時に、SAS は DATA ステップで作成した各変数の値を欠損値に設定します。

関連項目

- 生データを読み取る場合

例: 特別な欠損値の作成

コード例

次の例では、マーケティング調査会社からのデータを使用します。5 人のテスターを雇用して、5 つの異なる製品の使いやすさと有効性をテストしました。テスターが不在の場合、報告する評価はなく、値は不在を表す **X** で記録されます。テスターが製品を適切にテストできなかった場合、評価は行われず、値は不完全なテストを表す **I** として記録されます。次のプログラムはデータを読み込み、結果の SAS データセットを表示します。最初と 3 番目のデータ行にある特別な欠損値に注意してください。

```
data period_a;
  missing X I;
  input Id $4. Foodpr1 Foodpr2 Foodpr3 Coffeem1 Coffeem2;
  datalines;
1001 115 45 65 I 78
1002 86 27 55 72 86
1004 93 52 X 76 88
1015 73 35 43 112 108
1027 101 127 39 76 79
  ;

proc print data=period_a;
  title 'Results of Test Period A';
  footnote1 'X indicates TESTER ABSENT';
  footnote2 'I indicates TEST WAS INCOMPLETE';
run;
```

次の出力が生成されます。

アウトプット 4.20　複数の欠損値を含む出力

Results of Test Period A

Obs	Id	Foodpr1	Foodpr2	Foodpr3	Coffeem1	Coffeem2
1	1001	115	45	65	I	78
2	1002	86	27	55	72	86
3	1004	93	52	X	76	88
4	1015	73	35	43	112	108
5	1027	101	127	39	76	79

X indicates TESTER ABSENT
I indicates TEST WAS INCOMPLETE

要点

- SAS に特殊欠損値として解釈させたい数値フィールドの文字がデータ値に含まれている場合、MISSING ステートメントを使用してそれらの文字を指定します。
- 特殊数値欠損値をピリオドで開始しない場合、SAS はそれを*変数*名として識別します。したがって、*SAS 式*または割り当て*ステートメント*で特殊数値欠損値を使用するには、値をピリオドで始め、その後に文字またはアンダースコアを付ける必要があります。次に例を示します。x=.d;

関連項目

- MISSING ステートメント
- 欠損値の表現

例: 欠損値のプロパゲーションの防止

コード例

欠損値が算術式にプロパゲーションすることを望まない場合は、サンプル統計関数を使用して計算から欠損値を省略できます。SUM ステートメントは欠損値も無視するため、c も **5** です。たとえば、次の DATA ステップを考えます。

```
data test;
```

```
  x=.;
  y=5;
  a=x+y;
  b=sum(x,y);
  c=5;
  c+x;
  put a= b= c=;
run;
```

式で x および y を加算すると、x の値が欠損しているため、欠損の結果が生成されます。したがって、a の値は欠損します。しかし、SUM 関数は欠損値を無視するため、x に y を加算すると、欠損値ではなく値 **5** が生成されます。

アウトプット 4.21　統計関数の欠損値の SAS ログ結果

```
184  data test;
185    x=.;
186    y=5;
187    a=x+y;
188    b=sum(x,y);
189    c=5;
190    c+x;
191    put a= b= c=;
192   run;

a=. b=5 c=5
NOTE: Missing values were generated as a result of performing
      an operation on missing values.
      Each place is given by:
      (Number of times) at (Line):(Column).
      1 at 187:6
NOTE: The data set WORK.TEST has 1 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

要点

- SUM 関数と SUM ステートメントは、計算時に欠損値を無視します。
- 算術計算で欠損値を使用すると、SAS はその計算の結果を欠損値に設定します。
- SAS ログには、欠損値を含む算術式と、作成日時が出力されます。

関連項目

- MISSING 関数
- 変数値の欠損

例: 精度に関連する問題の管理

例: SAS における不正確な値の比較

コード例

この例では、SAS は変数 point_three および three_time_point_one を、それぞれ **0.3** および**(3 x 0.1)**に設定します。次に、一方を他方から減算して 2 つの値を比較し、結果を SAS ログに書き込みます。

```
data a;
  point_three=0.3;
  three_time_point_one=3*0.1;
  difference=point_three - three_times_point_one;
  put 'The difference is ' difference;
run;
```

ログ出力には、10 進数の算術のように、**(3 x 0.1)** — **0.3** が **0** に等しくないことが示されています。その理由は、値が浮動小数点表現で保存されており、正確に保存できないためです。詳細については、“精度に関するエラーのトラブルシューティング”を参照してください。

アウトプット 4.22　SAS で不正確な値を比較するログ出力

```
The difference is -5.55112E-17
```

要点

- 10 進数で不正確な数値は、2 進数で不正確な数値と必ずしも同じではありません。
- SAS で不正確な数値に対して計算と比較を実行すると、予期しない結果が生じる可能性があります。
- 2 進数に相当する、2 進数が無限に繰り返される 10 進数の小数が多数あるため、一般的な有理数の結果を 10 進数で解釈する場合は注意が必要です。どちらの記数法でも問題にならない有理数がいくつかあります。

関連項目

- Numeric Precision

例: 10 進値を浮動小数点表現に変換する

コード例

この例では、10 進数値 255.75 を浮動小数点表現に変換するプロセスを示します。

1 基数 2 の記数法を使用して、値 255.75 を 2 進数で書き込みます。

注: 仮数の各ビットは、分子が 1、分母が 2 のべき乗である分数を表します。仮数は、1/2、1/4、1/8 などの一連の分数の合計です。したがって、浮動小数点数を正確に表現するには、前述の合計として表現する必要があります。

基数 2										
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	$.2^{-1}$	2^{-2}
	128	64	32	16	8	4	2	1	1/2	1/4
255.75 =	1 x 2^7	1 x 2^6	1 x 2^5	1 x 2^4	1 x 2^3	1 x 2^2	1 x 2^1	1 x 2^0	1 x 2^{-1}	1 x 2^{-2}

$$255.75 = (\mathbf{1} \times 2^7) + (\mathbf{1} \times 2^6) + (\mathbf{1} \times 2^5) + (\mathbf{1} \times 2^4) + (\mathbf{1} \times 2^3) + (\mathbf{1} \times 2^2) + (\mathbf{1} \times 2^1) + (\mathbf{1} \times 2^0) + (\mathbf{1} \times 2^{-1}) + (\mathbf{1} \times 2^{-2})$$

decimal point

したがって、値 255.75 はバイナリ形式で 1111 1111.11 と表されます。

2 1 桁だけになるまで左に小数点を移動します。このプロセスを、値の正規化と言います。指数表記における値の*正規化*は、仮数の絶対値が 1 以上 10 未満になるように指数を選択するプロセスです。この数値では、小数点を 7 桁移動します。

1.111 1111 11

小数点を 7 桁移動したため、指数は 7 になります。

3 バイアスは 1023 なので、1023 に 7 を加算すると次のようになります。

1030

4 10 進数値 1030 を 16 進数に変換するには、基数 16 の記数法を使用します。

基数 16						
16^7	...	16^4	16^3	16^2	16^1	16^0
268,435,456	...	65,536	4096	256	16	1

$$1030 = (\mathbf{4} \times 16^2) + (\mathbf{0} \times 16^1) + (\mathbf{6} \times 16^0)$$
$$= 1024 + 0 + 6$$

1030 に変換された 16 進値は、最終結果の指数部に配置されます。

5 406 をバイナリ形式に変換します。

```
0100 0000 0110
 4   0    6
```

変換する値が負の場合は、最初のビットを 1 に変更します。

```
1100 0000 0110
```

これを 16 進数に変換すると、次のようになります。

```
C   0   6
```

6 上記の手順 2 で、最初の桁と小数点(暗黙の 1 ビット)を削除します。

```
11111111
```

7 これらをニブル(半バイト)に分割すると、次のようになります。

```
1111 1111 1
```

8 最後に完全なニブルを持たせるには、4 ビットを完成させるのに十分なゼロを追加します。

```
 1111 1111 1000
```

9 次の値

```
1111 1111 1000
```

を、それに相当する 16 進数に変換して、仮数部分を取得します。

```
1111  1111  1000
F    F    8
```

255.75 の最終的な浮動小数点表現は、次のようになります。

```
406F F800 0000 0000
```

–255.75 の最終的な浮動小数点表現は、次のようになります。

```
C06F F800 0000 0000
```

この例では、開始時の 10 進数値である 255.75 は、2 進数と 16 進数の両方で丸めなしで表現できる有限の 2 進数値に変換されます。

要点

- プロセッサは拡大実精度で計算を実行します。
- Linux などのオペレーティングシステムで使用される、基本的な IEEE 浮動小数点形式よりも大きい数値を保存します。これが、同じ IEEE 標準を使用するオペレーティングシステム間でわずかに異なる値が表示される理由の 1 つです。

関連項目

- IEEE 規格を使用した浮動小数点表現

例: 10 進数値を 16 進浮動小数点表現に変換

コード例

次の例は、10 進数値 512.1 を 16 進浮動小数点表現に変換するプロセスを示しています。この例は、10 進数で正確に表現できる値が 16 進浮動小数点では正確に表現できないことを示しています。

1 基数は 16 であるため、最初に値 512.1 を 16 進表記に変換する必要があります。

2 まず、整数部分 512 を、基数 16 の記数法を使用して 16 進数に変換します。

基数 16						
16^7	...	16^4	16^3	16^2	16^1	16^0
268,435,456	...	65,536	4096	256	16	1

$$200 = .200 \times 16^3$$

値 512 は、16 進数では 200 と表されます。

3 16 進数 200 を浮動小数点表現で記述します。これを行うには、小数点を左端まで移動し、移動した位置の数を数えます。移動した数値が指数になります。

$$200 = .200 \times 16^3$$

4 元の数値 512.1 の小数部(.1)を 16 進数に変換します。

$$.1 = \frac{1}{10} = \frac{1.6}{16}$$

分子を小数にできないため、1 をそのままにして 0.6 の部分を再度変換します。

$$.6 = \frac{6}{10} = \frac{9.6}{16}$$

さらに、分子に小数を含めることはできないため、9 を保持し、0.6 の部分を再度変換します。

.6 は 9.6 として繰り返されます。これは、9 を保持して再変換することを意味します。.1 を 16 進数で表すことができる最も近い表現は次のとおりです。

$$.1 = .1999999 \times 16^{0}$$

5 値の指数は 3 です(上記の手順 2)。格納される実際の指数を決定するには、指数値を取得し、それにバイアスを加算します。

true exponent + bias = 3 + 40 = 43 (hexadecimal) = stored exponent

最後に決定する部分は仮数部の符号です。慣例により、正の仮数の符号ビットは 0、負の仮数の符号は 1 です。この情報は、最初のバイトの最初のビットに格納されます。手順 4 の 16 進値から、同等の 10 進数を計算し、バイナリ形式で書き込みます。最初の位置に符号ビットを追加します。保存された値は次のようになります。

$$43 \quad hexadecimal \quad = (4 \times 16^{1}) + (3 \times 16^{0}) = 67 \quad decimal \quad = 0100\,0003 \quad binary$$

$$11000003 = 195 \quad in \quad decimal \quad = C3 \quad in \quad hexadecimal$$

6 最後の手順では、すべてをまとめます。

4320019999999999 – floating point representation for 512.1

C320019999999999 – floating point representation for –512.1

したがって、10 進数値 512.1 は、2 進数または 16 進数の浮動小数点表記では正確に表現できません。数値 512.1 を変換すると、結果は無限に繰り返される数値になります。これは、分数 1/3 を 10 進数で表すことに似ています。

近似値は、3 を無限に繰り返す.33333333 になります。

要点

- 10 進数表記で正確に表現できる値が、浮動小数点表記では正確に表現できるとは限りません。
- 浮動小数点値に数値の繰り返しパターンがある場合、値を正確に表現できない可能性が高くなります。

関連項目

- IEEE 規格を使用した浮動小数点表現

例: 計算エラーを避けるために値を丸める

コード例

次の例は、ROUND 関数を使用して DATA ステップの 1 回の反復の結果を丸める方法を示しています。

```
data _null_;
  do i=-1 to 1 by .1;
    i=round(i, .1);
    put i=;
    if i=0 then put 'AT ZERO';
  end;
run;
```

次のように SAS ログに書き込まれます。

アウトプット 4.23　ROUND 関数の SAS ログ出力

```
i=-1
i=-0.9
i=-0.8
i=-0.7
i=-0.6
i=-0.5
i=-0.4
i=-0.3
i=-0.2
i=-0.1
i=0
AT ZERO
i=0.1
i=0.2
i=0.3
i=0.4
i=0.5
i=0.6
i=0.7
i=0.8
i=0.9
i=1
```

コード例

比較を実行する前に値を明示的に丸めることで、比較エラーを回避できます。次の例では、1/3 の計算結果を、割り当てられた値.33333.と比較します。 1/3 は不正確な数値なので、値は.33333,と等しくなく、PUT ステートメントは実行されません。しかし、次の例のように ROUND 関数を追加すると、PUT ステートメントは MATCH を出力します。

```
data _null_;
  x=1/3;
  if round(x, .00001)=.33333 then put 'MATCH';
run;
```

アウトプット 4.24　ログ出力: ROUND 関数を使用して比較エラーを回避

```
MATCH
```

要点

- 不正確な値に対する計算の累積によって発生するエラーは、丸めによって解決できます。
- 一般に、小数値との比較を行う場合は、計算や比較を実行する前に ROUND 関数を使用することをお勧めします。

関連項目

- “ROUND 関数” (*SAS 関数と CALL ルーチン: リファレンス*)

例: 値を比較する場合の LENGTH ステートメントの使用

コード例

257 から 271 までの数値を最初の 2 バイトに正確に保存することはできません。数値を正確に保存するには 3 番目のバイトが必要です。その結果、次のコードは誤解を招く結果を生成します。

```
data ab;
  length x 2;
  x=257;
  y1=x+1;
data abc;
  set ab;
  if x=257 then put 'FOUND';
    else put 'NOT FOUND';
  y2=x+1;
run;
```

次のように SAS ログに書き込まれます。

例のコード 4.3　SAS ログ

```
337  data ab;
338    length x 2;
             -
             352
ERROR 352-185: The length of numeric variables is 3-8.

339    x=257;
340    y1=x+1;

NOTE: The SAS System stopped processing this step because of
      errors.
WARNING: The data set WORK.AB may be incomplete.  When this
         step was stopped there were 0 observations and 2
         variables.
WARNING: Data set WORK.AB was not replaced because this step
         was stopped.
NOTE: DATA statement used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds



341  data abc;
342    set ab;
343    if x=257 then put 'FOUND';
344      else put 'NOT FOUND';
345    y2=x+1;
346  run;

NOTE: There were 0 observations read from the data set WORK.AB.
NOTE: The data set WORK.ABC has 0 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time          0.01 seconds
      cpu time           0.00 seconds
```

X の値は実際には 256(値 257 は 2 バイトに切り捨てられる)のため、PUT ステートメントは実行されません。256 は 4310 として 2 バイトに格納されますが、257 も 4310 として 2 バイトに格納され、10 の 3 番目のバイトが切り捨てられることに注意してください。

ただし、プログラムデータベクトルでは X の値が完全な 8 バイト浮動小数点表現で保持されるため、Y1 の値は 258 になることに注意してください。値は、SAS データセットに保存される場合にのみ切り捨てられます。X は数値がプログラムデータベクトルに読み込まれる前に切り捨てられるため、Y2 の値は 257 になります。

注意

変数値が整数でない場合は、LENGTH ステートメントを使用しないでください。 小数は切り捨てられると精度が失われます。また、ディスク容量が限られている場合にのみ、LENGTH ステートメントを使用して値を切り捨てます。最大値については、お使いの動作環境の SAS ドキュメントの長さの表を参照してください。

要点

- LENGTH ステートメントを使用すると、変数値の保存に使用されるバイト数を制御できます。ただし、誤差や重大なデータ損失を回避するには注意深く使用する必要があります。
- コンパイル中、SAS は、その変数で最初に検出された値に含まれる文字と同じ数の記憶領域を割り当てます。

関連項目

- “LENGTH ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- 値を比較する場合の LENGTH ステートメントの使用

例: 不正確な表現を持つ値を比較

コード例

10 進数の算術では、式は 15.7 – 11.9 = 3.8 という式が成り立ちます。ただし、SAS ではリテラル値を比較する場合、3.8 を式 15.7 – 11.9 と比較すると、not equal のエラーを返します。

。

```
data a;
  x=15.7-11.9;
  if x=3.8 then put 'equal';
   else put 'not equal';
run;
proc print data=a;
run;
```

ログ出力は、3.8 および(15.7 – 11.9) は等価ではないことを示しています。これは、計算に含まれる値が浮動小数点表現では正確に表現できないためです。

アウトプット 4.25　不正確な表現を持つ値を比較するログ出力

```
not equal
```

PROC PRINT ステートメントは、x の値を実際に格納されている値ではなく 3.8 として表示します。これは、プロシジャが表示前に自動的に形式を適用し、結果を丸め

るためです。この例では、PROC PRINT は計算後の最終結果のみを丸めるため、明示的でない丸めが混乱を引き起こす可能性があることを示しています。

アウトプット 4.26　値を比較する出力

The SAS System

Obs	x
1	3.8

要点

- 正確な浮動小数点表現を持たない非整数値を比較すると、予期しない結果が生じることがあります。

関連項目

- 浮動小数点表現

例: フォーマットを使用した精度エラーの確認

コード例

次の例では、2 つの異なる出力形式が与えられた結果に適用され、SAS ログに表示されます。最初の出力形式 10.8 は、x の値が 3.8 であることを示します。ただし、18.16 出力形式を使用して値を表示すると、x が 3.8 よりわずかに小さいことがわかります。

```
data a;
x=15.7-11.9;
if x=3.8 then put 'equal';
  else put 'not equal';
put x=10.8;
put x=18.16;
run;
```

アウトプット 4.27　ログ出力: 出力形式を使用して精度エラーを確認

```
not equal
x=3.80000000
x=3.7999999999999900
```

コード例

HEX*w.d* 出力形式で幅 16 を使用して、浮動小数点表現を表示することもできます。

```
data a;
  x=15.7-11.9;
  if x=3.8 then put 'equal';
    else put 'not equal';
put x=hex16.;
run;
```

アウトプット 4.28　HEX16 出力形式を使用した計算結果の検証

```
not equal
x=400E666666666664
```

要点

- HEX*w*.出力形式は、浮動小数点表現の表示に使用できる特別な出力形式です。
- 正確な浮動小数点表現を持たない非整数値を比較すると、予期しない結果が生じることがあります。

関連項目

- “HEXw. 出力形式” (*SAS 出力形式と入力形式: リファレンス*)
- “出力形式のディクショナリ” (*SAS 出力形式と入力形式: リファレンス*)
- 浮動小数点表現

例: 数値を正確に保存するための必要なバイト数の決定

コード例

TRUNC 関数を使用して、値を正確に保存するために必要な最小バイト数を決定することもできます。次のプログラムは、IBM メインフレーム環境の Numbers という名前のネイティブ SAS データセットに保存されている数値に、必要な最小バイト長(MinLen)を見つけます。データセット Numbers には変数 Value が含まれています。値には、269 から<inlineCode>272</inlineCode>の範囲の数値が含まれます。

```
data numbers;
 input value;
 datalines;
 269
 270
 271
 272
;
data temp;
 set numbers;
 x=value;
 do L=8 to 1 by -1;
  if x NE trunc(x,L) then
   do;
    minlen=L+1;
    output;
    return;
   end;
 end;
run;
proc print data=temp noobs;
 var value minlen;
run;
```

次の出力は、この例の結果を示しています。

アウトプット 4.29　数値を正確に保存するために必要なバイト数の決定

The SAS System

value	minlen
269	3
270	3
271	3
272	2

要点

- 値 271 に必要な最小長は、値 272 に必要な最小長よりも大きくなります。
- この事実は、数値範囲内の最大の数値が、より小さい数値よりも少ないストレージバイトを必要とする可能性があることを示しています。
- 範囲内のすべての数値に精度が必要な場合は、最大の数値だけでなく、すべての数値の最小の長さを取得する必要があります。

関連項目

- “TRUNC 関数” (*SAS 関数と CALL ルーチン: リファレンス*)

例: 大きなデータに対して不正確な結果を生成

コード例

次の例は、大きなデータの不正確な結果を示しています。

```
data _null_;
  lo2hi =0;
```

```
 hi2lo =0;
 do i = -10 to 10 by 0.1;
  lo2hi = lo2hi + 10**i;
 end;
 do i = 10 to -10 by -0.1;
  hi2lo = hi2lo + 10**i;
 end;
 diff = hi2lo-lo2hi;
put lo2hi;
put hi2lo;
put diff;
run;
```

次の出力は、この例のログ出力を示しています。

例のコード 4.4　SAS ログ

```
lo2hi=48621160939
hi2lo=48621160939
diff=0.0041885376
```

要点

- 計算された統計は、オブザベーションが処理される順序に応じて若干異なる場合があります。このような変動は浮動小数点演算によって生じる数値誤差によるものであり、その結果は正確なものではなく近似値であると考えるべきです。
- オブザベーションの処理の順序は、マルチスレッドまたは並列処理の非決定的な影響によって影響を受ける可能性があります。
- 処理の順序は、ACCESS エンジンを介してクエリ結果を配信する DBMS などのデータソースによって生成されるオブザベーションの順序が一貫していない、または非決定的であることによっても影響を受ける可能性があります。

関連項目

- “DO ステートメント: 反復” (*SAS DATA ステップステートメント: リファレンス*)
- “精度と大きさ”

例: 変数値の暗号化

例: TRANSLATE 関数を使用して単純な 1 バイト対 1 バイトのスワップを作成

コード例

この例では、TRANSLATE 関数を使用して単純な 1 バイト対 1 バイトのスワップを使用する方法を示します。

```
data sample1;                    /* 1 */
 input @1 name $;
 length encrypt decrypt $ 8;

/*ENCRYPT*/
  do i = 1 to 8;                 /* 2 */
   encrypt=strip(encrypt)||translate(substr(name,i,1),
    '0123456789!@#$%^&*()-=,./?<','ABCDEFGHIJKLMNOPQRSTUVWXYZ');
  end;

/*DECRYPT*/
  do j = 1 to 8;                 /* 3 */
   decrypt=strip(decrypt)||translate(substr(encrypt,j,1),
    'ABCDEFGHIJKLMNOPQRSTUVWXYZ','0123456789!@#$%^&*()-=,./?<');
  end;

  drop i j;
datalines;
ROBERT
JOHN
GREG
;
proc print;
run;*/
```

1 DATA ステップは、長さが 8 文字以下の名前を読み取り、DO ループを使用して TRANSLATE 関数と SUBSTR 関数を一度に 1 バイトずつ処理します。

2 最初の DO ループでは、暗号化された値が作成されます。

3 2 番目の DO ループは、順序を逆にして元の値を返すことにより、復号化された値を作成します。

アウトプット 4.30　PROC PRINT 結果: 単純な 1 バイト対 1 バイトのスワップ

Obs	name	encrypt	decrypt
1	ROBERT	*%14*)	ROBERT
2	JOHN	9%7$	JOHN
3	GREG	6*46	GREG

要点

- SAS は、ENCRYPT=データセットオプションを使用して SAS データセットの暗号化を提供しますが、このオプションは通常、データセットレベルでデータを暗号化するために使用されます。
- SAS 変数レベルでデータを暗号化するには、DATA ステップ関数とロジックを組み合わせて使用し、独自の暗号化および復号化アルゴリズムを作成できます。
- ただし、独自のアルゴリズムを作成する場合は、安全で公の場から隠蔽されるだけでなく、データの暗号化と復号化の両方のメソッドを含むプログラムを作成することが重要です。

関連項目

- SUBSTR 左関数
- SUBSTR 右関数
- TRANSLATE 関数

例: TRANWRD 関数を使用した 1 バイトから 2 バイトへのスワップの使用

コード例

この例では、TRANWRD 関数で 1 バイトから 2 バイトへのスワップを使用して値を暗号化する方法を示します。次のサンプルコードでは、DATA ステップで長さが 6 文字以下の ID を読み取ります。ただし、これは 1 バイトから 2 バイトへの交換であるため、変数には文字長を 2 倍にする 12 という長さが割り当てられます。DO ループは、TRANWRD 関数を一度に 1 バイトずつ処理します。

```
data sample2;                    /* 1 */
```

```
input @1 id $12.;

/*ENCRYPT*/
 encrypt=id;
  i=21;
 do from_1 = "C","F","E","A","D","B";   /* 2 */
  to_1=put(i,2.);
  encrypt=tranwrd(encrypt,from_1,to_1);
   i+1;
 end;

/*DECRYPT*/
 decrypt=encrypt;
  j=21;
 do to_2 = "C","F","E","A","D","B";     /* 3 */
  from_2=put(j,2.);
  decrypt=tranwrd(decrypt,from_2,to_2);
   j+1;
 end;
 drop i j to_1 from_1 to_2 from_2;

datalines;
ABCDEF
FEDC
ACE
BDFA
CAFDEB
BADCF
ABC
;
proc print;
run;
```

1 DATA ステップは、長さが 6 文字以下の ID を読み取ります。ただし、これは 1 バイトから 2 バイトへの交換であるため、変数には文字長を 2 倍にする 12 という長さが割り当てられます。

2 最初の DO ループでは、暗号化された値が作成されます。

3 2 番目の DO ループは、順序を逆にして元の値を返すことにより、復号化された値を作成します。元の ID 変数の上書きを避けるために、TRANWRD 関数の前に新しい変数が ID 変数に割り当てられます。

アウトプット 4.31　PROC PRINT 結果: 単純な 1 バイトから 2 バイトへのスワップ

Obs	id	encrypt	decrypt
1	ABCDEF	242621252322	ABCDEF
2	FEDC	22232521	FEDC
3	ACE	242123	ACE
4	BDFA	26252224	BDFA
5	CAFDEB	212422252326	CAFDEB
6	BADCF	2624252122	BADCF
7	ABC	242621	ABC

要点

- SAS は、ENCRYPT=データセットオプションを使用して SAS データセットの暗号化を提供しますが、このオプションは通常、データセットレベルでデータを暗号化するために使用されます。
- SAS 変数レベルでデータを暗号化するには、DATA ステップ関数とロジックを組み合わせて使用し、独自の暗号化および復号化アルゴリズムを作成できます。
- ただし、独自のアルゴリズムを作成する場合は、安全で公の場から隠蔽されるだけでなく、データの暗号化と復号化の両方のメソッドを含むプログラムを作成することが重要です。

関連項目

- TRANWRD 関数

例: さまざまな関数を使用して数値を文字列として暗号化する

コード例

この例では、数値を暗号化し、3 回ごとに異なる文字を使用して文字値を作成する方法を示します。この方法では、PUT、SUBSTR、INDEXC、TRANSLATE、CATS、および INPUT 関数と配列処理を使用します。

```
data sample3;                                          /* 1 */
 input num;
 array from(3) $ 10 from1-from3 ('0123456789','0123456789','0123456789');   /* 2 */
 array to(3) $ 10 to1-to3 ('ABCDEFGHIJ','KLMNOPQRST','UVWXYZABCD');
 array old(5) $ old1-old5;
 array new(5) $ new1-new5;
 char_num=put(num,5.);                                    /* 3 */
 do i = 1 to 5;                                        /* 4 */
  old(i)=substr(char_num,i,1);
 end;
 j=1;
 do k = 1 to 5;                                         /* 5 */
  if indexc(old(k),from(j)) > 0 then do;
    new(k)=translate(old(k),to(j),from(j));
    j+1;
    if j=4 then j=1;
  end;
 end;
 encrypt_num=cats(of new1-new5);                            /* 6 */
 keep num encrypt_num;
 datalines;
 12345
 70707
 99
 1111
 ;
run;

data sample4;                                          /* 7 */
 set sample3;
 array to(3) $ 10 to1-to3 ('0123456789','0123456789','0123456789');
 array from(3) $ 10 from1-from3 ('ABCDEFGHIJ','KLMNOPQRST','UVWXYZABCD');
 array old(5) $ old1-old5;
 array new(5) $ new1-new5;
 do i = 1 to 5;
  old(i)=substr(encrypt_num,i,1);
 end;
 j=1;
 do k = 1 to 5;
  if indexc(old(k),from(j)) > 0 then do;
    new(k)=translate(old(k),to(j),from(j));
    j+1;
    if j=4 then j=1;
  end;
 end;
 decrypt_num=input(cats(of new1-new5),5.);
 keep num encrypt_num decrypt_num;
run;

proc print;
run;
```

1 最初の DATA ステップでは、5 桁以下の数値を読み込みます。数値変数は文字変数に変換され、5 つの個別の値に分割されます。

2 4 つの ARRAY ステートメントが使用されます。最初の配列は **from** の値を設定します。2 番目は **to** の値を設定します。3 番目には 5 つの個別の数値が保持さ

れます。4 番目には、5 つの新しい個別の暗号化値が保持されます。配列 **from** および **to** は、それぞれ 3 つの要素で作成されます。配列 **from** には 3 つの要素すべてに同じ数字の文字列が割り当てられ、配列 **to** には 3 つの要素ごとに異なる文字列が割り当てられ、3 回ごとのパターンが繰り返されるように構成します。

3 PUT 関数は、数値を文字値に変換します。

4 最初の DO ループでは、SUBSTR 関数を使用して値を 5 つの個別の値に分割し、それぞれを配列 old に割り当てます。

5 2 番目の DO ループは、INDEXC 関数を使用して各値を変換し、配列 **from** 内の元の数値を見つけます。見つかった場合は、配列 **from** を使用して値を変換し、3 回ごとに要素のリストを繰り返します。

6 暗号化された値は、CATS 関数を使用して 5 つの変換された値を連結することによって作成されます。値の暗号化に使用されるのと同じ処理が、値の復号化にも使用されます。唯一の違いは、暗号化された変数が SUBSTR 関数に渡され、最終的に復号化された変数が CATS 関数に続いて INPUT 関数に渡されることです。これは、最終値が数値になるように行われます。

7 2 番目の DATA ステップでは、最初の DATA ステップで行われた暗号化を逆に行い、値を元の値に変換します。2 つの DATA ステップを比較すると、配列 **to** および配列 **from** の値が逆転していることがわかります。これは、2 番目の DATA ステップで最初の DATA ステップで行われた暗号化が逆に行われ、値が元の値に変換されるためです。

アウトプット 4.32　PROC PRINT 結果: さまざまな関数を使用して数値を文字列として暗号化する

Obs	num	encrypt_num	decrypt_num
1	12345	BMXEP	12345
2	70707	HKBAR	70707
3	99	JT	99
4	1111	BLVB	1111

要点

- SAS は、ENCRYPT=データセットオプションを使用して SAS データセットの暗号化を提供しますが、このオプションは通常、データセットレベルでデータを暗号化するために使用されます。
- SAS 変数レベルでデータを暗号化するには、DATA ステップ関数とロジックを組み合わせて使用し、独自の暗号化および復号化アルゴリズムを作成できます。
- ただし、独自のアルゴリズムを作成する場合は、安全で公の場から隠蔽されるだけでなく、データの暗号化と復号化の両方のメソッドを含むプログラムを作成することが重要です。

関連項目

- CATS 関数
- INDEXC 関数
- INPUT 関数
- PUT 関数
- SUBSTR 左関数
- SUBSTR 右関数
- TRANSLATE 関数

5

式

SAS 式

定義

式
結果の値を生成するために実行される一連の命令を形成する一連のオペランドと演算子です。

オペランド
数値または文字を使用できる定数または変数です。

演算子
比較、算術計算、または論理演算、SAS 関数、グループ化かっこを表す記号です。

単純式
演算子が 1 つだけの式です。単純式は、次の単一の演算子のいずれかで構成されます。

- 定数
- 変数
- 関数

複合式
　複数の演算子を含む式です。SAS は複合式に遭遇すると、ルールに従って式の各部分を評価する順序を決定します。

WHERE 式
　DATA または PROC ステップで処理するオブザベーションを選択するための条件を指定するために、WHERE ステートメントまたは WHERE=データセットオプション内で使用される SAS 式のタイプです。

関連項目

- 演算子
- “WHERE 式” (465 ページ)
- WHERE ステートメント

SAS 式の使用

SAS 式は、割り当てステートメントやその他の多くの SAS プログラミングステートメントで次のことを行うために使用されます。

- 変数の変換
- 新しい変数の作成
- 変数の条件付き処理
- 新しい値の計算
- 新しい値の割り当て

SAS 式は、数値、文字値、またはブール値に解決できます。

式中の SAS 定数

SAS 定数は、固定値を示す数値または文字列です。定数は、多くの SAS ステートメントで式として使用できます。詳細については、“式中の SAS 定数” (173 ページ)を参照してください。

式中の SAS 変数

SAS 変数は式で使用できます。ただし、変数値が要求されたデータ型と一致しない場合、SAS は値を期待される型に変換しようとします。詳細については、SAS 変数 (71 ページ)を参照してください。

式中の SAS 関数

SAS 関数は、特定の計算またはシステム操作を実行するために使用するキーワードです。関数は値を返し、1 つ以上の引数を必要とする場合があり、式で使用できます。SAS 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。

式中の SAS 演算子

SAS 演算子は、比較、算術計算、または論理演算、SAS 関数、グループ化かっこを表す記号です。詳細については、“SAS 演算子” (193 ページ)を参照してください。

6

定数

式中の SAS 定数

定義

SAS 定数は、固定値を示す数値または文字列です。定数は、変数の割り当てステートメントや IF-THEN ステートメントなど、多くの SAS ステートメントで式として使用されます。SAS 式で使用される定数の例を次に示します。

- x=10;
- name="James";
- date=01/23/2018;

これらは、特定のオプションの値としても使用できます。定数はリテラルとも呼ばれます。SAS 定数の種類は次のとおりです。

- 文字定数
- 数値定数
- 日付定数、時間定数、日時定数
- ビットテスト定数

文字定数

定義

文字定数は 1 文字から 32,767 文字で構成され、引用符で囲む必要があります。引用符は、単一引用符または二重引用符のいずれかです。'Tom'と"Tom"は同等です。

文字定数と文字変数

文字定数は引用符で囲まれていますが、変数の名前は引用符で囲まれていないことに注意してください。この区別は、タイトル、脚注、ラベル、その他の説明文字列、オプション値、ファイル指定やコマンドなどの動作環境固有の文字列など、文字定数を使用できるすべての 場所に適用されます。

次のステートメントは文字定数を使用しています。

- x='abc';
- if name='Smith' then do;

次のステートメントは文字変数を使用しています。

- x=abc;
- if name=Smith then do;

2 番目の例では、定数ではなく、ABC と SMITH という名前の変数が検索されます。

注: SAS では、文字定数を比較するときに大文字と小文字を区別します。たとえば、文字定数'Smith'と'SMITH'は同等ではありません。

文字定数内のアポストロフィと引用符

- 文字定数に単一引用符が含まれている場合は、二重引用符で囲みます。

- 文字定数にアポストロフィが含まれている場合は、文字列を単一引用符で囲み、アポストロフィを 2 つの連続した引用符として表すことができます。SAS では、2 つの連続する引用符を 1 つの引用符として扱います。

  name='Tom''s'

- 文字定数に単一引用符またはアポストロフィが含まれている場合は、文字列とアポストロフィを 2 つの連続する二重引用符で囲むことができます。SAS では、2 つの連続する二重引用符を 1 つの二重引用符として扱います。

注意
引用符を正しく一致させることが重要です。 SAS は、引用符がなかったり、余計な引用符があると、誤ったステートメントとそれに続く*ステートメント*の両方を正しく解釈できません。たとえば、**name='O'Brien';**の場合、**O** は Name の文字値、**Brien** は関係のない値、**';** は別の引用文字列の始まりとなります。

16 進表記で表現された文字定数

SAS 文字定数は、16 進表記で表現することができ、各文字は 16 進文字のペアで表現されます。次の例のように、文字定数の文字列は単一引用符または二重引用符で囲まれ、直後に X が続きます。

'534153'x

次の例のように、カンマを使用して文字列を読みやすくすることができますが、カンマは 16 進値の一部ではなく、16 進値を変更しません。

'31,32,33,34'x

注: 引用符内の末尾または先頭に空白があると、エラーメッセージがログに書き込まれます。

例については、“例: 16 進表記での文字定数の定義” (184 ページ)を参照してください。

文字定数でよくあるエラーの回避

文字定数の後には必ず空白を挿入する必要があります。そうしないと、SAS では、文字定数に続く文字を文字定数の値の種類として解釈する可能性があります。

次のステートメントでは、'821't は文字列と t の間にスペースがないため、時間定数として評価されます。

```
if flight='821'then flight='230';
```

次の行が SAS ログに出力されます。

```
ERROR: Invalid date/time/datetime constant '821't.
ERROR 77-185: Invalid number conversion on '821't.
```

終了引用符の後にスペースを 入れることで、このような誤った解釈を防ぐことができます。

表 6.1　文字定数に続くと誤って解釈される文字

文字	考えられる解釈	例
b	ビットテスト定数	'00100000'b
d	日付定数	'01jan04'd
dt	日時定数	'18jan2005:9:27:05am'dt
n	名前リテラル	'My Table'n
t	時間定数	'9:25:19pm't
x	16 進表記	'534153'x

関連項目

例

- “例: 式での文字定数の使用” (180 ページ)
- “例: 文字定数と文字変数の比較”
- “例: 文字列内で引用符を使用する”
- “例: 16 進表記での文字定数の定義”

数値定数

定義

数値定数は、SAS ステートメントに表示される数値です。数値定数は、次のようなさまざまな形式で表すことができます。

- 標準表記
- 指数(E)表記
- 16 進表記

標準表記で表現された数値定数

ほとんどの数値定数は、数値データ値と同じように記述されます。次の式の数値定数は 100 です。

```
part/all*100
```

数値定数は，標準表記で次のように表すことができます。

表 6.2 数値定数の標準表記

数値定数	説明
1	符号なし整数
-5	負符号を含む
+49	正符号を含む
1.23	小数点以下桁数を含む
01	有意でないゼロを先頭に含む

$(10^{32})-1$ より大きい数値定数は、指数表記を使用して記述する必要があります。たとえば、2E4 などの数値は指数表記で記述する必要があります。

指数表記で表現された数値定数

(2E4 などの)指数表記では、E の前の数値に、E の後の数値で示される 10 の累乗を掛けます。たとえば、2E4 は $2x10^4$ または 20,000 と同じです。

16 進表記で表現された数値定数

16 進表記で表現された数値定数は、数字(通常は 0)で始まります。その後にさらに 16 進文字を続けることができ、文字 X で終わります。定数には最大 16 個の有効な 16 進文字(0 から F)を含めることができます。

関連項目

例

- “例: 標準表記での数値定数の定義”

- “例: 指数表記での数値定数の定義”
- “例: 16 進表記での数値定数の定義”

ステートメント

- FORMAT ステートメン

日付定数、時間定数、日時定数

日付定数、時間定数、日時定数は、単一引用符または二重引用符で囲まれた日付、時間、日時であり、その後に値の種類を示す D(日付)、T(時間)、または DT(日時)が続きます。D、T、DT を使用して日付定数、時間定数、日時定数を作成する方法の例については、“例: 日付定数で日付、時間、および日時の値を定義する”を参照してください。

定数	形式	例
日付	*'ddmmm<yy>yy'*D *“ddmmm<yy>yy”*D	*'1jan2018'*D *"01jan18"*D
時間	*'hh:mm<:ss.s>'*T *“hh:mm<:ss.s>”*T	*'9:25'*T *"9:25:19pm"*T
日時	*'ddmmm<yy>yy:hh:mm<:ss.s>'*DT *“ddmmm<yy>yy:hh:mm<:ss.s>”*DT	*'01may18:9:30:00'*DT *"18jan2018:9:27:05am"*DT
タイムゾーン指定子を含む日時 (ISO 8601 規格)	*'yyyy-mm-ddThh:mm:ssZ'*DT *'yyyy-mm-ddThh:mm:ss+|-hh:ss'*DT	*'2018-07-20T12:00:00Z'*DT *'2018-05-17T09:15:30-05:00'*DT

> 重要　UTC または ISO 8601 の日時定数は、Zulu タイムゾーン表示または協定世界時からの数値オフセットを持ち、システムのタイムゾーンオフセットに従って内部値を調整することによってローカルタイムに変換されます。この調整は、システムの TIMEZONE オプションが明示的に設定されているかどうかに関係なく行われます。TIMEZONE=システムオプションが指定された場合、SAS では、TIMEZONE=システムオプションで指定した値に基づいて UTC と ISO 8601 の DATETIME 定数を変換します。TIMEZONE=システムオプションが指定されない場合、SAS では、システムの UTC タイムゾーンオフセットに基づいて UTC と ISO 8601 の DATETIME 定数を変換します。

引用符の中にある末尾の空白または先頭の空白は、日付定数、時間定数、日時定数の処理には影響しません。

関連項目

“例: 日付定数で日付、時間、および日時の値を定義する” (188 ページ)

ビットテスト定数

ビットテスト定数は、値の表現で内部ビットを比較するためにビットテストで使用されるビットマスクです。文字変数と数値変数の両方でビットテストを実行できます。

操作の一般的な形式は次のとおりです。

expression comparison-operator bit-mask

ビットテスト操作の構成要素は次のとおりです。

expression
: 任意の有効な SAS 式にすることができます。文字変数と数値変数の両方をビットテストできます。

  SAS が文字値をテストするとき、マスクの左端のビットを文字列の左端のビットに合わせ、対応するビットを右へ移動しながらテストを進めます。

  SAS が数値をテストするとき、値は浮動小数点数から 32 ビット整数に切り捨てられます。マスクの右端のビットを数値の右端のビットに合わせ、対応するビットを左へ移動しながらテストを進めます。

comparison-operator
: 式をビットマスクと比較します。詳細については、演算子を参照してください。

bit-mask
: '..1.0000'b のように、直後に B が続く引用符で囲まれた 0、1、ピリオドを含む文字列です。0 は、ビットがオフかをテストします。1 は、ビットがオンになっているかをテストします。ピリオドはビットを無視します。ビットマスクの意味を損なわずに読みやすくするために、カンマや空白を挿入できます。

注意
SAS でビットマスクを使用する場合、切り捨てが発生することがあります。 式がビットマスクよりも長い場合、SAS では式をビットマスクと比較する前に切り捨てます。誤った比較が行われる可能性があります。式の長さ(ビット単位)は、ビットマスクの長さ以下である必要があります。ビットマスクが文字式よりも長い場合、SAS ではログに警告を生成し、ビットマスクが左側で切り捨てられたことを示し、処理を続行します。

注: ビットマスクは、割り当てステートメントのビットリテラルとして使用することはできません。たとえば、次のステートメントは無効です。

```
x='0101'b;    /* incorrect*/
```

ヒント　$BINARY*w*.出力形式、BINARY*w*.出力形式、$BINARY*w*.入力形式、BINARY*w.d* 入力形式、BITS*w.d* 入力形式は、ビットテストに役立ちます。これらを使用して、文字値と数値を 2 進値に、またはその逆に変換したり、入力データから指定されたビットを抽出したりできます。これらの出力形式と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。

関連項目

例

- “例: 変数の値のビットテスト”

出力形式

- $BINARYw.出力形式

例: 式と定数

例: 式での文字定数の使用

コード例

次の例は、単一引用符と二重引用符を使用して文字定数を作成する方法を示しています。

```
data example;
  char_const = 'Tom';       /* 1 */
  apostrophe = "Tom's";     /* 2 */
  singlequote = 'Tom''s';   /* 3 */
run;
proc print data=example;
run;
```

1 文字定数は、文字列値を単一引用符で囲んで作成されます。

2 文字定数には、文字列の外側に二重引用符を使用して作成されるアポストロフィが含まれています。内側の単一引用符はアポストロフィを表します。

3 文字列内で 2 つの単一引用符を使用し、値全体を単一引用符で囲むこともできます。

アウトプット 6.1　文字定数における単一引用符および二重引用符の結果

Obs	char_const	apostrophe	singlequote
1	Tom	Tom's	Tom's

要点

- 文字定数は 1 文字から 32,767 文字で構成される固定値であり、引用符で囲む必要があります。文字定数は 16 進形式で表すこともできます。
- 文字*変数*には、英字、0 から 9 の数字、およびその他の特殊文字が含まれます。
- 引用符を正しく一致させることが重要です。
- 引用符がなかったり、余計な引用符があると、SAS がステートメントを誤って読み取り、エラーを生成します。
- 文字定数にアポストロフィ(または単一引用符)が含まれている場合は、文字列全体を二重引用符で囲むことによって文字定数を作成できます。

関連項目

- “文字定数と文字変数”
- “文字定数内のアポストロフィと引用符”
- “例: 文字列内で引用符を使用する”

例: 文字定数と文字変数の比較

コード例

次の例は、文字変数を使用して文字定数を作成する方法を示しています。

```
data compare;
  var1 = 'abc';    /* 1 */
  var2 = 'def';    /* 2 */
  name1 = var1;    /* 3 */
  name2 = var2;    /* 4 */
run;

proc print data=compare;
```

```
run;
```

1 文字変数 var1 と var2 を作成するには、割り当てられた値を単一引用符または二重引用符で囲みます。変数 var1 と var2 は、値が引用符で囲まれているため、文字定数から作成されます。

2 文字変数 name1 と name2 を作成します。値を引用符で囲まないでください。変数の値は、ステップ 1 で作成した文字定数の名前です。

3 変数 name1 は、値が var1 の値である文字変数です。var1 の値は'**abc**'です。

4 変数 name2 は、値が var2 の値である文字変数です。var2 の値は'**def**'です。

変数 name1 と name2 は文字定数からは作成されません。これらは、割り当てられる値が引用符で囲まれていないため、文字変数です。それらの値は、先に作成した文字変数 var1 と var2 の名前です。文字列を変数に割り当て、値を引用符で囲まない場合、SAS は値が既存の変数の名前であると想定します。

アウトプット 6.2　文字定数と文字変数の比較結果

Obs	var1	var2	name1	name2
1	abc	def	abc	def

要点

- 文字*定数*は引用符で囲まれますが、変数名は引用符で囲まれません。
- 文字定数は 1 文字から 32,767 文字で構成されます。文字定数は 16 進形式で表すこともできます。
- 文字*変数*は、英字、0 から 9 の数字、およびその他の特殊文字を含む文字型の変数です。
- この区別は、タイトル、脚注、ラベル、その他の説明文字列など、文字定数を使用できるあらゆる場所に適用されます。
- SAS では、文字式を比較するときに大文字と小文字を区別します。

関連項目

- “文字定数と文字変数”
- “文字定数内のアポストロフィと引用符”
- “例: 式での文字定数の使用”

例: 文字列内で引用符を使用する

コード例

次の例は、文字定数内での二重引用符と単一引用符の両方の使用を示しています。

```
data titles;
  book1 = "Uncle Tom's Cabin";   /* 1 */
  book2 = 'Uncle Tom''s Cabin';  /* 2 */
  book3 = '"Ben Hur"';           /* 3 */
  book4 =  """Ben Hur""";        /* 4 */
run;
proc print data=titles;
run;
```

1 book1 では、二重引用符と単一引用符を交互に使用します。

2 book2 では、単一引用符を使用して内側の引用符文字をエスケープします。

3 book3 では、単一引用符と二重引用符を使用します。

4 book4 では、3 つの二重引用符を使用します。

アウトプット 6.3　アポストロフィを含むリテラル定数の結果

Obs	book1	book2	book3	book4
1	Uncle Tom's Cabin	Uncle Tom's Cabin	"Ben Hur"	"Ben Hur"

要点

- 2 つの方法のいずれかを使用して、文字定数内の引用符をエスケープできます。二重引用符と単一引用符を交互に使用することも、引用符を 2 つ重ねて文字をエスケープすることもできます。
- 定数にアポストロフィまたは単一引用符が含まれている場合は、定数値全体を二重引用符で囲み、値の中に単一引用符を使用します。あるいは、別の単一引用符を使用して単一引用符を"エスケープ"することもできます。

関連項目

- “文字定数と文字変数”
- “文字定数内のアポストロフィと引用符”

- “16 進表記で表現された文字定数”
- “例: 16 進表記での文字定数の定義”

例: 16 進表記での文字定数の定義

コード例

次の例は、文字定数を 16 進表記で示しています。

```
data _null_;
  value1='534153'x;       /* 1 */
  value2='53,41,53'x;     /* 2 */
  put value1 "is identical to " value2;
  run;
```

1 value1 は、定数文字列を単一引用符で囲み、その直後に x を続けることによって定義されます。

2 value2 は、定数文字列を単一引用符で囲むことによって定義され、文字列を読みやすくするためにカンマを使用し、その直後に x が続きます。

次のように SAS ログに書き込まれます。

例のコード 6.1　SAS ログ

```
SAS is identical to SAS
```

要点

- SAS 文字定数は 16 進表記で表現することができ、各文字は 16 進表記で表されます。
- 文字定数の文字列は単一引用符または二重引用符で囲まれ、その直後に X が続きます。
- カンマを使用して文字列を読みやすくすることができますが、カンマは 16 進値の一部ではなく、16 進値を変更するものではありません。

関連項目

- “16 進表記で表現された文字定数”
- “16 進表記で表現された数値定数”

例: 標準表記での数値定数の定義

コード例

次の例では、標準表記で表現された数値定数を定義します。num1-num5 に割り当てられる数値定数の表現は異なります。num1 は符号なし整数、num2 には先頭のゼロが含まれ、num3 と num4 にはプラス記号が含まれ、num5 は符号付き整数(負の 1.25)を表します。マイナス記号(-)は、負の整数または負の数を表すために使用されます。

注: 変数の作成時に先頭のゼロを指定した場合でも、数値変数の先頭のゼロはデフォルトで削除されます。

```
data _null_;
  num1=1;
  num2=01;
  num3=+1;
  num4=+01;
  put num1 "= " num2 "= " num3 "= " num4;
  num5=-1.25;
  put num5;
run;
```

次のように SAS ログに書き込まれます。

例のコード 6.2　SAS ログ

```
1 = 1 = 1 = 1
-1.25
```

要点

- 正負の数を示すプラス記号とマイナス記号を使用して、標準表記で数値定数を表現できます。
- 数値定数を指数表記および 16 進表記で表現することもできます。

関連項目

- 標準表記で表現された数値定数
- 指数表記で表現された数値定数

- 16 進表記で表現された数値定数

例: 指数表記での数値定数の定義

コード例

次の例では、指数表記で数値定数を定義します。

```
data _null;
  large1=1.2e23;
  med1=0.5e-10;
  put "large1=" large1;
  put "med1=" med1;
run;
```

次のように SAS ログに書き込まれます。

```
large1=1.2E23
med1=5E-11
```

要点

- 指数表記では、E の前の数値に、E の後の数値で示される 10 の累乗を掛けます。
- (10^{32})-1 より大きい数値定数の場合は、指数表記を使用する必要があります。
- 指数表記を示すには、値の後に E と指数を使用します。
- 数値定数を標準表記および 16 進表記で表現することもできます。

関連項目

- “標準表記で表現された数値定数”
- “指数表記で表現された数値定数”
- “16 進表記で表現された数値定数”

例: 16 進表記での数値定数の定義

コード例

次の例では、16 進表記で数値定数を定義します。

```
data _null;
  hex1=0c1x;
  hex2=9x;
  put "hex1=" hex1;
  put "hex2=" hex2;
run;
```

次のように SAS ログに書き込まれます。

例のコード 6.3　SAS ログ

```
hex1=193
hex2=9
```

要点

- 16 進値として表現される数値定数は、数字(通常は 0)で始まり、その後にさらに 16 進文字が続き、文字 x で終わります。
- 数値定数には、最大 16 個の有効な 16 進文字(0-9、A-F)を含めることができます。
- 数値定数は標準表記および指数表記で表現できます。

関連項目

- “標準表記で表現された数値定数”
- “指数表記で表現された数値定数”
- “16 進表記で表現された数値定数”

例: 日付定数で日付、時間、および日時の値を定義する

コード例

次の例には、日付、時間、および日時定数の 3 つの図が含まれています。

```
data dtconsts;
   date='1jan2018'd;                /* 1 */
   time="9:25:19pm"t;               /* 2 */
   datetime='18jan2018:9:27:05am'dt;   /* 3 */
run;

proc print data=dtconsts;           /* 4 */
  format date date.
         time time.
         datetime datetime.;
run;
```

1 変数 date は、単一引用符で囲まれ、その後に日付を表す **d** が続く日付定数として表現されます。

2 変数 time は、二重引用符で囲まれ、その後に時間を表す **t** が続く時間定数として表現されます。

3 変数 datetime は、単一引用符で囲まれ、その後に日時を表す **dt** が続く日時定数として表現されます。

4 PROC PRINT は、FORMAT ステートメントを使用して、日付、時間、日時定数をフォーマットします。FORMAT ステートメントを使用しない場合、結果は SAS の日付、時間、日時の値を表す数値として表示されます。

アウトプット 6.4　日付定数、時間定数、および日時定数の結果

Obs	date	time	datetime
1	01JAN18	21:25:19	18JAN18:09:27:05

要点

- 定数は単一引用符または二重引用符で囲みます。
- 定数の種類を示すには、値の後に **d**、**t**、または **dt** を使用します。
- 引用符の中にある末尾の空白または先頭の空白は、日付定数、時間定数、日時定数の処理には影響しません。

関連項目

- “日付定数、時間定数、日時定数”
- “日付、時間、および間隔”
- “FORMAT ステートメント” (*SAS DATA ステップステートメント: リファレンス*)

例: 変数の値のビットテスト

コード例

次の例では、ビットマスクを使用してビットテストを実行します。この例では、値 8 と 7 に対してビット 4 が 1 (TRUE)であるかどうかをテストします。

```
data _null_;
  var=8;
  if var="1...."b then state="1";
    else state="O";
  put "For value " var "bit 4 is " state;

  var=7;
  if var="1...."b then state="1";
    else state="O";
  put "For value " var "bit 4 is " state;
run;
```

次のように SAS ログに書き込まれます。

例のコード 6.4　SAS ログ

```
For value 8 bit 4 is 1
For value 7 bit 4 is O
```

要点

- ビットテスト定数は、値の表現で内部ビットを比較するためにビットテストで使用されるビットマスクです。
- 文字変数と数値変数の両方でビットテストを実行できます。
- ビットマスクを引用符で囲み、**b** を使用してビットマスクを示します。

関連項目

- “ビットテスト定数”
- $BINARYw.出力形式

例: 定数でよくあるエラーの回避

コード例

次の例は、文字列を引用符で囲んだときに発生する可能性のあるよくあるエラーを示しています。その後に変数名が続きますが、引用符で囲まれた文字列と変数名の間にはスペースがありません。THEN ステートメントの数値 821 と t の間にスペースがないため、**‘821’t** は時間定数として評価されます。

```
data aireuro;
  set sasuser.europe;
  if flight='821'then flight='230';
run;
```

次のエラーが SAS ログに出力されます。

例のコード 6.5　SAS ログ

```
ERROR: Invalid date/time/datetime constant '821't.
ERROR 77-185: Invalid number conversion on '821't.
ERROR 388-185: Expecting an arithmetic operator.
ERROR 202-322: The option or parameter is not recognized and
               will be ignored.
```

このエラーを修正するには、THEN ステートメントの終了引用符と **t** の間に空白スペースを挿入します。これにより、誤った解釈がなくなります。エラーメッセージは生成されず、FLIGHT 値 821 を持つすべてのオブザベーションは値 230 に置き換えられます。

```
if flight='821' then flight='230';
```

要点

- エラーを避けるために、終了引用符と変数名の間には必ず空白スペースを 挿入してください。
- 空白スペースがないと、SAS は文字定数とそれに続く文字を特殊な SAS 定数として誤って解釈します。

関連項目

- 文字定数でよくあるエラーの回避

7

演算子

SAS 演算子

SAS 演算子の定義

SAS 演算子は、比較、算術計算、または論理演算を実行するために使用される記号です。SAS が使用する主な種類の演算子は、次の 2 つになります。

- 接頭辞演算子
- 挿入演算子

接頭辞演算子は、演算子の前にある変数、定数、関数、またはかっこ内の式に適用されます。プラス記号(+)とマイナス記号(-)は、接頭辞演算子として使用できます。単語 NOT、およびそれに相当する記号も接頭辞演算子として使用できます。接頭辞演算子の使用方法の例を次に示します。

- 変数:

  +y

- 定数:

  -25

- 関数:

  -cos(angle1)

- かっこ内の表現:

  +(x*y)

挿入演算子は、オペランドの間に配置される演算子です。挿入演算子には、次の種類の演算子が含まれます。

- 算術:

  a=b+c

- 比較:

  Weight>150

- 論理値またはブール値:

  if x or y eq 1

- 最小値と最大値:

  where a=(b max c)

- 連結:

  Name=Firstname || Lastname

SAS には、特定の SAS ステートメントでのみ使用される他の演算子もいくつか用意されています。WHERE ステートメントは、WHERE 式と一緒に使用する場合にのみ有効な、SAS 演算子の特別なグループを使用します。これらの演算子の詳細については、WHERE ステートメントを参照してください。

算術演算子

算術演算子は、次の表に示すような計算を実行します。

表 7.1　算術演算子

記号	説明	例
**	A を 3 乗します。	a**3
*[1]	Y の値に 2 を乗算します。	2*y

記号	説明	例
/	VAR の値を 5 で除算します。	var/5
+	NUM の値に 3 を加算します。	num+3
-	SALE の値から DISCOUNT の値を減算します。	sale-discount

1 アスタリスク(*)は乗算を示すために常に必要です。 2Y と 2(Y)は有効な式ではありません。

SAS がこれらの演算子を評価する順序については、“複合式の演算順序” (203 ページ)を参照してください。

注: 算術演算子で使用される値が欠損している場合、結果は欠損値になります。欠損値のプロパゲーションを防ぐ方法については、“欠損値” (95 ページ)を参照してください。

比較演算子

比較演算子は条件を表します。比較が true の場合、結果は 1 になります。比較が false の場合、結果は 0 になります。

比較方法

比較演算子は、次の表に示すように、記号または同等のニーモニックとして表現できます。

いずれかの演算子にコロン(:)修飾子を追加すると、文字列の指定された接頭辞のみを比較できます。詳細については、“文字比較” (196 ページ)を参照してください。

表 7.2 比較演算子

記号/ニーモニック	定義	例
=または EQ	等しい	a=3
^=、¬=、~=、または NE	等しくない	a ne 3
>または GT	より大きい	num>5
<または LT	未満	num<8

記号/ニーモニック	定義	例
>=または GE[1]	以上	sales>=300
<=または LE[2]	以下	sales<=100
または IN	リストの 1 つに等しい	num in (3, 4, 5)

1 記号=>も、SAS の以前のリリースとの互換性のために受け入れられます。WHERE 句または PROC SQL ではサポートされません。
2 記号=<も、SAS の以前のリリースとの互換性のために受け入れられます。WHERE 句または PROC SQL ではサポートされません。

数値比較

SAS は、true 値と false 値に基づいて数値比較を行います。式は、式が true の場合は 1 と評価され、式が false の場合は 0 と評価されます。たとえば、式 A<B において、A の値が 4、B の値が 3 である場合、A<B の値は 0、つまり false になります。

数値比較は通常、次の場合に使用されます。

- IF-THEN/ELSE ステートメント
- 割り当てステートメント中の式

数値比較の例については、次を参照してください。

- “例: 比較演算子を使用したデータのサブセット”
- “例: 比較演算子を使用した変数の作成”

8 バイト未満の値は 8 バイトを超える値よりも精度が低いため、異なる長さの数値を比較すると、不正確な結果が得られる可能性があります。丸めは数値比較の結果にも影響します。数値精度の詳細については、“数値精度” (100 ページ)を参照してください。

欠損数値は他の数値より小さく、欠損数値には独自の並べ替え順序があります。詳細については、“欠損値” (95 ページ)を参照してください。

文字比較

文字変数の値は、左から右に文字ごとに比較されます。文字の順序は、コンピューターで使用される照合順序と、SAS セッションエンコーディングのオプションによって異なります。

たとえば、Latin 1(cp1252 West European)および UTF-8(UTF-8 Unicode)照合順序では、G は A より大きくなります。したがって、次の式は true です。

```
'Gray'>'Adams'
```

文字を比較する際には、留意すべき重要な注意事項がいくつかあります。

- 文字列内の空白とピリオドは、文字列内の他の出力可能な文字よりも小さくなります。空白はピリオドよりも小さくなります。たとえば、次の式は true になります。

  ```
  'C.Jones'<'CharlesJones'
  'C Jones' < 'CJones'
  'C Jones' < 'C.Jones'
  ```

- 長さが等しくない文字値は、比較が行われる前に、短い方の値の末尾に空白が付加されて比較されます。たとえば、次の式は true です。

  ```
  'ab'<'abc'
  ```

- 比較では末尾の空白は無視されます。たとえば、'fox 'は'fox'と同等です。
- 比較演算子の後のコロン修飾子は、コロンの後の引用符付き文字を比較演算子の反対側の値と比較します。SAS は、比較中に長すぎる値を短い値の長さに切り捨てます。詳細については、“例: 文字式の指定された接頭辞の比較” (213 ページ)を参照してください。
- 文字値は大文字と小文字が区別されます。

IN 演算子

IN 演算子は、リストに値が存在するかどうかをチェックし、検索に一致するリスト内のオブザベーションを選択します。リストに対して確認した値は、式または式の結果になります。リスト内の個々の値は、カンマまたは空白で区切ることができます。コロンを使用して、連続する整数の範囲を指定できます。

IN 演算子構文には、次の 3 つの形式があります。

形式 1: 個々の値はカンマで区切ることができます。

expression IN(*value-1*<...,*value-n*>)

例:

```
number IN(1, 2, 3);
state IN('NY','NJ','SC');
```

形式 2: 個々の値は空白で区切ることができます。

expression IN(*value-1*<... *value-n*>)

例:

```
number IN(1  2  3);
state IN('NY' 'NJ' 'SC');
```

形式 3: 値の間にコロンを置くと、連続値の範囲を指定できます。

expression IN(*value-1*<...:*value-n*>)

- 例:

```
if score in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 99);
```

これは、次と同じです。

```
if score in (1:10, 99);
```

コンポーネント:

expression
IN()の左側の式で、変数名、リテラル、関数呼び出し、またはこれらと演算子の組み合わせなど、有効な SAS 式を指定できます。

value
IN の右側の括弧内の値または値のリストは、定数(リテラル)である必要があります。

IN 演算子を使用する例については、IN 演算子を使用した数値の配列の検索を参照してください。

IN 演算子を使用する理由

IN 演算子を使用すると、変数の値が文字値または数値のリストに含まれるかどうかを判断できます。IN 演算子とともに配列を使用して変数値を検索したり、範囲または文字列を使用して検索したりできます。

次の表は、数値変数と文字変数で IN 演算子を使用する方法を示しています。

表 7.3 IN 演算子の使用例

	使用例	例
数値変数	検索する連続した整数の範囲を指定します。同じ IN リスト内で複数の範囲を使用することもできます。	■ y=x in(1, 2, 3, 4, 5, 6, 7, 8, 9, 10); ■ y=x in(1 2 3 4 5 6 7 8 9 10); ■ y=x in(1:10);[1]
	比較する値のリストでは範囲と定数の両方を使用します。	if x in (0,9,1:5);
	数値の配列を検索します。	“例: IN 演算子を使用して数値の配列を検索する”
文字変数	変数の値が文字値のリストに含まれるかどうかを判断します。[2]	■ if state in ('NY','NJ','PA') then region+1; ■ if state in ('NY' 'NJ' 'PA') then region+1; ■ if state='NY' or state='NJ' or state='PA' then region+1; [1]
	文字値の配列を検索します。	“例: IN 演算子を使用して文字値の配列を検索する”

1 これらのステートメントは同じ結果を生成します。
2 IN 演算子は、異なる長さの文字変数値を受け入れます。

論理演算子

論理演算子はブール演算子とも呼ばれ、通常、一連の式を複合式にリンクするために式で使用されます。

論理演算子の使用方法

論理演算子を次の表に示します。

論理演算子のない数値式は、論理数値式として機能します。

表 7.4　論理演算子

記号/ニーモニック	説明	例
&または AND	AND でリンクされた両方の量が 1(true)の場合、AND 演算の結果は 1 になります。それ以外の場合、結果は 0 になります。	(a>b & c>d)
|または OR	OR でリンクされた量のいずれかが 1(true)の場合、OR 演算の結果は 1(true)になります。それ以外の場合、OR 演算により 0 が生成されます。	(a>b or c>d)
!または OR		
¦または OR		
¬または NOT		not(a>b)
◦または NOT		
~または NOT		

SAS がこれらの演算子を評価する順序については、“複合式の演算順序”を参照してください。

AND 演算子

AND でリンクされた共通変数との 2 つの比較は、暗黙の AND で要約できます。次の 2 つのサブセット化した IF ステートメントは、同じ結果を生成します。

- if 16<=age and age<=65;
- if 16<=age<=65;

OR 演算子

OR 演算子を使用した比較は、オペランドの 1 つだけが **true** の場合に **true** として解決されます。ゼロ以外で欠損していない定数は、常に true として評価されます。したがって、次のリストでは、最初のサブセット化した IF ステートメントは常に true ですが、2 番目は必ずしも true ではありません。

- if x=1 or 2;
- if x=1 or x=2;

NOT 演算子

NOT 演算子は、接頭辞演算子であり、論理演算子です。NOT 演算子は、ステートメント値の真偽を反転します。偽のステートメント(0)を否定した結果は true(1)になります。真のステートメント(1)を否定した結果は false(0)になります。

NOT 演算子を使用する比較は、別の方法で記述しても同じ結果が得られます。次の 2 つの式は同等です。

- not(a=b & c>d)は、a ne b | c le d と同じです。
- not(name='SMITH')は、name ne 'SMITH'と同じです。

詳細については、“例: NOT 演算子を使用して比較の論理を逆にする” (216 ページ)を参照してください。

論理数値式

計算用語では、true の値は 1、false の値は 0 です。SAS では、0 または欠損以外の数値は true であり、0 または欠損の値は false です。したがって、数値変数または数値式は条件内で単独で使用できます。その値が 0 以外の数値であるか、欠損している場合、条件は true になります。値が 0 または欠損している場合、条件は false になります。

たとえば、特定のオブザベーションに **Cost** の値が存在するかどうかに応じて、変数 **Remarks** に値を割り当てるとします。IF-THEN ステートメントは次のように記述できます。

```
if cost then remarks='Ready to budget';
```

このステートメントは次と同等です。

```
if cost ne . and cost ne 0
  then remarks='Ready to budget';
```

次のように、数値式を数値定数にすることができます。

```
if 5 then do;
```

関数によって返される数値も有効な数値式です。

```
if index(address,'Avenue') then do;
```

MIN 演算子および MAX 演算子

MIN 演算子と MAX 演算子の使用方法の概要表

MIN 演算子と MAX 演算子は、2 つの量の最小値または最大値を見つけるために使用されます。MIN 演算子と MAX 演算子は、WHERE 式およびサブセット化した IF ステートメントでよく使用されます。最小値または最大値を知りたい 2 つの量で演算子を囲みます。

表 7.5 MIN 演算子と MAX 演算子の使用方法

記号/ニーモニック	説明	例
><または MIN[1]	2 つの値のうち小さい方の値を返します。	■ where x = (b min c); ■ if x = (y><z);
<>または MAX[2]	2 つの値のうち大きい方の値を返します。	■ where a=(b max c) ■ if x = (a<>b)

1 WHERE 式では、記号表現><はサポートされていません。MIN ニーモニックは、LOG 内で><に変換されます。
2 WHERE 式では、記号表現<>は等しくないと解釈されます。

欠損値が比較の一部に含まれる場合、欠損値の並べ替え順序が使用されます。詳細については、“欠損値の順序” (96 ページ)を参照してください。

MIN 演算子と MAX 演算子および MIN 関数と MAX 関数の比較

MIN および MAX オペレータは、 MIN および MAX 関数と混同しないでください。

- MIN と MAX のオペレーターは、正確に 2 つの式の値を比較します。
- MIN 関数と MAX 関数は、1 つの呼び出しで任意の数の式を評価できます。
- どちらの演算子も、変数、リテラル定数、および有効な SAS 式を入力として受け入れます。

連結演算子

連結演算子は文字値を結合します。それは二重縦棒||で示されます。連結演算の結果は、通常は<inlineCode>level='grade '||'A'</inlineCode>のように、割り当てステートメントを使用して変数に格納されます。結果の変数の長さは、連結操作の各変数または定数の長さの合計です。LENGTH または ATTRIB ステートメントを使用して、新しい変数に異なる長さを指定できます。

連結演算子は、先頭または末尾の空白を削除しません。変数の末尾に空白が埋め込まれている場合は、変数の長さを確認し、TRIM 関数を使用して値を連結する前に値から末尾の空白を削除します。

連結演算子の概要表

連結演算子は、文字値を結合する場合に便利です。次の表は、連結演算子の使用例を示しています。

表 7.6　連結演算子の使用例

使用例	例
変数の値を文字定数と連結します。	newname='Mr. or Ms. '||oldname; において、ldName の値が 'Jones'の場合、NewName の値は'Mr. or Ms. Jones'になります。
PUT 関数を使用して数値を文字値に変換します。	month='sep '; year=99; date=trim(month)|| left(put(year,8.)); において、ATE の値は文字値'sep99'です。
連結操作で TRIM 関数を使用して末尾の空白を削除します。	連結操作で TRIM 関数を使用して末尾の空白を削除する (217 ページ)

連結演算子と連結関数の比較

連結関数は、TRIM 関数や PUT 関数を使用せずに連結操作を実行します。連結関数には次のものが含まれます。

- CAT 関数: 結演算子(| |)と同じ
- CATQ 関数: 個々の項目に区切り文字と引用符を追加します
- CATS 関数: 頭と末尾の空白を削除します
- CATT 関数: 尾の空白のみを削除します
- CATX 関数: 先頭と末尾の空白を削除し、区切り文字を挿入します。

複合式の演算順序

操作の順序は次の条件によって決定します。

- 複合式では、SAS は優先度 1 の演算子を含む式の部分を最初に評価し、次に各グループを順番に評価します。
- 同じ優先度を持つ連続した操作は、優先度 1 の場合は右から左に、優先度 2 と 3 の場合は左から右に実行されます。
- かっこ内の式は、かっこの外側の式よりも先に評価されます。
- SAS は、後続の式が評価される順序を保証しません。詳細については、"SAS における短絡評価"を参照してください。

表 7.7 複合式の演算順序

優先順位	評価の順序	記号/ニーモニック	アクション	例
グループ 1	右から左	**	指数	■ y=a**2; ■ x=2**3**4 は x=(2**(3**4))として評価されます。
		+	正の接頭辞 [1]	y=+(a*b);
		-	負の接頭辞 [2]	z=-(a+b);
		¬ ~ または NOT	論理否定 [3]	if not z then put x;
		><または MIN	最小	■ x=(a><b); ■ -3><-3 は-(3><-3)として評価されます。これら

優先順位	評価の順序	記号/ニーモニック	アクション	例
				は-(-3)、つまり+3 に等しくなります。
		<>または MAX	最大	x=(a<>b);
グループ 2	左から右	*	乗算	c=a*b;
		/	除算	f=g/h;
グループ 3	左から右	+	加算	c=a+b;
		-	減算	f=g-h;
グループ 4	左から右	|| ¦¦ !!	文字値の連結 [4]	name= 'J'||'SMITH';
グループ 5[5]	左から右 [6]	<または LT	未満	if x<y then c=5;
		<=または LE	以下	if x le y then a=0;
		=または EQ	等しい	if y eq (x+a) then output;
		¬=または NE	等しくない	if x ne z then output;
		>=または GE	以上	if y>=a then output;
		>または GT	より大きい	if z>a then output;
		IN	リストの 1 つに等しい	if state in ('NY','NJ','PA') then region='NE'; y = x in (1:10);
グループ 6	左から右	&または AND	論理積	if a=b & c=d then x=1;
グループ 7	左から右	| ¦ ! または OR	論理和 [7]	if y=2 or x=3

優先順位	評価の順序	記号/ニーモニック	アクション	例
				then a=d;

1 プラス(+)記号は、接頭辞または算術演算子のいずれかになります。プラス記号が接頭辞演算となるのは、それが式の先頭にある場合、またはその直前に開きかっこまたは別の演算子がある場合のみです。

2 マイナス(-)記号は、接頭辞または算術演算子のいずれかになります。マイナス記号が接頭辞演算子となるのは、式の先頭に現れる場合、またはその直前に開きかっこや別の演算子が置かれる場合に限られます。

3 キーボードで使用できる文字に応じて、記号は、否定記号(¬)、チルダ(~)、またはキャレット(^)になります。SAS システムオプションの CHARCODE を使用すると、使用できない特殊文字を他のさまざまな文字に置き換えることができます。

4 キーボードで使用できる文字に応じて、連結演算子として使用する記号は、二重縦棒(||)、縦破線(¦¦)、または感嘆符(!!)になります。

5 グループ 5 演算子は比較演算子です。比較演算の結果は、比較が true の場合は 1、false の場合は 0 になります。欠損値は、どの比較演算においても最も低くなります。以前のバージョンの SAS との互換性を保つために、記号=< (以下)も使用できます。

6 この規則の例外は、2 つの比較演算子が数量を囲む場合に発生します。たとえば、式 x<y<z は、(x<y)および(y<z)として評価されます。

7 キーボードで使用できる文字に応じて、論理和に使用する記号は、単一の縦棒(|)、縦破線(¦)、または感嘆符(!)になります。同等のニーモニック OR を使用することもできます。

注: 算術演算子で使用される値が欠損している場合、結果は欠損値になります。"欠損値の順序" (96 ページ)を参照してください。欠損値のプロパゲーションを防ぐ方法の詳細については、こちらを参照してくだ

SAS における短絡評価

最小評価(短絡評価)は、ブール演算子を評価するために一部のプログラミング言語で使用される方法です。短絡評価では、最初の引数が式全体の値を決定しない場合にのみ、ブール式の 2 番目の引数が実行されます。たとえば、式(X AND Y)では、最初の引数(X)が FALSE と評価される場合、式全体(X AND Y)が FALSE と評価される必要があります。したがって、2 番目の引数(Y)を計算する必要はありません。

SAS は短絡評価を保証しません。ブール演算子を使用して式を結合する場合、短絡すること、または 2 番目の式の評価を回避することが目的の場合、望ましくない結果が生じる可能性があります。SAS が式を評価する順序を保証するには、ネストされた IF ステートメントを使用して式を書き換えます。次の例は、SAS がどのように短絡評価を使用する場合および使用しない場合があるかを示しています。最後の例では、ネストされた IF ステートメントを使用して評価の順序を保証する方法を示します。

次の最初の例では、SAS は条件の最初の引数 a>0 を評価するときに短絡評価を使用します。式は FALSE と評価され、その結果、SAS は無効なゼロ除算演算を含む 2 番目の式 a=1/a を評価しません。SAS はこの 2 番目の式を評価しないため、プログラムはエラーを返しません。

```
data test;
  a=0;
  if (a>0 AND a=1/a) then put 'hello';
  else put 'goodbye';
run;
```

```
goodbye
NOTE: The data set WORK.TEST has 1 observations and 1 variables.
```

次の例では、短絡評価は使用されません。条件の最初の引数 a が FALSE と評価され、2 番目の引数 a=1/a が評価され、ゼロ除算エラーが返されます。

```
data test;
  a=0;
  if (a AND a=1/a) then put 'hello';
  else put 'goodbye';
run;
```

```
NOTE: Division by zero detected at line 12 column 19.
goodbye
a=0 _ERROR_=1 _N_=1
NOTE: Mathematical operations could not be performed at the following places.
The results of the operations have been set to missing values.
```

SAS がこのような式を評価する順序を保証するには、ネストされた IF ステートメントを使用して式を書き換えます。

```
data test;
  a=0;
  IF a>0 THEN
    DO;
      IF a=1/a THEN put 'hello';
        ELSE put 'goodbye';
    END;
    ELSE
      put 'goodbye again';
run;
```

```
goodbye again
NOTE: The data set WORK.TEST has 1 observations and 1 variables.
```

演算子の使用方法のまとめ

演算子の使用方法の概要の表

表 7.8　SAS 演算子の使用方法の概要

記号	説明	例
算術演算子		

記号	説明	例
**	A を 3 乗します。	a**3
*	Y の値に 2 を乗算します。	2*y
/	VAR の値を 5 で除算します。	var/5
+	NUM の値に 3 を加算します。	num+3
-	SALE の値から DISCOUNT の値を減算します。	sale-discount
比較演算子		
=または EQ	等しい	■ a=3 ■ 'Jones' EQ 'Jones'
^=、¬=、~=、または NE	等しくない	■ a ne 3 ■ 'Jones' NE 'JONES'
>または GT	より大きい	■ num>5 ■ 'CharlesJones'>'C.Jones'
<または LT	未満	num<8
>=または GE	以上	sales>=300
<=または LE	以下	sales<=100
IN	リストの 1 つに等しい	■ num in (3, 4, 5) ■ state in ('NY','NJ','PA')
論理演算子		
&または AND	AND でリンクされた両方の量が 1(true)の場合、AND 演算の結果は 1 になります。それ以外の場合、結果は 0 になります。	(a>b & c>d)
|または OR	OR でリンクされた量のいずれかが 1(true)の場合、OR 演算の結果は 1(true)になります。それ以外の場合、OR 演算により 0 が生成されます。	(a>b or c>d)
!または OR		
¦または OR		

記号	説明	例
¬または NOT		not(a>b)
◦または NOT		
~または NOT		
MIN 演算子および MAX 演算子		
><または MIN	2 つの値のうち小さい方の値を返します。	■ where x = (b min c); ■ if x = (y><z);
<>または MAX	2 つの値のうち大きい方の値を返します。	■ where a=(b max c) ■ if x = (a<>b)
連結演算子		
||	文字値を結合します。	Address= StreetNum || CityStateZip

例: 演算子

例: 比較演算子を使用したデータのサブセット

コード例

この例では、IF-THEN/ELSE ステートメントにおいて、データをサブセット化するのに比較演算子をどのように使用できるかを示します。

```
data greencars;
  set sashelp.cars;
  MPG_AVG=(MPG_City + MPG_Highway)/2;
  if MPG_AVG>30 then Green_Rating=1;              /* 1 */
  if MPG_AVG<30 and MPG_AVG>=25 then Green_Rating=2;   /* 2 */
  else if MPG_AVG<=25 then delete;                /* 3 */
proc print data=greencars;
  var Make Model MPG_City MPG_Highway MPG_AVG Green_Rating;
run;
```

1 MPG_AVG 値が 30 以上のオブザベーションには、Green_Rating に 1 が与えられます。

2 MPG_AVG 値が 30 未満および 25 以上のオブザベーションには、Green_Rating に 2 が与えられます。

3 MPG_AVG 値が 25 以下の値は削除されます。

Obs	Make	Model	MPG_City	MPG_Highway	MPG_AVG	Green_Rating
1	Acura	RSX Type S 2dr	24	31	27.5	2
2	Acura	TSX 4dr	22	29	25.5	2
3	Audi	A4 1.8T 4dr	22	31	26.5	2
4	Audi	A41.8T convertible 2dr	23	30	26.5	2
5	Audi	TT 3.2 coupe 2dr (convertible)	21	29	25.0	2
6	BMW	330i 4dr	20	30	25.0	2
7	BMW	330Ci 2dr	20	30	25.0	2
8	BMW	530i 4dr	20	30	25.0	2
9	BMW	Z4 convertible 3.0i 2dr	21	29	25.0	2
10	Buick	Century Custom 4dr	20	30	25.0	2
11	Buick	Regal LS 4dr	20	30	25.0	2
12	Chevrolet	Aveo 4dr	28	34	31.0	1
13	Chevrolet	Aveo LS 4dr hatch	28	34	31.0	1
14	Chevrolet	Cavalier 2dr	26	37	31.5	1
15	Chevrolet	Cavalier 4dr	26	37	31.5	1

要点

- 比較演算子は条件を表します。
- 数値変数または数値式は、条件内で単独で使用できます。
- 新しい変数を作成するときに、割り当てステートメントで算術演算子を使用できます。
- 数値比較では、結果が true の場合は 1 が返され、結果が false の場合は 0 が返されます。

関連項目

- “演算子の使用方法の概要の表” (206 ページ)
- 割り当てステートメントの式で、比較を使用できます。詳細については、“例: 比較演算子を使用した変数の作成” (210 ページ)を参照してください。
- 欠損している数値には独自の並べ替え順序があります。詳細については、“欠損値” (95 ページ)を参照してください。

例: 比較演算子を使用した変数の作成

コード例

次の例は、での比較の使用方法を示します。 割り当てステートメントの比較を使用して変数(この場合の変数は c)を作成する方法を示しています。

```
data test;       /* 1 */
  x=6;
  y=8;
  c=5*(x<y)+12*(x>=y);   /* 2 */
  put c;              /* 3 */
run;
```

1 変数 x および y に、x が 6 および y が 8 になるように値を割り当てます。

2 割り当てステートメントで、x および y の値を比較します。x の値(6)は y の値(8)より小さいので、式(x<y)は 1 (true)と評価されます。同様に、6 は 8 以上ではないため、式(x>=y)は 0(false)と評価されます。そのため、c=(5*1)+(12*0)は 5 と評価されます。

3 ログに c の値を出力します。

例のコード 7.1　ログ

```
314 data test;
315   x=6;
316   y=8;
317   c=5*(x<y)+12*(x>=y);
318   put c;
319 run;
5
```

要点

- 比較演算子は条件を表します。
- 比較では、結果が true の場合は 1 が返され、結果が false の場合は 0 が返されます。
- SAS は、演算を実行する前にかっこ内の数量を評価します。

関連項目

- “演算子の使用方法の概要の表” (206 ページ)

例: IN 演算子を使用して数値の配列を検索する

コード例

この例では、IN 演算子を使用して数値の配列を検索する方法を示します。次のコードは配列 **a** を作成し、定数 x を定義し、IN 演算子を使用して配列内の x の値を検索します。

```
data test;
  array a{10} (2*1:5);   /* 1 */
  x=99;                  /* 2 */
  y= x in a;             /* 3 */
  put y=;
run;
proc print data=test;run;
```

1 次の 10 個の値(1、2、3、4、5、1、2、3、4、5)を含む **a** という名前の配列を作成します。

2 変数 x を作成し、値 99 を割り当てます。

3 ステートメントの x in a の部分は、配列 **a** で値 99 を検索します。99 は配列に存在しないため、変数 y に 0 または false と評価されます。

出力は次のとおりです。

Obs	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	x	y
1	1	2	3	4	5	1	2	3	4	5	99	0

前述のコードは、配列を作成して値を割り当てる方法を示しています。次のコードは、a{5}=98 を追加して値を割り当てる方法を示しています。

```
data test;
  array a{10} (2*1:5);
  x=99;
  a{5} = 99;          /* 1 */
  y = x in a;
  put y=;
run;
proc print data=test;
run;
```

1 割り当てステートメントは、値 99 を配列の 5 番目の要素に割り当て、ARRAY ステートメントで割り当てられた元の値 5 を上書きします。

出力は次のとおりです。

Obs	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	x	y
1	1	2	3	4	99	1	2	3	4	5	99	1

```
376 data test;
377    array a{10} (2*1:5);
378  x=99;
379  a{5} = 99;
380  y = x in a;
381  put y=;
382 run;
y=1
```

要点

- 配列の作成時にかっこ内に値を指定することで、配列内の変数に値を割り当てることができます。
- PROC SQL は IN 演算子をサポートしません。

関連項目

- “IN 演算子” (197 ページ)

例: IN 演算子を使用して文字値の配列を検索する

コード例

この例では、配列 **a**、定義された定数 x、および IN 演算子を使用して、配列内の x を検索しています。

```
data _null_;
  array a{5} $ (5*'');
  x='b1';
  y = x in a;
  put y=;
  a{5} = 'b1';
```

```
    y = x in a;
    put y=;
run;
```

例のコード 7.2　IN 演算子を使用して文字値の配列を検索した結果(部分出力)

```
190  data _null_;
191     array a{5} $ (5*'');
192     x='b1';
193     y = x in a;
194     put y=;
195     a{5} = 'b1';
196     y = x in a;
197     put y=;
198  run;
y=0
y=1
```

要点

- IN 演算子を使用して数値の配列を検索することもできます。

関連項目

- “IN 演算子を使用する理由” (198 ページ)

例: 文字式の指定された接頭辞の比較

コード例

この例では、比較演算子の後にコロン(:)を使用して、文字式に指定された接頭辞のみを比較する方法を示します。

```
data restaurantratings;
length Action $10;
input Name $1-18 Eval_1 20-21 Eval_2 Eval_3 Status $;
   if Status=:'F' then Action='Contact';
   if Status=:'P' then Action='Print Card';
datalines;
Lilac and Lavender 97 95 99 Passed
The Salty Pearl    85 70 65 F
Taste the Range    90 92 90 Pass
The Underground    85 90 90 P
```

```
When Pigs Fly     70 70 67 Fail
Basil             85 90 90 Passed
;

proc print data=restaurantratings;
run;
```

出力は次のとおりです。

Obs	Action	Name	Eval_1	Eval_2	Eval_3	Status
1	Print Card	Lilac and Lavender	97	95	99	Passed
2	Contact	The Salty Pearl	85	70	65	F
3	Print Card	Taste the Range	90	92	90	Pass
4	Print Card	The Underground	85	90	90	P
5	Contact	When Pigs Fly	70	70	67	Fail
6	Print Card	Basil	85	90	90	Passed

要点

- この例では、等号の後のコロン修飾子により、SAS に最初の文字のみを選択するように指示します。ただし、コロン修飾子の後に引用符で囲まれた文字と同じ数の文字を比較する機能があります。
- SAS は、比較中に長すぎる値を短い値の長さに切り捨てます。
- IN:比較または EQ:比較で、長さ 0 の文字値を他の文字値と比較するときに、2 つの文字値は等しいとは見なされません。結果は常に 0、または false と評価されます。

関連項目

- *SAS 関数と CALL ルーチン: リファレンス*
- “文字比較” (196 ページ)

例: 論理演算子を使用して値を比較する

コード例

この例では、論理演算子を使用して変数を比較する方法を示します。

```
data inventory;
length Restock $ 3;
input ItemNum 1-4 Quantity 5-7 PriorityLev 8-10;
 if Quantity<=10 and PriorityLev>=5
```

```
 or Quantity<=10 then Restock=1;
 else Restock=0;
datalines;
6530 10  8
8759  3  1
4573 22 10
9237  2 10
4329 12  4
9831  9  7
9830 15  4
9458 25  1
5673  4 10
7562  7   3
3291  3 10
;

proc print data=inventory;
run;
```

Obs	Restock	ItemNum	Quantity	PriorityLev
1	1	6530	10	8
2	1	8759	3	1
3	0	4573	22	10
4	1	9237	2	10
5	0	4329	12	4
6	1	9831	9	7
7	0	9830	15	4
8	0	9458	25	1
9	1	5673	4	10
10	1	7562	7	3
11	1	3291	3	10

要点

- AND でリンクされた両方の量が 1(true)の場合、AND 演算の結果は 1 になります。それ以外の場合、結果は 0 になります。
- OR でリンクされた量のいずれかが 1(true)の場合、OR 演算の結果は 1(true)になります。それ以外の場合、OR 演算により 0 が生成されます。

関連項目

- “論理演算子” (199 ページ)。

例: NOT 演算子を使用して比較の論理を逆にする

コード例

この例では、NOT 演算子を IN 演算子とともに使用して、比較の論理を逆にする方法を示します。

```
data productreviews;
  length Category $ 10;
  input ProdID Satisfaction Quality Safety Usability;
  Avg= round((Satisfaction+Quality+Safety+Usability)/4,0.1);
  if (avg>=3) then Category='Pass';                    /* 1 */
  else if (avg) NOT IN (3,4,5) then Category='Fail';
datalines;
8954 5 3 2 5
9183 5 5 5 5
6839 1 1 1 1
3493 2 1 3 2
2908 3 2 3 3
5419 5 4 5 5
3759 3 4 3 3
5301 4 3 4 4
;
run;
proc print data=productreviews;
   var ProdID avg Category;
run;
```

1 変数 category は、satisfaction、quality、safety、および usability の値の平均に依存します。

Obs	ProdID	Avg	Category
1	8954	3.8	Pass
2	9183	5.0	Pass
3	6839	1.0	Fail
4	3493	2.0	Fail
5	2908	2.8	Fail
6	5419	4.8	Pass
7	3759	3.3	Pass
8	5301	3.8	Pass

要点

- NOT 演算子を他の演算子とともに使用すると、比較の論理を逆にすることができます。

関連項目

- “NOT 演算子” (200 ページ)

例: 連結操作で TRIM 関数を使用して末尾の空白を削除する

コード例

この例では、TRIM 関数を連結演算子とともに使用して、変数 color および name の連結後に表示される末尾の空白を削除します。

```
data namegame;
  length color name $8 game $12;
  color='black';        /* 1 */
  name='jack';
  game=trim(color)||name;/* 2 */
  put game=;
run;
```

1 変数 color の長さは 8 なので、値 **black** の末尾には 3 つの空白があります。

2 game の値は、**black jack** になります。TRIM 関数は空白を削除します。

要点

- 連結演算子は、先頭または末尾の空白を削除しません。
- CAT 関数を使用すると、TRIM 関数や PUT 関数を使用せずに連結操作を実行できます。

関連項目

- *SAS 関数と CALL ルーチン: リファレンス*

8

日時

日付、時間、および間隔

SAS は、さまざまな経過時間をカウントするための日付、時間、日時間隔を提供します。詳細については、次のリソースを参照してください。

- “日付定数、時間定数、日時定数”
- “例: 日付定数で日付、時間、および日時の値を定義する”
- “SAS 日付値、時間値、日時値について” (*SAS 出力形式と入力形式: リファレンス*)

9

コンポーネントオブジェクト

DATA ステップコンポーネントオブジェクト

DATA ステップコンポーネントオブジェクトは、DATA ステップのステートメント、属性、メソッド、演算子を使用して作成および操作できる Base SAS の要素です。

SAS では、DATA ステップで使用する 5 つの事前定義コンポーネントオブジェクト(ハッシュ、ハッシュ反復子、ロガー、アペンダー、Java オブジェクト)を提供します。

詳細については、*SAS コンポーネントオブジェクト: リファレンス*を参照してください。

10

SAS コメント

SAS コメント

SAS コードにコメントを追加すると、コードの説明と文書化に役立ちます。コードが単純でも複雑でも、コメントはユーザーにコードの理解を提供します。

SAS コードにコメントを挿入する一般的な方法を次に示します。

Block comment
/* *Write your comment here* */

/* ... */にテキストを挿入して、SAS コードにコメントを追加します。

SAS Data and AI Studio(2026.06 より前は SAS Studio という名前)では、キーボードショートカットの Ctrl + *を使用できます。

Asterisk comment
* *Write your comment here*;

アスタリスクの後にコメントを挿入し、コメントをセミコロンで終了します。

複数のアスタリスクを使用することもできます: ******Comment in SAS code***** ;

Multiple comments
複数のコメントを追加する場合は、/*...*/が有効です。この形式を使用して SAS コードをコメントアウトすることもできます。コメントアウトしたい行を選択し、 Ctrl + *を押します。

複数行コメントの例を次に示します。

/* *SAS comment* */
/* *SAS comment* */
/* *SAS comment* */

アスタリスクを使用した例を次に示します。

**SAS comment*;
**SAS comment*;

*SAS comment;

次は、/* */の単一セットを使用している例です:

```
/* SAS comment
SAS comment
SAS comment */
```

% Macro comment *

コメントの追加例を次に示します。

```
%* SAS comment;
```

ステートメントをコメントアウトすることもできます。

```
%*%macro_name() ;
```

COMMENT keyword

キーワード COMMENT を使用して、コメントを追加できます。次は 2 つのコメントを追加する例です。

```
data test;
    comment first SAS comment ;
  dt=today();
    comment second SAS comment ;
  format dt date9.;
run;
```

重要

- ブロックコメントを別のブロックコメント内に指定しないでください。
- アスタリスクコメントを別のアスタリスクコメント内に指定しないでください。
- ブロックコメント内にアスタリスクコメントを配置できます。
- アスタリスクコメントの内部にブロックコメントを配置できます。
- ブロックコメントは、複数の SAS ステートメントをコメントアウトできます。
- ステートメントの先頭にアスタリスクを置くと、単一の SAS ステートメントをコメントアウトできます。
- 各ステートメントの先頭にアスタリスクを置くことで、複数の SAS ステートメントをコメントアウトできます。

3 部

データへのアクセス

11

SAS ライブラリ

SAS ライブラリの定義

*SAS ライブラリ*は、SAS データセットを含む 1 つ以上の SAS ファイルのコレクションであり、1 つのユニットとして参照および保存されます。ディレクトリベースの環境では、SAS ライブラリは、同じフォルダーまたはディレクトリに保存されている SAS ファイルのグループです。

各 SAS ライブラリは、ライブラリ参照名、エンジン、物理的な場所、エンジンまたは環境に固有のオプションに関連付けられています。

*ライブラリ参照名*は、データの物理的な場所を参照するために作成するショートカット名またはニックネームです。たとえば、2 レベル名 mylib.myfile では、ライブラリ参照名は mylib で、ライブラリメンバーは myfile です。

*ライブラリメンバー*は、次の表にある SAS ファイルタイプのいずれかになります。他のファイルもディレクトリに保存できますが、SAS ライブラリの一部として認識されるのは SAS ファイルのみです。各 SAS メンバータイプには、固有のファイル拡張子があります。したがって、ライブラリには、名前は同じでもメンバータイプが異なるファイルを含めることができます。

表 11.1 最も一般的な SAS ライブラリメンバー

ファイル	メンバータイプ [1]	Linux ファイル拡張子	例
データセットまたはテーブル	DATA	.sas7bdat	■ “Example: Create and Print a V9 Engine Data Set” (*SAS V9 LIBNAME Engine: Reference*) ■ “例: DBMS データに SAS ライブラリとしてアクセスする” ■ “例: SPD Engine を使用した Hadoop での SAS データの読み取りおよび書き込み” ■ “XMLMap を使用して XML ドキュメントを 1 つの SAS データセットとしてインポートする” (*SAS XMLV2 および XML LIBNAME エンジン: ユーザーガイド*)

ファイル	メンバータイプ [1]	Linux ファイル拡張子	例
ビュー	VIEW	.sas7bvew	■ “Examples: SAS Views” (*SAS V9 LIBNAME Engine: Reference*) ■ “例: DBMS データにアクセスして SAS ビューを作成する”
カタログ	CATALOG	.sas7bcat	■ “Examples: Catalogs” (*SAS V9 LIBNAME Engine: Reference*)
ストアドコンパイル DATA ステッププログラム	PROGRAM	.sas7bpgm	■ “Examples: Stored, Compiled DATA Step Programs” (*SAS V9 LIBNAME Engine: Reference*)
アイテムストア	ITEMSTOR	.sas7bitm	■ “Listing Templates in a Template Store” (*SAS Output Delivery System: Procedures Guide*) ■ “SAS レジストリの保存場所” ■ 線形モデルのアイテムストアの詳細については、*SAS/STAT ユーザーガイド*の PLM プロシジャを参照してください。

1 この表と次の表のファイル拡張子は、ランダムアクセスファイル用です。順次アクセスファイルは、あまり一般的ではありませんが、これらの拡張子の文字 b を文字 s に置き換えます。

ライブラリの内容をリストする DATASETS プロシジャからの出力では、追加のファイルがデータセットの下にリストされ、異なるメンバータイプを持つことに気づく場合があります。これらのファイルはデータセットの属性であり、別のファイルに保存されます。これらのメンバータイプはデータセットに関連付けられているため、MEMTYPE=オプションでは指定できません。

表 11.2 別のファイルに保存されるデータセット属性

SAS ファイル	メンバータイプ	Linux ファイル拡張子
監査証跡	AUDIT	.sas7baud
拡張属性	XATTR	.sas7bxat
インデックス	INDEX	.sas7bndx

ライブラリ割り当ての要素

ライブラリの割り当ての概要

次の表は、ライブラリを割り当てる一般的な方法をまとめたものです。割り当てを永続化する方法を選択しない限り、割り当ては現在のセッションでのみ有効です。

表 11.3 SAS ライブラリを割り当てて永続化する方法

方法	現在のセッションを超えた永続化
LIBNAME ステートメント(例) LIBNAME 関数(例)	LIBNAME ステートメントを SAS レジストリ、autoexec ファイル、または構成に配置します。
ライブラリの作成ウィンドウ	SAS Data and AI Studio(2026.06 より前は SAS Studio という名前)で**起動時にこのライブラリを再作成する**チェックボックスを選択します。
管理インターフェイス	SAS 管理者は、ライブラリを事前に割り当てて永続化します。
環境変数	ライブラリの場所を環境変数に保存し、その変数名をライブラリ参照名のかわりに使用できます。ただし、エンジン名やライブラリオプションを含めることはできません。デフォルトのエンジンが使用されます。

ライブラリ名(ライブラリ参照名)

ライブラリ参照名の命名規則

ライブラリ参照名は、ライブラリ割り当ての必須要素です。

libname *libref engine* '*location*' *options*;

*ライブラリ参照名*は、データの物理的な場所を参照するために作成するショートカット名またはニックネームです。たとえば、2 レベル名 mylib.myfile では、ライブラリ参照名は mylib で、ライブラリメンバーは myfile です。

ライブラリ参照名の規則は次のとおりです。

- ライブラリ参照名は 8 バイト以下にすることができます。
- 最初の文字は英字またはアンダースコアにする必要があります。後続の文字には、文字、数字、またはアンダースコアを使用できます。詳細については、“ほとんどの SAS 名の規則” (36 ページ)を参照してください。
- 意図した場合を除き、予約名 Sashelp、Sasuser、または Work をライブラリ参照名として使用しないでください。(“SAS デフォルトライブラリ” (236 ページ).を参照してください。)

ライブラリ参照名の使用規則

ライブラリ参照名の使用に関するいくつかの規則を次に示します。

- SAS ファイルの 2 レベル名の最初の要素としてライブラリ参照名を指定します。
- ライブラリ割り当てを永続化する方法を選択しない限り、ライブラリ参照名は現在のセッションでのみ有効です。
- セッションが終了する前に、ライブラリ参照名を割り当て解除(クリア)できます。
- ライブラリ参照名の割り当てが解除されるか、セッションが終了しても、ライブラリメンバーは物理的な保存場所から削除されません。(ただし、Work ライブラリの内容はセッション終了時に削除されます。)
- セッション内でライブラリ参照名を繰り返し参照できます。
- マクロ変数をライブラリ参照名として使用する場合は、変数を引用符で囲まないでください。変数の後に区切り文字を置きます。たとえば、2 レベル名 &mylib..mydata は、mydata ファイルが配置されているライブラリのライブラリ参照名を見つけるために&mylib 変数を参照します。“解決済みテキストの後ろにピリオドを挿入する” (*SAS マクロ言語: リファレンス*)を参照してください。

ライブラリエンジン

SAS は、SAS ファイル、または別のアプリケーションや DBMS によってフォーマットされたファイルにアクセスするための多数のエンジンを提供します。ライブラリ割り当てではエンジン名が必要ない場合もありますが、エンジンを指定することをお勧めします。

libname *libref engine* '*location*' *options*;

SAS ライブラリエンジンに関する重要な概念をいくつか示します。

- 出荷時のデフォルトの Base SAS エンジンは BASE であり、これは V9 エンジンのエイリアスです。

- 新しいライブラリを作成するときにエンジン名を指定せず、ENGINE=システムオプションを指定していない場合は、V9 エンジンが自動的に選択されます。
- ライブラリの場所にすでに SAS ファイルが含まれている場合は、SAS でそれらのファイルに基づいて正しいエンジンを割り当てることもできます。たとえば、場所に V9 データセットのみが含まれている場合、SAS は V9 エンジンを割り当てます。ただし、ライブラリの場所に異なるエンジンファイルが混在している場合、SAS では必要なエンジンを割り当てられないことがあります。したがって、エンジンを指定することがベストプラクティスです。

詳細については、12 章, “SAS エンジン” (269 ページ)を参照してください。

ライブラリの場所

ライブラリ割り当てには物理的な場所が必要です。

libname *libref engine* '*location*' *options*;

データを作成またはアクセスする物理的な場所のパス名を指定します。相対パス名を指定すると、現在の作業ディレクトリが参照されます。これは、SAS インターフェイスと配置に応じて異なります。

物理的な場所を単一引用符または二重引用符で囲みます。ライブラリを連結している場合、ライブラリの場所の構文は少し異なります。

LIBNAME ステートメントをサブミットする前にライブラリの物理的な場所が存在しなかった場合、SAS はログに注記を書き込みます。場合によっては、このような LIBNAME ステートメントをサブミットした後に場所を作成すると、ライブラリ参照名を正常に使用できるようになります。ただし、多くの処理モードおよびインターフェイスでは、物理的な場所を作成した後に LIBNAME ステートメントを再サブミットする必要があります。

DLCREATEDIR システムオプションを設定して、自動的にライブラリの新しいサブフォルダーを作成できます。

ライブラリオプション

エンジンと環境によっては、ライブラリ割り当てでライブラリオプションが必要になる場合があります。

libname *libref engine* '*location*' *options*;

DBMS またはクラウドストレージシステムに保存されているデータにアクセスする場合、通常は接続情報を指定する必要があります。エンジンに固有の追加の LIBNAME ステートメントオプションが必要になる場合があります。

さらに、一部の SAS データセットオプションは、LIBNAME ステートメントオプションとしても使用できます。LIBNAME ステートメントオプションはシステムオプションよりも優先されることに注意してください。データセットオプションは、LIBNAME ステートメントオプションよりも優先されます。

SAS エンジンに固有の LIBNAME ステートメントオプションについては、次のようなドキュメントを参照してください。

- *リレーショナルデータベース用 SAS/ACCESS: リファレンス*
- *SAS V9 LIBNAME エンジン: リファレンス*
- *SAS/ACCESS Interface to PC Files: SAS Viya プラットフォームのリファレンス*
- *ORC および Parquet 用の SAS Viya LIBNAME エンジン: リファレンス*

ライブラリ参照名を使用する利点

SAS ライブラリを介してデータにアクセスすると、次の利点があります。

- SAS コードでライブラリ参照名を指定すると、特にプログラム内でファイルを繰り返し参照する場合、パス名を指定するよりも便利です。
- ライブラリ全体のオプションを指定できます。DBMS またはクラウドストレージシステムに保存されているデータにアクセスするには、通常、複数のオプションが必要です。
- データの場所が変更された場合、コード内で複数の場所を変更するかわりに、ライブラリ割り当てでそのパス名を 1 回変更するだけで済みます。
- 関連ファイルは同じ物理的な場所にグループ化できます。
- ユーザーインターフェイスに加えて、SAS は、ライブラリの割り当て、 to ライブラリに関する情報の表示、およびライブラリの管理を行うためのプログラムによる方法を提供します。

ライブラリ参照名を使用せずにデータにアクセスする

場合によっては、ライブラリ割り当てが必要ない場合があります。

- ライブラリ参照名を省略して 1 レベル名を指定した場合、SAS は一時 Work ライブラリを使用します。これは通常、セッションの終了時に削除されます。この動作は変更できます。SAS ファイルを永久保存するデフォルトライブラリを設定するには、User ライブラリを割り当てます。User ライブラリが割り当てられている場合は、Work ライブラリよりも優先されます。
- ファイル名のみ(パス名なし)を引用符で囲んで指定した場合、デフォルトの場所は現在の作業ディレクトリになります。この場所は、SAS インターフェイスと配置によって異なります。
- ほとんどの言語要素では、ライブラリ参照名を省略し、物理的な場所を指定することでファイルを直接参照できます。
- テキストファイルに保存されている生データなどの外部の非構造化データの場合は、ライブラリ参照名のかわりにファイル参照名を使用します。

ライブラリ参照名のかわりに物理的な場所を指定するための規則は次のとおりです。

- パス名とファイル名は引用符で囲みます。ファイルが SAS データセットの場合は、ファイル拡張子を省略できます。
- 引用符で囲まれたパス名とファイル名は、動作環境またはクラウドストレージシステムの命名規則に準拠している必要があります。
- ライブラリ割り当ての利点により、物理的な場所を指定するよりもライブラリ参照名を指定することをお勧めします
- 引用符で囲まれた物理的な場所は、次の SAS 機能ではサポートされていません。
  - デフォルトの Base SAS エンジン(V9)以外のエンジン
  - PROC COPY の SELECT ステートメントやほとんどの PROC DATASETS ステートメントなど、ライブラリ参照名を受け入れないコンテキスト
  - PROC SQL
  - 特定のデータセットオプション
  - SAS ビュー
  - ストアドコンパイル DATA ステッププログラム
  - SAS カタログ
  - MDDB および FDB 参照

ライブラリ連結

SAS ライブラリ連結の定義

2 つ以上の SAS ライブラリを連結すると、ライブラリの内容が論理的に結合されます。ライブラリは異なる場所に保存されていますが、1 つのライブラリ参照名ですべてを参照できます。

ライブラリを連結する方法

LIBNAME ステートメントまたは LIBNAME 関数では、各ライブラリを、以前に割り当てられたライブラリ参照名またはその物理的な場所(完全なパス名)で指定します。

- ライブラリ参照名と物理的な場所を組み合わせて使用できます。
- 物理的な場所を指定する場合は、単一引用符または二重引用符で囲みます。
- 各指定は空白またはカンマで区切ります。

- リスト全体をかっこで囲みます。

3 つのライブラリを連結する例を次に示します。この指定には 2 つのライブラリ参照名と 1 つの物理的な場所が含まれています。

```
libname year (quarter1 quarter2 '/quarter3/sales');
```

より詳細な例については、“例: SAS ライブラリを連結する” (240 ページ)を参照してください。

ライブラリ連結の規則

ライブラリ連結を作成した後、ライブラリ参照名を受け入れるコンテキストでライブラリ参照名を指定できます。SAS は次の規則を適用します。

- オプションまたはエンジンを指定した場合、それらは物理的な場所で指定したライブラリにのみ適用され、ライブラリ参照名で指定したライブラリには適用されません。
- 連結内で割り当てられた後にライブラリ参照名を変更しても、連結には影響しません。
- 2 つ以上の SAS ライブラリを論理的に連結すると、SAS カタログも連結されます。“Catalog Concatenation” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。
- 連結内のいずれかのライブラリが順次である場合、すべてのライブラリが順次として扱われます。

ライブラリをリストする順序によって、SAS ファイルの処理方法が決まります。

- データセットが入力または更新のために開かれると、連結ライブラリは連結内にリストされている順序で検索されます。最初に出現したデータセットが使用されます。
- データセットが作成されると、連結内の別のライブラリに同じ名前のファイルが存在する場合でも、そのデータセットは連結内にリストされている最初のライブラリに作成されます。
- SAS ファイルを削除または名前変更すると、最初に出現したファイルのみが影響を受けます。
- SAS ファイルのリストが表示されるときは、最初に出現したファイル名のみが表示されます。
- 別のファイル(データセットのインデックスなど)に論理的に関連付けられている SAS ファイルは、親ファイルが同じライブラリ内に存在する場合にのみ連結に含まれます。この規則は、最初に出現したファイルのみが使用されるという規則の影響を受けます。たとえば、2 つのライブラリが連結されます。どちらのライブラリにも、Mytable という名前のデータセットが含まれています。2 番目のライブラリでは、Mytable にインデックスが付けられます。インデックスは連結に含まれません。
- 最初のライブラリの属性によって、連結の属性が決まります。たとえば、最初のライブラリが読み取り専用の場合、連結ライブラリ全体が読み取り専用になります。

SAS デフォルトライブラリ

SAS デフォルトライブラリの概要

次のライブラリが SAS によって提供されます。

- Work はデフォルトで割り当てられます。
- User は、ユーザーまたは管理者によって割り当てられます。
- Sahelp はデフォルトで割り当てられます。
- Sasuser はデフォルトで割り当てられます。

Work ライブラリ(一時)

SAS は Work ライブラリを自動的に割り当てます。Work ライブラリには、次の 2 種類の一時ファイルが保存されます。

- ユーザーが作成したファイル
- 処理の一部として SAS によって内部的に作成されるファイル

Work ライブラリを使用すると、一時ストレージに 1 レベル名を指定できます。*1 レベル名*を指定するということは、ライブラリ参照名を省略し、引用符なしでファイル名のみを指定することを意味します。“例: 一時データに Work ライブラリを使用する” (251 ページ)を参照してください。

デフォルトでは、セッションが正常に終了した場合、Work ライブラリは各セッションの終了時に削除されます。Work ライブラリのデフォルトの動作を変更するには、*SAS システムオプション: リファレンス*の WORK=、WORKINIT、および WORKTERM システムオプションを参照してください。

Work ライブラリとは対照的に、ほとんどの SAS ライブラリは一時的ではなく永久的です。

User ライブラリ

User ライブラリを使用すると、永久ストレージに 1 レベル名を指定できます。User ライブラリを割り当てる場合、1 レベル名で作成されたすべてのファイルは、一時的な Work ライブラリではなく User ライブラリに保存されます。1 レベル名でファイルを参照すると、SAS は User ライブラリでファイルを探します。User ライブラリを割り当てていない場合、SAS は Work ライブラリでファイルを探します。

User ライブラリの例 (253 ページ)のいずれかの方法を使用して、User ライブラリを割り当てることができます。“USER= システムオプション” (*SAS システムオプション: リファレンス*)も参照してください。

User ライブラリに保存されているファイルは、セッションが終了しても SAS によって削除されません。SAS が内部で作成したデータファイルは、引き続き Work ライブラリに保存されます。User ライブラリを割り当てた後、Work に一時ファイルを作成する場合は、Work.MyFile などの 2 レベル名で Work を指定する必要があります。

SPD Engine の場合、TEMP=YES LIBNAME ステートメントオプションを USER=システムオプションとともに使用して、1 レベル名で参照できる一時データセットを保存できます。“TEMP= LIBNAME Statement Option” (*SAS Scalable Performance Data Engine: Reference*)を参照してください。

Sashelp ライブラリ

SAS は Sahelp ライブラリを自動的に割り当てます。Sashelp ライブラリには次のアイテムが含まれています。

- SAS ドキュメントの一部の例で使用されるサンプルデータ。
- カタログ、アイテムストア、およびサイト全体の SAS 設定を保存するその他のファイル。Sashelp ライブラリのデフォルトは、オンサイトの SAS サポート担当者がカスタマイズできます。(多くの設定は、2 つのアイテムストアである SAS レジストリに保存されます。35 章, “SAS レジストリ” (797 ページ)を参照してください。)

PROC DATASETS または PROC CATALOG を使用して、ライブラリ内のカタログをリストします。SASHELP システムオプションは、Sashelp ライブラリの場所を指定します。このオプションはインストールプロセス中に設定され、通常はインストール後に変更されることはありません。

Sasuser ライブラリ

SAS は Sasuser ライブラリを自動的に割り当てます。Sasuser ライブラリには、パーソナル設定やカスタマイズを保存するカタログやその他のファイルが含まれています。これらの値は、Sasuser.Profile という名前のカタログに保存されます。“Sasuser.Profile Catalog” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。(多くの設定は、2 つのアイテムストアである SAS レジストリに保存されます。35 章, “SAS レジストリ” (797 ページ)を参照してください。)

SASUSER システムオプションは、Sasuser ライブラリの場所を指定します。

RSASUSER システムオプションを使用すると、システム管理者は Sasuser ライブラリへのアクセスモードを制御できます。RSASUSER は、複数のユーザー用に 1 つの Sasuser ライブラリがあるインストール環境で、それらのユーザーがそれを変更できないようにするのに役立ちます。

例: ライブラリ参照名を使用してデータにアクセスする

例: LIBNAME ステートメントを使用してライブラリ参照名を割り当てる

コード例

この例では、ライブラリ参照名を割り当てて、DATA ステップと PROC ステップでそれを参照します。

```
libname sales 'library-path';   /* 1 */
data sales.quarter1;            /* 2 */
  length mileage 4;
  input account mileage;
  datalines;
1 932
2 563
;
proc print data=sales.quarter1; /* 3 */
run;
```

1 LIBNAME ステートメントは、ライブラリ参照名 sales をライブラリの場所に割り当てます。*library-path* をライブラリの場所に置き換えます。この場所はすでに存在しており、SAS Compute Server からアクセスできる必要があります。

2 DATA ステップでは、データセット sales.quarter1 を作成し、それをライブラリの物理的な場所に保存します。

3 PROC PRINT ステップは、2 レベル名 sales.quarter1 によってデータセットを参照します。

要点

- LIBNAME ステートメントは、ライブラリ参照名をライブラリの物理的な場所に割り当てるために一般的に使用される方法です。
- ライブラリが管理者によって割り当てられている場合でも、ライブラリの概念を理解しておくと役立ちます。

- SAS® Viya®でライブラリを割り当てるときは、その場所がすでに存在しており、計算サーバーからアクセスできる必要があります。

関連項目

- “LIBNAME Statement: V9 Engine” (*SAS V9 LIBNAME Engine: Reference*)
- “ライブラリ割り当ての要素” (230 ページ)
- “例: 永久データ用の User ライブラリの割り当て” (253 ページ)
- “ライブラリの操作” (*SAS Studio: ユーザーガイド*)

例: 関数を使用してライブラリ参照名を割り当てる

コード例

この例では、関数を使用してライブラリ参照名を割り当て、その割り当てを検証する test という名前のマクロを作成します。

```
%macro test;
  %let mylibref=new;                          /* 1 */
  %let mydirectory=library-location;          /* 2 */
  %if %sysfunc(libname(&mylibref,&mydirectory)) %then /* 3 */
    %put %sysfunc(sysmsg());
  %else %put success;
  %if %sysfunc(libref(&mylibref)) %then       /* 4 */
    %put %sysfunc(sysmsg());
  %else %put library &mylibref is assigned to &mydirectory;
%mend test;                                   /* 5 */

%test
```

1 マクロ変数 mylibref は、ライブラリ参照名 new を指定します。

2 マクロ変数 mydirectory は、ライブラリの場所を指定します。*library-path* をライブラリの場所に置き換えます。この場所はすでに存在しており、計算サーバーからアクセスできる必要があります。

3 IF-THEN-ELSE ステートメントは%SYSFUNC マクロ関数を呼び出し、次に LIBNAME 関数を呼び出してライブラリ割り当てを試みます。エラーまたは警告が発生した場合、メッセージは SAS ログに書き込まれます。エラーや警告が発生しない場合は、success がログに書き込まれます。マクロステートメントでは、文字列を引用符で囲まないことに注意してください。

4 別の IF-THEN-ELSE ステートメントは、%SYSFUNC マクロ関数を呼び出します。この関数は、ライブラリ割り当てを検証するために LIBREF 関数を呼び出します。

この場合も、エラーまたは警告が発生した場合、メッセージは SAS ログに書き込まれます。エラーや警告が発生しない場合は、情報メッセージがログに書き込まれます。

5 test マクロが実行されます。

例のコード 11.1 マクロの実行によるログの出力

```
12  %test
success
library new is assigned to library-location
```

要点

- プログラマの中には、ステートメントよりも SAS 関数を使用することを好む人もいます。LIBNAME 関数は、ライブラリ割り当てを割り当てまたは割り当て解除(クリア)できます。LIBREF 関数はライブラリ割り当てを検証できます。
- LIBNAME 関数の動作は引数の数によって異なります。
- マクロ関数内で DATA ステップ関数を使用する場合は、追加のルールに注意してください。

関連項目

- “LIBNAME 関数” (*SAS 関数と CALL ルーチン: リファレンス*)
- “LIBREF 関数” (*SAS 関数と CALL ルーチン: リファレンス*)
- “%SYSFUNC マクロ関数” (*SAS マクロ言語: リファレンス*)
- “マクロ関数内で DATA ステップ関数を使用する” (*SAS 関数と CALL ルーチン: リファレンス*)
- “ライブラリ割り当ての要素” (230 ページ)

例: SAS ライブラリを連結する

コード例

この LIBNAME ステートメントは、2 つの SAS ライブラリを連結します。

```
libname lib3 (lib1 lib2);
```

次の図は、連結を示しています。

アウトプット 11.1 Lib1 と Lib2 の連結

lib1

Name	Member Type
APPLES	DATA
FORMATS	CATALOG
PEARS	DATA

+

lib2

Name	Member Type
APPLES	DATA
APPLES	INDEX
FORMATS	CATALOG
FRUIT	CATALOG
ORANGES	DATA
ORANGES	INDEX

=

lib3

Name	Member Type
APPLES	DATA
FORMATS	CATALOG
FRUIT	CATALOG
ORANGES	DATA
ORANGES	INDEX
PEARS	DATA

apples のインデックスが連結に表示されないことに注意してください。lib2.apples データセットにはインデックスがあります。ただし、lib1.apples データセットにはインデックスがなく、lib1 が連結の最初にリストされます。SAS は、関連するデータセットが連結の一部ではない場合、インデックスを抑制します。

複数のカタログが同じ名前を持つ場合、それらのエントリは連結されます。lib3.formats カタログは、lib1.formats カタログと lib2.formats カタログのエントリを組み合わせたものです。詳細については、“Catalog Concatenation” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。

要点

- ライブラリ連結により、異なる物理的な場所に保存されている複数のライブラリを参照できます。
- データセットが入力または更新のために開かれると、連結ライブラリが検索され、最初に出現したデータセットが使用されます。データセットが作成されると、連結内の別のライブラリに同じ名前のファイルが存在する場合でも、そのデータセットは連結内にリストされている最初のライブラリに作成されます。同じ名前のデータセットが異なる場所に存在する場合、望ましくない動作が発生する可能性があります。

関連項目

- “ライブラリ連結” (234 ページ)
- “Catalog Concatenation” (*SAS V9 LIBNAME Engine: Reference*)

例: SAS/CONNECT ソフトウェアを使用してリモート SAS ライブラリにアクセスする

コード例

この例では、SAS/CONNECT スポーナーを使用して、リモートコンピューターに保存されている SAS ライブラリにアクセスします。

```
options comamid=tcp;                              /* 1 */
%let myserver=host.name.com;                      /* 2 */
signon myserver.__1234 user=userid password='mypw';    /* 3 */
libname reports '/myremotedata' server=myserver.__1234; /* 4 */
proc datasets library=reports;                    /* 5 */
run;
quit;
signoff myserver.__1234;
```

1 COMAMID=システムオプションは、通信アクセス方式として TCP/IP を指定します。

2 myserver マクロ変数は、リモート SAS/CONNECT サーバーのホスト名に割り当てられます。

3 SIGNON ステートメントは、myserver マクロ変数の後に、SAS/CONNECT スポーナーがリッスンしているポート番号を参照します。ポート番号またはサービス名がマクロ変数で定義されている場合は、それを SIGNON ステートメントから省略します。

4 LIBNAME ステートメントでは、エンジンを指定しないでください。データの場所を指定し、SERVER=オプションで myserver マクロ変数を指定します。SIGNON ステートメントでポート番号が指定されている場合は、ポート番号を含めます。

5 PROC DATASETS はクライアントで実行されます。クライアントはサーバー上のデータにアクセスしますが、データはクライアントのローカルディスクには書き込まれません。

PROC DATASETS からの次の出力は、REMOTE エンジンがディレクトリに割り当てられていることを示しています。

アウトプット 11.2 リモートディレクトリ情報を示す PROC DATASETS 出力の一部

Directory	
Libref	REPORTS
Engine	REMOTE
Physical Name	/myremotedata
Accessed through server	MYSERVER
Server's libref	REPORTS
Server's engine	V9
Server's SAS release	9.04.01M6P11072018 (160)
Server's host type	z/OS - IBM 8561
Server's versus user's data representation	DIFFERENT
Views interpreted in server's execution	YES
Owner	USERID
Group	.
File Size	8192
Device number (dev)	1120
Dir serial number (ino)	1

要点

- SAS/CONNECT ソフトウェアを構成すると、LIBNAME ステートメントを使用して、サーバー(リモート)データがクライアントのディスクに保存されているかのように、読み取り、書き込み、および更新を行うことができます。
- SAS はクライアントメモリ内のデータを処理します。このデータは、後続のクライアントによるサーバーデータ要求で上書きされます。
- リモートアクセスは、異なるアーキテクチャを持つコンピューター間でデータセットにアクセスする場合に便利です。一部のカタログエントリタイプは読み取り専用アクセスが可能です。

関連項目

- “Types of Sign-Ons” (*SAS/CONNECT for the SAS Viya Platform: User’s Guide*)

例: WebDAV サーバー上のリモート SAS ライブラリにアクセスする

コード例

次の LIBNAME ステートメントは WebDAV サーバーにアクセスします。

```
libname davdata v9 "https://www.webserver.com/datadir" /* 1 */
      webdav user="userid" pw="12345";          /* 2 */
```

1 LIBNAME ステートメントは、ライブラリ参照名 davdata を WebDAV サーバーの URL の場所に割り当てます。

2 WebDAV サーバーにアクセスするには、WEBDAV オプションが必要です。

要点

- WebDAV (Web 分散オーサリングとバージョン管理)は、HTTP プロトコルを拡張したプロトコルです。インターネット上での共同オーサリングのための標準インフラストラクチャを提供します。WebDAV を使用すると、Web ドキュメントを編集したり、後で取得できるようにバージョンを保存したり、上書きを防止するロックメカニズムを提供したりできます。
- WebDAV サーバー上のファイルにアクセスすると、SAS は処理のためにファイルをサーバーからローカルディスクにプルします。一時ストレージに別の場所を指定する LIBNAME ステートメントで LOCALCACHE=オプションを使用しない限り、ファイルは Work ライブラリに一時的に保存されます。ファイルの更新が終了すると、SAS はファイルを WebDAV サーバーにプッシュして保存し、ローカルディスクからファイルを削除します。

関連項目

- “LIBNAME ステートメント: WebDAV サーバーアクセス” (*SAS グローバルステートメント: リファレンス*)
- “FILENAME ステートメント: WebDAV アクセスメソッド” (*SAS グローバルステートメント: リファレンス*)

例: DBMS データに SAS ライブラリとしてアクセスする

コード例

この例では、SAS DATA ステップによって Teradata テーブルが作成されます。この例を実行するには、SAS/ACCESS 用に Teradata インターフェイスを構成する必要があります。

```
libname mytddata teradata server=mytera user=myid password=mypw; /* 1 */
data mytddata.grades;                                  /* 2 */
  input student $ test1 test2 final;
  datalines;
Fred 66 80 70
Wilma 97 91 98
;
proc datasets library=mytddata;                        /* 3 */
run;
quit;
```

1 LIBNAME ステートメントは、mytddata ライブラリ参照名と、SAS/ACCESS Interface to Teradata のエンジンニックネームである TERADATA を指定します。このステートメントでは、Teradata の接続オプションも指定します。これらのオプションを変更して、SAS/ACCESS 接続値および必要なその他のオプションを指定します。

2 DATA ステップでは、grades という名前のテーブルを作成します。テーブルは Teradata DBMS 内にあり、SAS データセットではありません。

3 mytddata ライブラリの PROC DATASETS 出力は、エンジンが Teradata であることを示しています。grades テーブルの場合、SAS メンバータイプは DATA、DBMS メンバータイプは TABLE です。

アウトプット 11.3　DBMS ディレクトリ情報を示す PROC DATASETS 出力

Directory	
Libref	MYTDDATA
Engine	TERADATA
Physical Name	mytera
Schema/User	myid

アウトプット 11.4 DBMS コンテンツのライブラリコンテンツを示す PROC DATASETS 出力

#	Name	Member Type	DBMS Member Type
1	grades	DATA	TABLE

要点

- DBMS データ用に SAS/ACCESS インターフェイスを構成すると、SAS で LIBNAME ステートメントをサブミットしてテーブルにアクセスできます。SAS は、SAS データセットにアクセスする場合と同様の方法で DBMS テーブルにアクセスできます。ただし、既存の DBMS テーブルを置き換えることはできないことに注意してください。SAS/ACCESS インターフェイスは、REPLACE= SAS データセットオプションをサポートしません。
- 一部の SAS/ACCESS インターフェイスでは大文字と小文字が区別されます。DBMS の要件に準拠するために、テーブル名または列名の大文字と小文字を変更する必要がある場合があります。
- SAS 以外の DBMS またはアプリケーションに保存されているデータにアクセスする場合は、通常、接続情報を指定する必要があります。エンジンに固有の追加の LIBNAME ステートメントオプションが必要になる場合があります。

関連項目

- SAS/ACCESS ドキュメント
- 12 章, “SAS エンジン” (269 ページ)

例: DBMS データにアクセスして SAS ビューを作成する

コード例

この例では、“例: DBMS データに SAS ライブラリとしてアクセスする” (245 ページ)で作成された Teradata テーブルを参照する SAS ビューを作成します。このコードは、ネイティブの Teradata ビューではなく、SAS ビューを作成します。PROC SQL ビューを作成するには、“例: PROC SQL ビューに SAS/ACCESS LIBNAME ステートメントを埋め込む” (291 ページ)を参照してください。

```
libname target 'library-path';                          /* 1 */
libname mytddata teradata server=mytera user=myid password=mypw; /* 2 */
data target.highgrades / view=target.highgrades;        /* 3 */
  set mytddata.grades;
  where final gt 80;                                    /* 4 */
run;
proc print data=target.highgrades;                      /* 5 */
run;
```

1 LIBNAME ステートメントでは、ライブラリ参照名 target が SAS ライブラリの場所に割り当てられます。*library-path* をライブラリの場所に置き換えます。エンジンが指定されていないため、SAS はデフォルトの V9 エンジンを割り当てます。

2 これは、“例: DBMS データに SAS ライブラリとしてアクセスする” (245 ページ) と同じ SAS/ACCESS LIBNAME ステートメントです。

3 DATA ステップでは、grades という名前の Teradata テーブルを参照する highgrades という名前の SAS ビューを作成します。

4 ビューには、final 変数が 80 より大きい行が含まれます。

5 PROC PRINT はビューを実行します。DATA ステップビューは LIBNAME ステートメントを保持しないことに注意してください。したがって、このビューを参照するときは、まず mytddata ライブラリと target ライブラリの LIBNAME ステートメントをサブミットする必要があります。

PROC PRINT 出力は、Wilma の final 成績が 80 より大きいことを示しています。Fred の final 成績は 80 以下のため、Fred はビューに含まれません。

アウトプット 11.5 Teradata テーブルを参照する SAS ビューの PROC PRINT

Obs	Student	test1	test2	final
1	Wilma	97	91	98

PROC DATASETS を実行すると、highgrades のメンバータイプが VIEW であることがわかります。

```
proc datasets library=target;
run;
quit;
```

アウトプット 11.6 Target ライブラリの PROC DATASETS 出力

#	Name	Member Type	File Size	Last Modified
1	HIGHGRADES	VIEW	5KB	03/19/2019 12:47:03

比較のために、Teradata BTEQ ツールを使用して、Teradata でネイティブテーブルを作成し、表示することができます。BTEQ で CREATE TABLE および CREATE VIEW コマンドを実行して、gradessql という名前のテーブルと gradessqlview という名前のビューを作成します。

mytddata ライブラリに対して PROC DATASETS を実行すると、SAS メンバータイプが DBMS メンバータイプとは異なることがわかります。

```
proc datasets library=mytddata;
```

```
run;
quit;
```

アウトプット 11.7　Mytddata ライブラリの PROC DATASETS 出力

#	Name	Member Type	DBMS Member Type
1	grades	DATA	TABLE
2	gradessql	DATA	TABLE
3	gradessqlview	DATA	VIEW

要点

- 多くの顧客は、DBMS を大規模で継続的なデータストアとして使用しています。DBMS テーブル全体を処理するのではなく、SAS ビューを作成してデータのサブセットをクエリできます。
- SAS/ACCESS LIBNAME ステートメントをサブミットした後、DATA ステップビューまたは PROC SQL ビューを作成できます。もう 1 つのオプションは、SQL パススルー機能を使用する PROC SQL ビューで、DBMS 固有の構文をサブミットします。
- SAS ビューは、基になるデータソース内のデータの現在の状態を反映します。あるいは、基になる DBMS データが変更されることが予想されない場合は、DBMS データを読み取って永久 SAS データセットを作成することもできます。

関連項目

- “SAS Views of DBMS Data” (*SAS/ACCESS for Relational Databases: Reference*)
- “SAS Views” (*SAS V9 LIBNAME Engine: Reference*)
- 12 章, “SAS エンジン” (269 ページ)

例: ライブラリの新しいサブフォルダーを自動的に作成する

コード例

この例では、ライブラリのサブフォルダーを作成するために DLCREATEDIR システムオプションを設定します。

この例では、フォルダー/home/userid/mydata/は存在しますが、サブフォルダー project は存在しません。DLCREATEDIR システムオプションが設定されているため、SAS は project を作成します。

```
options dlcreatedir;
libname mynewlib '/home/userid/mydata/project';
```

例のコード 11.2　ライブラリが作成されたことを示すログ情報

```
NOTE: Library MYNEWLIB was created.
NOTE: Libref MYNEWLIB was successfully assigned as follows:
      Engine:       V9
      Physical Name: /home/userid/mydata/project
```

要点

- DLCREATEDIR システムオプションを設定すると、LIBNAME ステートメントで指定された SAS ライブラリのサブフォルダーが存在しない場合にそのサブフォルダーを作成できます。
- SAS は、既存のフォルダーの新しいサブフォルダーを 1 つ作成できます。つまり、SAS はパス名の最後のコンポーネントのみを作成します。

関連項目

- “DLCREATEDIR システムオプション” (*SAS システムオプション: リファレンス*)

例: ライブラリ参照名の割り当て解除(クリア)

コード例

次のステートメントは、ライブラリ参照名 mylib を物理的な場所から割り当て解除します。

```
libname mylib clear;
```

すべてのライブラリ割り当て(システムライブラリ以外)を割り当て解除するには、LIBNAME ステートメントで_ALL_キーワードを使用します。

```
libname _all_ clear;
```

LIBNAME 関数を使用することもできます。次のコードは、“例: 関数を使用してライブラリ参照名を割り当てる” (239 ページ)で割り当てられたライブラリ参照名 new の割り当てを解除します。

```
%macro test;
  %if (%sysfunc(libname(new))) %then
  %put %sysfunc(sysmsg());
%mend test;
%test
```

要点

- ライブラリ参照名の割り当てを解除すると、リソース、特に共有データを解放するのに役立ちます。
- デフォルトでは、SAS は各セッションの終了時にライブラリ参照名の割り当てを自動的に解除します。
- セッションが終了する前に、ライブラリ参照名の割り当てを解除するように要求できます。
  - LIBNAME ステートメントで、ライブラリ参照名と CLEAR を指定して、単一のライブラリ参照名の割り当てを解除します。現在割り当てられているすべてのライブラリ参照名の割り当てを解除するには、_ALL_および CLEAR を指定します。Sashelp や Sasuser などのシステムライブラリは割り当て解除されません。
  - LIBNAME 関数では、引数 1 つの形式を使用してライブラリ参照名の割り当てを解除します。
- 現在のセッションを超えてライブラリ割り当てを永続化する方法を使用する場合、LIBNAME ステートメントまたは LIBNAME 関数を使用してライブラリ参照名の割り当てを永久に解除することはできません。ライブラリ参照名は現在のセッションに対してのみ割り当て解除され、新しいセッションを開始するときに再割り当てされます。

関連項目

- “LIBNAME Statement: V9 Engine” (*SAS V9 LIBNAME Engine: Reference*)
- “LIBNAME 関数” (*SAS 関数と CALL ルーチン: リファレンス*)

例: ライブラリ参照名を使用せずにデータにアクセスする

例: 一時データに Work ライブラリを使用する

コード例

User ライブラリを設定していない場合、次のコードは mytable を Work ライブラリに書き込みます。

1 レベル名 mytable にはライブラリ参照名がないことに注意してください。このコードは、work.mytable を指定しても同じように動作します。

```
data mytable;
  x=1;
run;
proc contents data=mytable;
run;
```

次は PROC CONTENTS 出力の一部であり、Work ライブラリが使用されていることを示しています。

アウトプット 11.8　Work 内のデータセットの PROC CONTENTS 出力の一部

Data Set Name	WORK.MYTABLE	**Observations**	1
Member Type	DATA	**Variables**	1
Engine	V9	**Indexes**	0
Created	03/30/2020 13:25:07	**Observation Length**	8
Last Modified	03/30/2020 13:25:07	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64, LINUX_POWER_64		
Encoding	utf-8 Unicode (UTF-8)		

要点

- Work ライブラリは中間結果または一時的な結果に使用できます。
- デフォルトでは、データセットの作成時に 1 レベル名を指定すると、データセットは一時的に Work ライブラリに保存されます。1 レベル名を指定して一時ファイルではなく永久ファイルを作成および使用する場合は、User ライブラリを割り当てます。
- Work ライブラリは、各セッションの開始時に SAS によって自動的に定義されます。通常、Work ライブラリ内のファイルは、セッションが正常に終了した場合、各セッションの終了時に削除されます。

関連項目

- “Work ライブラリ(一時)” (236 ページ)
- “例: 永久データ用の User ライブラリの割り当て” (253 ページ)
- “例: パス名を引用符で指定してデータセットにアクセスする” (255 ページ)

例: 永久データ用の User ライブラリの割り当て

コード例

1 レベル名を指定して一時ファイルではなく永久ファイルを作成および使用する場合は、User ライブラリを割り当てます。

この例を実行するには、まず、データセット quarter1 を“例: LIBNAME ステートメントを使用してライブラリ参照名を割り当てる” (238 ページ)で作成します。この例では、*library-path* を同じライブラリの場所に置き換えます。

```
libname sales 'library-path'; /* 1 */
options user=sales;          /* 2 */
proc print data=quarter1;    /* 3 */
run;
```

1 LIBNAME ステートメントは、ライブラリ参照名 sales をライブラリの場所に割り当てます。

2 USER=システムオプションは、sales ライブラリを 1 レベル名のデフォルトとして指定します。

3 PROC PRINT ステップは、1 レベル名 quarter1 によってデータセットを参照します。

sales が PROC PRINT で指定されていない場合でも、ログ出力により、データが sales.quarter1 から読み取られたことが確認されます。

例のコード 11.3 ログ出力

```
1    libname sales 'library-path';
NOTE: Libref SALES was successfully assigned as follows:
      Engine:       V9
      Physical Name: library-path
1 !                         /*1*/
2    options user=sales; /*2*/
3
4    proc print data=quarter1; /*3*/
NOTE: Writing HTML Body file: sashtml.htm
5    run;

NOTE: There were 2 observations read from the data set SALES.QUARTER1.
```

USER=システムオプションを設定するかわりに、LIBNAME ステートメントまたは LIBNAME 関数を使用して User ライブラリを割り当てることができます。次の 2 つの例を試してください。

```
libname user 'library-path';
proc print data=quarter1;
run;

data _null_;
```

```
  x=libname ('user', 'library-path');
run;
proc print data=quarter1;
run;
```

この動作は、ログにライブラリ参照名が user として表示されることを除いて、USER=システムオプションを設定する場合と同じです。

例のコード 11.4　ログ出力

```
NOTE: There were 2 observations read from the data set USER.QUARTER1.
```

要点

- デフォルトでは、データセットの作成時に 1 レベル名を指定すると、データセットは一時的に Work ライブラリに保存されます。1 レベル名を指定して一時ファイルではなく永久ファイルを作成および使用する場合は、User ライブラリを割り当てます。
- USER=システムオプションを使用して、User ライブラリを設定できます。LIBNAME ステートメントなど、ライブラリを割り当てる一般的な方法を使用することもできます。以前に割り当てられたライブラリ参照名または物理的な場所を指定します。
- User ライブラリを設定した後、一時データセットを Work ライブラリに保存する場合は、Work ライブラリ参照名を 2 レベル名で指定する必要があります。

関連項目

- “User ライブラリ” (236 ページ)
- “USER= システムオプション” (*SAS システムオプション: リファレンス*)
- “例: 一時データに Work ライブラリを使用する” (251 ページ)
- “例: パス名を引用符で指定してデータセットにアクセスする” (255 ページ)

例: パス名を引用符で指定してデータセットにアクセスする

コード例

この例では、ライブラリ参照名とデータセット名のかわりに完全なパス名とファイル名で識別されるデータセットを出力します。データセット quarter1 は“例: LIBNAME ステートメントを使用してライブラリ参照名を割り当てる” (238 ページ)で作成されます。

```
proc print data='file-path/file-name.sas7bdat';
run;
```

要点

- 2 レベル名 *libref.file-name* を使用するかわりに、ライブラリ参照名を省略し、引用符で囲んだ完全なパス名とファイル名をハードコーディングすることができます。ファイルが SAS データセットの場合は、ファイル拡張子を省略できます。
- 多くの言語要素は物理的な場所を受け入れません。制限事項については、“ライブラリ参照名を使用せずにデータにアクセスする” (233 ページ)を参照してください。

関連項目

- “LIBNAME Statement: V9 Engine” (*SAS V9 LIBNAME Engine: Reference*)
- “ライブラリ割り当ての要素” (230 ページ)
- “例: 永久データ用の User ライブラリの割り当て” (253 ページ)

例: ライブラリメンバーではないファイルのファイル参照名を指定する

コード例

この例では、ファイル参照名を使用して生データの場所を特定します。ファイル sampdata.txt は外部テキストファイルです。

```
filename test url "http://support.sas.com/publishing/cert/sampdata.txt"
  ;                                                  /* 1 */
data credit;                                           /* 2 */
  infile test firstobs=945 obs=954;                          /* 3 */
  input Account $ 1-4 Name $ 6-22 Type $ 24 Transaction $ 26-31;
run;
proc print data=credit;                                    /* 4 */
run;
```

1 FILENAME ステートメントはファイルの場所を指定します。ファイル参照名は test、アクセス方式は URL、場所はファイルの完全な URL です。

sampdata.txt をセッションからアクセス可能なファイルシステム(NFS マウントされたディレクトリなど)にダウンロードする場合、URL アクセス方式は必要ありません。

2 DATA ステップは、Work ライブラリに credit という名前の一時データセットを作成します。

3 INFILE ステートメントは、FILENAME ステートメントで割り当てられたファイル参照名 test を指定します。FIRSTOBS=および OBS=データセットオプションは、外部ファイルから読み取る行を指定します。ファイル sampdata.txt には、多くのサンプルデータセットとプログラムが含まれています。この例では、スペースで区切られた生データである行 945-954 のみを使用します。

4 PROC PRINT は、外部データが SAS データセットとしてインポートされていることを検証します。

アウトプット 11.9　外部ファイルからのインポートが成功したことを示す PROC PRINT 出力

Obs	Account	Name	Type	Transaction
1	1118	ART CONTUCK	D	57.69
2	2287	MICHAEL WINSTONE	D	145.89
3	6201	MARY WATERS	C	45.00
4	7821	MICHELLE STANTON	A	304.45
5	6621	WALTER LUND	C	234.76
6	1086	KATHERINE MORRY	A	64.98
7	0556	LEE McDONALD	D	70.82
8	7821	ELIZABETH WESTIN	C	188.23
9	0265	JEFFREY DONALDSON	C	78.90
10	1010	MARTIN LYNN	D	150.55

要点

- 区切りデータ(生データとも呼ばれる)を含むテキストファイルは、SAS ライブラリメンバーではありません。したがって、ライブラリ参照名を使用してファイルの場所を参照することはできません。かわりに、FILENAME ステートメントを使用して SAS ファイル参照名を割り当てます。
- FILENAME ステートメントでは、ファイル参照名を外部ファイルまたは出力デバイスに割り当てたり、ファイル参照名と外部ファイルの割り当てを解除したり、外部ファイルの属性をリストしたりできます。
- ファイル参照名を再利用する必要がなく、URL などのアクセス方式も必要ない場合は、FILENAME ステートメントを省略できる場合があります。多くの言語要素では、引用符で囲まれたパス名とファイル名を指定できます。ただし、ファイル参照名はライブラリ参照名と同じ理由で役立つ場合があります。

関連項目

- 14 章, “生データ” (341 ページ)
- “FILENAME ステートメント: URL アクセスメソッド” (*SAS グローバルステートメント: リファレンス*)

例: ライブラリに関する情報の表示

例: ライブラリとそのメンバーに関する情報を出力する

コード例

DATASETS プロシジャは、SAS ライブラリ内のメンバーのリストまたはディレクトリを出力します。

```
libname myfiles 'library-path';
proc datasets library=myfiles;
run;
quit;
```

PROC DATASETS からの出力例では、ディレクトリ情報には、ライブラリ参照名、物理的な場所、およびライブラリのその他の属性が含まれています。出力の 2 番目のセクションには、各メンバーの名前とそのメンバータイプが表示され、その後に関連ファイルが表示されます。出力では、flowers データセットにインデックスがあります。

アウトプット 11.10　ディレクトリ情報を示す PROC DATASETS 出力

Directory	
Libref	MYFILES
Engine	V9
Physical Name	*library-path*
Filename	*library-path*
Inode Number	113544585
Access Permission	rwxr-xr-x
Owner Name	*userid*
File Size	0KB
File Size (bytes)	105

アウトプット 11.11　ライブラリメンバーを示す PROC DATASETS 出力

#	Name	Member Type	File Size	Last Modified
1	CONTAINERS	VIEW	16KB	03/30/2020 19:08:22
2	FLOWERS	DATA	192KB	03/30/2020 19:08:22
	FLOWERS	INDEX	24KB	03/30/2020 19:08:22
3	RESTOCK	DATA	128KB	03/30/2020 19:08:22

要点

- CONTENTS ステートメントを使用しない場合、PROC DATASETS はディレクトリに関する情報を返します。ライブラリメンバーに関する詳細情報が必要な場合は、CONTENTS ステートメントを追加します。多くの PROC DATASETS ステートメントと同様、CONTENTS はスタンドアロンプロシジャ PROC CONTENTS としても使用できます。
- DETAILS オプションを指定すると、オブザベーションの数、変数の数、インデックスの数、およびデータセットラベルに関する情報が含まれます。DETAILS は、PROC DATASETS オプションおよびシステムオプションとして使用できます。
- 返される情報はエンジンによって異なります。PROC DATASETS 出力の例については、エンジンのドキュメントを参照してください。
- ライブラリメンバーと見なされないファイルは、ライブラリディレクトリにリストされません。例としては、内部 SAS ユーティリティファイルや SAS プログラムを含むテキストファイルなどがあります。これらのファイルはファイルシステム内に表示される場合がありますが、PROC DATASETS 出力にはリストされません。

関連項目

- “DATASETS プロシジャ” (*Base SAS プロシジャガイド*)
- “DETAILS システムオプション” (*SAS システムオプション: リファレンス*)

例: LIBNAME ステートメントを使用してライブラリ属性を返す

コード例

この例では、LIBNAME ステートメントで LIST 引数を使用します。LIST 引数は、ライブラリの属性をログに出力します。

```
libname myfiles 'library-path';
libname myfiles list;
```

次の情報がログに書き込まれます。

例のコード 11.5　SAS ログからのライブラリ情報

```
NOTE: Libref=   MYFILES
      Scope=    Compute Server
      Engine=   V9
      Physical Name= library-path
      Filename= library-path
      Inode Number= 113544585
      Access Permission= rwxr-xr-x
      Owner Name= userid
      File Size=            0KB
      File Size (bytes)= 105
```

要点

- 単一の SAS ライブラリの属性をリストするには、ライブラリ参照名を指定します。現在のセッションにライブラリ参照名を持つすべての SAS ライブラリの属性をリストするには、_ALL_を指定します。
- _ALL_を指定すると、環境変数として定義されたライブラリ参照名は、それらのライブラリ参照名を SAS ステートメントですでに使用している場合にのみ表示されます。

関連項目

- “LIBNAME Statement: V9 Engine” (*SAS V9 LIBNAME Engine: Reference*)
- “ライブラリ割り当ての要素” (230 ページ)
- “SAS ログ” (710 ページ)

例: 関数を使用してライブラリの場所を返す

コード例

この例では、PATHNAME 関数を使用して、ライブラリ参照名 myfiles に割り当てられている物理的な場所を返します。

```
libname myfiles '/home/userid/example';
data _null_;
  length path $ 100;
  path=pathname('myfiles');
put path;
run;
```

例のコード 11.6　パス名を示すログ出力

```
/home/userid/example
```

要点

- PATHNAME 関数は、ライブラリ参照名に割り当てられている物理的な場所を返します。PROC DATASETS を使用するか、LIBNAME ステートメントの LIST 引数を使用して、物理的な場所を表示することもできます。
- 他にもいくつかの SAS 関数を使用して、SAS ライブラリまたはライブラリメンバーに関する情報を返すことができます。

関連項目

- “ATTRC 関数” (*SAS 関数と CALL ルーチン: リファレンス*)
- “ATTRN 関数” (*SAS 関数と CALL ルーチン: リファレンス*)
- “EXIST 関数” (*SAS 関数と CALL ルーチン: リファレンス*)
- “PATHNAME 関数” (*SAS 関数と CALL ルーチン: リファレンス*)

例: SAS ライブラリの管理

例: SAS ライブラリの移行

コード例

次の MIGRATE プロシジャの例では、SAS ライブラリのメンバーを移行して、新しい SAS リリースで提供される機能を利用します。この例では、場合によっては必要になる SAS/CONNECT サーバーを使用しません。

この例では、セッションは Windows 版 SAS で作成されたファイルに直接アクセスします。

```
libname myfiles 'library-path-1';
libname target 'library-path-2';
proc migrate in=myfiles out=target;
run;
```

SAS ログには、移行の結果が表示されます。ログには、処理が停止したことが記載されています。ただし、カタログを除くすべてのファイルは正常に移行されます。

例のコード 11.7　ログ出力

```
NOTE: The BUFSIZE= option was not specified with the MIGRATE procedure. The migrated library members
will use the current value for
      BUFSIZE. For more information, see the PROC MIGRATE documentation.
NOTE: Migrating MYFILES.CONTAINERS to TARGET.CONTAINERS (memtype=VIEW).
NOTE: View file MYFILES.CONTAINERS.VIEW is in a format that is native to another host, or the file encoding
does not match the
      session encoding. Cross Environment Data Access will be used, which might require additional CPU
resources and might reduce
      performance.
NOTE: To complete the migration of your DATA step view, MYFILES.CONTAINERS, use the Migration
Validation tools to recompile the
      view.
NOTE: Migrating MYFILES.FLOWERS to TARGET.FLOWERS (memtype=DATA).
NOTE: Data file MYFILES.FLOWERS.DATA is in a format that is native to another host, or the file encoding
does not match the session
      encoding. Cross Environment Data Access will be used, which might require additional CPU resources
and might reduce
      performance.
NOTE: Simple index plantname has been defined.
NOTE: There were 7 observations read from the data set MYFILES.FLOWERS.
NOTE: The data set TARGET.FLOWERS has 7 observations and 3 variables.
ERROR: File MYFILES.FORMATS.CATALOG was created for a different operating system.
WARNING: No data is available.
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE MIGRATE used (Total process time):
      real time          0.03 seconds
      cpu time           0.05 seconds
```

異なる動作環境で作成されたライブラリのメンバーを移行する場合(たとえば、ライブラリを Windows から Linux に移行する場合)、PROC MIGRATE にはいくつかの制限があります。たとえば、次のログ情報には、クロス環境データアクセス(CEDA)が使用されたことが記載されています。このメッセージは、PROC MIGRATE のパフォーマンスが CEDA によって低下する可能性があることを通知します。ただし、移行が完了すると、移行されたファイルをターゲット環境で処理するときに CEDA は使用されなくなります。

例のコード 11.8　CEDA が使用されたことを示す情報ログメッセージ

```
NOTE: Data file MYFILES.FLOWERS.DATA is in a format that is native to another
host, or the file encoding does not match the session encoding. Cross Environment
Data Access will be used, which might require additional CPU resources and might
reduce performance.
```

次のようなエラーメッセージも CEDA が原因であり、formats カタログが移行されなかったことを示しています。PROC MIGRATE を使用してカタログを互換性のない動作環境に移行するには、SAS/CONNECT サーバーを使用して IN=ライブラリにアクセスする必要があります。

例のコード 11.9　エラーを示すログメッセージ

```
ERROR: File MYFILES.FORMATS.CATALOG was created for a different operating system.
WARNING: No data is available.
```

要点

- 通常、PROC MIGRATE は、SAS ライブラリのメンバーを現在の SAS バージョンに移行する最良の方法です。PROC MIGRATE は、ほとんどのユーザーがデータ移行で必要とするデータ属性を保持する 1 ステップのコピープロシジャです。移行されたデータセットとビューは、ターゲットライブラリのデータ表現とエンコーディング属性を引き継ぎます。
- 次の問題のいずれかが存在する場合は、SAS/CONNECT サーバーが必要となり、別の構文が使用されます。“SAS/CONNECT を使用して SAS®9 から移行する” (*Base SAS プロシジャガイド*)を参照してください。
  - ネットワークファイルシステム(NFS)経由でターゲットセッションからソースライブラリに直接アクセスできない場合
  - ソースライブラリに、ターゲットセッションと互換性のないデータ表現で作成されたカタログが含まれている場合
- あるいは、ソースライブラリに直接アクセスできる場合は、移行のかわりにクロス環境データアクセス(CEDA)を使用できます。このために、NFS は直接アクセスと見なされます。この読み取り専用アクセスは自動的かつ透過的ですが、制限に注意する必要があります。33 章, “クロス環境データアクセス” (773 ページ)を参照してください。
- 文字を表すためにより多くのバイトを使用する別の文字エンコーディングに変更する場合は、移行プロセスの一部として CVP エンジンを使用することをお勧めします。“Example: Avoid Truncation When Migrating a SAS Library by Using the CVP Engine” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。

関連項目

- “MIGRATE プロシジャ” (*Base SAS プロシジャガイド*)
- “Compatibility and Migration” (*SAS V9 LIBNAME Engine: Reference*)

例: SAS ライブラリのコピー

コード例

次の COPY プロシジャの例では、myfiles ライブラリ全体を target ライブラリにコピーします。PROC COPY には多くの便利なオプションがありますが、何も指定されていないため、CLONE などのデフォルトの動作が使用されます。

```
libname myfiles 'library-path-1';
libname target 'library-path-2';
proc copy in=myfiles out=target;
run;
```

この例では、ライブラリのすべてのメンバーのデータ表現とエンコーディングが現在のセッションと同じです。したがって、CEDA 注記はログに書き込まれません。

例のコード 11.10　コピーの結果を示すログ出力

```
NOTE: Copying MYFILES.CLASSSUBSET to TARGET.CLASSSUBSET (memtype=VIEW).
NOTE: Copying MYFILES.FORMATS to TARGET.FORMATS (memtype=CATALOG).
NOTE: Copying MYFILES.MYBASEBALL to TARGET.MYBASEBALL (memtype=DATA).
NOTE: Simple index Team has been defined.
NOTE: There were 322 observations read from the data set MYFILES.MYBASEBALL.
NOTE: The data set TARGET.MYBASEBALL has 322 observations and 24 variables.
```

要点

- PROC COPY などの SAS ファイル管理ユーティリティには、次の機能があります。
  - すべてのライブラリメンバーまたは選択したメンバーのみを操作できる
  - ほとんどの場合、データセットとその関連ファイル(インデックスなど)をコピー、名前変更、または移動できる
  - すべての動作環境でサポートされている
- ライブラリを別の動作環境または別の文字エンコーディングにコピーする場合は、PROC COPY に NOCLONE オプションを指定します。NOCLONE オプションは、デ

ータ表現とエンコーディングを OUT=ライブラリのものに変更します。さらに、クロス環境データアクセス(CEDA)が呼び出される場合は、CEDA の制限に注意する必要があります。

- SAS Data and AI Studio(2026.06 より前は SAS Studio という名前)を使用する場合は、ナビゲーションペインでいくつかのファイルのメンテナンスタスクを実行できます。
- SAS は、メンバーが大文字または大文字と小文字が混在した文字で作成された場合でも、Linux でファイル名を作成するときに小文字を使用します。ファイルシステムユーティリティまたはユーザーインターフェイスを 使用して、大文字または大文字と小文字が混在したファイル名を変更すると、SAS はファイルをデータセットとして認識できなくなります。(一般に、ファイルシステムユーティリティを 使用して SAS ファイルの名前変更、コピー、または移動を行うことはお勧めできません。)

関連項目

- “COPY プロシジャ” (*Base SAS プロシジャガイド*)
- “DATASETS プロシジャ” (*Base SAS プロシジャガイド*)
- “CATALOG プロシジャ” (*Base SAS プロシジャガイド*)

例: 移送ファイルを使用した SAS ライブラリのコピー

コード例

この例では、CPORT および CIMPORT プロシジャを使用して、環境間で SAS ライブラリをコピーします。マルチステッププロセスが必要です。

1 ソース環境では、PROC CPORT を使用して移送ファイルを作成します。
2 通信ソフトウェア(FTP など)または記憶装置を使用して、移送ファイルをターゲット環境に移動します。FTP を使用する場合は、バイナリモードでファイルを転送します。
3 ターゲット環境では、PROC CIMPORT を使用して移送ファイルからライブラリをインポートします。

ステップ 1 では、PROC CPORT は、ファイル参照名 tranfile によって参照される移送ファイル mytransfer を作成します。

```
libname source 'c:\example';
filename tranfile 'c:\myfiles\mytransfer';
proc cport library=source file=tranfile;
run;
```

ログでは、SAS ビューであるため、containers が移植されていないことに注意してください。

例のコード 11.11　ログ出力

```
NOTE: Not porting SOURCE.CONTAINERS because memtype is not supported.

NOTE: PROC CPORT begins to transport data set SOURCE.FLOWERS
NOTE: The data set contains 3 variables and 7 observations.
      Logical record length is 24.
NOTE: Transporting data set index information.

NOTE: PROC CPORT begins to transport catalog SOURCE.FORMATS
NOTE: The catalog has 1 entries and its maximum logical record length is 120.
NOTE: Entry TESTFMT.FORMAT has been transported.

NOTE: PROC CPORT begins to transport data set SOURCE.RESTOCK
NOTE: The data set contains 4 variables and 7 observations.
      Logical record length is 40.
```

ステップ 2 では、ユーザーは mytransfer ファイルを Windows 環境から Linux 環境にコピーします。

ステップ 3 では、PROC CIMPORT がファイル mytransfer の内容をインポートして target ライブラリを作成します。

```
libname target '/mydata/example';
filename tranfile '/mydata/mytransfer';
proc cimport library=target infile=tranfile;
run;
```

例のコード 11.12　インポートの成功を示すログ出力

```
NOTE: PROC CIMPORT begins to create/update data set TARGET.FLOWERS
NOTE: The data set index plantname is defined.
NOTE: Data set contains 3 variables and 7 observations.
      Logical record length is 24

NOTE: PROC CIMPORT begins to create/update catalog TARGET.FORMATS
NOTE: Entry TESTFMT.FORMAT has been imported.
NOTE: Total number of entries processed in catalog TARGET.FORMATS: 1

NOTE: PROC CIMPORT begins to create/update data set TARGET.RESTOCK
NOTE: Data set contains 4 variables and 7 observations.
      Logical record length is 40
```

要点

- PROC CPORT と PROC CIMPORT には、PROC MIGRATE と比較していくつかの制限があります。PROC MIGRATE に SAS/CONNECT ソフトウェアが必要で、そのソフトウェアにアクセスできない場合にのみ、この方法を移行に使用してください。PROC MIGRATE は、データセット、カタログ、およびその他のほとんどのメンバータイプを含むライブラリ全体を移行します。“Example: Migrate a SAS Library across Environments by Using SAS/CONNECT” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。

- PROC CPORT は SAS データセットとカタログをサポートしますが、他のメンバータイプはサポートしません。カタログのみを含めるには、CAT=を指定できます。
- FTP を使用して移送ファイルをターゲット環境に移動する場合は、ファイルをバイナリモードで転送します。
- 新しいエンコーディングにトランスコーディングすると、切り捨てが発生する可能性があります。切り捨てが発生した場合は、変数長を拡張する必要があります。CVP エンジンを PROC CPORT で使用することも、EXTENDVAR=オプションを PROC CIMPORT で使用することもできます。
- CPORT プロシジャによって作成された移送ファイルは、XPORT エンジンによって作成された移送ファイルと互換性がありません。

関連項目

- “CPORT プロシジャ” (*Base SAS プロシジャガイド*)
- “CIMPORT プロシジャ” (*Base SAS プロシジャガイド*)
- “PROC CPORT and PROC CIMPORT” (*Moving and Accessing SAS Files*)
- “Compatibility and Migration” (*SAS V9 LIBNAME Engine: Reference*)

12

SAS エンジン

SAS エンジンの定義

*エンジン*は、特定のファイル形式でファイルの読み取りと書き込みを行う SAS ソフトウェアのコンポーネントです。(一部のエンジンは読み取り専用です。)

ほとんどのエンジンは、*SAS ライブラリ*として使用される SAS ファイルのグループにアクセスするため、*ライブラリエンジン*と呼ばれます。詳細については、11 章, “SAS ライブラリ” (227 ページ)を参照してください。

SAS *マルチエンジンアーキテクチャ*を使用すると、さまざまなファイル形式にアクセスできます。

- 特定のエンジンは SAS データのみを処理します。“例: SAS エンジンを使用して SAS データを処理する” (277 ページ)を参照してください。
- 他のエンジンは、他のアプリケーション(DBMS、XML、JSON、Microsoft Excel など)からのデータを解釈します。これらのエンジンは抽象化レイヤーを適用するため、SAS は外部データを SAS データセットまたはデータセットの SAS ライブラリであるかのように処理できます。“例: SAS エンジンを使用して外部データを処理する” (286 ページ)を参照してください。

ライブラリ割り当てまたは ENGINE システムオプションでエンジンを指定します。

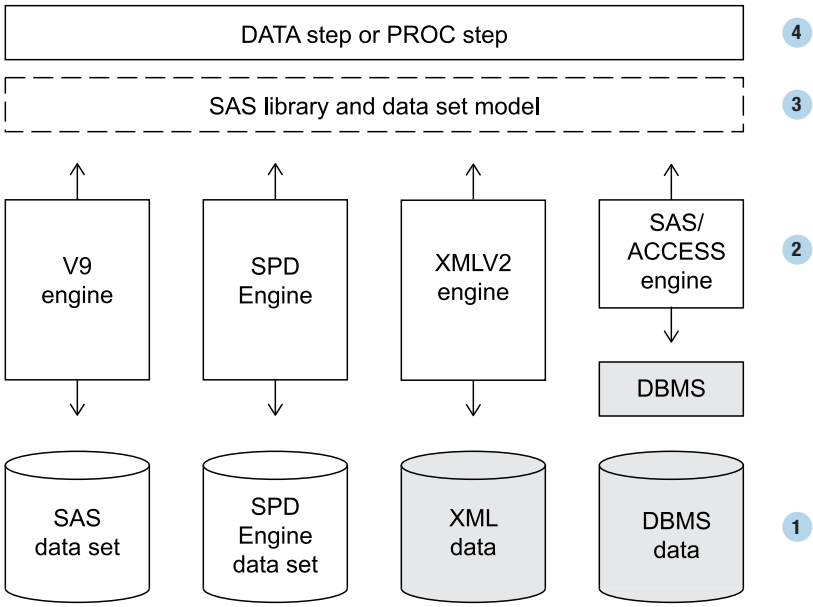
- *ライブラリ割り当て*は、ライブラリ参照名、エンジン、物理的な場所、エンジンまたは環境に固有のオプションで構成されます。“ライブラリ割り当ての要素” (230 ページ)を参照してください。
- *デフォルトエンジン*はソフトウェアに同梱されています。デフォルトを変更するには、SAS Environment Manager を使用して、SAS Compute Server 構成で ENGINE システムオプションを指定します。

エンジンがファイルを処理する仕組み

次の図は、SAS Compute Server のエンジンを介してファイルにアクセスする方法を示しています。この図では、影付きのコンポーネントは外部ファイルタイプまたはアプリケーションを表します。

SAS Cloud Analytic Services (CAS)はこの図には示されていません。*SAS Cloud Analytic Services: 基本*を参照してください。

図 12.1 SAS エンジンがデータにアクセスする方法



1 SAS データセットまたはテーブルは、エンジンと属性に応じて 1 つ以上の物理ファイルに保存されます。適切な SAS エンジンを構成している場合、SAS は、DBMS などの他のアプリケーションによって作成されたデータの読み取りおよび書き込みが可能です。

Base SAS は一部の生データを読み取ることができます。たとえば、DATA ステップまたは IMPORT プロシジャは、テキストファイルからカンマ区切りデータを読み取ることができます。DATA ステップまたはプロシジャは、SAS データセットへの出力のためにデータを V9 エンジンに提供します。14 章, “生データ” (341 ページ)を参照してください。

2 SAS データセット名を指定すると、エンジンは保存されているファイルを検索してメタデータを取得します。V9 エンジンデータセットには、データセットファイル内にメタデータ(ディスクリプター情報とも呼ばれる)が含まれています。他のファイルタイプは、メタデータを別のファイルに保存します。SAS は多くの外部ファイルタイプのメタデータを判別できますが、追加の指示が必要になる場合があります。メタデータは、変数名や属性、ファイルにインデックスや圧縮されたオブザベーションなどの特別な処理特性があるかどうかなどの情報を提供します。

複数のエンジンが処理に関与する可能性があることに注意してください。たとえば、DATA ステップでは、1 つのエンジンを使用してデータを読み取り、別のエンジンを使用してデータを書き込むことができます。

3 エンジンはメタデータを使用して、SAS 処理用の標準論理形式でデータを編成します。この標準形式は *SAS データセットモデル*です。SAS データセットは、変数(列)とオブザベーション(行)に編成されたデータ値で構成されます。

SAS データセットモデルと同様に、*SAS ライブラリモデル*は、処理のために論理形式に編成されたデータセットおよびその他のライブラリメンバーのグループです。ファイルが SAS ライブラリとしてアクセスされる場合、DATASETS プロシジャなどの SAS ユーティリティを使用して、ファイルの内容をリストし、管理できます。

4 SAS プロシジャと DATA ステップステートメントは、この論理形式である SAS データセットモデルでデータを処理します。処理中に、エンジンは物理ファイルの開閉やデータの読み取りと書き込みに必要な命令をすべて渡します。データが SAS データセットとして物理的に保存されていなくても、SAS データセットモデルで処理を実行できます。データが DBMS などの外部アプリケーションに保存されている場合、一部の SAS プロシジャはそのアプリケーションに処理を渡すことができます。

エンジン特性

SAS エンジンの概要

LIBNAME エンジンとデータコネクタのリストについては、*データソースはアルファベット順にリストされています*を参照してください。

表 12.1 一般的に使用される SAS エンジン

LIBNAME エンジンのニックネーム	用途	例とドキュメント
V9 または BASE	出荷されたデフォルトの Base SAS エンジン SAS データセット	“例: LIBNAME ステートメントで V9 エンジンを割り当てる” (277 ページ) “例: カンマ区切りファイルを読み取る” (286 ページ) “デフォルトの Base SAS エンジン(V9 エンジン)” (274 ページ) *SAS V9 LIBNAME エンジン: リファレンス*
CAS	ビッグデータ マルチスレッド分散処理 クラウドコンピューティング	“例: SAS データセットを CAS サーバーにロードする” (284 ページ) “CAS LIBNAME エンジン” (*SAS Cloud Analytic Services: ユーザーガイド*)
SAS/ACCESS エンジン(複数)	他のアプリケーションからの外部データ	“例: SAS/ACCESS エンジンと PROC IMPORT を使用して Microsoft Excel データを読み取る” (288 ページ) “例: SAS/ACCESS エンジンを使用して DBMS にデータを作成する” (289 ページ)

LIBNAME エンジンのニックネーム	用途	例とドキュメント
		“例: PROC SQL ビューに SAS/ACCESS LIBNAME ステートメントを埋め込む” (291 ページ) “例: SQL パススルー機能を使用して DBMS データにアクセスする” (293 ページ) *リレーショナルデータベース用 SAS/ACCESS: リファレンス* *非リレーショナルデータベース用 SAS/ACCESS: リファレンス* *SAS/ACCESS Interface to PC Files: SAS Viya プラットフォームのリファレンス* *SAS/ACCESS Interface to R/3: ユーザーガイド* SAS/ACCESS ドキュメント
Parquet ORC	Base SAS でのオープンソースファイル形式のインポートとエクスポート ビッグデータ クラウドストレージ	“Example: Create a Parquet Table in Google Cloud Storage” (*SAS Viya LIBNAME Engines for ORC and Parquet: Reference*) “Example: Read an ORC Table from ADLS” (*SAS Viya LIBNAME Engines for ORC and Parquet: Reference*) *ORC および Parquet 用の SAS Viya LIBNAME エンジン: リファレンス*
SPDE	代替 Base SAS エンジン SPD Engine のデータセット ビッグデータ マルチ CPU スレッド処理 オプションの Hadoop ストレージ	“例: LIBNAME ステートメントで SPD Engine を割り当てる” (279 ページ) “例: SPD Engine を使用した Hadoop での SAS データの読み取りおよび書き込み” (280 ページ) *SAS Scalable Performance Data Engine: リファレンス* *SAS SPD Engine: Hadoop 分散ファイルシステムでのデータの保存*
CVP	各国語サポート	“例: CVP エンジンと V9 エンジンを使用して切り捨てを回避する” (281 ページ) *SAS 各国語サポート(NLS): リファレンスガイド*
JSON	Base SAS での JSON のインポートとエクスポート	“例: JSON エンジンを使用して JSON データをインポートする” (296 ページ)

LIBNAME エンジンのニックネーム	用途	例とドキュメント
		“LIBNAME ステートメント: JSON エンジン” (*SAS グローバルステートメント: リファレンス*)
XMLV2	Base SAS での XML のインポートとエクスポート	“例: XMLV2 エンジンを使用して XML データをインポートする” (294 ページ)
		SAS XMLV2 および XML LIBNAME エンジン: ユーザーガイド
SASESOCK	SAS/CONNECT TCP/IP パイピング	“LIBNAME Statement: SASESOCK Engine” (*SAS/CONNECT for the SAS Viya Platform: User's Guide*)
SASIOLA と SASHDAT	SAS LASR Analytic Server マルチスレッド分散処理 オプションの Hadoop ストレージ	*SAS LASR Analytic Server: リファレンスガイド*
FEDSVR	SAS Federation Server を使用した外部データまたは SAS データ	*SAS Federation Server の SAS LIBNAME エンジン: ユーザーガイド*
SASEDOC	ODS DOCUMENT 出力オブジェクト	“LIBNAME ステートメント: SASEDOC” (*SAS Output Delivery System: ユーザーガイド*)
JMP	JMP 統計検出ソフトウェア	“LIBNAME Statement: JMP Engine” (*SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform*)

デフォルトの Base SAS エンジン(V9 エンジン)

デフォルトの Base SAS エンジンの重要な概念は次のとおりです。

- 出荷時のデフォルトの Base SAS エンジンは BASE であり、これは V9 エンジンのエイリアスです。
- ほとんどの SAS 処理では、新しいライブラリを作成するときにエンジン名を指定せず、ENGINE システムオプションを指定していない場合は、V9 エンジンが自動的に選択されます。
- ライブラリの場所にすでに SAS ファイルが含まれている場合は、SAS でそれらのファイルに基づいて正しいエンジンを割り当てることもできます。たとえば、場所に V9 データセットのみが含まれている場合、SAS は V9 エンジンを割り当てます。ただし、ライブラリの場所に異なるエンジンファイルが混在している場

合、SAS では必要なエンジンを割り当てられないことがあります。したがって、エンジンを指定することがベストプラクティスです。

“例: LIBNAME ステートメントで V9 エンジンを割り当てる” (277 ページ)を参照してください。デフォルトのエンジンを変更するには、SAS Environment Manager を使用して、SAS Compute Server 構成で ENGINE システムオプションを指定します。

ログあるいは PROC CONTENTS または PROC DATASETS からの出力に予期しないエンジン名が表示された場合、そのエンジンは V9 エンジンから内部的に呼び出された可能性があります。これらの内部エンジンは、ユーザーが指定することはできません。たとえば、DATA ステップビューを作成すると、SAS は SASDSV エンジンを呼び出します。PROC SQL ビューを作成すると、SAS は SQLVIEW を呼び出します。

SAS/CONNECT ソフトウェアを使用すると、SAS は REMOTE エンジンを呼び出します。通常、LIBNAME ステートメントで REMOTE エンジンを指定しないでください。使用するエンジンはクライアントが自動的に決定するためです。

レガシーエンジン

移送エンジン

XPORT エンジンは、文字エンコーディングと数値表現に環境に依存しない標準を使用する移送形式でファイルを作成します。XPORT エンジンによって作成された移送ファイルは、動作環境間で転送できます。移送ファイルは、DATA ステップまたは COPY プロシジャ(または DATASETS プロシジャの COPY ステートメント)で XPORT エンジンを使用することにより、ターゲット環境で読み取ることができます。

XPORT エンジンは、環境間で SAS ファイルを移動するためのベストプラクティスではありません。“Strategies for Moving and Accessing SAS Files” (*Moving and Accessing SAS Files*)を参照してください。

新しい SAS リリースに移行する場合は、“MIGRATE プロシジャ” (*Base SAS プロシジャガイド*)を参照してください。

SPSS エンジン

SPSS エンジンは、外部アプリケーション SPSS で作成されたデータを読み取ります。エンジンは、.por 拡張子を持つ SPSS ポータブルファイル形式を読み取ります。このファイル形式は、SAS データセットの移送形式に似ています。SPSS ポータブルファイル(エクスポートファイルとも呼ばれる)は、SPSS EXPORT コマンドを使用して作成する必要があります。SPSS エンジンは順次エンジンです。

SAS で SPSS ファイルを読み取るには、次のいずれかの方法を選択します。

- .por ファイルがある場合は、SPSS エンジンまたは CONVERT プロシジャを使用して、SPSS から SAS データセットに変換できます。PROC CONVERT は SAS Viya プラットフォームには含まれていません。

- .por ファイルではなく.sav ファイルがある場合は、SAS/ACCESS to PC Files を構成する必要があります。IMPORT プロシジャを使用して、SPSS から SAS データセットに変換します。*SAS/ACCESS Interface to PC Files: SAS Viya プラットフォームのリファレンス*を参照してください。
- 例については、Sample 34629: Coming to SAS from SPSS も参照してください。

アクセスパターン

SAS プロシジャとステートメントは、次の 4 つの一般的なパターンのいずれかで SAS データセット内のオブザベーションを読み取ることができます。

順次アクセス
: ファイルの先頭から開始してファイルの末尾まで順番に継続して、オブザベーションを次々に処理します。たとえば、JSON エンジンと XMLV2 エンジンは順次処理のみを実行します。

ランダムアクセス
: 以前のオブザベーションを処理せずに、なんらかのインジケーター変数の値に従ってオブザベーションを処理します。たとえば、SET ステートメントの POINT=オプションには、オブザベーションへのランダムアクセスと、ファイル内のレコード識別子からオブザベーション番号を計算する機能が必要です。

BY グループアクセス
: BY ステートメントで指定された変数の値の順序でオブザベーションをグループ化して処理します。

複数パス
: SAS ステートメントまたはプロシジャで必要な場合、データに対して 2 つ以上のパスを実行します。

ステートメントまたはプロシジャが、必要なアクセスパターンをサポートしていないエンジンのデータセットにアクセスしようとすると、SAS は適切なエラーメッセージをログに出力します。

ロックのレベル

SAS データセットを複数のセッション、または単一セッション内の複数のステートメントまたはプロシジャによって同時に開くことができる場合、ロックのレベルが重要です。ロックのレベルによって、ファイルに対して同時に読み書きできるセッション、プロシジャ、またはステートメントの数が決まります。一部のエンジンはロックをまったくサポートしていないか、これらのレベルの一部をサポートしていません。詳細については、エンジンのドキュメントを参照してください。

ライブラリレベルのロック
: 同時アクセスがライブラリレベルで制御されることを指定します。このロックにより、ライブラリへの同時アクセスが 1 つの更新プロセスのみに制限されます。

メンバーレベル(データセット)のロック
多くのセッション、ステートメント、またはプロシジャへの読み取りアクセスを有効にします。このロックにより、セッション、ステートメント、またはプロシジャが更新アクセスまたは出力アクセスを取得した場合、SAS データセットへの他のすべてのアクセスが制限されます。

レコードレベル(行)のロック
複数のセッション、ステートメント、またはプロシジャによる SAS データセットへの同時読み取りアクセスと更新アクセスを有効にします。このロックにより、同じオブザベーションへの同時更新アクセスが防止されます。

デフォルトでは、SAS はデータの一貫性を保証しながら、可能な限り最大レベルの同時アクセスを提供します。ロックレベルを制御する方法は次のとおりです。

- 通常、アクセスレベルを自分で制御する必要はありませんが、CNTLLEV=データセットオプションを使用すると、ライブラリ、レコード、またはメンバーレベルでのロックが可能になります。
- LOCK ステートメントは、既存の SAS ファイルに対する排他的ロックを取得、リスト、または解放します。
- SAS/ACCESS などの一部の SAS 製品には、拡張セッション管理サービスとファイルロック機能をサポートするエンジンが含まれています。

例: SAS エンジンを使用して SAS データを処理する

例: LIBNAME ステートメントで V9 エンジンを割り当てる

コード例

このライブラリ割り当ては、V9 エンジンを指定します。

```
libname myfiles v9 'library-path'; /* 1 */
data myfiles.myclass;              /* 2 */
  set sashelp.class;
run;
```

1 LIBNAME ステートメントは、myfiles ライブラリ参照名と V9 エンジンをライブラリの場所に割り当てます。*library-path* をライブラリの場所に置き換えます。この場所はすでに存在しており、SAS Compute Server からアクセスできる必要があります。

2 DATA ステップは、sashelp ライブラリから class データセットをコピーすることにより、myfiles ライブラリに myclass データセットを作成します。

例のコード 12.1 V9 ライブラリ割り当てが成功したことを示す SAS ログ

```
1   libname myfiles v9 'library-path';
NOTE: Libref MYFILES was successfully assigned as follows:
      Engine:       V9
      Physical Name: library-path
2 !
3   data myfiles.myclass;
4     set sashelp.class;
5   run;

NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set MYFILES.MYCLASS has 19 observations and 5 variables.
```

要点

- LIBNAME ステートメントは、プログラムでライブラリを割り当てる一般的な方法です。ライブラリ割り当ては、ライブラリ参照名、エンジン、物理的な場所、エンジンまたは環境に固有のオプションで構成されます。
- SAS® Viya®でライブラリを割り当てるときは、その場所がすでに存在しており、計算サーバーからアクセスできる必要があります。
- 出荷時のデフォルトの Base SAS エンジンは BASE であり、これは V9 エンジンのエイリアスです。
- 新しいライブラリを作成するときにエンジン名を指定せず、ENGINE システムオプションを指定していない場合は、V9 エンジンが自動的に選択されます。
- ライブラリの場所にすでに SAS ファイルが含まれている場合は、SAS でそれらのファイルに基づいて正しいエンジンを割り当てることもできます。たとえば、場所に V9 データセットのみが含まれている場合、SAS は V9 エンジンを割り当てます。ただし、ライブラリの場所に異なるエンジンファイルが混在している場合、SAS では必要なエンジンを割り当てられないことがあります。したがって、エンジンを指定することがベストプラクティスです。

関連項目

- “例: ライブラリ参照名を使用してデータにアクセスする” (238 ページ)
- “ライブラリ割り当ての要素” (230 ページ)
- “デフォルトの Base SAS エンジン(V9 エンジン)” (274 ページ)

例: LIBNAME ステートメントで SPD Engine を割り当てる

コード例

SPD Engine の次の LIBNAME ステートメントは、V9 エンジンの LIBNAME ステートメントと非常に似ています。

```
libname mylib spde 'library-path'       /* 1 */
  datapath=('path-for-data-partitions') /* 2 */
  indexpath=('path-for-indexes');       /* 3 */
```

1 LIBNAME ステートメントのこの部分では、mylib ライブラリ参照名と SPD Engine をプライマリパス名に割り当てます。データセットの最初(通常は唯一)のメタデータファイルは、常にライブラリのプライマリパスに保存されます。

2 オプションで、DATAPATH=オプションで 1 つ以上のパス名を割り当てて、データパーティションを保存できます。それ以外の場合、データパーティションファイルはプライマリパスに保存されます。

3 オプションで、INDEXPATH=オプションで 1 つ以上のパス名を割り当てて、インデックスファイルを保存できます。それ以外の場合、インデックスファイルはプライマリパスに保存されます。

例のコード 12.2 SPD Engine ライブラリ割り当てが成功したことを示す SAS ログ

```
1   libname mylib spde 'library-path'
2     datapath=('path-for-data-partitions')
3     indexpath=('path-for-indexes');
NOTE: Libref MYLIB was successfully assigned as follows:
    Engine:       SPDE
    Physical Name: library-path
3 !
```

要点

- SPD Engine は、代替 Base SAS エンジンです。
- SPD Engine は、非常に大きなテーブルを高速に処理できるように設計されています。このエンジンはスレッドを使用してデータを非常に高速かつ並列に読み取り、複数の CPU で実行されます。このパフォーマンスに貢献しているのは、分散環境を活用できるパーティションファイル形式です。
- SPD Engine はデータセットを複数のファイルに保存しますが、SPD Engine データセットは V9 エンジンデータセットと非常によく似た方法で処理できます。Base

SAS 言語のほとんどは、SPD Engine データセットと非常にうまく連携します。ただし、エンジンは、その処理とストレージの最適化に固有のいくつかの言語要素をサポートしています。V9 エンジン機能との違いについては、ドキュメントを参照してください。

関連項目

- *SAS Scalable Performance Data Engine: リファレンス*
- “エンジン特性” (272 ページ)

例: SPD Engine を使用した Hadoop での SAS データの読み取りおよび書き込み

コード例

次の例では、Base SAS ライブラリを Hadoop クラスターに割り当てます。

```
options set=SAS_HADOOP_CONFIG_PATH='/myconfigpath';       /* 1 */
options set=SAS_HADOOP_JAR_PATH='/myjarpath';

libname mydata spde '/data/abcdef' hdfs=yes accelwhere=yes; /* 2 */
```

1 SET=システムオプションは、Hadoop の環境変数を定義します。これらの環境変数がすでに設定されている場合(たとえば、構成中)、これらのコード行をサブミットしないでください。これらの環境変数が正しく設定されていない場合、LIBNAME ステートメントにより SAS ログにエラーが生成されます。

2 LIBNAME ステートメントは、mydata ライブラリ参照名を SPD Engine と Hadoop クラスター内のディレクトリに割り当てます。HDFS=YES 引数は、Hadoop クラスター構成ファイルで定義されている Hadoop クラスターに接続することを指定します。ACCELWHERE=YES オプションは、Hadoop クラスター内の MapReduce プログラムによってデータのサブセット化が実行されることを要求します。

要点

- SPD Engine は、従来のファイルシステムまたは Hadoop 上で SAS データを読み書きできる代替 Base SAS エンジンです。このエンジンでは、SAS/ACCESS などの追

加の SAS 製品を構成する必要はありませんが、サポートされている Hadoop ディストリビューションを実行している必要があります。

- 顧客は多くの場合、非常に大規模なデータを低コストで保存するために Hadoop を選択します。SPD Engine の分散ストレージと処理は、Hadoop ファイルシステム (HDFS)とうまく連携します。さらに、このエンジンは、Hadoop クラスターに MapReduce プログラムを自動的にサブミットすることで、ほとんどの WHERE 式を最適化できます。
- このエンジンは、Hadoop 上の SPD Engine データセットを読み取ることができます。エンジンを使用してデータセットを Hadoop に保存すると、ほとんどの Base SAS 言語を使用してデータを処理できるようになります。ただし、このエンジンは、Hadoop での処理とストレージに固有のいくつかの言語要素をサポートしています。

関連項目

- *SAS SPD Engine: Hadoop 分散ファイルシステムでのデータの保存*
- *Base SAS および SAS/ACCESS 用 SAS Hadoop 構成ガイド*

例: CVP エンジンと V9 エンジンを使用して切り捨てを回避する

コード例

この例を実行するには、まず、“例: LIBNAME ステートメントで V9 エンジンを割り当てる” (277 ページ)にあるように、myclass という名前のデータセットを作成します。PROC CONTENTS を実行して変数の長さを確認します。

```
libname myfiles v9 'library-path-1';
proc contents data=myfiles.myclass;
run;
```

PROC CONTENTS 出力では、2 つの文字変数に注目してください。Name の長さは 8、Sex の長さは 1 です。

アウトプット 12.1　拡張前の変数長を示す PROC CONTENTS

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
3	Age	Num	8
4	Height	Num	8
1	Name	Char	8
2	Sex	Char	1
5	Weight	Num	8

次の例では、CVP エンジンと V9 エンジンを使用して文字変数のサイズを拡張しています。CVP エンジンは、文字を表現するためにより多くのバイトを使用するエンコーディングにデータセットをコピーする場合に、切り捨てを回避するのに役立ちます。

```
libname srclib cvp 'library-path-1' cvpengine=v9 cvpmult=2.5; /* 1 */
libname target v9 'library-path-2';                          /* 2 */
proc copy in=srclib out=target;                              /* 3 */
  select myclass;
run;

proc contents data=target.myclass;                           /* 4 */
run;
```

1 この LIBNAME ステートメントは、srclib ライブラリを CVP エンジンとコピーするデータの場所に割り当てます。CVPENGINE=オプションは、データを処理する基になるエンジンとして V9 エンジンを指定します。CVPMULT=オプションは、すべての文字変数を拡張するための乗算係数 2.5 を指定します。このオプションが指定されていない場合、CVP エンジンは自動的に乗数値を選択します。

2 この LIBNAME ステートメントは、コピーされたデータを含む target ライブラリを割り当てます。

3 SELECT ステートメントを使用した COPY プロシジャは、myclass データセットを target ライブラリにコピーします。コピー中に、CVP エンジンは文字変数の長さを 2.5 倍に拡張します。

4 CONTENTS プロシジャは、文字変数の長さが 2.5 倍になったことを示します。

Name の場合、8 × 2.5 = 20。

Sex の場合、1 × 2.5 = 2.5 となり、整数に切り上げると 3 になります。

アウトプット 12.2　拡張後の変数長を示す PROC CONTENTS

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
3	Age	Num	8
4	Height	Num	8
1	Name	Char	20
2	Sex	Char	3
5	Weight	Num	8

要点

- 文字を表現するためにより多くのバイトを使用するエンコーディングにデータセットをコピーする場合、列の長さがより大きな文字サイズに対応できないと切り捨てが発生する可能性があります。たとえば、文字は wlatin1 エンコーディングでは 1 バイトとして表現されますが、UTF-8 では 2 バイトとして表現される場合があります。
- ログ内のエラーに character data was lost during transcoding と記載されている場合は、通常、切り捨てが発生したことを示しています。CVP エンジンを使用して文字変数の長さを拡張することで、エラーをトラブルシューティングできます。
- デフォルトでは、CVP エンジンは自動的に乗数値を選択します。通常、自動値は切り捨てを回避するのに十分です。CVPMULTIPLIER=を使用して自分で指定することもできます。
- デフォルトの CVPFORMATWIDTH=YES オプションは、出力形式の長さを拡張しますが、ユーザー定義の出力形式には影響しません。ユーザー定義の出力形式については、“Example: Avoid Truncation in Formats When Using PROC FORMAT” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。
- データセットを別の動作環境または別の文字エンコーディングにコピーする場合は、PROC COPY に NOCLONE などのオプションを指定することをお勧めします。NOCLONE は、ライブラリがターゲット環境で互換性があるように、データ表現や文字エンコーディングなどの特定のデータセット属性をコピーしません。

  NOCLONE オプションのかわりに、OVERRIDE=オプションを使用して ENCODING=および OUTREP=オプションを指定できます。この方法では、ソースライブラリの他のデータセット属性が保持されるため、それらの属性が必要であることを確認してください。
- PROC COPY は監査証跡を保持しません。CEDA 処理では、PROC COPY はインデックスや一貫性制約をコピーしません。移行する場合は、PROC MIGRATE の方がよい選択となる可能性があります。“Example: Avoid Truncation When Migrating a SAS Library by Using the CVP Engine” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。

関連項目

- *SAS 各国語サポート(NLS): リファレンスガイド*
- “Compatibility and Migration” (*SAS V9 LIBNAME Engine: Reference*)
- “COPY プロシジャ” (*Base SAS プロシジャガイド*)

例: SAS データセットを CAS サーバーにロードする

コード例

次の例では、DATA ステップを使用して、SAS データセットを SAS Cloud Analytic Services (CAS)テーブルとしてメモリにロードします。

```
cas casauto host="cloud.example.com" port=5570; /* 1 */

libname mycas cas;                  /* 2 */
data mycas.cars (promote=yes);          /* 3 */
  set sashelp.cars;
run;
proc contents data=mycas.cars;          /* 4 */
run;
```

1 CAS ステートメントは CAS セッションを開始し、CAS セッション名として casauto を指定します。HOST=および PORT=オプションで接続情報を使用します。

2 LIBNAME ステートメントは、mycas ライブラリ参照名を CAS エンジンに割り当てます。SESSREF= LIBNAME オプションが指定されていないため、エンジンは casauto セッションを使用します。

3 DATA ステップは、SAS データセット sashelp.cars を CAS セッションにコピーします。PROMOTE=YES データセットオプションは、テーブルをグローバルスコープでプロモートします。

4 PROC CONTENTS は、mycas.cars テーブルがセッション中に CAS サーバー上で使用できることを示します。データがメモリにロードされた後、後続のステップでメモリ内のデータを処理できます。ロードと処理は別のステップで行われます。

アウトプット 12.3　mycas.cars の PROC CONTENTS 出力の一部

Data Set Name	MYCAS.CARS	Observations	428
Member Type	DATA	Variables	15
Engine	CAS	Indexes	0
Created	05/17/2019 17:52:37	Observation Length	160
Last Modified	05/17/2019 17:52:37	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB
Caslib	CASUSERHDFS(userid)
Scope	Session

要点

- CAS エンジンを使用する LIBNAME ステートメントをサブミットして、SAS Compute Server セッションを CAS セッションに接続できます。CAS サーバーと既存の CAS セッションにアクセスできる必要があります。
- DATA ステップを備えた CAS LIBNAME エンジンは、SAS データをインメモリテーブルとして CAS サーバーにロードする 1 つの方法です。大きなテーブルの場合は、他の方法の方が効率的である可能性があります。
- データを CAS サーバーにロードした後、SAS ライブラリ参照名とテーブル名を参照することで、SAS セッションから SAS プロシジャまたは DATA ステップを実行できます。テーブルをメモリにロードするために使用するのと同じ DATA ステップでは、メモリ内のテーブルを処理しません。ロードと処理は別のステップで行われます。
- テーブルは、caslib にロードされるときに自動的に保存されません。CASUTIL プロシジャを使用してテーブルを保存できます。ネイティブ CAS テーブルのファイル拡張子は.sashdat です。

関連項目

- “LIBNAME ステートメント: CAS エンジン” (*SAS Cloud Analytic Services: ユーザーガイド*)
- “DATA ステップと CAS” (*SAS Cloud Analytic Services: DATA ステッププログラミング*)
- “Example: Load SAS Data to CAS by Using the CASUTIL Procedure” (*SAS V9 LIBNAME Engine: Reference*)
- *SAS Viya プラットフォームプログラミング: 入門ガイド*

例: SAS エンジンを使用して外部データを処理する

例: カンマ区切りファイルを読み取る

コード例

この例では、IMPORT プロシジャは、カンマ区切り値を含むファイルをインポートします。

```
filename chol temp;              /* 1 */
proc http                     /* 2 */
   url="http://support.sas.com/documentation/onlinedoc/viya/exampledatasets/
cholesterol.csv"
   out=chol;
quit;
proc import datafile=chol        /* 3 */
   out=work.mycholesterol
   dbms=csv
   replace;
run;
proc print data=work.mycholesterol; /* 4 */
run;
```

1 FILENAME ステートメントは、chol ファイル参照名を割り当てます。TEMP オプションは、ファイルが一時的なものであることを指定するため、パス名は必要ありません。

2 HTTP プロシジャは、cholesterol.csv 入力ファイルの URL を指定します。データは chol ファイル参照名に書き込まれます。

3 PROC IMPORT は、カンマ区切りデータを読み取り、mycholesterol データセットを作成します。

4 PRINT プロシジャはデータセットを出力します。出力には、ファイルが列名も含めて正しくインポートされたことが示されます。

アウトプット 12.4　work.mycholesterol の PROC PRINT

patnr	measurement	cholesterol
A2	1	150
A2	2	145
A2	3	148
B1	1	301
B1	2	280
B1	3	275
C1	1	212
C1	2	220
C1	3	240

要点

- Base SAS ソフトウェアは、SAS/ACCESS エンジンを使用せずに、一部の外部テキストファイルをインポートおよびエクスポートできます。これらのファイルは通常、生データまたは区切りデータと呼ばれます。DATA ステップまたは PROC IMPORT は、テキストファイルからデータを読み取り、そのデータを V9 エンジンに提供して SAS データセットに出力できます。
- Base SAS で Microsoft Excel ファイルの読み取りまたは書き込みを行う場合は、ファイルの拡張子が.csv である必要があります。Excel ファイル拡張子が.xls または.xlsx のファイルをインポートするには、SAS/ACCESS Interface to PC Files を構成しておく必要があります。ただし、Excel を使用して.xls または.xlsx ファイルを.csv ファイルとして保存すると、Base SAS はそれを読み取ることができます。
- Excel ファイルから SAS データセットを作成した場合、データは静的であり、基になる Excel データを反映するように変更されません。SAS からクエリできる継続的なデータストアとしてデータを Excel に保持したい場合は、XLSX エンジンを使用します。SAS/ACCESS Interface to PC Files を構成しておく必要があります。

関連項目

- “デフォルトの Base SAS エンジン(V9 エンジン)” (274 ページ)
- 14 章, “生データ” (341 ページ)
- “IMPORT プロシジャ” (*Base SAS プロシジャガイド*)

例: SAS/ACCESS エンジンと PROC IMPORT を使用して Microsoft Excel データを読み取る

コード例

この例では、XLSX エンジンと PROC IMPORT を使用して Excel データをインポートします。この例のデータを作成するには、Microsoft Excel を起動し、“例: カンマ区切りファイルを読み取る” (286 ページ)で使用される cholesterol.csv ファイルを開きます。(cholesterol.csv ファイルは、http://support.sas.com/documentation/onlinedoc/viya/exampledatasets/cholesterol.csv からダウンロードできます。)Excel では、.xlsx 拡張子を付けてファイルを保存します。次のコードは、cholesterol.xlsx を SAS データセット mycholesterol としてインポートします。

```
options validvarname=v7;                    /* 1 */
proc import datafile='file-path/cholesterol.xlsx' /* 2 */
   dbms=xlsx                          /* 3 */
   out=work.mycholesterol                   /* 4 */
   replace;                           /* 5 */
run;
```

1 VALIDVARNAME=V7 ステートメントは、SAS が列名を変数名に変換するときに、スペースを強制的にアンダースコアに変換します。

2 DATAFILE=オプションは、入力ファイルのパスを指定します。

3 DBMS=オプションは、インポートするデータのタイプを指定します。Excel ワークブックをインポートする場合は、DBMS=XLSX を指定します。PROC IMPORT は XLSX エンジンを呼び出すため、LIBNAME ステートメントをサブミットする必要はありません。

4 OUT=オプションは、出力 SAS データセットを識別します。

5 REPLACE オプションは、既存の SAS データセットを上書きします。

要点

- XLSX エンジンを使用すると、Excel の.xlsx ファイルからデータを直接読み取ることができます。SAS/ACCESS Interface to PC Files を構成しておく必要があります。XLSX エンジンは、Microsoft Excel 2007、2010、およびそれ以降のファイルへの接続をサポートします。
- Excel の命名規則は SAS とは異なります。Excel データを読み取るためのベストプラクティスは、VALIDVARNAME=V7 を設定することです。このオプションはスペースをアンダースコアに変換し、32 文字を超える名前を切り捨てます。
- 日付値は、SAS 日付数値に自動的に変換されます。

関連項目

- "LIBNAME Statement: XLSX Engine" (*SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform*)
- "IMPORT プロシジャ" (*Base SAS プロシジャガイド*)
- "VALIDVARNAME= システムオプション" (*SAS システムオプション: リファレンス*)

例: SAS/ACCESS エンジンを使用して DBMS にデータを作成する

コード例

この例では、SAS DATA ステップによって Teradata テーブルが作成されます。この例を実行するには、SAS/ACCESS 用に Teradata インターフェイスを構成する必要があります。

```
libname mytddata teradata server=mytera user=myid password=mypw; /* 1 */
data mytddata.grades;                                            /* 2 */
  input student $ test1 test2 final;
  datalines;
Fred 66 80 70
Wilma 97 91 98
;
proc datasets library=mytddata;                                  /* 3 */
```

```
run;
quit;
```

1 LIBNAME ステートメントは、mytddata ライブラリ参照名と、SAS/ACCESS Interface to Teradata のエンジンニックネームである TERADATA を指定します。このステートメントでは、Teradata の接続オプションも指定します。これらのオプションを変更して、SAS/ACCESS 接続値および必要なその他のオプションを指定します。

2 DATA ステップでは、grades という名前のテーブルを作成します。テーブルは Teradata DBMS 内にあり、SAS データセットではありません。

3 mytddata ライブラリの PROC DATASETS 出力は、エンジンが Teradata であることを示しています。grades テーブルの場合、SAS メンバータイプは DATA、DBMS メンバータイプは TABLE です。

アウトプット 12.5　DBMS ディレクトリ情報を示す PROC DATASETS 出力

Directory	
Libref	MYTDDATA
Engine	TERADATA
Physical Name	mytera
Schema/User	myid

アウトプット 12.6　DBMS コンテンツのライブラリコンテンツを示す PROC DATASETS 出力

#	Name	Member Type	DBMS Member Type
1	grades	DATA	TABLE

要点

- DBMS データ用に SAS/ACCESS インターフェイスを構成すると、SAS で LIBNAME ステートメントをサブミットしてテーブルにアクセスできます。SAS は、SAS データセットにアクセスする場合と同様の方法で DBMS テーブルにアクセスできます。ただし、既存の DBMS テーブルを置き換えることはできないことに注意してください。SAS/ACCESS インターフェイスは、REPLACE= SAS データセットオプションをサポートしません。
- ほとんどの SAS/ACCESS エンジンは、Base SAS 言語を完全にサポートしています。カタログなどのいくつかの Base SAS 機能は V9 エンジンに固有であり、SAS/ACCESS エンジンでは使用できません。SAS/ACCESS エンジンは、データソースに固有のいくつかの言語要素をサポートします。詳細については、インターフェイスのドキュメントを参照してください。

- 一部の SAS/ACCESS インターフェイスでは大文字と小文字が区別されます。DBMS の要件に準拠するために、テーブル名または列名の大文字と小文字を変更する必要がある場合があります。

関連項目

- SAS/ACCESS ドキュメント
- *データソースはアルファベット順にリストされています*
- "DATA ステートメント" (*SAS DATA ステップステートメント: リファレンス*)

例: PROC SQL ビューに SAS/ACCESS LIBNAME ステートメントを埋め込む

コード例

次の例では、SAS/ACCESS LIBNAME ステートメントを PROC SQL ビューに埋め込みます。DATA ステップビュー(LIBNAME ステートメントを埋め込まない)を作成するには、"例: DBMS データにアクセスして SAS ビューを作成する" (246 ページ)を参照してください。

```
libname viewlib v9 'library-path';          /* 1 */
proc sql;
  create view viewlib.mygrades as           /* 2 */
    select *
      from mytddata.grades                  /* 3 */
      using libname mytddata teradata
                server=mytera
                user=myid password=mypw; /* 4 */
quit;
proc print data=viewlib.mygrades noobs;     /* 5 */
run;
```

1 V9 エンジンの LIBNAME ステートメントは、SAS ビューが保存される場所に viewlib ライブラリ参照名を割り当てます。

2 SQL プロシジャの CREATE VIEW ステートメントは、viewlib ライブラリに mygrades ビューを作成します。

3 mytddata.grades テーブルはビューで参照されます。

4 USING 引数には、SAS/ACCESS Interface to Teradata エンジンの LIBNAME ステートメントが埋め込まれます。

5 PRINT プロシジャは、埋め込み LIBNAME ステートメントを使用して mytddata.grades テーブルを参照する viewlib.mygrades ビューを実行します。

アウトプット 12.7　viewlib.mygrades の PROC PRINT

student	test1	test2	final
Wilma	97	91	98
Fred	66	80	70

要点

- ほとんどの SAS/ACCESS エンジンには、いくつかの接続オプションが必要です。LIBNAME ステートメントを埋め込むと、DBMS 接続情報をビューに保存して使いやすくすることができます。ただし、接続情報が変更されることが予想される場合 (たとえば、別のユーザー ID)、この方法はお勧めできません。
- LIBNAME ステートメントを PROC SQL ビューに埋め込むには、CREATE VIEW ステートメントの USING 句を指定します。LIBNAME ステートメントは、ほとんどの SAS/ACCESS エンジン、V9 エンジン、およびその他のエンジンに埋め込むことができます。
- PROC SQL が SAS ビューを実行するとき、SELECT ステートメントはライブラリ参照名を割り当て、DBMS への接続を確立します。ライブラリ参照名のスコープは SAS ビューに対してローカルであり、セッション内に存在する可能性のある同じ名前のライブラリ参照名と競合しません。クエリが終了すると、接続は終了し、ライブラリ参照名の割り当てが解除されます。
- DBMS テーブルの V9 エンジンビューを作成すると、データは DBMS 内に残ります。いつでもビューに対してプロシジャまたは DATA ステップを実行して、基になるデータソースから最新のデータを取得できます。

  ただし、複数のステップでデータを一定に保つ必要がある場合は、データを 1 回読み取って SAS データセットに保存することをお勧めします。この方法を実践すると、パフォーマンスにもメリットが得られます。通常、SAS データセットを複数回読み取る方が、DBMS テーブルを複数回読み取るよりも効率的です。

関連項目

- “LIBNAME Statement: External Databases” (*SAS/ACCESS for Relational Databases: Reference*)
- “CREATE VIEW ステートメント” (*SAS SQL プロシジャユーザーガイド*)
- “SAS Views of DBMS Data” (*SAS/ACCESS for Relational Databases: Reference*)
- “SAS Views” (*SAS V9 LIBNAME Engine: Reference*)

例: SQL パススルー機能を使用して DBMS データにアクセスする

コード例

この PROC SQL の例では、SQL パススルー機能を使用してクエリを Teradata テーブルに送信します。

```
proc sql;
  connect to teradata as myconn (server=mytera
    user=myid password=mypw);              /* 1 */
  select *
    from connection to myconn              /* 2 */
      (select *
        from grades
        where final gt 90);
  disconnect from myconn;                  /* 3 */
quit;
```

1 CONNECT ステートメントは DBMS への接続を確立します。

2 PROC SQL SELECT ステートメントの FROM 句の CONNECTION TO コンポーネントは、DBMS から直接データを取得します。DBMS 固有のクエリはかっこで囲まれています。

3 DISCONNECT ステートメントは、DBMS への接続を終了します。

アウトプット 12.8　PROC SQL パススルー機能の出力

student	test1	test2	final
Wilma	97	91	98

要点

- SQL パススルー機能は、SAS SQL 構文のかわりに DBMS 固有の SQL 構文を使用できるようにする PROC SQL の拡張機能です。
- SQL パススルー機能ステートメントを PROC SQL クエリで使用したり、PROC SQL ビューに保存したりできます。
- パススルー機能は、3 つのステートメントと 1 つのコンポーネントで構成されます。
  - CONNECT ステートメントは DBMS への接続を確立します。
  - EXECUTE ステートメントは、動的で非クエリ DBMS 固有の SQL ステートメントを DBMS に送信します。

- PROC SQL SELECT ステートメントの FROM 句の CONNECTION TO コンポーネントは、DBMS から直接データを取得します。
- DISCONNECT ステートメントは、DBMS への接続を終了します。

関連項目

- “SQL Pass-Through Facility” (*SAS/ACCESS for Relational Databases: Reference*)
- “SAS Views of DBMS Data” (*SAS/ACCESS for Relational Databases: Reference*)
- “SAS Views” (*SAS V9 LIBNAME Engine: Reference*)

例: XMLV2 エンジンを使用して XML データをインポートする

前提条件

XMLV2 コード例を実行するには、まず、次の XML データを含む nhl.xml という名前のファイルを作成します。このファイルは、セッションからアクセスできる場所に保存します。

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<NHL>
 <CONFERENCE> Eastern
  <DIVISION> Southeast
   <TEAM name="Thrashers"  abbrev="ATL" />
   <TEAM name="Hurricanes" abbrev="CAR" />
   <TEAM name="Panthers"   abbrev="FLA" />
   <TEAM name="Lightning"  abbrev="TB" />
   <TEAM name="Capitals"   abbrev="WSH" />
  </DIVISION>
 </CONFERENCE>

 <CONFERENCE> Western
  <DIVISION> Pacific
   <TEAM name="Stars"   abbrev="DAL" />
   <TEAM name="Kings"   abbrev="LA" />
   <TEAM name="Ducks"   abbrev="ANA" />
   <TEAM name="Coyotes" abbrev="PHX" />
   <TEAM name="Sharks"  abbrev="SJ" />
  </DIVISION>
 </CONFERENCE>
</NHL>
```

コード例

次の例では、XMLV2 エンジンを使用して XML データをインポートします。永久 SAS データセットを作成しなくても、XML データを SAS データセットとしてメモリ内で処理できます。

```
filename nhl 'file-path/nhl.xml';          /* 1 */
filename map 'file-path/nhlgenerate.map';  /* 2 */
libname nhl xmlv2 automap=replace xmlmap=map; /* 3 */
proc print data=nhl.team noobs;            /* 4 */
  var TEAM_name TEAM_abbrev;
run;
```

1 最初の FILENAME ステートメントは、ファイル参照名 nhl を、インポートされる XML ドキュメント nhl.xml の場所に割り当てます。

2 2 番目の FILENAME ステートメントは、ファイル参照名 map を、SAS によって生成される XMLMap nhlgenerate.map を保存する場所に割り当てます。

3 LIBNAME ステートメントは、nhl ライブラリを XMLV2 エンジンに割り当てます。AUTOMAP=REPLACE オプションは、XMLMap を自動的に生成し、XMLMap がすでに存在する場合はそれを置き換えるように指定します。XMLMAP=オプションは、XMLMap のファイル参照名を指定します。

4 PROC PRINT は出力を生成し、インポートが成功したことを確認します。

アウトプット 12.9　nhl.xml からインポートされたデータセットの PROC PRINT

TEAM_name	TEAM_abbrev
Thrashers	ATL
Hurricanes	CAR
Panthers	FLA
Lightning	TB
Capitals	WSH
Stars	DAL
Kings	LA
Ducks	ANA
Coyotes	PHX
Sharks	SJ

要点

- Base SAS XMLV2 エンジンは、XML データを読み取る(インポートする)ための順次アクセスを提供します。このエンジンは、SAS データセットを XML データとして書き込む(エクスポートする)こともできます。
- エンジンはメモリ内に一時データセットを作成します。永久データセットを作成するには、DATA ステップまたは PROC COPY で一時データセットを参照します。
- 1 つの XML ファイルに複数の関連データセットを含めることができます。XML ドキュメントを複数の SAS データセットとしてインポートする例については、ドキュメントを参照してください。
- エンジンは、XML ファイル内のデータの XMLMap を自動的に作成できます。マップを保存するには、LIBNAME ステートメントで MAP=オプションと AUTOMAP=REPLACE を指定します。その後、テキストエディターでマップを 開いてカスタマイズできます。マップを編集した後、カスタマイズが上書きされないように、MAP=オプションと AUTOMAP=REUSE を指定します。
- グラフィカルインターフェイスである SAS XML Mapper を使用して、XMLMap を生成およびカスタマイズすることもできます。

関連項目

- *SAS XMLV2 および XML LIBNAME エンジン: ユーザーガイド*
- “FILENAME ステートメント” (*SAS グローバルステートメント: リファレンス*)

例: JSON エンジンを使用して JSON データをインポートする

コード例

次の例では、SAS JSON エンジンを使用して JSON データを読み取ります。この例では、分析用に一時的なインメモリ SAS データセットを作成します。

この例を実行するには、まず“LIBNAME ステートメント: JSON エンジン” (*SAS グローバルステートメント: リファレンス*)から example.json の JSON コードをコピーします。計算サーバーセッションからアクセスできる場所にファイルを作成します。

```
libname mydata json '/file-path/example.json';
proc datasets lib=mydata;
run;
quit;
```

次は PROC DATASETS の出力で、example.json からインポートされた SAS データセットを示しています。

アウトプット 12.10 JSON からインポートされた SAS データセットの PROC DATASETS 出力

Directory	
Libref	MYDATA
Engine	JSON
Access	READONLY
Physical Name	/file-path/example.json

#	Name	Member Type
1	ALLDATA	DATA
2	INFO	DATA
3	INFO_PHONE	DATA
4	ROOT	DATA

要点

- Base SAS JSON エンジンは、JSON データへの読み取り専用の順次アクセスを提供します。
- エンジンはメモリ内に一時データセットを作成します。永久データセットを作成するには、DATA ステップまたは PROC COPY で一時データセットを参照します。
- 1 つの JSON ファイルに複数の関連データセットを含めることができます。インポートされたデータセットを単一のデータセットにマージする例については、ドキュメントを参照してください。
- エンジンは、JSON ファイル内のデータのマップを自動的に作成します。マップを保存するには、LIBNAME ステートメントで MAP=オプションと AUTOMAP=CREATE を指定します。その後、テキストエディターでマップを 開いてカスタマイズできます。マップを編集した後、カスタマイズが上書きされないように、MAP=オプションと AUTOMAP=REUSE を指定します。
- SAS データセットから JSON ファイルをエクスポートするには、JSON プロシジャを使用します。

関連項目

- “LIBNAME ステートメント: JSON エンジン” (*SAS グローバルステートメント: リファレンス*)
- “JSON プロシジャ” (*Base SAS プロシジャガイド*)

13

SAS データセット

SAS データセットの定義

SAS データセットの定義

SAS データセットは、内容が SAS ファイル形式である表形式のデータセットです。SAS データセットには、データとデータに関する情報(メタデータ)が含まれています。

SAS データビューの詳細については、“SAS ビューの定義” (399 ページ)を参照してください。

SAS データセットの各部分

V9 エンジンを使用して SAS データセットを作成すると、データ値とメタデータが単一のファイルに保存されます。SAS Scalable Performance Data Engine を使用して SAS データセットを作成すると、データ値とメタデータは複数の個別のファイルに保存されます。メタデータはディスクリプター情報とも呼ばれます。エンジンに基づいてデータがどのように保存されるかに関する具体的な情報については、次のドキュメントを参照してください。

- SAS V9 エンジンについては、SAS データセットの各部分を参照してください。
- SPD Engine については、“Differences between the Default Base SAS Engine Data Sets and the SPD Engine Data Sets” (*SAS Scalable Performance Data Engine: Reference*)および“Comparing the Default Base SAS Engine and the SPD Engine” (*SAS Scalable Performance Data Engine: Reference*)を参照してください。
- “データ” (*SAS Cloud Analytic Servcies: 基本*)。

SAS データセットに関するディスクリプター情報(メタデータ)を表示するには、CONTENTS プロシジャまたは DATASETS プロシジャ内の CONTENTS ステートメントを使用します。

```
proc contents data=sashelp.air;
run;
```

データセットが並べ替えられると、追加の並べ替え情報がデータセットのメタデータに追加されます。SAS データセットの並べ替えの詳細については、BY グループ処理 (413 ページ)を参照してください。

SAS データセットの管理

SAS データセットの読み取りと作成の方法

次の表には、SAS データセットを管理するための一般的なタスクがいくつか含まれています。この表には、SAS データセットを操作および管理できる方法の小さなサンプリングが含まれています。SAS エンジンの詳細については、12 章, “SAS エンジン” (269 ページ)を参照してください。

表 13.1　データセットを管理する方法

タスク	詳細情報
“例: 単一の SAS データセットの読み取り” (310 ページ)	V9 エンジン LIBNAME ステートメント
SAS データセットのコピー	DATASETS プロシジャ
SAS データセットに関する情報の取得	

データセットの管理に使用する方法の多くは、SAS ライブラリ (262 ページ)の管理に使用する方法と同じです。

SAS データセット内の変数を管理する方法

デフォルトでは、SAS は入力データセットのすべての変数とすべての行を出力データセットに書き込みます。次の言語要素のいずれかを使用して、SAS が読み書きする変数と行を制御できます。

表 13.2　変数と行の読み書きを制御する方法

カテゴリ	タスク	詳細情報
変数(列)の制御	出力から変数を削除する 入力から変数を削除する	DROP ステートメントと DROP= データセットオプション

カテゴリ	タスク	詳細情報
	変数を出力データセットに保持する 変数を入力データセットに保持する	KEEP ステートメントと KEEP=データセットオプション
	変数の名前変更 出力データセット出力内の変数の名前を変更する	RENAME ステートメントと RENAME=データセットオプション
行の制御	WHERE ステートメントを使用して条件付きで行を選択する IF ステートメントを使用して条件付きで行を選択する	WHERE ステートメント WHERE=データセットオプション サブセット化 IF ステートメント
	条件に基づいて行を削除する	DELETE ステートメント
	現在の行を出力データセットに書き込む 指定された条件が真の場合に現在の行を書き込む 1 行の入力から複数の行を作成する 複数行の入力から 1 行を作成する	OUTPUT ステートメント
	行への直接アクセス	POINT=ステートメントオプション
行番号に基づいて行を変更する		KEY=ステートメントオプション
データセットの特定行を最初に処理する		FIRSTOBS=データセットオプション(V9 エンジンのみ)
行の処理を停止するタイミングを指定する		OBS=データセットオプション(V9 エンジンのみ)

SAS データセット内の行を読み取る方法

データセット内の行には、順次または直接アクセスできます。

Sequential access
: 物理ファイルに出現する順序で行を順次読み取ります。SET、MERGE、UPDATE、MODIFY ステートメントは、デフォルトで行を順次読み取ります。SAS 関数の OPEN、FETCH、および FETCHOBS も行を順次読み取ります。

Direct access

by row number
: DATA ステップで、行番号によって行に直接アクセスするには、SET または MODIFY ステートメントで POINT=オプションを使用します。POINT=オプションは、SET または MODIFY ステートメントが読み取る行を現在の値によって決定する一時変数を指定します。

  次のように、直接アクセス方法を使用して、あるデータセットの行をサブセット化し、別のデータセットの行と結合できます。

```
data south;
  set revenue;
  if region=4;
  set expense point=_n_;
run;
```

by index
: 1 つ以上の指定された変数の値に基づく行に直接アクセスするには、まず変数のインデックスを作成する必要があります。インデックスは、キー変数のデータ値を含む別の構造であり、値を含む行の位置識別子と組み合わせられています。

  インデックスが作成されたら、KEY=オプションを SET または MODIFY ステートメントで指定した DATA ステップを使用して、インデックス付き変数の値に基づいて行に直接アクセスできます。

  たとえば、1 つのデータセット内の情報を 2 番目のデータセット内の特定の値と照合する必要があるとします。2 番目のデータセットに適切にインデックスが作成されている場合は、SET ステートメントで KEY=オプションを使用してインデックス付きテーブルに対して"テーブルルックアップ"を実行し、それらの行のみを最初のデータセットと結合できます。

```
data combine;
  set invtory(keep=partno instock price);
  set partcode(keep=partno desc)  key=partno;
run;
```

  別の例については、*"インデックスにより検索されるオブザベーションの変更" (SAS DATA ステップステートメント: リファレンス)*を参照してください。

  インデックスを作成するためのいくつかの方法を次に示します。

表 13.3　インデックスを作成する方法

使用される言語要素	例
INDEX=データセットオプション	INDEX=データセットオプションを使用してインデックスを作成する
INDEX CREATE ステートメント(DATASETS プロシジャ)	“SAS データセットの変更” (*Base SAS プロシジャガイド*)
CREATE INDEX ステートメント(SQL プロシジャ)	“インデックスの作成” (*SAS SQL プロシジャユーザーガイド*)

インデックスは、Base SAS V9 Engine を使用して作成された SAS データセットに作成できます。SAS データセットでのインデックスの作成の詳細については、“Indexes” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。

SAS データセットを CAS サーバーにロードする方法

このセクションでは、SAS データセットを CAS サーバーにロードする 3 つの方法を説明します。このセクションでは、mycas は mysess CAS セッションに接続されている caslib の名前です。

- **次のように、単一の DATA ステップを使用して、CAS サーバー上にデータテーブルを作成できます。**

  ```
  data mycas.Sample;
  input y x @@;
  datalines;
  .46 1 .47 2 .57 3 .61 4 .62 5 .68 6 .69 7 ;
  ```

  DATA ステップ操作は、SAS クライアントではなく CAS サーバーで実行すると、意図したとおりに機能しない可能性があることに注意してください。

- **次のように、最初に SAS データセットを作成し、そこに必要な内容が含まれている場合は別の DATA ステップを使用して CAS サーバーにロードすることができます。**

  ```
  data Sample;
  input y x @@;
  datalines;
  .46 1 .47 2 .57 3 .61 4 .62 5 .68 6 .69 7 .78 8 ;

  data mycas.Sample;
  set Sample;
  run;
  ```

- **CASUTIL プロシジャは次のように使用できます。**

```
proc casutil sessref=mysess;
load data=Sample casout="Sample";
quit;
```

CASUTIL プロシジャは、DATA ステップよりも効率的にデータを CAS サーバーにロードできます。CASUTIL プロシジャの詳細については、“CASUTIL プロシジャ” (*SAS Cloud Analytic Services: ユーザーガイド*)を参照してください。 *SAS Cloud Analytic Services: ユーザーガイド*.

mycas caslib には、多数のマシンノードに分散できる Sample データテーブルが保存されます。SAS クライアントマシンが CAS セッションと通信できるようにするには、プロシジャで caslib 参照を使用する必要があります。たとえば、次の FACTMAC プロシジャステートメントでは、mycas caslib に存在するデータテーブルを使用します。

```
proc factmac data = mycas.Sample;
 ...statements...;
run;
```

次のように DELETE プロシジャを使用すると、データテーブルを削除できます。

```
proc delete data = mycas.Sample;
run;
```

これらの例では、Sample データテーブルは mysess セッションでのみアクセスできます。mysess セッションを終了すると、CAS サーバーから Sample データテーブルにアクセスできなくなります。Sample データテーブルを他の CAS セッションで使用できるようにするには、データテーブルをプロモートする必要があります。CASUTIL プロシジャを使用してデータテーブルをプロモートする方法を示す例については、“テーブルのプロモート” (*SAS Cloud Analytic Services: ユーザーガイド*)を *SAS® Cloud Analytic Services: ユーザーガイド*で参照してください。

SAS データセットの命名規則

- “データセット名” (49 ページ)
- SAS データセットの命名規則の概要 (54 ページ)
- “特殊データセット名” (52 ページ)
- “例: 特殊文字を含む SAS データセット名の作成” (57 ページ)および“例: 2 レベルデータセット名の作成” (63 ページ)

データセット名の前に、データセットが関連付けられているライブラリを付けることができます(Sashelp.cars データセットなど)。これを 2 レベル名と呼びます。SAS ライブラリおよび SAS データセット名は、“ほとんどの SAS 名の規則” (36 ページ)に従う必要があります。_NULL_データセットや_LAST_データセットなどの特殊データセットもあります。これらの名前は、プログラム内の特殊データセットを参照するために使用できます。データセットの命名規則や特殊なデータセット名の詳細については、“データセット名” (49 ページ)のトピックを参照してください。

データセット名リスト

データセット名リストは、ステートメントまたはプロシジャで複数の SAS データセット名を指定する短縮方法です。

データセット名リストは、*番号付き範囲リスト*または*名前接頭辞リスト*のいずれかになります。

番号付き範囲リスト

```
data combine;
  set sales0 sales1 sales2 sales3 sales4 sales5;
run;

data combine;
  set sales0-sales5;
run;
```

- 番号付き範囲リスト内のデータセット名は、連続する番号である最後の文字を除いて同じ名前になります。
- 番号付き範囲リスト内のデータセット名は、番号が連続している限り、任意の番号で始まり、任意の番号で終わることができます。たとえば、次の SET ステートメントは同じデータセットを参照します。

```
proc datasets;
  copy in=work out=mysas;
  select d1-d3;
run; quit;
```

注: 数値接尾辞の先頭にゼロが含まれる場合、最後のデータセット名の接尾辞の桁数は、最初のデータセット名の桁数以上である必要があります。たとえば、データセットリスト sales001–sales99 ではエラーが発生します。データセットリスト sales001–sales999 は有効です。

名前接頭辞リスト(コロンリスト)

名前接頭辞リストは、単一の名前接頭辞とその後に続くコロンを使用して複数のデータセット名を指定する短縮方法です。短縮(接頭辞)名は、それが表す名前と同じ文字で始まる必要があります。コロン(:)は文字列の接頭辞の末尾を指定するために使用されます。たとえば、次の 2 つの DATA ステップは同じデータセットを参照します。

```
data combine;
  set sales5 sales3 sales2 sal sal_weekly sales_monthly;
run;

data combine;
  set sal:;
run;
```

データセット名リストは通常、次の SAS 言語要素で使用されます。

PROC DATASETS のステートメント:

- COPY ステートメント(“選択した SAS ファイルのコピー” (*Base SAS プロシジャガイド*))
- SELECT ステートメント(“複数の同じ名前のファイルの選択” (*Base SAS プロシジャガイド*))
- EXCLUDE ステートメント(“複数の同じ名前のファイルを除外” (*Base SAS プロシジャガイド*))
- DELETE ステートメント
- REPAIR ステートメント
- MODIFY ステートメントの SORTEDBY オプション

DATA ステップステートメント:

- MERGE ステートメント、(MERGE でのデータセットリストの使用)
- SET ステートメント

SAS コードでの文字変数リストと数値変数リストの作成の詳細については、“変数リスト” (91 ページ)を参照してください。

関連項目

“SAS データセットの定義” (300 ページ)

並べ替えられたデータセット

並べ替えインジケーター

データセットが並べ替えられた後、*並べ替えインジケーター*が、データセットのデータセットディスクリプター情報に追加されます。これは、CONTENTS プロシジャを指定することで確認できます。

```
proc contents data=air; run;
```

図 13.1 **検証済み**フィールドを含む並べ替え情報を示す PROC CONTENTS 出力

The CONTENTS Procedure

Data Set Name	WORK.AIR	Observations	144
Member Type	DATA	Variables	2
Engine	V9	Indexes	0
Created	04/13/2020 17:47:20	Observation Length	16
Last Modified	04/13/2020 17:47:20	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	YES
Label	airline data (monthly: JAN49-DEC60)		
Data Representation	SOLARIS		
Encoding	utf-8 Unicode (UTF-8)		

• • •

Sort Information	
Sortedby	AIR
Validated	YES
Character Set	ASCII

並べ替えインジケーターには、データの並べ替えに使用された方法に応じて **YES** または **NO** に設定される**検証済み**フィールドが含まれます。次の表は、データを並べ替える方法と、その方法が使用される**検証済み**フィールドの設定方法を示しています。

表 13.4 SAS データセットに並べ替えインジケーターを設定する方法

並べ替え方法	並べ替え情報	例

SORT プロシジャ	**検証済み** = **YES**	proc sort data=sashelp.air out=air;
		by air;
		run;
		proc contents data=air; run;
		
		Sort Information
		Sortedby AIR
		Validated YES
		Character Set ASCII

ORDER BY を使用した SQL プロシジャ	**検証済み** = **YES**	proc sql; select * from air order by air; proc contents data=air; run; Sort Information Sortedby AIR Validated YES Character Set ASCII

MODIFY SORTEDBY=オプションを使用した	**検証済み** = **NO**	proc datasets library=work memtype=data;
		modify air (sortedby=air);
		run;
		proc contents data=air; run;

並べ替え方法	並べ替え情報	例
DATASETS プロシジャ	**ソート順**並べ替え情報は、データセットオプションで指定された変数で更新されます。	Sort Information Sortedby: AIR Validated: NO Character Set: ASCII

SORTEDBY=データセットオプション	**検証済み** = **NO**
	ソート順並べ替え情報は、データセットオプションで指定された変数で更新されます。
		data air (sortedby=air);
		set sashelp.air;
		run;
		proc contents data=air; run;
		Sort Information
		Sortedby: AIR
		Validated: NO
		Character Set: ASCII

並べ替えインジケーターには、SAS データセットの次の並べ替え情報の一部またはすべてが含まれます。

- データセットがどの変数によってどのように並べ替えらえるか
- 変数の順序が降順か昇順(デフォルト)か
- 文字データの順序付けに使用される文字セット
- 文字データの順序付けに使用される照合順序
- データセットが言語的に並べ替えられている場合の照合ルール
- 特定の BY グループにオブザベーションが 1 つだけあるかどうか(SORT プロシジャでの NODUPKEY オプションの使用)

SAS が並べ替えインジケーターを使用してパフォーマンスを向上させる方法

並べ替えインジケーターによって提供される並べ替え情報は、パフォーマンス向上のために内部で使用されます。並べ替えインジケーターを使用してパフォーマンスを向上させる方法はいくつかあります。

- SAS は、並べ替えインジケーターを使用して、以前に並べ替えがあったかどうかを検証します。SORT プロシジャまたは ORDER BY 句を使用した SQL プロシジャによる以前の並べ替えがあった場合、SAS は別の並べ替えを実行しません。
- SORT プロシジャは、並べ替えが行われるときに並べ替えインジケーターを設定します。SORT プロシジャは、データが不必要に再並べ替えされないように、データセットを並べ替える前に並べ替えインジケーターをチェックします。詳細については、“SORT プロシジャ” (*Base SAS プロシジャガイド*)を参照してください。
- SQL プロシジャは並べ替えインジケーターを使用してクエリをより効率的に処理し、結合を実行する前に内部並べ替えが必要かどうかを判断します。

- インデックス作成時に並べ替えインジケーターを使用する場合、SAS は、データファイル内の並べ替えインジケーターをチェックすることにより、データがすでにキー変数によって昇順に並べ替えられているかどうかを判断します。並べ替えインジケーターの値が昇順である場合、SAS はインデックスファイルの値を並べ替えません。
- インデックスなしで WHERE 式を処理する場合、SAS は最初に並べ替えインジケーターをチェックします。**検証済み**並べ替え情報が **YES** の場合、SAS は WHERE 式を満たす値がなくなるとファイルの読み取りを停止します。
- WHERE 式の処理にインデックスが選択されている場合、そのデータセットの並べ替えインジケーターは、インデックスで指定された順序を反映するように変更されます。
- BY グループ処理では、データセットがすでに BY 変数で並べ替えられている場合、データセットが BY 変数でインデックス付けされていても、SAS はインデックスを使用しません。
- **検証済み**並べ替え情報が **YES** に設定されている場合、SAS は別の並べ替えを実行する必要はありません。

世代データセット

世代データセットは、SAS データセットの更新時に作成される、SAS データセットのアーカイブされたバージョンです。世代データセットの詳細については、“Definitions for Generation Data Sets” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。

例: SAS データセットの作成と読み取り

例: 単一の SAS データセットの読み取り

コード例

この例では、sashelp ライブラリから SAS データセットを読み取り、出力を SAS Work ライブラリに書き込みます。

SET ステートメントは、sashelp.shoes データセットを DATA ステップに読み込み、WHERE ステートメントで処理します。WHERE ステートメントは、変数 sales に 500,000 より大きい値を含むオブザベーションのみを選択します。次に、DATA ステ

ップは、DATA ステートメントで指定されたデータセット(work.shoes)に出力を書き込みます。

```
data work.shoes;
  set sashelp.shoes;
  where sales>500000;
run;
proc print data=shoes; run;
```

アウトプット 13.1 Sashelp ライブラリから読み取られた Work.shoes データセットの PROC PRINT 出力

Obs	Region	Product	Subsidiary	Stores	Sales	Inventory	Returns
1	Canada	Men's Dress	Vancouver	28	$757,798	$1,847,559	$16,833
2	Canada	Slipper	Vancouver	27	$700,513	$2,520,085	$21,247
3	Canada	Women's Dress	Vancouver	21	$756,347	$2,503,387	$19,378
4	Central America/Caribbean	Men's Casual	Kingston	28	$576,112	$1,159,556	$20,005
5	Middle East	Men's Casual	Tel Aviv	11	$1,298,717	$2,881,005	$57,362
6	Western Europe	Women's Casual	Copenhagen	26	$502,636	$1,110,412	$17,448

要点

- SET ステートメントは、SAS データセットを DATA ステップに読み込んで処理します。また、MERGE ステートメント、MODIFY ステートメント、および UPDATE ステートメントを使用して、SAS データセットを DATA ステップに読み込むこともできます。
- DATA ステートメントは、DATA ステップによって処理された SAS データセットを書き込みます。
- 出力データセットの場所を指定しない場合、DATA ステートメントは出力を SAS Work ライブラリに自動的に書き込みます。Work ライブラリに書き込まれたデータセットは、現在の SAS セッションの間のみ保存されます。永久出力場所を指定するには、LIBNAME ステートメントを使用して SAS ライブラリ(ライブラリ参照名)を作成する必要があります。SAS のライブラリの詳細については、11 章, “SAS ライブラリ” (227 ページ)を参照してください。
- デフォルトでは、DATA ステップは、順次アクセスを使用して SAS データセットからオブザベーションを読み取ります。順次および直接データアクセスの詳細については、“SAS データセット内の行を読み取る方法” (303 ページ)を参照してください。

関連項目

- SET ステートメント、 DATA ステートメント、および“WHERE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- *SAS プログラマガイド: エッセンシャル*の“Sashelp ライブラリ” (237 ページ)

■ “SAS データセット内の行を読み取る方法” (303 ページ)

例: DATA ステッププログラミングステートメントを使用したデータの作成

コード例

データを読み取るのではなく、プログラミングステートメントを使用してオブザベーションを生成することによって、SAS データセットのデータを作成できます。入力を読み取らない DATA ステップは 1 回だけ反復されます。

```
data investment;                        /* 1 */
  begin='01JAN1990'd;
  end='31DEC2009'd;
  do year=year(begin) to year(end);     /* 2 */
    Capital+2000 + .07*(Capital+2000);
    output;                             /* 3 */
  end;
  put 'The number of DATA step iterations is '_n_;  /* 4 */
run;                                    /* 5 */

proc print data=investment;             /* 6 */
  format Capital dollar12.2;            /* 7 */
run;                                    /* 8 */
```

1 DATA ステップを開始して、SAS データセット Investment を作成します。

2 1990 年から 2009 年まで、2,000 ドルの資本投資と毎年 7%の利息に基づいて価値を計算します。DO ループの反復ごとに 1 つのオブザベーションの変数値を計算します。

3 各オブザベーションをデータセット Investment に書き込みます。

4 DATA ステップが 1 回だけ反復されることを証明する注記を SAS ログに書き込みます。

5 DATA ステップを実行します。

6 出力を確認するには、PRINT プロシジャを使用して Investment データセットを出力します。

7 FORMAT ステートメントを使用して、ドル記号、カンマ、および小数点を使用して数値を書き込みます。

8 PRINT プロシジャを実行します。

要点

- SET ステートメントは、SAS データセットを DATA ステップに読み込んで処理します。また、MERGE ステートメント、MODIFY ステートメント、および UPDATE ステートメントを使用して、SAS データセットを DATA ステップに読み込むこともできます。
- DATA ステートメントは、DATA ステップによって処理された SAS データセットを書き込みます。
- 出力データセットの場所を指定しない場合、DATA ステートメントは出力を SAS Work ライブラリに自動的に書き込みます。Work ライブラリに書き込まれたデータセットは、現在の SAS セッションの間のみ保存されます。永久出力場所を指定するには、LIBNAME ステートメントを使用して SAS ライブラリ(ライブラリ参照名)を作成する必要があります。SAS のライブラリの詳細については、11 章, “SAS ライブラリ” (227 ページ)を参照してください。
- デフォルトでは、DATA ステップは、順次アクセスを使用して SAS データセットからオブザベーションを読み取ります。順次および直接データアクセスの詳細については、“SAS データセット内の行を読み取る方法” (303 ページ)を参照してください。

関連項目

- SET ステートメント、 DATA ステートメント、および“WHERE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- *SAS プログラマガイド: エッセンシャル*の“Sashelp ライブラリ” (237 ページ)
- “SAS データセット内の行を読み取る方法” (303 ページ)

例: 複数の SAS データセットの読み取り

コード例

この例では、Sashelp ライブラリから 3 つのデータセットを読み取り、それらを concat という名前の単一の出力データセットに連結します。SAS ライブラリまたは出力場所が指定されていないため、出力データセット concat は一時的に SAS Work ライブラリに保存されます。

```
data concat;
  set sashelp.nvst1 sashelp.nvst2 sashelp.nvst3;
run;
```

```
proc print data=concat; run;
```

出力データセットは、3 つのデータセットすべてからのオブザベーションで 構成されます。出力データセット内でデータセットが連結される順序は、SET ステートメントでのデータセットのリスト方法に基づきます。sashelp.nvst1 からのオブザベーションが最初で、次に sashelp.nvst2 からのオブザベーションが続き、その後に sashelp.nvst3 からのオブザベーションが続きます。

データセット concat の PROC PRINT 出力は次のとおりです。DATA ステップが複数の入力データセットをどのように連結するかを示す注釈が付けられています。

アウトプット 13.2　データセット Concat の PROC PRINT 出力

Obs	DATE	AMOUNT	
1	01JAN1997	-30000	Sashelp.nvst1
2	01JAN1998	7500	
3	01JAN1999	7500	
4	01JAN2000	7500	
5	01JAN2001	7500	
6	01JAN2002	7500	
7	01JAN1997	-60000	Sashelp.nvst2
8	01JAN1998	13755	
9	01JAN1999	13755	
10	01JAN2000	13755	
11	01JAN2001	13755	
12	01JAN2002	23755	
13	01JAN1997	-20000	Sashelp.nvst3
14	01JAN1998	5000	
15	01JAN1999	5000	
16	01JAN2000	5000	
17	01JAN2001	5000	
18	01JAN2002	5000	

ログ出力は次のとおりです。

アウトプット 13.3　複数の SAS データセットを読み取るためのログ出力

```
NOTE: There were 6 observations read from the data set SASHELP.NVST1.
NOTE: There were 6 observations read from the data set SASHELP.NVST2.
NOTE: There were 6 observations read from the data set SASHELP.NVST3.
NOTE: The data set WORK.CONCAT has 18 observations and 2 variables.
```

要点

- 複数の SAS データセットから読み取り、さまざまな方法でデータを結合および変更できます。詳細については、“SAS データセットを結合する方法の概要” (515 ページ)を参照してください。
- SET ステートメントは、SAS データセットを DATA ステップに読み込んで処理します。また、MERGE ステートメント、MODIFY ステートメント、および UPDATE ステートメントを使用して、SAS データセットを DATA ステップに読み込むこともできます。
- 出力データセットの場所を指定しない場合、DATA ステートメントは出力を SAS Work ライブラリに自動的に書き込みます。Work ライブラリに書き込まれたデータセットは、現在の SAS セッションの間のみ保存されます。永久出力場所を指定するには、LIBNAME ステートメントを使用して SAS ライブラリ(ライブラリ参照名)を作成する必要があります。SAS のライブラリの詳細については、11 章, “SAS ライブラリ” (227 ページ)を参照してください。
- DATA ステートメントは、DATA ステップによって処理された SAS データセットを書き込みます。
- DATA ステートメントは、DATA ステップによって処理された SAS データセットを書き込みます。

関連項目

- “SET ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “WHERE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “Sashelp ライブラリ” (237 ページ)

例: ユーザー定義ライブラリから SAS データセットを読み取る

コード例

この例では、ユーザー定義ライブラリ mysas から永久データセット quakes を読み取り、それを同じライブラリに quakes_mag として書き込みます。

この例を設定するには、まず LIBNAME ステートメントを使用してユーザー定義ライブラリ(ライブラリ参照名)mysas を作成します。次に、データセット sashelp.quakes を作成し、出力データセットとして mysas.quakes を指定します。

```
libname mysas "<path-to-file>";              /* 1 */

data mysas.quakes;                           /* 2 */
  set sashelp.quakes;
run;

proc sort data=mysas.quakes;
  by Magnitude;
run;                                  /* 3 */

data mysas.quakes_mag;                       /* 4 */
  set mysas.quakes;                     /* 5 */
  by Magnitude;                        /* 6 */
run;
proc print data=mysas.quakes_mag(obs=10);      /* 7 */
  title "Earthquakes by Magnitude";
run;
```

1 LIBNAME ステートメントは、ライブラリ参照名 mysas を作成します。ライブラリへのパスは単一引用符または二重引用符で指定します。

2 この例の永久データセットを作成するには、SET ステートメントで Sashelp ライブラリから quakes データセットを読み取ります。DATA ステートメントは、データセット quakes を mysas ライブラリに書き込み、そこでディスクに保存されます。

3 SORT プロシジャは、mysas.quakes データセットを変数 Magnitude の値順に並べ替えます。

4 DATA ステートメントは、quakes データセットを同じライブラリ内のデータセット quakes_mag に書き込みます。

5 SET ステートメントは、データセット quakes を mysas ライブラリから読み取ります。

6 BY ステートメントは、変数 Magnitude の値によってオブザベーションをグループ化し、順序付けします。

7 PRINT プロシジャは、quakes_mag データセットの結果を出力します。PRINT ステートメントの OBS=データセットオプションにより、PRINT プロシジャは出力データセットの最初の 10 個のオブザベーションのみを表示します。

次の PROC PRINT 出力は結果を示しています。

アウトプット 13.4 Magnitude 順の mysas.quakes の PROC PRINT 出力

Earthquakes by Magnitude

Obs	Latitude	Longitude	Depth	Magnitude	dNearestStation	RootMeanSquareTime	Type
1	36.8408	-97.875	4.2510	2.5	.	0.4600	earthquake
2	36.0210	-97.097	4.0420	2.5	.	0.6300	earthquake
3	36.7038	-98.041	12.7600	2.5	0.20300	0.2500	earthquake
4	36.7554	-97.556	6.2220	2.5	.	0.7600	earthquake
5	41.9035	-119.662	1.7742	2.5	0.47700	0.2545	earthquake
6	36.3510	-84.995	20.7000	2.5	0.34136	0.2000	earthquake
7	35.7968	-97.444	5.4500	2.5	.	0.3500	earthquake
8	36.7484	-97.649	3.3550	2.5	.	0.5500	earthquake
9	41.8692	-119.615	5.0000	2.5	0.65577	0.6500	earthquake
10	41.8652	-119.673	0.2000	2.5	0.46712	0.4500	earthquake

注: 出力は、出力データセットの一部にすぎません。PROC PRINT ステートメントの OBS=データセットオプションは、表示されるオブザベーションの数を制限します。

要点

- 出力データセットの場所を指定しない場合、DATA ステートメントは出力を SAS Work ライブラリに自動的に書き込みます。Work ライブラリに書き込まれたデータセットは、現在の SAS セッションの間のみ保存されます。永久出力場所を指定するには、LIBNAME ステートメントを使用して SAS ライブラリ(ライブラリ参照名)を作成する必要があります。SAS のライブラリの詳細については、11 章, “SAS ライブラリ” (227 ページ)を参照してください。
- SET ステートメントは、SAS データセットを DATA ステップに読み込んで処理します。また、MERGE ステートメント、MODIFY ステートメント、および UPDATE ステートメントを使用して、SAS データセットを DATA ステップに読み込むこともできます。
- DATA ステートメントは、DATA ステップによって処理された SAS データセットを書き込みます。
- デフォルトでは、DATA ステップは、順次アクセスを使用して SAS データセットからオブザベーションを読み取ります。順次および直接データアクセスの詳細については、“SAS データセット内の行を読み取る方法” (303 ページ)を参照してください。

関連項目

- 11 章, “SAS ライブラリ” (227 ページ)

- “BY ステートメント” (*SAS DATA ステップステートメント: リファレンス*)

例: ライブラリ参照名を使用せずに永久 SAS データセットを読み取る

コード例

この例では、Windows ディレクトリ demo に保存されている永久 SAS データセットを読み取り、それを別のディレクトリ(demo2 ディレクトリ)に書き込みます。ライブラリ参照名を使用するかわりに、SAS データセットへのパス名が DATA ステートメントで引用符で囲まれて指定されます。

SET ステートメントは、フォルダー demo 内のデータセット shoesales を読み取ります。DATA ステートメントは、同じ構文を使用して、フォルダー demo2 内の shoesales2 に出力を書き込みます。

```
data "c:\Users\demo\shoesales2.sas7bdat";
  set "c:\Users\demo2\shoesales.sas7bdat";
run;

data "c:\Users\Jdoe\courses2.sas7bdat";
  set "c:\Users\Jdoe\sasuser\courses.sas7bdat";
run;
proc print data="c:\Users\Jdoe\courses2.sas7bdat"; run;
```

次にログ出力を示します。

アウトプット 13.5　ログ出力

```
NOTE: There were 395 observations read from the data set c:\Users\demo
\shoesales.sas7bdat.
NOTE: The data set c:\Users\demo2\shoesales2.sas7bdat has 395 observations and 8
variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

要点

- 出力データセットの場所を指定しない場合、DATA ステートメントは出力を SAS Work ライブラリに自動的に書き込みます。Work ライブラリに書き込まれたデータセットは、現在の SAS セッションの間のみ保存されます。永久出力場所を指定するには、LIBNAME ステートメントを使用して SAS ライブラリ(ライブラリ参照名)

を作成する必要があります。SAS のライブラリの詳細については、11 章, “SAS ライブラリ” (227 ページ)を参照してください。

- SET ステートメントは、SAS データセットを DATA ステップに読み込んで処理します。また、MERGE ステートメント、MODIFY ステートメント、および UPDATE ステートメントを使用して、SAS データセットを DATA ステップに読み込むこともできます。
- DATA ステートメントは、DATA ステップによって処理された SAS データセットを書き込みます。
- デフォルトでは、DATA ステップは、順次アクセスを使用して SAS データセットからオブザベーションを読み取ります。順次および直接データアクセスの詳細については、“SAS データセット内の行を読み取る方法” (303 ページ)を参照してください。

関連項目

- 11 章, “SAS ライブラリ” (227 ページ)
- “SAS デフォルトライブラリ” (236 ページ)

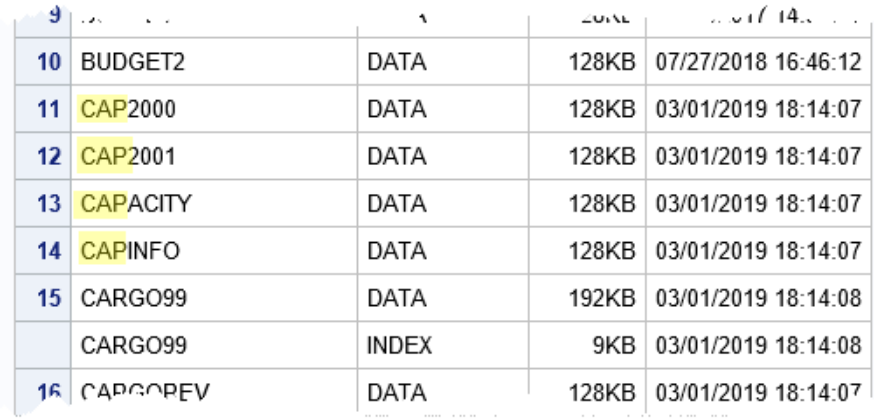
例: データセット名リストの使用

例: データセット名接頭辞リストを使用して複数の SAS データセットをコピーする

例

この例では、名前接頭辞リストを使用して、PROC COPY 操作で複数の SAS データセットを参照します。名前接頭辞リストは、Sasuser (237 ページ)システムライブラリ内の Cap という文字で始まるすべてのデータセットがユーザー定義ライブラリ mylib にコピーされることを指定する短縮方法です。次のデータセットは、Cap2000、Cap2001、Capacity、Capinfo です。

図 13.2　Cap の文字で始まる Sasuser のデータセット



10	BUDGET2	DATA	128KB	07/27/2018 16:46:12
11	CAP2000	DATA	128KB	03/01/2019 18:14:07
12	CAP2001	DATA	128KB	03/01/2019 18:14:07
13	CAPACITY	DATA	128KB	03/01/2019 18:14:07
14	CAPINFO	DATA	128KB	03/01/2019 18:14:07
15	CARGO99	DATA	192KB	03/01/2019 18:14:08
	CARGO99	INDEX	9KB	03/01/2019 18:14:08
16	CARGOREV	DATA	128KB	03/01/2019 18:14:07

```
libname mylib 'c:\Users\sasuser\mysas';  /* 1 */

proc copy in=sasuser out=mylib;        /* 2 */
  select Cap:;                         /* 3 */
run;
quit;
proc datasets library=mylib;  run;     /* 4 */
```

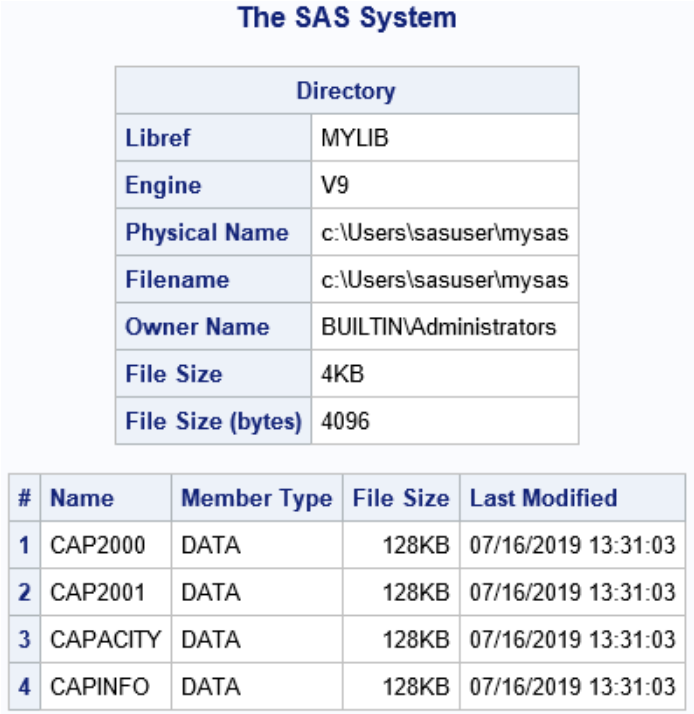
1 LIBNAME ステートメントを使用して、(238 ページ)ライブラリ参照名 mylib を物理的な場所 **c:\Users\sasuser\mysas** に割り当てます。ライブラリ参照名は、データの物理的な場所を参照するために作成するショートカット名またはニックネームです。この場合、それはデータセットファイルのコピー先の場所です。

2 COPY プロシジャを使用して、Cap という文字で始まるデータセットを Sasuser (237 ページ)システムライブラリから mylib ライブラリにコピーします。

3 SELECT ステートメントを使用して、Sasuser (237 ページ)システムライブラリ内のどのデータセットをコピーするかを指定します。Cap の文字に続けてコロン(:)を指定すると、Cap の文字で始まる Sasuser システムライブラリのデータセットがライブラリ mylib にコピーされます。

4 DATASETS プロシジャを使用して、データセットがコピーされていることを確認します。PROC DATASETS の LIBRARY=オプションは、ライブラリ mylib 内のすべてのファイルをリストします。

図 13.3 名前接頭辞(コロン)リストを使用して PROC COPY で指定されたデータセットを示す PROC DATASETS 出力



The SAS System

Directory	
Libref	MYLIB
Engine	V9
Physical Name	c:\Users\sasuser\mysas
Filename	c:\Users\sasuser\mysas
Owner Name	BUILTIN\Administrators
File Size	4KB
File Size (bytes)	4096

#	Name	Member Type	File Size	Last Modified
1	CAP2000	DATA	128KB	07/16/2019 13:31:03
2	CAP2001	DATA	128KB	07/16/2019 13:31:03
3	CAPACITY	DATA	128KB	07/16/2019 13:31:03
4	CAPINFO	DATA	128KB	07/16/2019 13:31:03

要点

- データセット名リスト (306 ページ)は、ステートメントまたはプロシジャで複数の SAS データセット名を指定する短縮方法です。
- データセット名リストは、*番号付き範囲リスト*または*名前接頭辞リスト*のいずれかになります。
- 一般的な文字の後にコロン(:)を入力することで、名前が同じ文字で始まるデータセットのグループを選択できます。

関連項目

- “複数の同じ名前のファイルの選択” (*Base SAS プロシジャガイド*)
- “選択した SAS ファイルのコピー” (*Base SAS プロシジャガイド*)
- MERGE ステートメント(MERGE でのデータセットリストの使用)
- (番号付き範囲)データセットリストの使用

例: 番号付き範囲リストを使用して複数のデータセットを参照する

例

この例では、複数の入力データセットを指定するためのショートカット方法として番号付き範囲リストを使用します。すべてのデータセットの名前は同じ文字で始まり、連続した番号で終わります。

```
libname mylib 'c:\Users\sasuser\mysas';  /* 1 */

data mylib.Eisobj;
  set Sashelp.Eisobj1-Sashelp.Eisobj8;  /* 2 */
run;
```

1 LIBNAME ステートメントを使用して、(238 ページ)ライブラリ参照名 mylib を物理的な場所 **c:\Users\sasuser\mysas** に割り当てます。ライブラリ参照名は、データの物理的な場所を参照するために作成するショートカット名またはニックネームです。この場合、それはデータセットファイルのコピー先の場所です。

2 SET ステートメントを使用して、複数の入力データセットを Sashelp ライブラリから mysas ライブラリ内の単一の SAS データセットに読み込みます。8 つのデータセット(Eisobj1 から Eisobj8)を、番号付き範囲リスト (306 ページ)を使用して指定します。

要点

- 番号付き範囲リストは、同じ名前の接頭辞が付いているが、最後の文字の数値が異なる変数で構成されます。値は連続している必要があります。
- 2147483647 より大きい数字で終わるデータセットまたは変数には、番号付き範囲リストを指定できません。
- OF 演算子を使用すると、番号付き範囲リストを引数として一部の関数に渡すことができます。

関連項目

- “複数の同じ名前のファイルの選択” (*Base SAS プロシジャガイド*)
- “選択した SAS ファイルのコピー” (*Base SAS プロシジャガイド*)
- MERGE ステートメント(MERGE でのデータセットリストの使用)
- (番号付き範囲)データセットリストの使用

例: データセット内の変数とオブザベーションを制御する

例: 特定の変数をデータセットに保持する

コード例

この例では、KEEP ステートメントを使用して、どの変数を出力データセットに書き込むかを制御します。

LIBNAME ステートメントは、出力データセットを書き込む場所を指定し、その場所に mysas というを名前関連付けます。SET ステートメントは入力データセット Sashelp.Cars を読み取り、DATA ステートメントは結果を出力データセット mysas.cars に書き込みます。

この例では、KEEP ステートメントを使用して、変数 Make、Mpg、および MSRP のみを出力データセットに含めます。SAS は、Sashelp.Cars データセットからすべての変数をメモリに読み込み、出力データセットを作成するときに不要な変数を削除します。変数 MPG_City および MPG_Highway は、DATA ステップによってメモリに読み込まれ、重み付き平均 MPG の計算に使用されますが、出力には含まれません。

```
libname mysas "c:\Users\demo";
data cars;
  set sashelp.cars;
  keep make mpg MSRP;
  Mpg=(MPG_City*.45)+(MPG_Highway*.55)/2;
run;
proc print data=cars(obs=10); run;
```

アウトプット 13.6　KEEP ステートメントで指定された変数のみを表示する mysas.cars データセットの PROC PRINT 出力

Obs	Make	MSRP	Mpg
1	Acura	$36,945	13.975
2	Acura	$23,820	19.325
3	Acura	$26,990	17.875
4	Acura	$33,195	16.700
5	Acura	$43,755	14.700
6	Acura	$46,100	14.700
7	Acura	$89,765	14.250
8	Audi	$25,940	18.425
9	Audi	$35,940	18.600
10	Audi	$31,840	16.700

注: 出力は、出力データセットの一部にすぎません。PROC PRINT ステートメントの OBS=データセットオプションは、表示されるオブザベーションの数を制限します。

変数の選択を制御する別の方法は、KEEP=データセットオプションを使用することです。SET ステートメントの入力データセットの KEEP=データセットオプションは、どの変数が出力データセットに読み取られ、出力データセットに書き込まれるかを制御します。DATA ステートメントの出力データセットの KEEP=データセットオプションは、どの変数を出力データセットに書き込むかを制御します。

要点

- 特に指示しない場合、SAS は入力データセットのすべての変数とすべてのオブザベーションを出力データセットに書き込みます。
- SAS ステートメント、データセットオプション、および関数を使用して、どの変数やオブザベーションを読み書きするかを制御できます。これらの言語要素のリストについては、“SAS データセット内の変数を管理する方法” (301 ページ)を参照してください。
- DROP ステートメントと DROP=データセットオプションは、選択した変数が保持されずに削除されることを除いて、KEEP=ステートメントおよび KEEP=データセットオプションと同じように機能します。

関連項目

- DROP=データセットオプションと DROP ステートメントの比較
- KEEP=データセットオプションと KEEP ステートメントの比較
- “入力から変数を除外する ” (*SAS データセットオプション: リファレンス*)

- “変数をデータセットに書き込まずに処理する” (*SAS データセットオプション: リファレンス*)

例: WHERE ステートメントを使用してどのオブザベーションを読み取るかを制御する

コード例

この例では、WHERE ステートメントを使用して、変数 age と height の値に基づいてオブザベーションを選択します。

```
data class;
  set sashelp.class;
  where age>12 and height>=67;
run;
proc print data=class; run;
```

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Philip	M	16	72	150.0
3	Ronald	M	15	67	133.0

次のログが示すように、DATA ステップがこのプログラムを実行する場合、入力データセット内のすべての行が読み取られるわけではありません。

アウトプット 13.7　ログ出力

```
NOTE: There were 3 observations read from the data set SASHELP.CLASS.
      WHERE (age>12) and (height>=67);
```

WHERE ステートメントを使用すると、DATA ステップで読み取る必要があるオブザベーションが少なくなった場合に、SAS プログラムの効率を向上させることができます。

WHERE=データセットオプションを使用して、入力データセットまたは出力データセットのいずれかのオブザベーションを条件付きで選択することもできます。

```
data class;
  set sashelp.class(where=(age>12 and height >=67));
run;
```

WHERE ステートメントと同様、WHERE=データセットオプションを入力データセットで指定すると、DATA ステップですべてのオブザベーションを 読み取る必要はありません。次は、入力データセットの SET ステートメントで WHERE=データセットオプションが指定された場合のログ出力です。

アウトプット 13.8　ログ出力

```
NOTE: There were 3 observations read from the data set SASHELP.CLASS.
      WHERE (age>12) and (height>=67);
```

DATA ステートメントの出力データセットで WHERE=データセットオプションが指定されている場合、DATA ステップは入力データセットのすべての行を読み取り、基準を満たすオブザベーションのみを出力データセットに書き込みます。

```
data class(where=(age>12 and height >=67));
  set sashelp.class;
run;
```

アウトプット 13.9　ログ出力

```
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set WORK.CLASS has 3 observations and 5 variables.
```

要点

- 特に指示しない場合、SAS は入力データセットのすべての変数とすべてのオブザベーションを出力データセットに書き込みます。
- SAS ステートメント、データセットオプション、および関数を使用して、どの変数やオブザベーションを読み書きするかを制御できます。これらの言語要素のリストについては、“SAS データセット内の変数を管理する方法” (301 ページ)を参照してください。
- WHERE ステートメントは、変数の値に基づいて、SET ステートメントによってどのオブザベーションが読み取られるかを制御します。
- WHERE ステートメントが DATA ステップで使用される場合、DATA ステップは入力データセット内のすべてのオブザベーションを 読み取るわけではありません。したがって、WHERE ステートメントを使用すると、SAS プログラムの効率を向上させることができます。
- DATA ステートメントの WHERE=データセットオプションは、DATA ステートメントによって出力データセットに書き込まれるオブザベーションを制御します。DATA ステートメントで WHERE=データセットオプションが指定されている場合、DATA ステップは入力データセット内のすべてのオブザベーションを 読み取ります。

関連項目

- “WHERE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)および “SAS DATA ステップで WHERE ステートメントを指定する” (*SAS DATA ステップステートメント: リファレンス*)

- “WHERE= データセットオプション” (*SAS データセットオプション: リファレンス*)

例: 最初に特定のオブザベーションを読み取る (FIRSTOBS)

コード例

この例では、DATA ステップは入力データセット内の特定のオブザベーションに直接アクセスし、その指定されたオブザベーションから始まる新しいデータセットに出力を書き込みます。FIRSTOBS=データセットオプションは、オブザベーションが入力データセット Sashelp.quakes のオブザベーション番号 5 から始まる出力データセット quakes2 に書き込まれることを指定します。

この例の入力データセットを作成するには、次の DATA ステップを使用して、Sashelp.Quakes データセットのサブセットを作成します。DATA ステップでは、変数 Magnitude の値が **6.0** より大きいオブザベーションを条件付きで選択することにより、サブセットを作成します。

```
data quakes;
  set sashelp.quakes(where=(Magnitude>6.0));
  keep Depth Type Magnitude;
run;
proc print data=quakes; run;
```

アウトプット 13.10　Quakes データセットの PROC PRINT 出力

Obs	Depth	Magnitude	Type
1	11.25	6.02	earthquake
2	9.45	6.60	earthquake
3	13.00	6.30	earthquake
4	13.00	7.00	earthquake
5	4.00	7.20	earthquake
6	10.00	6.20	earthquake
7	10.00	6.90	earthquake
8	14.00	6.60	earthquake

次の DATA ステップでは、SET ステートメントで FIRSTOBS=データセットオプションを指定して、入力データセットのオブザベーション 5 で始まる新しい出力データセットを作成します。出力データセットには、入力データセットの残りのオブザベーションがすべて 含まれます。

```
data quakes2;
  set quakes(firstobs=5);
run;
proc print data=quakes2; run;
```

アウトプット 13.11　Quakes データセットの行 5 が Quakes2 データセットの最初の行として示される PROC PRINT 出力

Obs	Depth	Magnitude	Type
1	4	7.2	earthquake
2	10	6.2	earthquake
3	10	6.9	earthquake
4	14	6.6	earthquake

また、NOTE 関数、CUROBS 関数、POINT 関数、FETCHOBS 関数を使用して、オブザベーション番号でオブザベーションに直接アクセスすることもできます。

要点

- OBS=データセットオプションが処理の終了点を指定する場合、FIRSTOBS=データセットオプションは開始点を指定します。2 つのオプションは、処理するオブザベーションの範囲を定義するために一緒に使用されることがよくあります。
- OBS=データセットオプションを使用すると、SAS データセットからオブザベーションを選択できます。INFILE ステートメントの OBS=オプションを使用すると、外部データファイルから読み取るオブザベーションを選択できます。

関連項目

- “FIRSTOBS= システムオプション” (*SAS システムオプション: リファレンス*)および“OBS= システムオプション” (*SAS システムオプション: リファレンス*)
- “FIRSTOBS= データセットオプション” (*SAS データセットオプション: リファレンス*)および“OBS= データセットオプション” (*SAS データセットオプション: リファレンス*)

例: オブザベーションの範囲の読み取り(FIRSTOBS および OBS)

コード例

この例では、DATA ステップは、FIRSTOBS=および OBS=データセットオプションを一緒に使用して、データセット内のオブザベーションの範囲にアクセスします。FIRSTOBS=データセットオプションは、入力データセット Sashelp.quakes のオブザベーション番号 2 から始まるオブザベーションが出力データセット quakes2 に書き

込まれることを指定します。OBS=データセットオプションは、指定された数のオブザベーションを読み取った後にオブザベーションの処理を停止するように SAS に指示します。

SAS は、OBS=および FIRSTOBS=データセットオプションを一緒に使用するときに読み取るオブザベーションの数を決定するために次の式を使用します: **(obs - firstobs) + 1 = 行数**。

この例の入力データセットを作成するには、最初の DATA ステップで Sashelp.Quakes データセットのサブセットを作成します。2 番目の DATA ステップでは、オブザベーションの範囲を作成します。

```
data quakes;
 set sashelp.quakes(where=(Magnitude>6.0));
 keep Depth Type Magnitude;
run;
proc print data=quakes; run;

data quakes2;
  set quakes(firstobs=2 obs=4);
run;
proc print data=quakes2; run;
```

アウトプット 13.12　Quakes から選択され、Quakes2 に書き込まれたオブザベーションの範囲を示す PROC PRINT 出力

Quakes

Obs	Depth	Magnitude	Type
1	11.25	6.02	earthquake
2	9.45	6.60	earthquake
3	13.00	6.30	earthquake
4	13.00	7.00	earthquake
5	4.00	7.20	earthquake
6	10.00	6.20	earthquake
7	10.00	6.90	earthquake
8	14.00	6.60	earthquake

Quakes2

FIRSTOBS=2, OBS=4

Obs	Depth	Magnitude	Type
1	9.45	6.6	earthquake
2	13.00	6.3	earthquake
3	13.00	7.0	earthquake

また、NOTE 関数、CUROBS 関数、POINT 関数、FETCHOBS 関数を使用して、オブザベーション番号でオブザベーションに直接アクセスすることもできます。

要点

- OBS=データセットオプションが処理の終了点を指定する場合、FIRSTOBS=データセットオプションは開始点を指定します。2 つのオプションは、処理するオブザベーションの範囲を定義するために一緒に使用されることがよくあります。
- OBS=データセットオプションを使用すると、SAS データセットからオブザベーションを選択できます。INFILE ステートメントの OBS=オプションを使用すると、外部データファイルから読み取るオブザベーションを選択できます。

関連項目

- “FIRSTOBS= システムオプション” (*SAS システムオプション: リファレンス*)および“OBS= システムオプション” (*SAS システムオプション: リファレンス*)
-

例: POINT=オプションを使用してオブザベーションを直接読み取る

コード例

この例では、DATA ステップが Sashelp.Comet データセットの 3 行目に直接アクセスします。

一時数値変数 num は、Sashelp.Comet データセットから直接アクセスされるオブザベーションの値を保持するために作成されます。割り当てステートメントは変数を作成し、Sashelp.Comet データセットの 3 番目のオブザベーションを割り当てます。次に、SET ステートメントで POINT=オプションが指定され、一時変数 num と等しく設定されます。OUTPUT ステートメントは、現在のオブザベーションを出力データセット Comet に書き込みます。STOP ステートメントは、DATA ステップの継続的な処理を防止するために使用されます。

CALL SYMPUT ルーチンは、行番号の値をマクロ変数に割り当て、TITLE ステートメントで使用して、アクセスされている行を示すことができます。

次の最初の PRINT プロシジャは、DATA ステップによって 3 行目が直接アクセスされる入力データセットの最初の数行を表示するために使用されます。

```
proc print data=sashelp.comet(obs=5);
title "Sashelp.Comet Data Set";


run;

data comet;
  num=3;
  set sashelp.comet point=num;
  call symput('num',num);
  output;
  stop;
run;

proc print data=comet;
title "Row &num from Sashelp.Comet Data Set";
```

```
run;
```

アウトプット 13.13　sachelp.comet データセットの部分的な PROC PRINT 出力 (比較用)

Comet Data Set

Obs	Dose	Rat	Sample	Length
1	0	1	1	15.3527
2	0	1	1	16.1826
3	0	1	1	14.9378
4	0	1	1	12.4481
5	0	1	1	12.8631

アウトプット 13.14　直接アクセスされた行 3 を示す Comet データセットの PROC PRINT 出力

Row 3 of Comet Data Set

Obs	Dose	Rat	Sample	Length
1	0	1	1	14.9378

要点

- 特に指示しない場合、SAS は入力データセットのすべての変数とすべてのオブザベーションを出力データセットに書き込みます。
- SAS ステートメント、データセットオプション、および関数を使用して、どの変数やオブザベーションを読み書きするかを制御できます。“SAS データセット内の変数を管理する方法” (301 ページ)を参照してください。
- POINT=オプションを使用してオブザベーションに直接アクセスする場合、SAS はファイルの終わりを検出しないため、DATA ステップの継続的な処理を防ぐために、POINT=オプションを使用して STOP ステートメントを指定する必要があります。
- WHERE ステートメントは、変数の値に基づいて、SET ステートメントによってどのオブザベーションが読み取られるかを制御します。

関連項目

- POINT=オプション
- STOP ステートメント
- OUTPUT ステートメント

例: データセットのディスクリプター情報の並べ替えと表示

例: SORTEDBY=データセットオプションを使用したデータセットの並べ替え

コード例

この例では、SORTEDBY=データセットオプションを使用して、priority と indate によってデータを並べ替えています。CONTENTS プロシジャは、並べ替えられた出力データセットに関する情報を表示します。

```
data sorttest (sortedby=priority descending indate);
  input priority indate date7. office $ code $;
  format indate date7.;
  datalines;
1 03may01 CH J8U
1 21mar01 LA M91
1 01dec00 FW L6R
1 27feb99 FW Q2A
2 15jan00 FW I9U
2 09jul99 CH P3Q
3 08apr99 CH H5T
3 31jan99 FW D2W
;
proc contents data=sorttest; run;
```

アウトプット 13.15　Sorttest データセットの並べ替え情報を示す PROC CONTENTS 出力

The CONTENTS Procedure

Data Set Name	WORK.SORTTEST	Observations	8
Member Type	DATA	Variables	4
Engine	V9	Indexes	0
Created	04/13/2020 19:28:06	Observation Length	32
Last Modified	04/13/2020 19:28:06	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	YES
Label			
Data Representation	SOLARIS_X86_64, LINU		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	2038
Obs in First Data Page	8
Number of Data Set Repairs	0
Filename	/opt/sas/viya/config/var/tmp/compsrv/default/70faad
Release Created	V.0400M0
Host Created	Linux
Inode Number	130663431
Access Permission	rw-r--r--
Owner Name	UNKNOWN
File Size	128KB
File Size (bytes)	131072

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format
4	code	Char	8	
2	indate	Num	8	DATE7.
3	office	Char	8	
1	priority	Num	8	

Sort Information	
Sortedby	priority DESCENDING indate
Validated	NO
Character Set	ASCII

要点

- データセットが並べ替えられた後、*並べ替えインジケーター*が、データセットのデータセットディスクリプター情報に追加されます。これは、CONTENTS プロシジャを指定することで確認できます。
- SORTEDBY=データセットオプションは、**検証済み**並べ替え情報を **NO** に設定し、**ソート順**フィールドを更新します。

関連項目

- “並べ替えられたデータセット” (307 ページ)

例: データセットのディスクリプター情報の表示

コード例

この例では、CONTENTS プロシジャは SAS データセット Sashelp.Snacks に関する情報を表示します。

```
proc contents data=sashelp.snacks;
run;
```

アウトプット 13.16 sashelp.snacks データセットのディスクリプター情報を示す PROC CONTENTS 出力

The CONTENTS Procedure

Data Set Name	SASHELP.SNACKS	Observations	35770
Member Type	DATA	Variables	6
Engine	V9	Indexes	0
Created	06/25/2018 03:57:43	Observation Length	80
Last Modified	06/25/2018 03:57:43	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	YES
Label	Daily snack food sales		
Data Representation	WINDOWS_64		
Encoding	us-ascii ASCII (ANSI)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	44
First Data Page	1
Max Obs per Page	817
Obs in First Data Page	796
Number of Data Set Repairs	0
ExtendObsCounter	YES
Filename	c:\snacks.sas7bdat
Release Created	9.0501M0
Host Created	X64_SR12R2
Owner Name	sasuser
File Size	3MB
File Size (bytes)	2949120

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
3	Advertised	Num	8		Advertised (1=yes)
5	Date	Num	8	DATE9.	Date of sale
4	Holiday	Num	8		Holiday (1=yes)
2	Price	Num	8		Retail price of product
6	Product	Char	40		Product name
1	QtySold	Num	8		Quantity sold

Sort Information	
Sortedby	Product Date
Validated	YES
Character Set	ANSI

要点

- ディスクリプター情報は、データセットの内容に関する情報を含む SAS データセットの一部です。
- ディスクリプター情報には、オブザベーションの数、オブザベーションの長さ、データセットが最後に変更された日付、およびその他の事実が含まれます。

- 個々の変数のディスクリプター情報には、名前、型、長さ、出力形式、ラベル、変数にインデックスが付けられているかどうかなどの属性が含まれます。
- データセットが並べ替えられると、並べ替え情報テーブルには、データの並べ替えに使用された BY 変数が表示されます。

関連項目

- “CONTENTS プロシジャ” (*Base SAS プロシジャガイド*)
- “OPTIONS ステートメント” (*SAS グローバルステートメント: リファレンス*)

例: データセットの並べ替え情報の表示

コード例

この例では、CONTENTS プロシジャを使用して、Sashelp.Air データセットに関する情報を表示します。PROC CONTENTS 出力の**ソート済み**フィールドは、データセットが並べ替えられていないことを示します。したがって、出力には追加の**並べ替えインジケーター**テーブルはありません。

```
proc contents data=sashelp.air; run;
```

アウトプット 13.17　データセット sashelp.air が並べ替えられていないことを示す PROC CONTENTS 出力

The CONTENTS Procedure

Data Set Name	SASHELP.AIR	Observations	144
Member Type	DATA	Variables	2
Engine	V9	Indexes	0
Created	06/03/2018 21:41:44	Observation Length	16
Last Modified	06/03/2018 21:41:44	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label	airline data (monthly: JAN49-DEC60)		
Data Representation	WINDOWS_64		
Encoding	us-ascii ASCII (ANSI)		

次の DATA ステップでは、Sashelp.Air データセットからデータセット air が作成され、変数 air によって並べ替えられます。CONTENTS プロシジャを再度使用して、並べ替え情報を表示します。

```
data air(sortedby=air);
```

```
  set sashelp.air;
run;

proc contents data=air; run;
```

PROC CONTENTS 出力は、データセットが SORTEDBY=データセットオプションを使用して並べ替えられたことを示します。**ソート情報**テーブル(並べ替えインジケーター)が含まれるようになりました。**ソート済み**フィールドが **NO** に設定されていることに注意してください。これは、データセットの並べ替えに SORTEDBY=データセットオプションが使用されたためです。並べ替え情報は、データセットが PROC SORT または PROC SQL を使用して並べ替えられている場合にのみ有効な並べ替えを示します。

アウトプット 13.18　データセットが SORTEDBY を使用して並べ替えられたことを示す PROC CONTENTS 出力

The CONTENTS Procedure

Data Set Name	WORK.AIR	Observations	144
Member Type	DATA	Variables	2
Engine	V9	Indexes	0
Created	06/07/2018 19:15:45	Observation Length	16
Last Modified	06/07/2018 19:15:45	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	YES
Label			
Data Representation	WINDOWS_64		
Encoding	wlatin1 Western (Windows)		

Sort Information	
Sortedby	AIR
Validated	NO
Character Set	ANSI

次に、SORT プロシジャを使用して、変数 air の値によってデータセットを降順に並べ替えます。

```
proc sort data=air; by descending air; run;
proc contents data=air; run;
```

アウトプット 13.19　検証済み並べ替えを示す部分的な PROC CONTENTS 出力

The CONTENTS Procedure

Data Set Name	WORK.AIR	Observations	144
Member Type	DATA	Variables	2
Engine	V9	Indexes	0
Created	06/07/2018 19:19:44	Observation Length	16
Last Modified	06/07/2018 19:19:44	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	YES
Label			
Data Representation	WINDOWS_64		
Encoding	wlatin1 Western (Windows)		

Sort Information	
Sortedby	DESCENDING AIR
Validated	YES
Character Set	ANSI

PROC CONTENTS 出力には、データセットが並べ替えられ、並べ替えが検証されたことが示されます。Validated フィールドの YES は、データが PROC SORT または PROC SQL を使用して SAS によって並べ替えられたことを示します。

要点

- 並べ替えインジケーターは、データセットが次のいずれかの方法で並べ替えられるときに設定されます。
  - SORT プロシジャ
  - ORDER BY 句(PROC SQL)
  - MODIFY ステートメント(PROC DATASETS)
  - SORTEDBY=データセットオプション(DATA ステップ DATA ステートメント)
- PROC SORT および PROC SQL は検証済み並べ替えを生成します。これは、ディスクリプター情報の**検証済み**フィールドで確認できます。
- SORTEDBY=データセットオプション と PROC DATASETS の SORTEDBY=オプションでは、検証済み並べ替えは生成されません。
- 一部の SAS プロシジャでは、PROC ステップで処理する前にデータセットを並べ替える必要があります。パフォーマンスを向上させるために、これらのプロシジャはデータセットのディスクリプター情報(並べ替えインジケーターフィールド)の値をチェックして、データが並べ替えられていることを確認します。行が並べ替えられていることを確認するためにデータセット全体を読み取る必要がないため、このプロシジャはより効率的に実行されます。
- OPTIONS ステートメントで SORTVALIDATE システムオプションを指定して、並べ替えを検証し、データセットが BY ステートメントの変数に従って並べ替えられて

いることを確認できます。データセットが正しく並べ替えられていない場合、SORTVALIDATE により、SAS はデータセットを自動的に並べ替えられます。

関連項目

- “CONTENTS プロシジャ” (*Base SAS プロシジャガイド*)

14

生データ

生データの定義

raw data
SAS データセットに読み取られていない未処理のデータ。IMPORT プロシジャまたは SAS DATA ステップを使用して、生データを SAS データセットに読み取ることができます。SAS が読み取ることができる生データのタイプは次のとおりです。

- インストリームデータ
- 外部ファイル

生データの種類

生データの種類

データ値
文字値または数値です。

数値
数字のみを含み、場合によっては小数点、マイナス記号、またはその両方を含みます。SAS データセットに読み取られると、数値は動作環境固有の浮動小数点形式で保存されます。非標準の数値には、他の文字を数字として含めることができます。フォーマットされた入力を使用して、SAS がそれらを読み取れるようにすることができます。

文字値
文字値は文字列です。

標準データ
リスト、カラム、フォーマット、または名前付き入力で読み取ることができる文字または数値です。標準データの例は次のとおりです。

- ARKANSAS
- 1166.42

非標準データ
入力形式を使用して読み取り専用にできるデータです。非標準データの例には、カンマ、ドル記号または空白、日時値、16 進数値と 2 進数値が含まれます。

数値データ

数値データは、いくつかの方法で表すことができます。標準数値データを読み取る場合、特殊な命令を指定する必要はありません。非標準数値データを読み取るには、入力形式の形式で特殊な命令を指定する必要があります。すべての SAS 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。

表 14.1 標準数値データの読み取り

データ	説明	対策
23	入力右揃え	不要
23	入力非整列	不要
23	入力左揃え	不要
00023	先頭にゼロを含む入力	不要
23.0	小数点付き入力	不要
2.3E1	指数表記。2.30 (ss1)	不要
230E-1	指数表記。230x10 (ss-1)	不要
-23	負の数のマイナス記号	不要

表 14.2 非標準数値データの読み取り

データ	説明	対策
2 3	空白を含む	COMMA.入力形式、または BZ.入力形式
- 23	空白を含む	COMMA.入力形式、または BZ.入力形式
2,341	カンマ	COMMA.入力形式
(23)	かっこ	COMMA.入力形式
C4A2	16 進値	HEX.入力形式
1MAR90	日付値	DATE.入力形式

表 14.3　無効な数値データの読み取り

データ	説明	対策
23 -	数字の後のマイナス記号	数値の前にマイナス記号を付けるか、プログラムで処理してください。S370FZDTw.d 入力形式を使用できる可能性がありますが、正の値には末尾のプラス記号(+)が必要です。
..	単一ピリオドの代わりに二重ピリオド	欠損値を単一のピリオドとするプログラムを記述するか、INPUT ステートメントで??修飾子を使用して、無効な入力値を欠損値とするプログラムを記述します。
J23	数ではない	文字値として読み取るか、生データを編集して有効な数値に変更します。

数値データを読み取る場合は、次の規則に注意してください。

- 数値の前にあるかっこまたはマイナス記号(空白を挟まない)は、負の値を示します。
- 先頭の 0 と入力フィールド内の値の配置は、変数に割り当てられる値には影響しません。先頭の 0、先頭および末尾の空白は値とともに保存されません。一部の言語とは異なり、SAS はデフォルトでは末尾の空白をゼロとして読み取りません。末尾の空白をゼロとして読み取るには、BZ.入力形式を使用します。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。
- 数値データには先頭と末尾に空白を含めることができますが、埋め込み空白を含めることはできません(COMMA.または BZ.入力形式で読み取られる場合を除く)。
- 明示的な小数点を含まない入力行から 10 進数値を読み取るには、カラム入力で 10 進数パラメーターを使用するか、フォーマットされた入力で入力形式を使用して、小数点が属する場所を指定します。詳細については、*SAS 出力形式と入力形式: リファレンス*の INPUT ステートメントの完全な説明を参照してください。入力データ内の明示的な小数点は、INPUT ステートメントの小数点指定をオーバーライドします。

文字データ

次の条件のいずれかに該当する場合、INPUT ステートメントで読み取られる値は文字値であると見なされます。

- INPUT ステートメントの変数名の後にドル記号($)が付く。
- 文字入力形式の使用。
- 以前に文字として定義されていた変数。たとえば、変数が以前に LENGTH ステートメント、RETAIN ステートメント、割り当てステートメント、または式で文字として定義されている場合、値は文字値であると見なされます。

文字変数に保存する入力データには、任意の文字を含めることができます。生データに先頭の空白とセミコロンが含まれている場合は、次の表のガイドラインに従ってください。

表 14.4 先頭の空白とセミコロンを含む入力ストリームデータと外部ファイルの読み取り

データの文字	使用方法	理由
先頭または末尾の空白の保持	フォーマットされた入力および$CHAR*w*.入力形式	リスト入力は、値が変数に割り当てられる前に、文字値から先頭と末尾の空白を削除します。
入力ストリームデータのセミコロン	DATALINES4 または CARDS4 ステートメントと、データの終わりを示す 4 つのセミコロン(;;;;)	通常の DATALINES および CARDS ステートメントでは、データ内のセミコロンがデータの終わりを途中で通知します。
区切り文字、空白文字、または引用符で囲まれた文字列	INFILE ステートメントで、DLM=または DLMSTR=オプションを使用した DSD オプション	これらのオプションにより、SAS は、引用符で囲まれた文字列内に区切り記号を含む文字値を読み取ることができます。これらのオプションは、2 つの連続する区切り文字を欠損値として扱い、文字値から引用符を削除することもできます。

文字データを読み取るときは、次の点に注意してください。

- DATA ステップで、INPUT ステートメントの変数名の後にドル記号($)を置くと、データ行から読み取られる文字データは元の大文字と小文字がそのまま残ります。SAS にデータ行からデータを大文字で読み取る場合は、CAPS システムオプションまたは$UPCASE 入力形式を使用します。
- 値が変数の長さより短い場合、SAS は値の末尾に空白を追加して、値を指定された長さにします。このプロセスは、値に空白を埋め込むと呼ばれます。

生データの読み取り

生データを SAS データセットに読み取る方法

次の項目のいずれかを使用して生データを読み取ることができます。

- インポートプロシジャ

  次の例を参照してください。

  - “区切り文字で区切られたファイルのインポート” (*Base SAS プロシジャガイド*)
  - “ファイル参照名を使用した、特定の区切り文字で区切られたファイルのインポート” (*Base SAS プロシジャガイド*)
  - “タブ区切り文字で区切られたファイルのインポート” (*Base SAS プロシジャガイド*)
- DATA ステップステートメント (347 ページ)
- FOPEN、 FGET、FCLOSE 関数などの SAS I/O 関数。FCLOSE

  使用可能な関数の説明については、“カテゴリ別の SAS 関数と CALL ルーチン” (*SAS 関数と CALL ルーチン: リファレンス*)の SAS ファイル I/O および外部ファイルカテゴリを参照してください。ステートメントと関数がファイルを操作する方法については、“関数を使用したファイルの操作” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

生のインストリームデータの読み取り

インストリームデータの定義

Instream data
: SAS プログラム内に含まれる生データレコードの行。インストリームデータは、DATALINES ステートメントで始まり、SAS DATA ステップの新しい行のセミコロンで終わります。

インストリームデータの読み取り

この例では、INPUT ステートメントと DATALINES ステートメントを使用してインストリームデータを読み取ります。INPUT ステートメントは変数(列)名を指定し、DATALINES ステートメントは従うべき生のインストリームデータを示します。

```
data weight;
  input PatientID $ State $;
  loss=Week1-Week16;
  datalines;
2477 TX
2431 TX
2456 NC
2412 NC
;
```

注: 最後のデータ行の直後にセミコロンが単独で現れるのが、この例で使用されている規則です。ただし、最後のデータ行の直後にあるセミコロンで終わる PROC ステートメント、DATA ステートメント、またはグローバルステートメントは、前の DATA ステップもサブミットします。

セミコロンを含むインストリームデータの読み取り

この例では、セミコロンを含むインストリームデータを読み取ります。

```
data weight;
  input PatientID $ Week1 Week8 Week16;
  loss=Week1-Week16;
  datalines4;
24;77 195 177 163
24;31 220 213 198
24;56 173 166 155
24;12 135 125 116
;;;;
```

DATA ステップ(INPUT ステートメント)による生データの読み取り

SAS DATA ステップは、インストリームデータ行または外部ファイルから生データを読み取ります。DATA ステップの INPUT ステートメントは、これらのソースから SAS データセットにデータを読み取ります。

入力スタイルの選択

DATA ステップで生データを読み取る場合は、INPUT ステートメント、 DATALINES ステートメント、および INFILE ステートメントを組み合わせて使用できます。レコード内のデータ値のレイアウトに応じて、次のさまざまな入力スタイルを使用できます。

- リスト入力
- “修飾リスト入力” (349 ページ)
- カラム入力
- フォーマット入力
- 名前付き入力

単一の INPUT ステートメントで入力のスタイルを組み合わせることもできます。入力のスタイルの詳細については、INPUT ステートメントを参照してください。

リスト入力

リスト入力では、データ値を見つけるためにスキャン方法を使用します。データ値はカラムで整列する必要はありませんが、少なくとも 1 つの空白(またはその他の定義された区切り文字)で区切る必要があります。リスト入力では、文字変数を定義する場合は、変数名とドル記号($)を指定するだけで済みます。データフィールドの場所を指定する必要はありません。

リスト入力を使用して生のインストリームデータを読み取る例は次のとおりです。

```
data scores;
  length name $ 12;
  input name $ score1 score2;

datalines;
Riley 1132 1187
Henderson 1015 1102
;
```

その他の例については、“単純リスト入力を使用した位置が揃っていないデータの読み込み” (*SAS DATA ステップステートメント: リファレンス*)を参照してください。

リスト入力には、読み取ることができるデータのタイプに関していくつかの制限があります。

- 入力値は、少なくとも 1 つの空白(デフォルトの区切り文字)または INFILE ステートメントの DLM=または DLMSTR=オプションで指定された区切り文字で区切る必要があります。SAS が連続する区切り文字を、それらの間に欠損値があるかのように読み取るようにする場合は、INFILE ステートメントで DSD オプションを指定します。
- 空白は欠損値を表すことはできません。かわりにピリオドなどの実数値を使用する必要があります。

- 8 バイトを超える文字値を読み取って保存するには、その長さを明示的に定義する必要があります。変数の長さは、LENGTH ステートメント、 INFORMAT ステートメント、または ATTRIB ステートメントのいずれかを使用して定義できます。これらのステートメントのいずれかを使用して文字変数を定義する場合は、変数の長さを定義するステートメントを、その変数への他の参照の前に、DATA ステップの最初に配置する必要があります。INPUT ステートメント内の入力形式とコロン修飾子で構成される修飾リスト入力を使用して、変数の長さを指定することもできます。
- ファイルが空白で区切られている場合、文字値に空白を含めることはできません。
- フィールドは順番に読み取る必要があります。
- データは標準の数値または文字形式である必要があります。

注: パック 10 進データなどの非標準数値には、フォーマットされた入力スタイルを使用する必要があります。“フォーマット入力” (351 ページ)を参照してください。

修飾リスト入力

リスト入力のより柔軟なバージョンは、修飾リスト入力と呼ばれ、フォーマット修飾子を含みます。次のフォーマット修飾子を使用すると、リスト入力で SAS 入力形式を使用して非標準データを読み取ることができます。

- **&** (アンパサンド)フォーマット修飾子を使用すると、リスト入力で 1 つ以上の空白を含む文字値を読み取り、文字入力形式を指定できます。SAS は、2 つの連続する空白、変数の定義された長さ、または入力行の終わりのいずれかが最初に現れるまで読み取ります。
- **:** (コロン)フォーマット修飾子を使用すると、リスト入力を使用できるだけでなく、変数名の後に文字または数値の入力形式を指定することもできます。SAS は、空白の列、変数の定義された長さ(文字のみ)、またはデータ行の終わりのいずれかが先に現れるまで読み取ります。
- **~** (チルダ)フォーマット修飾子を使用すると、文字値内の単一引用符、二重引用符、および区切り文字を読み取り、保持できます。

次は、:および~フォーマット修飾子の例です。INFILE ステートメントでは DSD オプションを使用する必要があります。それ以外の場合、INPUT ステートメントは~フォーマット修飾子を無視します。この例では、修飾リスト入力を使用して生のインストリームデータを読み取ります。

```
data scores;
  infile datalines dsd;
  input Name : $9. Score1-Score3 Team ~ $25. Div $;

datalines;
Smith,12,22,46,"Green Hornets, Atlanta",AAA
Mitchel,23,19,25,"High Volts, Portland",AAA
Jones,09,17,54,"Vulcans, Las Vegas",AA
;
proc print data=scores;
```

Obs	Name	Score1	Score2	Score3	Team	Div
1	Smith	12	22	46	"Green Hornets, Atlanta"	AAA
2	Mitchel	23	19	25	"High Volts, Portland"	AAA
3	Jones	9	17	54	"Vulcans, Las Vegas"	AA

カラム入力

カラム入力を使用すると、データレコードのカラムに整列された標準データ値を読み取ることができます。変数名を指定し、文字変数の場合はドル記号($)を続けて、各レコード内のデータ値が配置されるカラムを指定します。

```
data scores;
  infile datalines truncover;
  input name $ 1-12 score2 17-20 score1 27-30;

datalines;
Riley         1132      987
Henderson     1015     1102
;
```

その他の例については、例を参照してください。

注: INFILE ステートメントで TRUNCOVER オプションを使用すると、SAS がさまざまな長さのデータ値を適切に処理できるようになります。

カラム入力を使用するには、データ値は次のとおりである必要があります。

- すべての入力行の同じフィールドに存在する
- 標準の数値または文字形式である

注: カラム入力で入力形式を使用することはできません。

カラム入力の機能は次のとおりです。

- 文字値には空白を含めることができます。
- 文字値の長さは 1-32,767 文字です。
- 単一のピリオド(.)などのプレースホルダーは、欠損データには必要ありません。
- 入力値は、レコード内の位置に関係なく、任意の順序で読み取ることができます。
- 値または値の一部を再読み取りできます。
- フィールド内の先頭と末尾の空白は両方とも無視されます。
- 値を空白やその他の区切り文字で区切る必要はありません。

注意

DATALINES ステートメントにカラム形式でデータを入力するときにタブを挿入すると、予期しない結果が生じる可能性があります。この問題は、SAS 拡張エディタ

ーまたは SAS プログラムエディターを使用する場合に発生します。この問題を回避するには、次のいずれかを実行します。

- SAS 外部の別のエディターを使用して、データ内のすべてのタブを単一のスペースに置き換えます。
- SAS エディターから%INCLUDE ステートメントを指定してコードをサブミットします。
- SAS 拡張エディターを使用している場合は、**ツール** ⇨ **オプション** ⇨ **拡張エディター**を選択してタブサイズを 4 から 1 に変更します。

フォーマット入力

フォーマット入力は、入力形式を使用する柔軟性とカラム入力の多くの機能を組み合わせたものです。フォーマット入力を使用すると、SAS が追加の命令を必要とする非標準データを読み取ることができます。フォーマット入力は通常、データを読み取るときに入力バッファー内の入力ポインターの位置を制御できるポインターコントロールとともに使用されます。

次の DATA ステップの INPUT ステートメントは、フォーマット入力およびポインターコントロールを使用して、DATALINES ステートメントにリストされている生のインストリームデータを読み取ります。

$12.と COMMA5.は入力形式、+4 と+6 はカラムポインターコントロールであることに注意してください。

```
data scores;
  input name $12. +4 score1 comma5. +6 score2 comma5.;

datalines;
Riley         1,132     1,187
Henderson     1,015     1,102
;
```

その他の例については、""例"" (*SAS DATA ステップステートメント: リファレンス*)を参照してください。

注: 入力形式を使用して、カラムに整列していないデータを読み取ることもできます。詳細については、"修飾リスト入力" (349 ページ)を参照してください。

フォーマット入力に関する重要な点は次のとおりです。

- 文字値には空白を含めることができます。
- 文字値の長さは 1-32,767 文字です。
- 単一のピリオド(.)などのプレースホルダーは、欠損データには必要ありません。
- ポインターコントロールを使用してポインターを配置すると、レコード内の位置に関係なく、入力値を任意の順序で読み取ることができます。
- 値または値の一部を再読み取りできます。

- フォーマット入力を使用すると、パック 10 進数やカンマ付き数値など、非標準形式で保存されたデータを読み取ることができます。

名前付き入力

名前付き入力を使用すると、データ値の前に変数名と等号(=)が付いているレコードを読み取ることができます。次の INPUT ステートメントは、等号を含むデータ行を読み取ります。

```
data games;
  input name=$ score1= score2=;

datalines;
name=riley score1=1132 score2=1187
;
```

その他の例については、*""例"" (SAS DATA ステップステートメント: リファレンス)*を参照してください。

注: INPUT ステートメントの変数の後に等号が続く場合、SAS は入力行に残っているデータには名前付き入力値のみが含まれていると想定します。名前付き入力を使用した後は、同じ INPUT ステートメント内で別の形式の入力に切り替えることはできません。また、入力データに存在する変数が INPUT ステートメントで定義されていない場合、フィールドが欠損していることを示す注記が SAS ログに生成されることに注意してください。

追加のデータ読み取り機能

さまざまな入力スタイルに加えて、さまざまなデータ読み取り状況のニーズを満たすツールが多数あります。INFILE ステートメントのオプションを INPUT ステートメントと組み合わせて使用すると、データレコードの読み取りをさらに制御できます。次の表に、一般的なデータ読み取りタスクと、INPUT ステートメントと INFILE ステートメントで使用できる適切な機能を示します。

表 14.5 追加のデータ読み取り機能

入力	目標	使用
複数レコード	単一のオブザベーションを作成する	DO ループを使用した INPUT ステートメント内の #n または/行ポインターコントロール。
単一レコード	複数のオブザベーショ	INPUT ステートメントの後置@@、 複数の INPUT ステートメント OUTPUT ステートメントで使用する後置@。

入力	目標	使用
	ンを作成する	(例)
可変長データフィールドとレコード	区切りデータを読み取る	INPUT ステートメントのフォーマット修飾子を使用または使用しないリスト入力、および INFILE ステートメントの TRUNCOVER、DLM=、DLMSTR=、または DSD オプション。 (例)
	非区切りデータを読み取る	INPUT ステートメントの$VARYING*w*.入力形式と、INFILE ステートメントの LENGTH=および TRUNCOVER オプション。 (例)
さまざまなレコードレイアウトを持つファイル		複数の INPUT ステートメントを含む IF-THEN ステートメント。必要に応じて後置@または@@を使用します。
階層ファイル		複数の INPUT ステートメントを含む IF-THEN ステートメント。必要に応じて後置@を使用します。
複数の入力ファイルか、または EOF でプログラムフローを制御		INFILE ステートメントの EOF=オプションまたは END=オプション。 (例) 複数の INFILE ステートメントと INPUT ステートメント。 INFILE ステートメントの FILEVAR=。 (例) 連結、ワイルドカード、またはパイピングを使用した FILENAME ステートメント。
各レコードの一部のみ		INFILE ステートメントの LINESIZE=オプション。
ファイル内のすべてのレコードではなく一部のレコード		INFILE ステートメントの FIRSTOBS=および OBS=オプション、FIRSTOBS=および OBS=システムオプション、#n 行ポインターコントロール。 (例)
インストリームデータ行	読み取りを制御する (特別オプション)	DATALINES と適切なオプションを指定した INFILE ステートメント。

入力	目標	使用
特定のカラムから開始		@カラムポインターコントロール。
先頭の空白	維持する	INPUT ステートメントの$CHARw.入力形式。
空白以外の区切り文字(リスト入力または コロン修飾子を使用する修飾リスト入力)		INFILE ステートメントの DLM=または DLMSTR= オプション、DSD オプション、あるいはその両方。
標準タブ文字		INFILE ステートメントの DLM=または DLMSTR= オプション、あるいは INFILE ステートメントの EXPANDTABS オプション。
欠損値(リスト入力または コロン修飾子を使用する修飾リスト入力)	データの一貫性を損なうことなくオブザベーションを作成する デフォルトの動作をオーバーライドしてデータの一貫性を保護する	INFILE ステートメントの TRUNCOVER オプション。DLM=、 DLMSTR=、または DSD も必要になる場合があります。

データ読み取り機能の詳細については、*SAS DATA ステップステートメント: リファレンス*の INPUT および INFILE ステートメントを参照してください。

SAS が無効なデータを処理する方法

入力値が次のいずれかの特性を持つ場合、その値は無効です。

- 指定されていない入力形式が必要です。
- 指定された入力形式に準拠していません。
- 使用されている入力スタイルと一致しません。例としては、標準数値データ(ドル記号や入力形式なし)として読み取られるが、標準の SAS 数値の規則に準拠していない場合があります。
- 範囲外です(大きすぎるか小さすぎます)。

SAS がその変数に指定された型と互換性のないデータ値を読み取った場合、SAS はその値を指定された型に変換しようとします。変換できない場合はエラーが発生し、SAS は次のアクションを実行します。

- 読み取られる変数の値を欠損値、または INVALIDDATA=システムオプションで指定された値に設定します。
- 無効なデータの注記を SAS ログに出力します。
- 現在のオブザベーションに対して自動変数_ERROR_を 1 に設定します。
- SAS ログ内の無効な値を含む入力行とカラム番号を出力します。行に表示不能文字が含まれている場合は、16 進形式で出力されます。カラム番号を決定しやすくするために、入力行の上にスケールが出力されます。

SAS が欠損値を処理する方法

入力データ内の欠損値の表現

データのコレクションの多くには、欠損値が含まれています。SAS は、これらの値を読み取るときに、これらの値が欠損していると認識できます。生データを読み取るときに欠損値を表すには、次の文字を使用します。

欠損数値
: 単一の小数点(.)で表されます。リスト入力を除くすべての入力スタイルでは、欠損数値を空白で表すこともできます。

文字欠損値
: 1 つの例外を除き、空白で表されます。リスト入力では、欠損値を表すためにピリオド(.)を使用する必要があります。

特殊欠損数値
: 小数点(.)の後に文字またはアンダースコア(_)が続く 2 つの文字で表されます。

欠損値の詳細については、“欠損値” (95 ページ)を参照してください。

数値入力データ内の特殊欠損値

SAS を使用すると、数値データ内の欠損値のクラスを区別できます。数値変数の場合、A - Z (大文字または小文字)とアンダースコア文字(_)を使用して、最大 27 個の特殊欠損値を指定できます。

次の例は、DATA ステップで MISSING ステートメントを使用して欠損値をコーディングする方法を示しています。

```
data test_results;
  missing a b c;
  input name $8. Answer1 Answer2 Answer3;

datalines;
```

```
Smith   2 5 9
Jones   4 b 8
Carter  a 4 7
Reed    3 5 c
;
```

次のように、式または割り当てステートメントで特殊欠損数値を指定する場合は、ピリオドを使用する必要があることに注意してください。

```
x=.d;
```

ただし、入力データ内で特殊欠損数値データ値をピリオドで指定する必要はありません。たとえば、次の DATA ステップでは、入力データ内の特殊欠損値にピリオドを使用しますが、ピリオドなしの入力データと同じ結果が生成されます。

```
data test_results;
  missing a b c;
  input name $8. Answer1 Answer2 Answer3;
  datalines;
Smith   2 5 9
Jones   4 .b 8
Carter  .a 4 7
Reed    3 5 .c
;

proc print;
run;
```

アウトプット 14.1　特殊欠損数値を含むデータの出力

The SAS System

Obs	name	Answer1	Answer2	Answer3
1	Smith	2	5	9
2	Jones	4	B	8
3	Carter	A	4	7
4	Reed	3	5	C

注: SAS が表示され、大文字を使用する特殊欠損値が出力されます。

外部ファイルの定義

binary data
: 外部ファイルにバイナリ形式で保存される数値データ。2 進数は 2 を底とし、0 と 1 の数字で表されます。

column-binary data storage
古い形式のデータストレージは、現在では広く使用されておらず、ほとんどの SAS ユーザーには必要ありません。カラムバイナリデータストレージでは、80 アイテムを超えるデータを保存できるようにデータが圧縮されます。この方法の利点は、同じ量のスペースにより 多くのデータを保存できることです。カードイメージデータセットは存在し続けるため、SAS はカラムバイナリデータを読み取るための入力形式を提供します。カラムバイナリデータストレージの詳細については、“カラムバイナリデータの読み取り” (363 ページ)を参照してください。

external files
SAS ではなくオペレーティングシステムによって 管理および維持されるファイル。これらには、SAS ジョブへの入力としてデータまたはテキストが含まれるか、SAS ジョブの結果として作成される出力ファイルです。

SAS セッションへの入力としての外部ファイルには次のものが含まれます。

- プレーン ASCII テキストファイルやバイナリファイルの形式のデータなど、入力として SAS に読み取る外部ファイル内のデータのレコード。
- 実行のために SAS にサブミットする外部ファイル内のプログラミングステートメント。

SAS セッションからの出力としての外部ファイルには次のものが含まれます。

- プレーン ASCII テキストファイルのレコードとして書き出されたデータやバイナリ形式で書き出されたデータなど、データまたはテキストを保存するファイル。
- SAS ログファイルや SAS プロシジャの結果など、SAS プログラムの実行の結果として作成されるファイル。たとえば、PRINTTO プロシジャを使用すると、プロシジャ出力を外部ファイルに送信できます。すべての SAS ジョブは、少なくとも 1 つの外部ファイルである SAS ログを作成します。SAS カタログファイルや ODS 出力先も外部ファイルの例です。

packed decimal data
各バイトを使用して 2 つの 10 進数を表すことによってエンコードされた 2 進 10 進数。パック 10 進数表現では、10 進データが正確な精度で保存されます。個別の仮数と指数がないため、数値の小数部は入力形式または出力形式を使用して決定する必要があります。

zoned decimal data
各桁に 1 バイトのストレージが必要になるようにエンコードされた 2 進 10 進数。最後のバイトには、数値の符号と最後の数字が含まれます。ゾーン 10 進データは、印刷可能な表現を生成します。

外部ファイルの読み取りと書き込み

DATA ステップを使用した外部ファイルの読み取り

次の例は、INFILE および INPUT ステートメントを使用して、外部ファイルから生データを読み取る方法を示しています。この例は、SAS 入力形式でフォーマット入力を使用して生データを読み取るという点で、フォーマット入力の例に似ています。ただし、この例では、ソースデータが外部テキストファイル内にあるため、DATALINES ステートメントのかわりに INFILE ステートメントが必要であることに注意してください。

```
data weight;
  infile <fileref> or <path-to-file;
  input PatientID $ Week1 Week8 Week16;
  loss=Week1-Week16;
run;
```

外部ファイルへの直接の読み取りと書き込み

SAS ステートメントまたはコマンドでファイルを直接参照するには、その物理名を引用符で囲んで指定します。これは、次の表に示すように、動作環境で認識される名前です。

表 14.6　外部ファイルからのデータの直接読み取り

タスク	言語要素	例

入力データを含むファイルを指定します。	INFILE ステートメント	data weight;
		  infile '*input-file*';
		  input idno $ week1 week16;
		  loss=week1-week16;
		run;

表 14.7　外部ファイルへのデータの直接書き込み

タスク	言語要素	例

PUT ステートメントが書き込むファイルを特定します。	FILE ステートメント	file '*output-file*';
		data status;
		  if loss ge 5 and loss le 9 then
		  put idno loss 'AWARD STATUS=3';
		  else if loss ge 10 and loss le 14 then

タスク	言語要素	例
		put idno loss 'AWARD STATUS=2'; else if loss ge 15 then put idno loss 'AWARD STATUS=1'; run;

表 14.8 プログラミングステートメントを含む外部ファイルを含める

タスク	言語要素	例
別のファイルからステートメントまたは生データを SAS ジョブに取り込んで実行します。	%INCLUDE ステートメント	%include '*source-file*';

ファイル参照名を使用して外部ファイルを間接的に参照する

別のジョブまたは後で実行するためにファイルを簡単に変更できるように、プログラム内の 1 か所だけでファイルを参照する場合は、ファイル名を間接的に参照できます。FILENAME ステートメント、FILENAME 関数、または適切なオペレーティングシステムコマンドを使用して、ファイルにファイル参照名またはニックネームを割り当てます。[1] 次の表に示すように、外部ファイルである SAS カタログまたは出力デバイスにファイル参照名を割り当てることができることに注意してください。

表 14.9 ファイル参照名を使用した外部参照

タスク	言語要素	例

入力データを含むファイルにファイル参照名を割り当てます。	FILENAME ステートメント	filename mydata '*input-file*';
		data weight;
		infile mydata;
		input idno $ week1 week16;
		loss=week1-week16;
		run;
		
		filename mydata '*input-file*';
		proc import datafile=mydata
		out=shoes dbms=dlm replace;
		run;
		proc print data=shoes; run;

出力データのファイルにファイル参	FILENAME ステートメント	filename myreport '*output-file*'; data myreport;

1. 一部の動作環境では、コマンド'&'を使用してファイル参照名を割り当てることもできます。

タスク	言語要素	例

照名を割り当てます。		infile mydata;
		input idno $ week1 week16;
		loss=week1-week16;
		run;
		
		proc export data=sashelp.shoes
		outfile=myreport dbms=dlm replace;
		delimiter=' ';
		run;

プログラムステートメントを含むファイルにファイル参照名を割り当てます。	FILENAME ステートメント	filename mypgm '*source-file*';
ファイル参照名を出力デバイスに割り当てます。	FILENAME ステートメント	filename myprinter <*device-type*> <*host-options*>;
入力データを含むファイルを指定します。	INFILE ステートメント	data weight; infile mydata; input idno $ week1 week16; loss=week1-week16;
PUT ステートメントが書き込むファイルを指定します。	FILE ステートメント	file myreport; if loss ge 5 and loss le 9 then put idno loss 'AWARD STATUS=3'; else if loss ge 10 and loss le 14 then put idno loss 'AWARD STATUS=2'; else if loss ge 15 then put idno loss 'AWARD STATUS=1'; run;
別のファイルからステートメントまたは生データを SAS ジョブに取り込んで実行します。	%INCLUDE ステートメント	%include mypgm;

多くの外部ファイルを効率的に参照する

ディレクトリやパーティションデータセット (PDS または MACLIB)など、単一の集合保存場所にある多数のファイルを使用する場合、単一のファイル参照名とそれに続くかっこで囲まれたファイル名を使用して、個々のファイルにアクセスできます。これにより、長いファイルの保存場所名を繰り返し入力する必要がなくなり、時間が節約されます。また、ファイルの保存場所を変更すると、後でプログラムを変更

するのも簡単になります。次の表は、ファイル参照名を集合保存場所に割り当てる例を示しています。

表 14.10 多くのファイルを効率的に参照する

タスク	言語要素	例
ファイル参照名を集合保存場所に割り当てます。	FILENAME ステートメント	filename mydir '*directory-or-PDS-name*';

入力データを含むファイルを指定します。	INFILE ステートメント	data weight;
		infile mydir(qrt1.data);
		input idno $ week1 week16;
		loss=week1-week16;
		run;

PUT ステートメントが書き込むファイルを指定します。[1]	FILE ステートメント	file mydir(awards); if loss ge 5 then put idno loss 'AWARD STATUS=3'; else if loss ge 10 then put idno loss 'AWARD STATUS=2'; else if loss ge 15 then put idno loss 'AWARD STATUS=1'; run;
別のファイルからステートメントまたは生データを SAS ジョブに取り込んで実行します。	%INCLUDE ステートメント	%include mydir(whole.program);

1 SAS は、動作環境に適した拡張子が付いた名前のファイルを作成します。

他のアクセス方式による外部ファイルの参照

次の FILENAME アクセス方式を使用してアクセスする外部ファイルにファイル参照名を割り当てることができます。

表 14.11 他のアクセス方式による外部ファイルの参照

タスク	言語要素	例
集合保存場所である SAS カタログにファイル参照名を割り当てます。	FILENAME ステートメント(CATALOG 指定子あり)	filename mycat catalog '*catalog*' <*catalog-options*>;
データ URL によってアクセスされる外部ファイルにフ	FILENAME ステートメント(DATAURL 指定子あり)	filename myfile dataurl '*external-file*' <*dataurl-options*>;

タスク	言語要素	例
ァイル参照名を割り当てます。		
FTP でアクセスされる外部ファイルにファイル参照名を割り当てます。	FILENAME ステートメント(FTP 指定子あり)	filename myfile FTP '*external-file*' *<ftp-options>*;
Hadoop 分散ファイルシステム上でアクセスされる外部ファイルにファイル参照名を割り当てます。	FILENAME ステートメント(Hadoop 指定子あり)	filename myfile hadoop '*external-file*' *<hadoop-options>*;
クライアントモードまたはサーバーモードで TCP/IP SOCKET によってアクセスされる外部ファイルにファイル参照名を割り当てます。	FILENAME ステートメント(SOCKET 指定子あり)	filename myfile SOCKET '*hostname: portno*' *<tcpip-options>*; または filename myfile SOCKET ':*portno*' SERVER *<tcpip-options>*;
URL によってアクセスされる外部ファイルにファイル参照名を割り当てます。	FILENAME ステートメント(URL 指定子あり)	filename myfile URL '*external-file*' *<url-options>*;
WebDAV サーバー上でアクセスされる外部ファイルにファイル参照名を割り当てます。	FILENAME ステートメント(WebDAV 指定子あり)	filename myfile WEBDAV '*external-file*' *<webdav-options>*;
Zlib サービスを使用してアクセスされる ZIP ファイルにファイル参照名を割り当てます。	FILENAME ステートメント(ZIP 指定子あり)	filename myfile ZIP '*external-file*' *<zip-options>*;

バイナリデータの読み取り

SAS 入力形式を使用したバイナリデータの読み取り

SAS は、SAS 入力形式によって提供される特別な命令を使用してバイナリデータを読み取ることができます。フォーマット入力を使用し、INPUT ステートメントで入力形式を指定できます。選択する入力形式は、次の要素によって決まります。

- 読み取られる数値のタイプ: 2 進数、パック 10 進数、ゾーン 10 進数、またはこれらのいずれかのバリエーション
- データが作成されたシステムのタイプ
- データの読み取りに使用するシステムのタイプ

コンピュータープラットフォームが異なれば、数値バイナリデータは異なる形式で保存されます。バイトの順序は、"ビッグエンディアン"または"リトルエンディアン"と呼ばれるプラットフォームによって異なります。詳細については、*“ビッグエンディアンプラットフォームとリトルエンディアンプラットフォーム 上でのバイナリ整数データのバイトオーダリング” (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。

バイナリデータを読み取る SAS プログラムを作成し、1 種類のシステムのみで実行する場合は、ネイティブモードの入力形式と出力形式を使用できます。ただし、異なるバイト格納方式を使用する複数のシステム上で実行できる SAS プログラムを作成する場合は、IBM 370 入力形式を使用します。IBM 370 入力形式を使用すると、この形式でデータを読み取ることができ、数値データの保存標準に関係なく、任意の SAS 環境で実行できる SAS プログラムを作成できます。

注: テキストファイルがローカルエンコーディング環境以外の場所から生成された場合は、EBCDIC または ASCII システムのいずれかで ENCODING=データセットオプションを指定する必要がある場合があります。ASCII プラットフォームで EBCDIC テキストファイルを読み取る場合は、FILENAME または INFILE ステートメントで ENCODING=オプションを指定することをお勧めします。ただし、INFILE ステートメントで DSD=オプション、、DLM=オプション、または DLMSTR=オプションを使用する場合は、これらのオプションがセッションエンコーディングで特定の文字(引用符、カンマ、空白など)を使用するため、ENCODING=オプションが必要です。文字フィールドと非文字フィールドの両方を含む真のバイナリファイルで使用するために、エンコーディング固有の入力形式を予約します。

カラムバイナリデータの読み取り

SAS でカラムバイナリデータを読み取るには、次のことを知っておく必要があります。

- 適切な SAS カラムバイナリ入力形式を選択する方法
- INFILE ステートメントで RECFM=および LRECL=オプションを設定する方法
- ポインターコントロールの使用方法

次の表に、SAS カラムバイナリ入力形式のリストと説明を示します。

表 14.12　カラムバイナリデータを読み取るための SAS 入力形式

入力形式名	説明
$CB*w*.	カラムバイナリファイルから標準文字データを読み取る
CB*w*.	カラムバイナリファイルから標準数値データを読み取る
PUNCH.*d*	行がパンチされているかどうかを読み取る
ROW*w.d*	カードカラムの下のカラムバイナリフィールドを読み取る

カラムバイナリデータを読み取るには、INFILE ステートメントで 2 つのオプションを設定する必要があります。

- RECFM=を固定の F に設定します。
- カラムバイナリデータの各カードカラムはフィールドが読み取られる前に 2 バイトに拡張されるため、LRECL=を 160 に設定します。

たとえば、ファイルからカラムバイナリデータを読み取るには、データを読み取る INPUT ステートメントの前に、次の形式で INFILE ステートメントを使用します。

```
data out;
  infile file-specification or path-to-file recfm=f lrecl=160;
  input var1;
run;
```

注: カラムバイナリデータの各カラムを 2 バイトに拡張しても、カラムポインターの位置には影響しません。入力形式は二重化されたレコード上の実際の位置を自動的に計算するため、通常どおり、絶対カラムポインターコントロール@を使用します。値がカラム 23 にある場合は、ポインターコントロール@23 を使用してポインターをそこに移動します。

CAS 内のデータへのアクセス

CAS 内のデータへのアクセスについては、“SAS Cloud Analytic Services を使用したデータへのアクセス” (*SAS Cloud Analytic Services: ユーザーガイド*)を参照してください。

例: CAS テーブルの読み取りと作成

例: CSV ファイルを読み取り、CAS テーブルとして保存する

コード例

次の例は、カンマ区切り値ファイル(CSV)を Web サイトから CAS にロードする方法を示しています。プログラムはまず、FILENAME URL アクセス方式と DATA ステップを使用して、ファイルを SAS に読み込みます。DATA ステップは SAS Compute Server 上で実行されています。DATA ステップがコンパイルされて変数が読み込まれると、CAS エンジンライブラリ参照名である mycas を使用して、データがインメモリ CAS テーブルとして書き出されます。

この例では、names.csv ファイルを使用します。このファイルは、http://support.sas.com/documentation/onlinedoc/viya/examples.htm にあります。

```
filename names url
 "http://support.sas.com/documentation/onlinedoc/viya/exampledatasets/names.csv";

data mycas.names;
 infile names dsd truncover firstobs=2;              /* 1 */
 input BRTH_YR :$10. GNDR :$10. ETHCTY :$10. NM :$10.
       CNT :$10.  RNK :$10.;                         /* 2 */
run;

proc casutil incaslib='casuser';                     /* 3 */
  save casdata='names' outcaslib='casuser' replace;
  list;
run;
```

CSV ファイルとその内容の一部を次に示しています。

```
BRTH_YR,GNDR,ETHCTY,NM,CNT,RNK
2011,FEMALE,HISPANIC,GERALDINE,13,75
2011,FEMALE,HISPANIC,GIA,21,67
2011,FEMALE,HISPANIC,GIANNA,49,42
2011,FEMALE,HISPANIC,GISELLE,38,51
      <more rows>
2014,MALE,WHITE NON HISPANIC,Youssef,24,88
2014,MALE,WHITE NON HISPANIC,Yusuf,16,96
2014,MALE,WHITE NON HISPANIC,Zachary,90,39
2014,MALE,WHITE NON HISPANIC,Zev,49,65
```

1 SAS では、INFILE ステートメントを使用して外部カンマ区切りファイルをロードします。出力テーブルで CAS エンジンライブラリ参照名を指定します。TRUNCOVER オプションを使用すると、SAS は可変長レコードを正しく読み取ることができます。値が割り当てられていない変数は欠損値に設定されます。

2 INPUT ステートメントを指定して列名をリストし、入力形式として読み取ります。

3 インメモリ CAS テーブルの永久コピーを保存します。

注: この時点では、CAS ではデータの処理が行われていないことに注意してください。CAS エンジンライブラリ参照名を使用してデータを CAS にロードする場合(たとえば、DATA ステップまたは CASUTIL プロシジャを使用する場合)、処理は計算サーバーで行われます。

要点

- 出力テーブルで INFILE ステートメントと CAS エンジンライブラリ参照名を指定することにより、DATA ステップを使用して外部ファイルを CAS にロードできます。
- CASUTIL プロシジャを使用して、外部ファイルを CAS にロードすることもできます。

関連項目

- “FILENAME ステートメント: URL アクセスメソッド” (*SAS グローバルステートメント: リファレンス*)
- “CSV ファイルを CAS にロード” (*SAS Cloud Analytic Services: ユーザーガイド*)
- “CASUTIL プロシジャ” (*SAS Cloud Analytic Services: ユーザーガイド*)。

例: CAS テーブルを SAS データセットとして保存する

コード例

次の例は、DATA ステップを使用してインメモリ CAS テーブルを SAS データセットに変換する方法を示しています。最初の DATA ステップでは、この例の変換に使用

できるインメモリ CAS テーブルを作成します。2 番目の DATA ステップでは、CAS テーブルを読み取り、SAS ライブラリ mySAS に SAS データセットを作成します。

```
cas casauto sessopts=(caslib='casuser');  /* 1 */
libname mycas cas;                        /* 2 */
caslib _all_ assign;

data mycas.earnings;
  Amount=1000;                            /* 3 */
  Rate=.075/12;
  do month=1 to 12;
    Earned +(amount+earned)*(rate);
  end;
run;
proc print data=mycas.earnings;
run;

libname mySAS "u/user/myfiles/";          /* 4 */

data mySAS.earnings;                      /* 5 */
  set mycas.earnings;
run;
```

1 Casauto という名前の CAS セッションを開始し、パーソナル caslib である Casuser をアクティブな caslib として指定します。

2 CAS LIBNAME ステートメントを使用して、CAS エンジンライブラリ参照名を作成します。

3 この例で使用する mycas.earnings という名前の CAS テーブルを作成します。

4 テーブルを SAS データセットとして保存するための mySAS という名前のライブラリ参照名を作成します。ライブラリ参照名 mySAS は、データセットが保存されている物理的な場所を表します。

5 テーブル mycas.earnings を読み取り、mySAS.earnings という名前の SAS データセットとして書き出します。

Obs	Amount	Rate	month	Earned
1	1000	.00625	13	77.6326

要点

- 出力テーブルはあるが入力テーブルがない CAS DATA ステップは、デフォルトで単一スレッドで実行されます。DATA ステートメントで SINGLE=NO オプションを指定すると、DATA ステップを強制的に複数スレッドで実行できます。SINGLE=オプションの指定については、“SINGLE=NO | NOINPUT | YES” (*SAS DATA ステップステートメント: リファレンス*)を参照してください。
- DATA ステップが実行される場所と DATA ステップ処理モードの制御の詳細については、“CAS での DATA ステップ処理” (*SAS Cloud Analytic Services: DATA ステッププログラミング*)を参照してください。

- CAS DATA ステップは、CAS テーブルでのみ機能します。

関連項目

- “CAS ステートメント” (*SAS Cloud Analytic Services: ユーザーガイド*)
- CASLIB ステートメントオプション“_ALL_ ” (*SAS Cloud Analytic Services: ユーザーガイド*)
- “LIBNAME ステートメント” (*SAS グローバルステートメント: リファレンス*)
- “SAS Cloud Analytic Services を使用したデータへのアクセス” (*SAS Cloud Analytic Services: ユーザーガイド*)
- “DATA ステートメント” (*SAS DATA ステップステートメント: リファレンス*)

例: SAS エンジンを使用して SAS データを処理する

例: LIBNAME ステートメントで V9 エンジンを割り当てる

コード例

このライブラリ割り当ては、V9 エンジンを指定します。

```
libname myfiles v9 'library-path'; /* 1 */
data myfiles.myclass;             /* 2 */
  set sashelp.class;
run;
```

1 LIBNAME ステートメントは、myfiles ライブラリ参照名と V9 エンジンをライブラリの場所に割り当てます。*library-path* をライブラリの場所に置き換えます。この場所はすでに存在しており、SAS Compute Server からアクセスできる必要があります。

2 DATA ステップは、sashelp ライブラリから class データセットをコピーすることにより、myfiles ライブラリに myclass データセットを作成します。

例のコード 14.1 V9 ライブラリ割り当てが成功したことを示す SAS ログ

```
1    libname myfiles v9 'library-path';
NOTE: Libref MYFILES was successfully assigned as follows:
      Engine:       V9
      Physical Name: library-path
2 !
3    data myfiles.myclass;
4      set sashelp.class;
5    run;

NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set MYFILES.MYCLASS has 19 observations and 5 variables.
```

要点

- LIBNAME ステートメントは、プログラムでライブラリを割り当てる一般的な方法です。ライブラリ割り当ては、ライブラリ参照名、エンジン、物理的な場所、エンジンまたは環境に固有のオプションで構成されます。
- SAS® Viya®でライブラリを割り当てるときは、その場所がすでに存在しており、SAS からアクセスできる必要があります。
- 出荷時のデフォルト Base SAS エンジンは BASE です。BASE エンジンは V9 エンジンのエイリアスです。新しいライブラリを作成するときにエンジン名を指定せず、ENGINE システムオプションを指定していない場合は、V9 エンジンが自動的に選択されます。ライブラリの場所にすでに SAS ファイルが含まれている場合は、SAS でそれらのファイルに基づいて正しいエンジンを割り当てることもできます。たとえば、場所に V9 データセットのみが含まれている場合、SAS は V9 エンジンを割り当てます。ただし、ライブラリの場所に異なるエンジンファイルが混在している場合、SAS では必要なエンジンを割り当てられないことがあります。したがって、エンジンを指定することがベストプラクティスです。

関連項目

- “例: ライブラリ参照名を使用してデータにアクセスする” (238 ページ)
- “ライブラリ割り当ての要素” (230 ページ)
- “デフォルトの Base SAS エンジン(V9 エンジン)” (274 ページ)

例: LIBNAME ステートメントで SPD Engine を割り当てる

コード例

SPD Engine の次の LIBNAME ステートメントは、V9 エンジンの LIBNAME ステートメントと非常に似ています。

```
libname mylib spde 'library-path'       /* 1 */
  datapath=('path-for-data-partitions') /* 2 */
  indexpath=('path-for-indexes');       /* 3 */
```

1 LIBNAME ステートメントのこの部分では、mylib ライブラリ参照名と SPD Engine をプライマリパス名に割り当てます。データセットの最初(通常は唯一)のメタデータファイルは、常にライブラリのプライマリパスに保存されます。

2 オプションで、DATAPATH=オプションで 1 つ以上のパス名を割り当てて、データパーティションを保存できます。それ以外の場合、データパーティションファイルはプライマリパスに保存されます。

3 オプションで、INDEXPATH=オプションで 1 つ以上のパス名を割り当てて、インデックスファイルを保存できます。それ以外の場合、インデックスファイルはプライマリパスに保存されます。

例のコード 14.2　SPD Engine ライブラリ割り当てが成功したことを示す SAS ログ

```
1   libname mylib spde 'library-path'
2     datapath=('path-for-data-partitions')
3     indexpath=('path-for-indexes');
NOTE: Libref MYLIB was successfully assigned as follows:
    Engine:       SPDE
    Physical Name: library-path
3 !
```

要点

- SPD Engine は、代替 Base SAS エンジンです。
- SPD Engine は、非常に大きなテーブルを高速に処理できるように設計されています。このエンジンはスレッドを使用してデータを非常に高速かつ並列に読み取り、複数の CPU で実行されます。このパフォーマンスに貢献しているのは、分散環境を活用できるパーティションファイル形式です。
- SPD Engine はデータセットを複数のファイルに保存しますが、SPD Engine データセットは V9 エンジンデータセットと非常によく似た方法で処理できます。Base

SAS 言語のほとんどは、SPD Engine データセットと非常にうまく連携します。ただし、エンジンは、その処理とストレージの最適化に固有のいくつかの言語要素をサポートしています。V9 エンジン機能との違いについては、ドキュメントを参照してください。

関連項目

- *SAS Scalable Performance Data Engine: リファレンス*
- “エンジン特性” (272 ページ)

例: SPD Engine を使用した Hadoop での SAS データの読み取りおよび書き込み

コード例

次の例では、Base SAS ライブラリを Hadoop クラスターに割り当てます。

```
options set=SAS_HADOOP_CONFIG_PATH='/myconfigpath';       /* 1 */
options set=SAS_HADOOP_JAR_PATH='/myjarpath';

libname mydata spde '/data/abcdef' hdfs=yes accelwhere=yes; /* 2 */
```

1 SET=システムオプションは、Hadoop の環境変数を定義します。これらの環境変数がすでに設定されている場合(たとえば、構成中)、これらのコード行をサブミットしないでください。これらの環境変数が正しく設定されていない場合、LIBNAME ステートメントにより SAS ログにエラーが生成されます。

2 LIBNAME ステートメントは、mydata ライブラリ参照名を SPD Engine と Hadoop クラスター内のディレクトリに割り当てます。HDFS=YES 引数は、Hadoop クラスター構成ファイルで定義されている Hadoop クラスターに接続することを指定します。ACCELWHERE=YES オプションは、Hadoop クラスター内の MapReduce プログラムによってデータのサブセット化が実行されることを要求します。

要点

- SPD Engine は、従来のファイルシステムまたは Hadoop 上で SAS データを読み書きできる代替 Base SAS エンジンです。このエンジンでは、SAS/ACCESS などの追

加の SAS 製品を構成する必要はありませんが、サポートされている Hadoop ディストリビューションを実行している必要があります。

- 顧客は多くの場合、非常に大規模なデータを低コストで保存するために Hadoop を選択します。SPD Engine の分散ストレージと処理は、Hadoop ファイルシステム (HDFS)とうまく連携します。さらに、このエンジンは、Hadoop クラスターに MapReduce プログラムを自動的にサブミットすることで、ほとんどの WHERE 式を最適化できます。
- このエンジンは、Hadoop 上の SPD Engine データセットを読み取ることができます。エンジンを使用してデータセットを Hadoop に保存すると、ほとんどの Base SAS 言語を使用してデータを処理できるようになります。ただし、このエンジンは、Hadoop での処理とストレージに固有のいくつかの言語要素をサポートしています。

関連項目

- *SAS SPD Engine: Hadoop 分散ファイルシステムでのデータの保存*
- *Base SAS および SAS/ACCESS 用 SAS Hadoop 構成ガイド*

例: CVP エンジンと V9 エンジンを使用して切り捨てを回避する

コード例

この例を実行するには、まず、“例: LIBNAME ステートメントで V9 エンジンを割り当てる” (277 ページ)にあるように、myclass という名前のデータセットを作成します。PROC CONTENTS を実行して変数の長さを確認します。

```
libname myfiles v9 'library-path-1';
proc contents data=myfiles.myclass;
run;
```

PROC CONTENTS 出力では、2 つの文字変数に注目してください。Name の長さは 8、Sex の長さは 1 です。

アウトプット 14.2　拡張前の変数長を示す PROC CONTENTS

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
3	Age	Num	8
4	Height	Num	8
1	Name	Char	8
2	Sex	Char	1
5	Weight	Num	8

次の例では、CVP エンジンと V9 エンジンを使用して文字変数のサイズを拡張しています。CVP エンジンは、文字を表現するためにより多くのバイトを使用するエンコーディングにデータセットをコピーする場合に、切り捨てを回避するのに役立ちます。

```
libname srclib cvp 'library-path-1' cvpengine=v9 cvpmult=2.5; /* 1 */
libname target v9 'library-path-2';                           /* 2 */
proc copy in=srclib out=target;                               /* 3 */
  select myclass;
run;

proc contents data=target.myclass;                            /* 4 */
run;
```

1 この LIBNAME ステートメントは、srclib ライブラリを CVP エンジンとコピーするデータの場所に割り当てます。CVPENGINE=オプションは、データを処理する基になるエンジンとして V9 エンジンを指定します。CVPMULT=オプションは、すべての文字変数を拡張するための乗算係数 2.5 を指定します。このオプションが指定されていない場合、CVP エンジンは自動的に乗数値を選択します。

2 この LIBNAME ステートメントは、コピーされたデータを含む target ライブラリを割り当てます。

3 SELECT ステートメントを使用した COPY プロシジャは、myclass データセットを target ライブラリにコピーします。コピー中に、CVP エンジンは文字変数の長さを 2.5 倍に拡張します。

4 CONTENTS プロシジャは、文字変数の長さが 2.5 倍になったことを示します。

Name の場合、8 × 2.5 = 20。

Sex の場合、1 × 2.5 = 2.5 となり、整数に切り上げると 3 になります。

アウトプット 14.3　拡張後の変数長を示す PROC CONTENTS

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
3	Age	Num	8
4	Height	Num	8
1	Name	Char	20
2	Sex	Char	3
5	Weight	Num	8

要点

- 文字を表現するためにより多くのバイトを使用するエンコーディングにデータセットをコピーする場合、列の長さがより大きな文字サイズに対応できないと切り捨てが発生する可能性があります。たとえば、文字は wlatin1 エンコーディングでは 1 バイトとして表現されますが、UTF-8 では 2 バイトとして表現される場合があります。
- ログ内のエラーに character data was lost during transcoding と記載されている場合は、通常、切り捨てが発生したことを示しています。CVP エンジンを使用して文字変数の長さを拡張することで、エラーをトラブルシューティングできます。
- デフォルトでは、CVP エンジンは自動的に乗数値を選択します。通常、自動値は切り捨てを回避するのに十分です。CVPMULTIPLIER=を使用して自分で指定することもできます。
- デフォルトの CVPFORMATWIDTH=YES オプションは、出力形式の長さを拡張しますが、ユーザー定義の出力形式には影響しません。ユーザー定義の出力形式については、“Example: Avoid Truncation in Formats When Using PROC FORMAT” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。
- データセットを別の動作環境または別の文字エンコーディングにコピーする場合は、PROC COPY に NOCLONE などのオプションを指定することをお勧めします。NOCLONE は、ライブラリがターゲット環境で互換性があるように、データ表現や文字エンコーディングなどの特定のデータセット属性をコピーしません。

  NOCLONE オプションのかわりに、OVERRIDE=オプションを使用して ENCODING=および OUTREP=オプションを指定できます。この方法では、ソースライブラリの他のデータセット属性が保持されるため、それらの属性が必要であることを確認してください。
- PROC COPY は監査証跡を保持しません。CEDA 処理では、PROC COPY はインデックスや一貫性制約をコピーしません。移行する場合は、PROC MIGRATE の方がよい選択となる可能性があります。“Example: Avoid Truncation When Migrating a SAS Library by Using the CVP Engine” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。

関連項目

- “Compatibility and Migration” (*SAS V9 LIBNAME Engine: Reference*)
- “COPY プロシジャ” (*Base SAS プロシジャガイド*)

CAS 内のデータへのアクセス

CAS 内のデータへのアクセスについては、“SAS Cloud Analytic Services を使用したデータへのアクセス” (*SAS Cloud Analytic Services: ユーザーガイド*)を参照してください。

15

データベースと PC ファイル

SAS のデータベースおよび PC ファイルの定義

SAS/ACCESS ソフトウェア
: 他のベンダーのデータベース管理システム(DBMS)や一部の PC ファイル形式との間でデータの読み書きを行うことができます。LIBNAME ステートメントを割り当てた後、SAS の 2 レベル名を使用してデータベース内のテーブルまたはビューを指定できます。その後、SAS データセットの場合と同様に、テーブルまたはビューを操作できるようになります。DBMS データを参照する SAS ビューを作

成することもできます。詳細については、“SAS/ACCESS インターフェイスの使用方法” (378 ページ)のドキュメントリストを参照してください。

SQL パススルー機能
SAS セッションを離れることなく、ネイティブ SQL 構文を使用してデータソースと対話できます。PROC SQL を使用して、SQL ステートメントをデータソースに直接渡して処理します。PROC SQL ビューで SQL パススルー機能を使用できます。詳細については、“How the SQL Pass-Through Facility Works” (*SAS/ACCESS for Relational Databases: Reference*)を参照してください。

SAS ビュー
後で使用するために名前を付けて保存される仮想データセットを指定します。ビューにはデータは含まれませんが、かわりに他の場所に保存されているデータが記述されます。SAS ビューは、DATA ステップまたは PROC SQL を使用して作成されます。SAS 言語要素を使用してネイティブ DBMS ビューを作成することはできません。詳細については、“SAS Views” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。

SAS エンジン
特定のファイル形式でファイルの読み取りと書き込みを行う SAS のコンポーネントです。(一部のエンジンは読み取り専用です。)SAS/ACCESS エンジンは、他のベンダーのアプリケーションと通信します。エンジンの詳細については、12 章, “SAS エンジン” (269 ページ)を参照してください。

いくつかの重要な概念を次に示します。

- このドキュメントで説明されている SAS/ACCESS 機能を使用するには、データソースの SAS/ACCESS インターフェイスを構成する必要があります。
- 一部の外部ファイルは SAS/ACCESS を必要としません。区切りデータは、Base SAS 言語要素による入出力処理でサポートされています(14 章, “生データ” (341 ページ)を参照)。JSON、ORC、XML などの他のファイルタイプは、LIBNAME ステートメントをサブミットした後、Base SAS によってサポートされます。
- LIBNAME ステートメントオプション、データセットオプション、およびその他の機能は、データソースと環境によって異なります。データソースの SAS/ACCESS ドキュメントを参照してください。http://support.sas.com で入手できる SAS のシステム要件と構成ガイドのドキュメントも参照してください。

SAS/ACCESS インターフェイスの使用方法

SAS/ACCESS 機能の可用性と動作は、インターフェイスごとに異なります。したがって、適切な SAS/ACCESS ドキュメントの一般セクションと DBMS 固有セクションの情報を参照してください。

- *リレーショナルデータベース用 SAS/ACCESS: リファレンス*
- *非リレーショナルデータベース用 SAS/ACCESS: リファレンス*
- *SAS/ACCESS Interface to PC Files: SAS Viya プラットフォームのリファレンス*

表 15.1 SAS/ACCESS で一般的に使用されるアクセス方式

アクセス方式	サポートされている SAS/ACCESS データソース	例
LIBNAME ステートメント: 1 SAS/ACCESS LIBNAME ステートメントをサブミットして、SAS ライブラリをデータソースに割り当てます。 2 SAS プロシジャおよび DATA ステップの LIBNAME ステートメントのライブラリ参照名を使用して、データに直接アクセスします。	ほとんどのリレーショナルおよび非リレーショナル DBMS、および PC ファイル	■ “例: SAS/ACCESS エンジンを使用して DBMS にデータを作成する” (386 ページ) ■ “例: DATA ステップを使用して SAS データセットから Excel データを作成する” (382 ページ)
PROC SQL の SQL パススルー機能: 1 CONNECT ステートメントを使用して、DBMS 名と接続オプションを指定します。(LIBNAME ステートメントは必要ありません。) 2 データを読み取るには、SELECT ステートメントで CONNECTION TO コンポーネントを使用します。SELECT ステートメントのクエリでは、DBMS にネイティブな SQL を使用します。 3 EXECUTE ステートメントを使用すると、動的な非クエリ SQL ステートメント (INSERT、DELETE、UPDATE など)をデータソースに渡すことができます。 4 接続を終了するには、DISCONNECT ステートメントを指定します。	ほとんどのリレーショナルおよび非リレーショナル DBMS、および PCFILES エンジン	■ “例: SQL パススルー機能を使用して DBMS データにアクセスする” (388 ページ)
LIBNAME ステートメントと SAS ビュー: 1 V9 エンジンの LIBNAME ステートメントを使用して、ビューを保存できる SAS ライブラリの場所を割り当てます。 2 SAS/ACCESS LIBNAME ステートメントを使用して、SAS ライブラリをデータソースに割り当てます。あるいは、PROC SQL ビューの場合は、USING 句に LIBNAME を埋め込むことを選択できます。 3 データのクエリとして DATA ステップビューまたは PROC SQL ビューを作成します。 4 ビューで SAS プロシジャと DATA ステップを使用して、データのサブセットにアクセスします。	ほとんどのリレーショナルおよび非リレーショナル DBMS、および PC ファイル	■ “例: PROC SQL ビューに SAS/ACCESS LIBNAME ステートメントを埋め込む” (389 ページ)

アクセス方式	サポートされている SAS/ACCESS データソース	例
IMPORT および EXPORT プロシジャ	PC ファイル	■ “例: SAS/ACCESS エンジンと PROC IMPORT を使用して Microsoft Excel データを読み取る” (381 ページ) ■ “例: PROC EXPORT を使用して SAS データセットから Excel データを作成する” (384 ページ)

表 15.2　SAS/ACCESS および CAS サーバーで一般的に使用されるアクセス方式

SAS Cloud Analytic Services (CAS)にロードするためのアクセス方式	サポートされている SAS/ACCESS データソース	例
SAS データコネクタまたはデータ接続アクセラレータの使用をお勧めします: 1 CASLIB ステートメントをサブミットして、データソースと CAS サーバー間の接続を開きます。 2 CASUTIL または CAS プロシジャを使用して、データを CAS にロードします。	ほとんどのリレーショナルおよび非リレーショナル DBMS、および PC ファイル	■ “range=を CASUTIL プロシジャで使用してインポートするデータを指定” (*SAS Cloud Analytic Services: ユーザーガイド*) ■ “Suggested Workflow for Data Sources with Data Connectors” (*SAS/ACCESS for Relational Databases: Reference*)
SAS データコネクタまたはデータ接続アクセラレータを使用しない場合: 1 SAS/ACCESS LIBNAME ステートメントをサブミットして、SAS ライブラリをデータソースに割り当てます。SAS プロシジャと DATA ステップを使用してデータを操作できます。 2 CAS ステートメントをサブミットして、SAS Compute Server と CAS サーバー間の接続を開きます。 3 CASUTIL または CAS プロシジャを使用して、データを CAS にロードします。	ほとんどのリレーショナルおよび非リレーショナル DBMS、および PC ファイル	■ “例: CASUTIL プロシジャを使用して DBMS データを CAS にロードする” (391 ページ) ■ “Suggested Workflow for Data Sources without Data Connectors” (*SAS/ACCESS for Relational Databases: Reference*)

例: Microsoft Excel ファイルの読み取りと書き込み

例: SAS/ACCESS エンジンと PROC IMPORT を使用して Microsoft Excel データを読み取る

コード例

この例では、XLSX エンジンと PROC IMPORT を使用して Excel データをインポートします。この例のデータを作成するには、Microsoft Excel を起動し、“例: カンマ区切りファイルを読み取る” (286 ページ)で使用される cholesterol.csv ファイルを開きます。(cholesterol.csv ファイルは、http://support.sas.com/documentation/onlinedoc/viya/exampledatasets/cholesterol.csv からダウンロードできます。)Excel では、.xlsx 拡張子を付けてファイルを保存します。次のコードは、cholesterol.xlsx を SAS データセット mycholesterol としてインポートします。

```
options validvarname=v7;                    /* 1 */
proc import datafile='file-path/cholesterol.xlsx' /* 2 */
    dbms=xlsx                           /* 3 */
    out=work.mycholesterol                  /* 4 */
    replace;                            /* 5 */
run;
```

1 VALIDVARNAME=V7 ステートメントは、SAS が列名を変数名に変換するときに、スペースを 強制的にアンダースコアに変換します。

2 DATAFILE=オプションは、入力ファイルのパスを指定します。

3 DBMS=オプションは、インポートするデータのタイプを指定します。Excel ワークブックをインポートする場合は、DBMS=XLSX を指定します。PROC IMPORT は XLSX エンジンを呼び出すため、LIBNAME ステートメントをサブミットする必要はありません。

4 OUT=オプションは、出力 SAS データセットを識別します。

5 REPLACE オプションは、既存の SAS データセットを上書きします。

要点

- XLSX エンジンを使用すると、Excel の.xlsx ファイルからデータを直接読み取ることができます。SAS/ACCESS Interface to PC Files を構成しておく必要があります。XLSX エンジンは、Microsoft Excel 2007、2010、およびそれ以降のファイルへの接続をサポートします。
- Excel の命名規則は SAS とは異なります。Excel データを読み取るためのベストプラクティスは、VALIDVARNAME=V7 を設定することです。このオプションはスペースをアンダースコアに変換し、32 文字を超える名前を切り捨てます。
- 日付値は、SAS 日付数値に自動的に変換されます。

関連項目

- “LIBNAME Statement: XLSX Engine” (*SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform*)
- “IMPORT プロシジャ” (*Base SAS プロシジャガイド*)
- “VALIDVARNAME= システムオプション” (*SAS システムオプション: リファレンス*)

例: DATA ステップを使用して SAS データセットから Excel データを作成する

コード例

次のコードは、demo.xlsx という名前の Excel ファイルを作成します(まだ存在しない場合)。DATA ステップでは、SAS データセット sashelp.class を使用して、ファイル内に My Worksheet という名前のワークシートを作成します。ワークシート名には空白スペースが含まれるため、n リテラルとして指定されます。

```
libname myfiles xlsx "library-path/demo.xlsx";
data myfiles.'My Worksheet'n;
  set sashelp.class;
run;
proc print data=myfiles.'My Worksheet'n (obs=10);
run;
```

PROC PRINT はシート My Worksheet の最初の 10 個のオブザベーションを出力します。

アウトプット 15.1　My Worksheet の PROC PRINT 出力

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84
3	Barbara	F	13	65.3	98
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84
10	John	M	12	59	99.5

要点

- DATA ステップで Excel データを作成するには、まず、ファイル拡張子.xlsx を持つファイル名の場所に XLSX LIBNAME ステートメントを割り当てます。ファイルがすでに存在している必要はありませんが、場所(フォルダーまたはディレクトリ)が存在している必要があります。
- DATA ステップでは、ワークシート名に特殊文字またはスペースが含まれている場合、名前を n リテラルとして指定します。
- SAS から Excel データを作成するために一般的に使用されるその他の方法には、PROC EXPORT または ODS EXCEL 出力先があります。

関連項目

- “LIBNAME Statement: XLSX Engine” (*SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform*)
- “SAS Name Literals” (41 ページ)

例: PROC EXPORT を使用して SAS データセットから Excel データを作成する

コード例

次のコードは、“例: DATA ステップを使用して SAS データセットから Excel データを作成する” (382 ページ)で作成された demo.xlsx という名前の Excel ファイルを参照します。ファイルがまだ存在しない場合は、PROC EXPORT によって作成されます。

PROC EXPORT は SAS データセット sashelp.class を使用して、ファイル内に Demo Sheet 2 という名前のワークシートを作成します。

```
proc export outfile="library-path/demo.xlsx"
  data=sashelp.class
  dbms=xlsx;
  sheet="Demo Sheet 2";
run;
```

要点

- PROC EXPORT を使用して Excel データを作成するには、DBMS=XLSX を指定します。OUTFILE=を使用して、以前に定義したファイル参照名、または完全なパス名とファイル名を指定します。ファイル名を指定する場合は、.xlsx ファイル拡張子を使用します。ファイルがすでに存在している必要はありませんが、場所(フォルダーまたはディレクトリ)が存在している必要があります。Linux で SAS/ACCESS を使用して PC サーバーに保存されているデータにアクセスする場合、ファイル参照名はサポートされません。
- 小文字、特殊文字、またはスペースを保持するには、SHEET=オプションの値を引用符で囲みます。
- DBMS=XLSX を指定する場合、SAS PC Files Server は必要ありません。
- SAS から Excel データを作成するために一般的に使用されるその他の方法には、XLSX LIBNAME ステートメントを使用した DATA ステップ、または ODS EXCEL 出力先が含まれます。

関連項目

- “EXPORT Procedure” (*SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform*)

- “Supported Data Sources and Environments” (*SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform*)
- “Microsoft Excel Workbook Files” (*SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform*)

例: PROC IMPORT を使用して Excel データを CAS にロードする

コード例

次のコードは、PROC IMPORT を使用して、Excel ファイルからメモリ内に CAS テーブルを作成します。この例を実行するには、まずファイル demo.xlsx を作成します。これには、“例: DATA ステップを使用して SAS データセットから Excel データを作成する” (382 ページ)を実行します。

```
cas casauto host="cloud.example.com" port=5570; /* 1 */
libname mycas cas;                              /* 2 */

proc import datafile='file-path/demo.xlsx'      /* 3 */
   dbms=xlsx                                    /* 4 */
   out=mycas.newdemo                            /* 5 */
   replace;                                     /* 6 */
run;
```

1 CAS ステートメントは CAS セッションを開始し、CAS セッション名として casauto を指定します。

2 LIBNAME ステートメントは、mycas ライブラリ参照名を CAS セッションに割り当てます。SESSREF= LIBNAME オプションが指定されていないため、エンジンは casauto セッションを使用します。

3 DATAFILE=オプションは、入力ファイルのパスを指定します。

4 DBMS=オプションは、インポートするデータのタイプを指定します。Excel ワークブックをインポートする場合は、DBMS=XLSX を指定します。PROC IMPORT は XLSX エンジンを呼び出すため、このデータソースに対して LIBNAME ステートメントをサブミットする必要はありません。

5 OUT=オプションは、出力 CAS テーブルを識別します。

6 REPLACE オプションは、既存のテーブルを上書きします。

要点

- PROC IMPORT を使用して Excel データをインポートするには、DBMS=XLSX を指定します。OUT=を使用して、以前に定義したファイル参照名、または完全なパス名とファイル名を指定します。ファイル名を指定する場合は、.xlsx ファイル拡張子を使用します。ファイルがすでに存在している必要はありませんが、場所(フォルダーまたはディレクトリ)が存在している必要があります。Linux で SAS/ACCESS を使用して PC サーバーに保存されているデータにアクセスする場合、ファイル参照名はサポートされません。
- DBMS=XLSX を指定する場合、SAS PC Files Server は必要ありません。
- Excel ファイルは、CAS サーバー上のメモリ内の CAS テーブルにインポートするために CAS エンジンによってサポートされています。

関連項目

- “IMPORT プロシジャ” (*Base SAS プロシジャガイド*)
- “Supported Data Sources and Environments” (*SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform*)
- “Microsoft Excel Workbook Files” (*SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform*)

例: DBMS ファイルへのアクセス

例: SAS/ACCESS エンジンを使用して DBMS にデータを作成する

コード例

この例では、SAS DATA ステップによって Teradata テーブルが作成されます。この例を実行するには、SAS/ACCESS 用に Teradata インターフェイスを構成する必要があります。

```
libname mytddata teradata server=mytera user=myid password=mypw; /* 1 */
data mytddata.grades;                                  /* 2 */
  input student $ test1 test2 final;
  datalines;
Fred 66 80 70
Wilma 97 91 98
;
proc datasets library=mytddata;                        /* 3 */
run;
quit;
```

1 LIBNAME ステートメントは、mytddata ライブラリ参照名と、SAS/ACCESS Interface to Teradata のエンジンニックネームである TERADATA を指定します。このステートメントでは、Teradata の接続オプションも指定します。これらのオプションを変更して、SAS/ACCESS 接続値および必要なその他のオプションを指定します。

2 DATA ステップでは、grades という名前のテーブルを作成します。テーブルは Teradata DBMS 内にあり、SAS データセットではありません。

3 mytddata ライブラリの PROC DATASETS 出力は、エンジンが Teradata であることを示しています。grades テーブルの場合、SAS メンバータイプは DATA、DBMS メンバータイプは TABLE です。

アウトプット 15.2 DBMS ディレクトリ情報を示す PROC DATASETS 出力

Directory	
Libref	MYTDDATA
Engine	TERADATA
Physical Name	mytera
Schema/User	myid

アウトプット 15.3 DBMS コンテンツのライブラリコンテンツを示す PROC DATASETS 出力

#	Name	Member Type	DBMS Member Type
1	grades	DATA	TABLE

要点

- DBMS データ用に SAS/ACCESS インターフェイスを構成すると、SAS で LIBNAME ステートメントをサブミットしてテーブルにアクセスできます。SAS は、SAS データセットにアクセスする場合と同様の方法で DBMS テーブルにアクセスできます。ただし、既存の DBMS テーブルを置き換えることはできないことに注意してください。SAS/ACCESS インターフェイスは、REPLACE= SAS データセットオプションをサポートしません。

- ほとんどの SAS/ACCESS エンジンは、Base SAS 言語を完全にサポートしています。カタログなどのいくつかの Base SAS 機能は V9 エンジンに固有であり、SAS/ACCESS エンジンでは使用できません。SAS/ACCESS エンジンは、データソースに固有のいくつかの言語要素をサポートします。詳細については、インターフェイスのドキュメントを参照してください。
- 一部の SAS/ACCESS インターフェイスでは大文字と小文字が区別されます。DBMS の要件に準拠するために、テーブル名または列名の大文字と小文字を変更する必要がある場合があります。

関連項目

- SAS/ACCESS ドキュメント
- *データソースはアルファベット順にリストされています*
- "DATA ステートメント" (*SAS DATA ステップステートメント: リファレンス*)

例: SQL パススルー機能を使用して DBMS データにアクセスする

コード例

この PROC SQL の例では、SQL パススルー機能を使用してクエリを Teradata テーブルに送信します。

```
proc sql;
  connect to teradata as myconn (server=mytera
    user=myid password=mypw);              /* 1 */
  select *
    from connection to myconn              /* 2 */
      (select *
        from grades
        where final gt 90);
  disconnect from myconn;                  /* 3 */
quit;
```

1 CONNECT ステートメントは DBMS への接続を確立します。

2 PROC SQL SELECT ステートメントの FROM 句の CONNECTION TO コンポーネントは、DBMS から直接データを取得します。DBMS 固有のクエリはかっこで囲まれています。

3 DISCONNECT ステートメントは、DBMS への接続を終了します。

アウトプット 15.4 PROC SQL パススルー機能の出力

student	test1	test2	final
Wilma	97	91	98

要点

- SQL パススルー機能は、SAS SQL 構文のかわりに DBMS 固有の SQL 構文を使用できるようにする PROC SQL の拡張機能です。
- SQL パススルー機能ステートメントを PROC SQL クエリで使用したり、PROC SQL ビューに保存したりできます。
- パススルー機能は、3 つのステートメントと 1 つのコンポーネントで構成されます。
  - CONNECT ステートメントは DBMS への接続を確立します。
  - EXECUTE ステートメントは、動的で非クエリ DBMS 固有の SQL ステートメントを DBMS に送信します。
  - PROC SQL SELECT ステートメントの FROM 句の CONNECTION TO コンポーネントは、DBMS から直接データを取得します。
  - DISCONNECT ステートメントは、DBMS への接続を終了します。

関連項目

- “SQL Pass-Through Facility” (*SAS/ACCESS for Relational Databases: Reference*)
- “SAS Views of DBMS Data” (*SAS/ACCESS for Relational Databases: Reference*)
- “SAS Views” (*SAS V9 LIBNAME Engine: Reference*)

例: PROC SQL ビューに SAS/ACCESS LIBNAME ステートメントを埋め込む

コード例

次の例では、SAS/ACCESS LIBNAME ステートメントを PROC SQL ビューに埋め込みます。DATA ステップビュー(LIBNAME ステートメントを埋め込まない)を作成するには、“例: DBMS データにアクセスして SAS ビューを作成する” (246 ページ)を参照してください。

```
libname viewlib v9 'library-path';          /* 1 */
proc sql;
  create view viewlib.mygrades as           /* 2 */
    select *
      from mytddata.grades                  /* 3 */
      using libname mytddata teradata
                server=mytera
                user=myid password=mypw; /* 4 */
quit;
proc print data=viewlib.mygrades noobs;     /* 5 */
run;
```

1 V9 エンジンの LIBNAME ステートメントは、SAS ビューが保存される場所に viewlib ライブラリ参照名を割り当てます。

2 SQL プロシジャの CREATE VIEW ステートメントは、viewlib ライブラリに mygrades ビューを作成します。

3 mytddata.grades テーブルはビューで参照されます。

4 USING 引数には、SAS/ACCESS Interface to Teradata エンジンの LIBNAME ステートメントが埋め込まれます。

5 PRINT プロシジャは、埋め込み LIBNAME ステートメントを使用して mytddata.grades テーブルを参照する viewlib.mygrades ビューを実行します。

アウトプット 15.5　viewlib.mygrades の PROC PRINT

student	test1	test2	final
Wilma	97	91	98
Fred	66	80	70

要点

- ほとんどの SAS/ACCESS エンジンには、いくつかの接続オプションが必要です。LIBNAME ステートメントを埋め込むと、DBMS 接続情報をビューに保存して使いやすくすることができます。ただし、接続情報が変更されることが予想される場合 (たとえば、別のユーザー ID)、この方法はお勧めできません。
- LIBNAME ステートメントを PROC SQL ビューに埋め込むには、CREATE VIEW ステートメントの USING 句を指定します。LIBNAME ステートメントは、ほとんどの SAS/ACCESS エンジン、V9 エンジン、およびその他のエンジンに埋め込むことができます。
- PROC SQL が SAS ビューを実行するとき、SELECT ステートメントはライブラリ参照名を割り当て、DBMS への接続を確立します。ライブラリ参照名のスコープは SAS ビューに対してローカルであり、セッション内に存在する可能性のある同じ名前のライブラリ参照名と競合しません。クエリが終了すると、接続は終了し、ライブラリ参照名の割り当てが解除されます。

- DBMS テーブルの V9 エンジンビューを作成すると、データは DBMS 内に残ります。いつでもビューに対してプロシジャまたは DATA ステップを実行して、基になるデータソースから最新のデータを取得できます。

  ただし、複数のステップでデータを一定に保つ必要がある場合は、データを 1 回読み取って SAS データセットに保存することをお勧めします。この方法を実践すると、パフォーマンスにもメリットが得られます。通常、SAS データセットを複数回読み取る方が、DBMS テーブルを複数回読み取るよりも効率的です。

関連項目

- “LIBNAME Statement: External Databases” (*SAS/ACCESS for Relational Databases: Reference*)
- “CREATE VIEW ステートメント” (*SAS SQL プロシジャユーザーガイド*)
- “SAS Views of DBMS Data” (*SAS/ACCESS for Relational Databases: Reference*)
- “SAS Views” (*SAS V9 LIBNAME Engine: Reference*)

例: CASUTIL プロシジャを使用して DBMS データを CAS にロードする

コード例

この例では、CASUTIL プロシジャは Greenplum テーブルを SAS Cloud Analytic Services (CAS)にロードします。

```
libname mydblib greenplm server=mygpserver db=mydbms port=5432 /* 1 */
            user=myuser password=gppwd;

libname mylib v9 "/data";                          /* 2 */
data mylib.mydataset;
  set mydblib.mytable;
run;

cas casauto host="cloud.example.com" port=5570;          /* 3 */
proc casutil;
  load data=mylib.mydataset promote;                  /* 4 */
run;
```

1 この LIBNAME ステートメントは、mydblib ライブラリ参照名を Greenplum データベースに割り当てます。GREENPLM エンジンのニックネームと接続オプションが指定されます。

2 この LIBNAME ステートメントは、mylib ライブラリ参照名を SAS データの場所に割り当てます。V9 エンジンが指定されています。DATA ステップは、Greenplum テーブル mytable を中間 SAS データセット mydataset にコピーします。

3 CAS ステートメントは CAS セッションを開始し、CAS セッション名として casauto を指定します。

4 PROC CASUTIL では、DATA=オプションを指定した LOAD ステートメントは、mylib.mydataset からデータを読み取り、CAS のメモリにロードします。

PROMOTE オプションは、グローバルスコープでテーブルをロードします。

要点

- SAS/ACCESS LIBNAME ステートメントを使用して DBMS にアクセスできる場合は、データを中間 SAS データセットにインポートし、PROC CASUTIL を使用してデータを CAS にロードできます。この方法は、データを CAS にロードする前にローカル SAS クライアントでデータを処理する場合に使用します。この方法は、データコネクタを持たない少数のデータソースにも使用されます。これらの問題のどちらにも影響を受けない場合は、中間 SAS データセットを作成せずに、データコネクタを直接使用できます。
- LOAD ステートメントの LOAD DATA=形式は、SAS 言語要素を使用するプログラマに推奨される構文です。

関連項目

- “LOAD ステートメント” (*SAS Cloud Analytic Services: ユーザーガイド*)
- *SAS Viya プラットフォームプログラミング: 入門ガイド*

例: DBMS および PC ファイルに関する情報の表示

例: Excel ファイルに関する情報を出力する

コード例

次の PROC DATASETS コードは、ライブラリ myfiles に関する情報を出力します。このライブラリは Excel ファイル demo.xlsx に割り当てられます。

```
libname myfiles xlsx "library-path/demo.xlsx";
proc datasets lib=myfiles;
contents data='Demo Sheet 2'n;
run;
quit;
```

XLSX エンジンを使用する場合、SAS は Excel ファイルをライブラリとして処理します。ファイル内の各ワークシートは、DATA のメンバータイプと TABLE の DBMSTYPE です。

アウトプット 15.6　demo.xlsx の DATASETS ディレクトリ情報

Directory	
Libref	MYFILES
Engine	XLSX
Physical Name	*library-path*/demo.xlsx
Por	.

#	Name	Member Type	DBMSTYPE
1	DEMO SHEET 2	DATA	TABLE
2	MY WORKSHEET	DATA	TABLE

CONTENTS 出力には、他のデータソースと同様の情報が表示されます。

アウトプット 15.7　Demo Sheet 2 の CONTENTS

Data Set Name	MYFILES.'Demo Sheet 2'n	Observations	.
Member Type	DATA	Variables	5
Engine	XLSX	Indexes	0
Created	.	Observation Length	0
Last Modified	.	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	Default		
Encoding	Default		

Alphabetic List of Variables and Attributes						
#	Variable	Type	Len	Format	Informat	Label
3	Age	Num	8	BEST.		Age
4	Height	Num	8	BEST.		Height
1	Name	Char	7	$7.	$7.	Name
2	Sex	Char	1	$1.	$1.	Sex
5	Weight	Num	8	BEST.		Weight

要点

- PROC DATASETS を CONTENTS ステートメントとともに使用すると、データベースまたは PC ファイルの属性を記述することができます。
- ワークシート名またはテーブル名に特殊文字またはスペースが含まれている場合は、名前を n リテラルとして指定します。

関連項目

- “DATASETS プロシジャ” (*Base SAS プロシジャガイド*)
- “LIBNAME Statement: XLSX Engine” (*SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform*)
- “SAS Name Literals” (41 ページ)
- “Microsoft Excel Workbook Files” (*SAS/ACCESS Interface to PC Files: Reference for the SAS Viya Platform*)

データのアクセスと転送

データのアクセスと転送のために、管理者は SAS Cloud Analytic Services (CAS)のデータコネクタを構成する必要があります。データコネクタには Oracle や SAS データセットなどのデータソースに接続するための接続情報およびデータソースの特性が含まれます。データコネクタの構成方法については、*SAS Viya プラットフォーム: 配置ガイド*を参照してください。

データソースと CAS の間でデータを転送する方法は 2 つあります。データを並列に転送する方法と、逐次に転送する方法があります。データを並列に転送するには、ユーザーは SAS In-Database Technologies を使用し、管理者によって SAS Embedded Process が配置され、ユーザーは並行データ転送を利用できるデータソースにアクセスしている必要があります。並列データ転送が利用できない場合でも、ほとんどのデータソースではマルチノードデータ転送が利用でき、並列データ転送とほぼ同等の効率が得られます。詳細については、"DBMS データソースのデータコネクタでのデータ転送モードの使用" (*SAS Cloud Analytic Services: ユーザーガイド*)を参照してください。

データベーステーブルのロード

前提条件

次の例では、次のことを想定しています。

- データソース内のデータへのアクセスが許可されています。
- ホストやポートなどの接続情報がわかっています。
- SAS Cloud Analytic Services インストールはライセンスされ、アクセスするデータソースベンダーのクライアントソフトウェアを使用するように構成されています。インストール時の構成情報については、*SAS Viya プラットフォーム: 配置ガイド*を参照してください。

例

caslib のデータソースからサーバーにテーブルをロードすることが、データをロードする最も効率的な方法です。この例では、Oracle からテーブルが読み取られ、インメモリテーブルは同じ caslib に保持されます。

```
caslib oralib datasource=(                              /* 1 */
   srctype="oracle",
   uid="DBUSER",
   pwd="secret",
   path="//dbserver.example.com:1521/dbname",
   schema="DBUSER"
   );


proc casutil;
  list files;                                    /* 2 */

  droptable casdata="sales" quiet;                        /* 3 */
  contents casdata="sales";                              /* 4 */

  load casdata="sales" casout="sales" promote                    /* 5 */
     label="Fact table for User-to-Item Analysis"
     varlist=(
        (name="USERID"  label="User ID"),
        (name="ITEMID"  label="Item ID")
     );

  contents casdata="sales";                              /* 6 */
quit;

run;
```

1 データソースとして Oracle を使用する caslib を追加します。Oralib がセッションのアクティブ caslib になり、後続のプログラミングステートメントはそれを入出力に使用します。

2 LIST FILES ステートメントは、Oracle データベースで使用可能なテーブルを表示します。

3 この DROPTABLE ステートメントには QUIET オプションが含まれています。このステートメントの実行は、実行が繰り返される場合に役立ちます。Sales という名前のテーブルが、後続の LOAD CASDATA=ステートメントに干渉するインメモリではなくなります。

4 最初の CONTENTS ステートメントは DROPTABLE ステートメントの後に続くため、これにより、Oracle からのテーブル情報と列情報が確実に読み取られます。

5 LOAD ステートメントの CASDATA=引数は、Sales テーブルが caslib のデータソース(Oracle)から SAS Cloud Analytic Services に読み込まれることを示します。オプションは、テーブルと列にラベルを追加するために指定されています。

6 最後の CONTENTS ステートメントは LOAD ステートメントの後に続くため、Oracle から読み取られた Sales テーブルのインメモリコピーのテーブル情報と列情報が表示されます。

要点

- CASLIB ステートメントは、サーバー側データソースを SAS Cloud Analytic Services に追加します。
- この例では、アクティブ caslib は Oralib です。caslib を追加するとデフォルトでアクティブ caslib になることに注意してください。
- データソース接続パラメーターの詳細については、*"データコネクタ" (SAS Cloud Analytic Services: ユーザーガイド)*を参照してください。

16

SAS ビュー

SAS ビューの定義

*SAS ビュー*は、カスタマイズされた動的なデータ表現を提供するために、他のファイルからデータ値を抽出する仮想データセットです。

- ビューにはデータは含まれませんが、別の場所に保存されているデータを参照します。
- ほとんどの場合、SAS ビューは SAS データセットであるかのように使用できます。

SAS ビューのタイプは次のとおりです。

DATA ステップビュー
: ストアド DATA ステッププログラムです。“SAS Views” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。

PROC SQL ビュー
: PROC SQL で作成されるストアドクエリ式です。“SAS Views” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。“PROC SQL ビューの作成および使用” (*SAS SQL プロシジャユーザーガイド*)も参照してください。

SAS ビューは V9 エンジンでサポートされています。適切な SAS/ACCESS エンジンが構成されている場合、SAS ビューは DBMS データを参照できます。SAS ビューは、CAS エンジンまたは SPD Engine ではサポートされていません。

17

SAS 辞書テーブル

SAS DICTIONARY テーブルの定義

SAS DICTIONARY は、SAS セッションを開始するたびに SAS が作成する読み取り専用の SAS システムテーブルのコレクションです。テーブルには、現在の SAS セッションに関連付けられているすべての SAS ライブラリ、データセット、システムオプション、外部ファイルに関する情報が含まれています。すべての DICTIONARY テーブルには、Sashelp ライブラリに関連ビューがあります。PROC SQL を指定して SQL DICTIONARY テーブルにアクセスし、DATA ステップまたは PROC ステップを使用して関連するビューにアクセスします。

SAS は、次の DICTIONARY テーブルとビューをサポートしています。

表 17.1 DICTIONARY テーブルとそれに関連する 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	すべての既知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	既知のインフォメーションマップデータアイテム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。
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	既知のインフォメーションマップフィルターに関する情報が含まれます。
FORMATS	Vformat Vcformat	現在アクセス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	現在アクセス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ィックスオプション (SAS/GRAPH ソフトウェア)に関する情報が含まれます。Sashelp.Vallopt には、グラフィックスオプションだけでなく 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。
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。

DICTIONARY テーブル	Sashelp ビュー	説明
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	現在定義されている SAS ライブラリにあるすべてのオブジェクトに関する情報が含まれます。Sahelp.Vmember には、すべてのメンバータイプの情報が含まれています。他の Sashelp ビューは、特定のメンバータイプ(テーブルやビューなど)に固有です。
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だけでなくグラフィックス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	既知の記憶に関する情報が含まれます。
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	すべての既知のテーブルの一貫性制約に関する情報が含まれます。
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	使用不可	SQL または DATASTEP ビューによって参照されるテーブル(または他のビュー)のリストと、参照数のカウントが含まれます。
XATTRS	Vxattr	拡張属性に関する情報が含まれます。

DICTIONARY テーブルへのアクセス

辞書テーブルにアクセスするには、SQL プロシジャを使用します。FROM ステートメントでは、ライブラリ参照名“Dictionary”の後にピリオド(.)の後にテーブルの名前が続きます。

```
proc sql;
  select *
  from DICTIONARY.table-name;
quit;
```

例: COLUMNS 辞書テーブルへのアクセス

```
proc sql;
  select *
  from DICTIONARY.COLUMNS;
quit;
```

DICTIONARY ビューへのアクセス

DATA または PROC ステップを使用して、辞書テーブルの関連**ビュー**にアクセスします。ライブラリ参照名“Sashelp”の後にピリオド(.)の後にビューの名前が続きます。

```
data test;
  set sashelp.view-name;
run;
```

例: VCOLUMN 辞書テーブルビューへのアクセス

```
data test;
  set sashelp.VCOLUMN;
run;
```

Sashelp ビュー名は、関連付けられた DICTIONARY テーブル名の短縮版であり、名前の前に文字"V"が追加されていることに注意してください。

SAS DICTIONARY テーブルのパフォーマンス

DICTIONARY テーブルをクエリすると、SAS はそのテーブルに関連する情報を収集します。クエリ対象の DICTIONARY テーブルに応じて、このプロセスにはライブラリを検索したり、テーブルを開いたり、SAS ビューを実行したりする処理が含まれる場合があります。他の SAS プロシジャや DATA ステップとは異なり、PROC SQL では、選択プロセスが起動される前にクエリを最適化することで、このプロセスを向上させることができます。したがって、SAS プロシジャを使用したり、DATA ステップを使用して Sashelp ビューにアクセスしたりして DICTIONARY テーブル情報にアクセスすることは可能ですが、多くの場合、PROC SQL を使用する方が効率的です。

注: SAS は、クエリ間で DICTIONARY テーブル情報を維持しません。DICTIONARY テーブルの各クエリは、新しい検出プロセスを起動します。

同じ DICTIONARY テーブルを連続して複数回クエリする場合は、一時 SAS データセットを作成し、そのデータセットに対してクエリを実行することで、さらに高速なパフォーマンスを得ることができます。DATA ステップ SET ステートメントまたは PROC SQL CREATE TABLE AS ステートメントを使用して、一時データセットを作成できます。

例: DICTIONARY テーブルへのアクセス

例

SQL プロシジャを使用して、現在の SAS セッションの DICTIONARY ライブラリ内の MEMBERS テーブルにアクセスします。PROC SQL ステートメントで OUTOBS= オプションを使用して、クエリ出力に含まれるオブザベーションの数を制限します。

注: DICTIONARY テーブルにアクセスするには、PROC SQL を使用する必要があります。ただし、DATA ステップまたは PROC ステップを使用して、テーブルの関連ビューにアクセスできます。DICTIONARY テーブルビューは Sashelp ライブラリにあり、文字"V"で始まります。

```
proc sql outobs=10;
  select *
```

```
    from dictionary.members;
quit;
```

次のクエリ結果には、DICTIONARY ライブラリ内の MEMBERS テーブルの各データセットに関するメタデータ情報が含まれています。

アウトプット 17.1　クエリ結果

Library Name	Member Name	Member Type	DBMS Member Type	Engine Name	Indexes	Pathname
SASHELP	AACOMP	DATA		V9	yes	('/home/SASFoundation/sashelp')
SASHELP	AARFM	DATA		V9	yes	('/home/SASFoundation/sashelp')
SASHELP	AC	CATALOG		V9	no	('/home/SASFoundation/sashelp')
SASHELP	ADSMSG	DATA		V9	yes	('/home/SASFoundation/sashelp')
SASHELP	AFCLASS	CATALOG		V9	no	('/home/SASFoundation/sashelp')
SASHELP	AFMSG	DATA		V9	yes	('/home/SASFoundation/sashelp')
SASHELP	AFTOOLS	CATALOG		V9	no	('/home/SASFoundation/sashelp')
SASHELP	AIR	DATA		V9	no	('/home/SASFoundation/sashelp')
SASHELP	AIRLINE	DATA		V9	no	('/home/SASFoundation/sashelp')
SASHELP	APPLIANC	DATA		V9	no	('/home/SASFoundation/sashelp')

要点

- DICTIONARY テーブルには、DATA ステップまたは SAS プロシジャから直接アクセスできません。DICTIONARY テーブルにアクセスするには、PROC SQL を使用する必要があります。ただし、DATA または PROC ステップを使用すると、DICTIONARY テーブルに関連付けられたテーブルビューにアクセスできます。
- SAS のライブラリ参照名 DICTIONARY は、PROC SQL によってのみアクセスできる自動システムライブラリを表します。
- SAS は 32 の DICTIONARY テーブルをサポートします。サポートされているテーブルとビューのリストについては、表 17.1 (402 ページ)を参照してください。

例: DICTIONARY テーブルビューを CAS テーブルに書き込み、結果をフィルタリングする

例

SAS DATA ステップを使用して、DICTIONARY テーブルビュー Sashelp.vlibnam の内容を CAS テーブルに書き込みます。次に、CAS で実行されている DATA ステップを使用して、ライブラリが Sashelp で長さが 1200 より大きい行のみに結果をフィルタリングします。この結果は、長さが 1200 を超える変数を持つすべての Sashelp データセット内のすべての変数(列)を表します。

```
cas casauto;
libname mycas cas;

data mycas.columns;
  set sashelp.vcolumn;
  keep libname memname memtype length;
run;

data mycas.columns;
  set mycas.columns(where=(libname="SASHELP" and length>1200));
run;
proc print data=mycas.columns; run;
```

次のクエリ結果には、DICTIONARY ライブラリ内の MEMBERS テーブルの各データセットに関するメタデータ情報が含まれています。

Obs	libname	memname	memtype	length
1	SASHELP	VPRMXML	VIEW	32767

要点

- DICTIONARY テーブルには、DATA ステップまたは SAS PROC から直接アクセスできません。
- DICTIONARY テーブルにアクセスするには、PROC SQL を使用する必要があります。ただし、DATA または PROC ステップを使用すると、DICTIONARY テーブルに関連付けられたテーブルビューにアクセスできます。

- SAS のライブラリ参照名 DICTIONARY は、PROC SQL によってのみアクセスできる自動システムライブラリを表します。

例: DICTIONARY テーブルの要約の表示

例

PROC SQL で DESCRIBE TABLE ステートメントを使用して、DICTIONARY.MEMBERS テーブルの内容の要約を生成します。

```
proc sql;
  describe table dictionary.members;
quit;
```

次のようにログに出力されます。

例のコード 17.1　ログ

```
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',
  dbms_memtype char(32) label='DBMS Member Type',
  engine char(8) label='Engine Name',
  index char(3) label='Indexes',
  path char(1024) label='Pathname'
 );
```

- 各行の最初の単語は変数名です。たとえば、**libname** は変数名です。変数を参照する SAS ステートメントを記述する場合は、変数名を使用します。
- 変数名の後に変数のデータ型が続き、その後に幅が続きます。たとえば、**libname** 変数のデータ型は char (文字)で、**libname** 変数の幅は 8 です。
- label=は、出力が**結果**タブに表示されるときに、その変数に対して表示される名前です。たとえば、出力が表示されると、**libname** 変数が Library Name として表示されます。

要点

- DICTIONARY テーブルには、SQL プロシジャを除いて、DATA ステップまたは SAS PROC から直接アクセスすることはできません。

例: DICTIONARY テーブルのサブセットの表示

例

PROC SQL ステップで WHERE 句を使用して、DICTIONARY.MEMBERS テーブルから情報のサブセットを抽出します。

注意
CONTENTS OUT=データセットの GENNUM 変数値と、DICTIONARY テーブルの GEN 変数値を混同しないでください。 CONTENTS プロシジャまたはステートメントの GENNUM は、データセットの特定の世代を参照します。DICTIONARY テーブルの GEN は、データセットの総世代数を指します。

```
proc sql;
title1 'Subset of the DICTIONARY.MEMBERS Table';
  select libname, memname, index
    from dictionary.members
    where index='yes';
quit;
```

ヒント　WHERE 句の値では大文字と小文字が区別されます。大文字の YES を使用すると、結果は見つかりません。

以下のクエリ結果には、DICTIONARY.MEMBERS テーブル内のインデックスの値が yes である行のみが含まれます。

アウトプット 17.2　部分出力: クエリ結果

Subset of the DICTIONARY.MEMBERS Table

Library Name	Member Name	Indexes
SASHELP	AACOMP	yes
SASHELP	AARFM	yes
SASHELP	ADSMSG	yes
SASHELP	AFMSG	yes
SASHELP	BRM	yes
SASHELP	CLNMSG	yes
SASHELP	CREDIT	yes
SASHELP	DMCAS	yes
SASHELP	DMGMSG	yes
SASHELP	DMINE	yes
SASHELP	EISMBRP	yes
SASHELP	EISMSG	yes
SASHELP	ETSMSG	yes
SASHELP	GCDIRECT	yes
SASHELP	GEOEXM	yes
SASHELP	GNGMSG	yes
SASHELP	IMGMSG	yes
SASHELP	LSWMSG	yes

. . . more observations . . .

要点

- すべての DICTIONARY テーブルには、Sashelp ライブラリに関連付けられた PROC SQL ビューがあります。
- VIEWTABLE または FSVIEW ユーティリティを使用して Sashelp ビューを開くと、DICTIONARY テーブルの内容全体を確認できます。この方法では、DICTIONARY テーブルの要約を表示する方法に示すように、DESCRIBE TABLE ステートメントの出力で受け取るよりも詳細な情報が得られます。
- DICTIONARY テーブル内の多くの文字値は、すべて大文字として保存されます。それに応じてクエリを設計する必要があります。
- DICTIONARY テーブルには、SQL プロシジャを除いて、DATA ステップまたは SAS PROC から直接アクセスすることはできません。

4 部

データの操作

18

データのグループ化

BY グループ処理の定義

BY グループ処理
1 つ以上の共通変数の値によってグループ化または順序付けされた 1 つ以上の SAS データセットからのオブザベーションを処理する方法です。DATA ステップでの BY グループ処理の最も一般的な使用法は、2 つ以上の SAS データセットを

結合することです。これを行うには、SET、MERGE、MODIFY、UPDATE のいずれかのステートメントとともに BY ステートメントを使用します。“BY ステートメント” (*SAS DATA ステップステートメント: リファレンス*)(完全な構文情報)を参照してください。

BY 変数
データセットの並べ替えまたはインデックス付けに使用される変数名を指定します。SET、MERGE、または UPDATE ステートメントを使用する場合は、すべてのデータセットを BY 変数の値に基づいて並べ替えるか、インデックスを付ける必要があります。MODIFY を使用する場合、データを順序付ける必要はありません。ただし、プログラムは、順序付けされたデータを使用するとより効率的に実行される可能性があります。結合されるすべてのデータセットには、1 つ以上の BY 変数が含まれている必要があります。オブザベーション内の BY 変数の位置は重要ではありません。

BY 値
BY 変数の値またはフォーマット値です。

BY グループ
同じ BY 値を持つすべてのオブザベーションが含まれます。BY ステートメントで複数の変数を使用する場合、BY グループは、これらの変数の値の同じ組み合わせを持つオブザベーションのグループです。各 BY グループには、変数の値の一意の組み合わせがあります。

FIRST.変数と LAST.変数
SAS が BY 変数ごとに作成する変数です。SAS は BY グループ内の最初のオブザベーションを処理するときに FIRST.*variable* を設定し、BY グループ内の最後のオブザベーションを処理するときに LAST.*variable* を設定します。これらの割り当てにより、処理が新しい BY グループに対して開始されるのか、それとも BY グループに対して終了するのかに基づいて、さまざまなアクションを実行できるようになります。詳細については、“FIRST.と LAST.DATA ステップ変数” (416 ページ)を参照してください。

BY グループ処理の詳細については、“データの結合” (517 ページ)を参照してください。*SAS データセットの結合と変更: 例*も参照してください。

BY グループ処理を呼び出す方法

BY グループを作成するには、次の 2 つの方法のいずれかで BY ステートメントを使用します。

- DATA ステップ
- PROC ステップ(プロシジャによる BY グループ処理の詳細については、“BY グループの情報を含むタイトルの作成 ” (*Base SAS プロシジャガイド*)を参照してください。)

次の DATA ステッププログラムでは、SET ステートメントを使用して、ファイルをインターリーブすることで 3 つの SAS データセットからのオブザベーションを結合します。データは州、市、ZIP コードで順序付けされます。

```
data all_sales;
```

```
  set region1 region2 region3;
  by State City Zip;
  ... more SAS statements ...
run;
```

BY グループ処理のためのデータの前処理

データの前処理が必要かどうかを判断する

SET、MERGE、および UPDATE ステートメントを使用して複数のデータセットに対して BY グループ処理を実行する前に、データをチェックして前処理が必要かどうかを判断する必要があります。すべてのデータセットのオブザベーションが次のいずれかのパターンで出現する場合、データは前処理を必要としません。

- 数値の昇順または降順
- 文字の昇順または降順
- アルファベット順や数値順ではなく、カレンダー月ごとやフォーマット値ごとなど、なんらかの方法でグループ化されている

オブザベーションが希望の順序になっていない場合は、BY グループ処理を使用する前に、データセットを並べ替えるか、そのインデックスを作成する必要があります。

BY グループ処理で MODIFY ステートメントを使用する場合、入力データの事前並べ替えは必要ありません。ただし、事前並べ替えにより、処理がより効率的になり、コストが削減されます。

PROC SQL ビューは BY グループ処理で使用できます。完全な情報については、*SAS SQL プロシジャユーザーガイド*を参照してください。

注: SAS/ACCESS ユーザー: SAS ビューまたはライブラリ参照名を使用する場合、SAS プログラムでの BY グループの使用については、*リレーショナルデータベース用 SAS/ACCESS: リファレンス*を参照してください。

BY グループ処理のためのオブザベーションの並べ替え

SORT プロシジャを使用して、データセット内のオブザベーションの物理的な順序を変更できます。元のデータセットを置き換えることも、SORT プロシジャの OUT=オ

プションを使用して新しい並べ替えられたデータセットを作成することもできます。この例では、PROC SORT は、変数 State および ZipCode の昇順の値に基づいてデータセット INFORMATION 内のオブザベーションを再配置し、元のデータセットを置き換えます。

```
proc sort data=information;
  by State ZipCode;
run;
```

原則として、PROC SORT BY ステートメントの変数は、DATA ステップ BY ステートメントで指定するのと同じ順序で指定します。数値変数と文字変数のデフォルトの並べ替え順序の詳細については、*Base SAS プロシジャガイド*の SORT プロシジャを参照してください。

注: SORTSEQ=LINGUISTIC オプションを指定して SORT プロシジャを使用する場合、BY ステートメントは並べ替えられたデータの言語照合を尊重します。

BY グループ処理のインデックス

データセット内の 1 つ以上の変数に基づいてインデックスを作成することで、オブザベーションが昇順で処理されるようにすることもできます。DATA ステップで BY ステートメントを指定すると、SAS は適切なインデックスを検索します。インデックスが見つかった場合、SAS はインデックス付きの順序でデータセットからオブザベーションを自動的に取得します。

注: インデックスの作成と維持には追加のリソースが必要なため、それらを使用するとパフォーマンスが大幅に向上するかどうかを判断する必要があります。SAS データセット内のデータの性質によっては、PROC SORT を使用してデータ値を順序付けする方が、インデックス付けよりも有利な場合があります。インデックスの概要については、“SAS のインデックス” (609 ページ)を参照してください。

FIRST.と LAST.DATA ステップ変数

DATA ステップで BY グループを識別する方法

DATA ステップでは、SAS は BY 変数ごとに次の 2 つの一時変数を作成することによって、各 BY グループの始まりと終わりを識別します。

- FIRST.*variable*
- LAST.*variable*

たとえば、DATA ステップが BY ステートメントで変数 state を指定する場合、SAS は一時変数 FIRST.state と LAST.state を作成します。

これらの一時変数は DATA ステッププログラミングに使用できますが、出力データセットには追加されません。それらの値は、オブザベーションが次の位置のいずれかであるかを示します。

- BY グループの最初
- BY グループの最後
- BY グループの最初でも最後でもない
- 最初と最後の両方(BY グループにオブザベーションが 1 つだけある場合と同様)

BY グループ内の最初のオブザベーションを処理しているか最後のオブザベーションを処理しているかに基づいて、条件付きでアクションを実行できます。詳細については、“BY グループの条件付き処理” (418 ページ)を参照してください。

SAS による FIRST.*variable* と LAST.*variable* の決定方法

- オブザベーションが BY グループの最初である場合、SAS は FIRST.*variable* の値を 1 に設定します。これは、変数の値が前のオブザベーションから変化したときに発生します。
- BY グループ内の他のすべてのオブザベーションでは、FIRST.*variable* の値は 0 です。
- オブザベーションが BY グループの最後である場合、SAS は LAST.*variable* の値を 1 に設定します。これは、変数の値が次のオブザベーションで変化するか、データセット内の最後のオブザベーションである場合に発生します。
- BY グループ内の他のすべてのオブザベーションでは、LAST.*variable* の値は 0 です。
- データセット内の最後のオブザベーションでは、すべての LAST.*variable* 変数の値が 1 に設定されます。

名前リテラルを FIRST.と LAST.として使用する変数

BY グループ処理において、BY 変数として名前リテラルを指定し、対応する FIRST. または LAST.一次変数を参照する場合は、2 レベル変数名の FIRST.またはたは LAST. 部分を引用符で囲む必要があります。

```
data sedanTypes;
  set cars;
  by 'Sedan Types'n;
  if 'first.Sedan Types'n then type=1;
run;
```

DATA ステップでの BY グループの処理

概要

BY グループ処理の最も一般的な使用法は、BY ステートメントと SET、MERGE、MODIFY、UPDATE ステートメントのいずれかを使用してデータセットを結合することです。(SET、MERGE、または UPDATE ステートメントを BY ステートメントとともに使用する場合は、オブザベーションをグループ化し、順序付けする必要があります。)これらのステートメントを処理するとき、SAS は一度に 1 つのオブザベーションをプログラムデータベクトルに読み込みます。BY グループ処理を使用すると、SAS は BY 変数の値に従ってデータセットからオブザベーションを選択します。1 つの BY グループからのすべてのオブザベーションを 処理した後、SAS は次のオブザベーションが次の BY グループからのものであることを期待します。

BY ステートメントは、プログラムデータベクトル内の値がいつ欠損値に設定されるかを制御することにより、SET、MERGE、MODIFY、UPDATE ステートメントのいずれかのアクションを変更します。BY グループ処理中、SAS は、いずれかのデータセット内でその BY グループについて見つかった最後のオブザベーションをコピーするまで、変数の値を保持します。BY ステートメントを使用しないと、SET ステートメントは最後のオブザベーションを読み取るときに変数を欠損に設定します。MERGE ステートメントは、DATA ステップがプログラムデータベクトルへのオブザベーションの読み込みを開始した後、変数を欠損に設定しません。

BY グループの条件付き処理

一時変数 FIRST.*variable* および LAST.*variable* (BY グループ処理中に設定)を使用してサブセット化 IF ステートメントまたは IF-THEN ステートメント、あるいは SELECT ステートメントを使用することにより、条件付きでオブザベーションを処理できます。たとえば、IF または IF THEN ステートメントを使用して、各 BY グループの計算を実行し、BY グループの最初または最後のオブザベーションがプログラムデータベクトルに読み込まれたときにオブザベーションを書き込むことができます。

アルファベット順でも数字順でもないデータ

BY グループ処理では、カレンダー月順やカテゴリ順など、アルファベットや数字以外の順序で配置されたデータも使用できます。これを行うには、SET ステートメントを使用するときに BY ステートメントで NOTSORTED オプションを使用します。BY ステートメントの NOTSORTED オプションは、データがアルファベット順また

は数字順ではなく、BY 変数の値によってグループに 配置されていることを SAS に伝えます。NOTSORTED オプションは、MERGE ステートメント、UPDATE ステートメント、または SET ステートメントで複数のデータセットをリストする場合には使用できません。

フォーマット値によってグループ化されたデータ

BY ステートメントで GROUPFORMAT オプションを使用して、次の方法が確実に適用されるようにします。

- フォーマット値は、DATA ステップで FORMAT ステートメントと BY ステートメントが一緒に使用される場合に、オブザベーションをグループ化するために使用される
- FIRST.*variable* と LAST.*variable* は、変数のフォーマット値によって割り当てられる

GROUPFORMAT オプションは、SAS データセットを作成する DATA ステップでのみ有効です。これは、ユーザー定義の出力形式の場合に特に便利です。例については、"BY ステートメント" (*SAS DATA ステップステートメント: リファレンス*)および"例: GROUPFORMAT と出力形式を使用した BY グループの処理" (431 ページ)を参照してください。

例: FIRST.と LAST.を使用した BY グループの処理 DATA ステップ変数

例: 州、市、および ZIP コードによるオブザベーションのグループ化

コード例

この例では、SAS が FIRST.*variable* と LAST.*variable* を使用して BY グループの開始と終了にフラグを設定する方法を示します。

```
data zip;
input State $ City $ ZipCode Street $20-29;
datalines;
FL Miami    33133 Rice St
FL Miami    33133 Thomas Ave
FL Miami    33133 Surrey Dr
```

```
FL Miami    33133 Trade Ave
FL Miami    33146 Nervia St
FL Miami    33146 Corsica St
FL Lakeland  33801 French Ave
FL Lakeland  33809 Egret Dr
AZ Tucson   85730 Domenic Ln
AZ Tucson   85730 Gleeson Pl
;
proc sort data=zip;
  by State City ZipCode;
run;

data zip2;
  set zip;
  by State City ZipCode;
  put _n_= City State ZipCode
  first.city= last.city=
  first.state= last.state=
  first.ZipCode= last.ZipCode= ;
run;
```

例のコード 18.1　州、市、および ZIP コードによるオブザベーションのグループ化

```
_N_=1  AZ Tucson   85730   FIRST.State=1 LAST.State=0   FIRST.City=1 LAST.City=0   FIRST.ZipCode=1
LAST.ZipCode=0
_N_=2  AZ Tucson   85730   FIRST.State=0 LAST.State=1   FIRST.City=0 LAST.City=1   FIRST.ZipCode=0
LAST.ZipCode=1
_N_=3  FL Lakeland 33801   FIRST.State=1 LAST.State=0   FIRST.City=1 LAST.City=0   FIRST.ZipCode=1
LAST.ZipCode=1
_N_=4  FL Lakeland 33809   FIRST.State=0 LAST.State=0   FIRST.City=0 LAST.City=1   FIRST.ZipCode=1
LAST.ZipCode=1
_N_=5  FL Miami    33133   FIRST.State=0 LAST.State=0   FIRST.City=1 LAST.City=0   FIRST.ZipCode=1
LAST.ZipCode=0
_N_=6  FL Miami    33133   FIRST.State=0 LAST.State=0   FIRST.City=0 LAST.City=0   FIRST.ZipCode=0
LAST.ZipCode=0
_N_=7  FL Miami    33133   FIRST.State=0 LAST.State=0   FIRST.City=0 LAST.City=0   FIRST.ZipCode=0
LAST.ZipCode=0
_N_=8  FL Miami    33133   FIRST.State=0 LAST.State=0   FIRST.City=0 LAST.City=0   FIRST.ZipCode=0
LAST.ZipCode=1
_N_=9  FL Miami    33146   FIRST.State=0 LAST.State=0   FIRST.City=0 LAST.City=0   FIRST.ZipCode=1
LAST.ZipCode=0
_N_=10 FL Miami    33146   FIRST.State=0 LAST.State=1   FIRST.City=0 LAST.City=1   FIRST.ZipCode=0
LAST.ZipCode=1
```

図 18.1　州、市、および ZIP コードの BY グループ

Observations in Three BY Groups			Corresponding FIRST. and LAST. Variables					
State	City	ZipCode	FIRST. State	LAST. State	FIRST. City	LAST. City	FIRST. ZipCode	LAST. ZipCode
AZ	Tucson	85730	1	0	1	0	1	0
AZ	Tucson	85730	0	1	0	1	0	1
FL	Lakeland	33801	1	0	1	0	1	1
FL	Lakeland	33809	0	0	0	1	1	1
FL	Miami	33133	0	0	1	0	1	0
FL	Miami	33133	0	0	0	0	0	0
FL	Miami	33133	0	0	0	0	0	0
FL	Miami	33133	0	0	0	0	0	1
FL	Miami	33146	0	0	0	0	1	0
FL	Miami	33146	0	1	0	1	0	1

注: ログの内容をわかりやすく表示するためのチャートです。これは出力データセットではありません。

要点

- FIRST 変数と LAST 変数は、SAS によって自動的に作成される一時変数で、各 BY グループの始まりと終わりを表します。
- DATA ステップで FIRST 変数と LAST 変数を参照できますが、これらは出力データセットの一部ではありません。
- 自動変数_N_は、DATA ステップ反復のカウンターとして使用されます。ログ出力で反復を確認できます。

関連項目

- “FIRST.と LAST.DATA ステップ変数” (416 ページ)
- “SORT プロシジャ” (*Base SAS プロシジャガイド*)

例: 州、市、および ZIP コードによるオブザベーションのグループ化

コード例

各 BY 変数は、FIRST.State、LAST.State、FIRST.City、LAST.City、FIRST.ZipCode、LAST.ZipCode の一時変数を作成します。

```
data zip;
input State $ City $ ZipCode Street $20-29;
datalines;
FL Miami    33133 Rice St
FL Miami    33133 Thomas Ave
FL Miami    33133 Surrey Dr
FL Miami    33133 Trade Ave
FL Miami    33146 Nervia St
FL Miami    33146 Corsica St
FL Lakeland  33801 French Ave
FL Lakeland  33809 Egret Dr
AZ Tucson    85730 Domenic Ln
AZ Tucson    85730 Gleeson Pl
;
proc sort data=zip;
by  City State ZipCode;
run;

data zip2;
  set zip;
```

例のコード 18.2　州、市、および ZIP コードによるオブザベーションのグループ化

```
_N_=1  Lakeland FL 33801 FIRST.City=1 LAST.City=0 FIRST.State=1 LAST.State=0 FIRST.ZipCode=1
LAST.ZipCode=1
_N_=2  Lakeland FL 33809 FIRST.City=0 LAST.City=1 FIRST.State=0 LAST.State=1 FIRST.ZipCode=1
LAST.ZipCode=1
_N_=3  Miami    FL 33133 FIRST.City=1 LAST.City=0 FIRST.State=1 LAST.State=0 FIRST.ZipCode=1
LAST.ZipCode=0
_N_=4  Miami    FL 33133 FIRST.City=0 LAST.City=0 FIRST.State=0 LAST.State=0 FIRST.ZipCode=0
LAST.ZipCode=0
_N_=5  Miami    FL 33133 FIRST.City=0 LAST.City=0 FIRST.State=0 LAST.State=0 FIRST.ZipCode=0
LAST.ZipCode=0
_N_=6  Miami    FL 33133 FIRST.City=0 LAST.City=0 FIRST.State=0 LAST.State=0 FIRST.ZipCode=0
LAST.ZipCode=1
_N_=7  Miami    FL 33146 FIRST.City=0 LAST.City=0 FIRST.State=0 LAST.State=0 FIRST.ZipCode=1
LAST.ZipCode=0
_N_=8  Miami    FL 33146 FIRST.City=0 LAST.City=1 FIRST.State=0 LAST.State=1 FIRST.ZipCode=0
LAST.ZipCode=1
_N_=9  Tucson   AZ 85730 FIRST.City=1 LAST.City=0 FIRST.State=1 LAST.State=0 FIRST.ZipCode=1
LAST.ZipCode=0
_N_=10 Tucson   AZ 85730 FIRST.City=0 LAST.City=1 FIRST.State=0 LAST.State=1 FIRST.ZipCode=0
LAST.ZipCode=1
```

図 18.2　州、市、および ZIP コードの BY グループ

Observations in Three BY Groups			Corresponding FIRST. and LAST. Variables					
State	City	ZipCode	FIRST. State	LAST. State	FIRST. City	LAST. City	FIRST. ZipCode	LAST. ZipCode
AZ	Tucson	85730	1	0	1	0	1	0
AZ	Tucson	85730	0	1	0	1	0	1
FL	Lakeland	33801	1	0	1	0	1	1
FL	Lakeland	33809	0	0	0	1	1	1
FL	Miami	33133	0	0	1	0	1	0
FL	Miami	33133	0	0	0	0	0	0
FL	Miami	33133	0	0	0	0	0	0
FL	Miami	33133	0	0	0	0	0	1
FL	Miami	33146	0	0	0	0	1	0
FL	Miami	33146	0	1	0	1	0	1

注: ログの内容をわかりやすく表示するためのチャートです。これは出力データセットではありません。

要点

- FIRST 変数と LAST 変数は、SAS によって自動的に作成される一時変数です。これらは、各 BY グループの始まりと終わりを表します。
- DATA ステップで FIRST 変数と LAST 変数を参照できますが、これらは出力データセットの一部ではありません。

- 自動変数_N_は、DATA ステップ反復のカウンターとして使用されます。ログ出力で反復を確認できます。

関連項目

- “FIRST.と LAST.DATA ステップ変数” (416 ページ)
- “SORT プロシジャ” (*Base SAS プロシジャガイド*)

例: FIRST.の値を変更する変数

コード例

変数の現在の値が同じままであっても、前の値を変更すると、FIRST.*variable* の値に影響を与える可能性があります。

この例では、FIRST.*variable* と LAST.*variable* の値は、並べ替え順序と BY 変数の値に依存します。

```
data fruit;
  input x $ y $ 10-18 z $ 21-29;
  datalines;
apple    banana     coconut
apple    banana     coconut
apple    blueberry  citron
apricot  blueberry  citron
;

data _null_;
  set fruit;
  by x y z;
  if _N_=1 then put 'Grouped by X Y Z';
  put _N_= x= first.x= last.x= first.y= last.y= first.z= last.z= ;
run;

data _null_;
  set fruit;
  by y x z;
  if _N_=1 then put 'Grouped by Y X Z';
  put _N_= first.y= last.y= first.x= last.x= first.z= last.z= ;
run;
```

オブザベーション 3 では、BLUEBERRY が Y の新しい値であるため、FIRST.*Y* の値は 1 に設定されます。この Y の変化により、Z の値が変化していないにもかかわらず、FIRST.*Z* も 1 に設定されます。

```
Grouped by X Y Z
_N_=1 FIRST.x=1 LAST.x=0 FIRST.y=1 LAST.y=0 FIRST.z=1 LAST.z=0
_N_=2 FIRST.x=0 LAST.x=0 FIRST.y=0 LAST.y=1 FIRST.z=0 LAST.z=1
_N_=3 FIRST.x=0 LAST.x=1 FIRST.y=1 LAST.y=1 FIRST.z=1 LAST.z=1
_N_=4 FIRST.x=1 LAST.x=1 FIRST.y=1 LAST.y=1 FIRST.z=1 LAST.z=1
```

```
Grouped by Y X Z
_N_=1 FIRST.y=1 LAST.y=0 FIRST.x=1 LAST.x=0 FIRST.z=1 LAST.z=0
_N_=2 FIRST.y=0 LAST.y=1 FIRST.x=0 LAST.x=1 FIRST.z=0 LAST.z=1
_N_=3 FIRST.y=1 LAST.y=0 FIRST.x=1 LAST.x=1 FIRST.z=1 LAST.z=1
_N_=4 FIRST.y=0 LAST.y=1 FIRST.x=1 LAST.x=1 FIRST.z=1 LAST.z=1
```

要点

- FIRST 変数と LAST 変数は、SAS によって自動的に作成される一時変数です。これらは、各 BY グループの始まりと終わりを表します。
- DATA ステップで FIRST 変数と LAST 変数を参照できますが、これらは出力データセットの一部ではありません。
- 自動変数_N_は、DATA ステップ反復のカウンターとして使用されます。ログ出力で反復を確認できます。

関連項目

- “FIRST.と LAST.DATA ステップ変数”

例: FIRST.と LAST.を使用する BY グループを条件付きで処理する変数

コード例

次の例では、部門ごとの年間給与を計算します。IF-THEN ステートメントと FIRST.*variable* および LAST.*variable* 自動変数の値を使用して、各 BY グループの開始時に PAYROLL の値を 0 にリセットし、BY グループ内の最後のオブザベーションが処理された後にオブザベーションを書き込みます。

```
data salaries;
  input Department $ Name $ WageCategory $ WageRate;
  datalines;
BAD Carol Salaried 20000
BAD Elizabeth Salaried 5000
```

```
BAD Linda Salaried 7000
BAD Thomas Salaried 9000
BAD Lynne Hourly 230
DDG Jason Hourly 200
DDG Paul Salaried 4000
PPD Kevin Salaried 5500
PPD Amber Hourly 150
PPD Tina Salaried 13000
STD Helen Hourly 200
STD Jim Salaried 8000
;

proc print data=salaries;
run;

proc sort data=salaries out=temp;
  by Department;
run;

data budget (keep=Department Payroll);
  set temp;
  by Department;
  if WageCategory='Salaried' then YearlyWage=WageRate*12;
  else if WageCategory='Hourly' then YearlyWage=WageRate*2000;

  if first.Department then Payroll=0;    /* 1 */
  Payroll+YearlyWage;
  if last.Department;                    /* 2 */
run;

proc print data=budget;
  format Payroll dollar10.;
  title 'Annual Payroll by Department';
run;
```

1 これが BY グループ内の新しい部門である場合、SAS は FIRST.**Department** を 1 に設定します。

2 これが現在の BY グループの最後の部門である場合、SAS は LAST.**Department** を 1 に設定します。

アウトプット 18.1　条件付き BY グループ処理からの出力

Annual Payroll by Department

Obs	Department	Payroll
1	BAD	$952,000
2	DDG	$448,000
3	PPD	$522,000
4	STD	$496,000

要点

- FIRST 変数と LAST 変数は、SAS によって自動的に作成される一時変数です。これらは、各 BY グループの始まりと終わりを表します。
- IF/THEN ステートメントは条件付きでステートメントを実行します。

関連項目

- “IF-THEN/ELSE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “FIRST.と LAST.DATA ステップ変数” (416 ページ)

例: DATA ステップでの BY グループの処理

例: 単一の BY 変数を使用したオブザベーションのグループ化

コード例

次の例は、DATA ステップで単一の BY 変数 zipCode を使用してデータをグループ化する方法を示しています。入力データセット zip には、通り名、市、州、ZIP コードが含まれます。グループは、BY ステートメントで変数 **zipCode** を指定することによって作成されます。DATA ステップでは、同じ値を持つ ZIP コードをグループに整理します。

```
data zip;
input zipCode State $ City $ Street $20-29;
datalines;
85730 AZ Tucson    Domenic Ln
85730 AZ Tucson    Gleeson Pl
33133 FL Miami     Rice St
33133 FL Miami     Thomas Ave
```

```
33133 FL Miami     Surrey Dr
33133 FL Miami     Trade Ave
33146 FL Miami     Nervia St
33146 FL Miami     Corsica St
33801 FL Lakeland  French Ave
33809 FL Lakeland  Egret Dr
;

proc sort data=zip;
  by zipCode;
run;

data zip;
  set zip;
  by zipCode;
run;

proc print data=zip noobs;
  title 'BY-Group Uing a Single Variable: ZipCode';
run;
```

この図は、作成された 5 つの BY グループを示しています。

図 18.3　単一の BY 変数(ZipCode)を使用した BY グループ

要点

- SORT プロシジャは、データセット内のオブザベーションの物理的な順序を変更します。OUT=オプションを使用すると、別の名前で新しい並べ替えられたデータセットを作成できます。ただし、この例では、zip データセットを置き換えます。
- 最初の BY グループには、BY 変数 State および City の最小値を持つすべてのオブザベーションが含まれます。2 番目の BY グループには、BY 変数の次に小さい値を持つすべてのオブザベーションが含まれ、以下同様に続きます。

関連項目

- “SORT プロシジャ” (*Base SAS プロシジャガイド*)
- “DATA ステップでの BY グループの処理” (418 ページ)

例: 複数の BY 変数を使用したオブザベーションのグループ化

コード例

次の例は、State と City の 2 つの BY 変数を含む zip データセットを処理した結果を示しています。

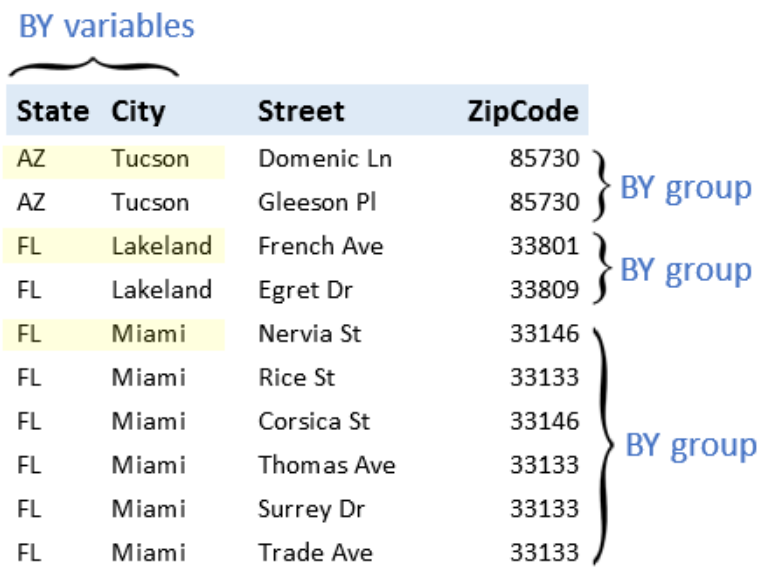
```
data zip;
input State $ City $ Street $13-22 ZipCode ;
datalines;
FL Miami    Nervia St  33146
FL Miami    Rice St    33133
FL Miami    Corsica St 33146
FL Miami    Thomas Ave 33133
FL Miami    Surrey Dr  33133
FL Miami    Trade Ave  33133
FL Lakeland French Ave 33801
FL Lakeland Egret Dr   33809
AZ Tucson   Domenic Ln 85730
AZ Tucson   Gleeson Pl 85730
;


proc sort data=zip;
```

```
  by State City;
run;

data zip;
  set zip;
  by State City;
run;
proc print data=zip noobs;
  title 'BY Groups with Multiple BY Variables: State City';
run;
```

この図には 3 つの BY グループが示されています。データセットは、読みやすいように左側に出力された BY 変数 State と City とともに示されています。オブザベーション内の BY 変数の位置は、値のグループ化と順序付けの方法には影響しません。

オブザベーションは、アリゾナのオブザベーションが最初に出現するように配置されます。State の各値内のオブザベーションは、City の値の順に配置されます。各 BY グループには、変数 State と City の値の一意の組み合わせがあります。たとえば、最初の BY グループの BY 値は AZ Tucson、2 番目の BY グループの BY 値は FL Lakeland になります。

図 18.4　複数の BY 変数(State と City)を持つ BY グループ



要点

- SORT プロシジャは、データセット内のオブザベーションの物理的な順序を変更します。OUT=オプションを使用すると、別の名前で新しい並べ替えられたデータセットを作成できます。ただし、この例では、zip データセットを置き換えます。
- 最初の BY グループには、BY 変数 **zipCode** の最小値を持つすべてのオブザベーションが含まれます。2 番目の BY グループには、BY 変数の次に小さい値を持つすべてのオブザベーションが含まれ、以下同様に続きます。

関連項目

- “SORT プロシジャ” (*Base SAS プロシジャガイド*)
- “DATA ステップでの BY グループの処理” (418 ページ)

例: GROUPFORMAT と出力形式を使用した BY グループの処理

コード例

この例では、FORMAT プロシジャ、GROUPFORMAT オプション、および FORMAT ステートメントを使用して、単純なデータセットを作成および出力します。入力 TEST データセットは昇順の値で並べ替えられます。NEWTEST データセットは、変数 Score のフォーマット値によって配置されます。

```
options
linesize=80 pagesize=60;

data test;
  input name $ Score;
datalines;
Jon    1
Anthony 3
Miguel  3
Joseph  4
Ian    5
Jan    6
;
proc format;
  value Range 1-2='Low'
              3-4='Medium'
              5-6='High';
run;

data newtest;
  set test;
  by groupformat Score;
  format Score Range.;
run;

proc print data=newtest;
  title 'Score Categories';
  var Name Score;
  by Score;
run;
```

アウトプット 18.2　BY ステートメント GROUPFORMAT オプションを使用した SAS 出力

Score Categories

Score=Low

Obs	name	Score
1	Jon	Low

Score=Medium

Obs	name	Score
2	Anthony	Medium
3	Miguel	Medium
4	Joseph	Medium

Score=High

Obs	name	Score
5	Ian	High
6	Jan	High

要点

- GROUPFORMAT オプションを使用した DATA ステップでの BY グループ処理は、SAS プロシジャでのフォーマット値を使用した BY グループ処理と同じです。GROUPFORMAT オプションの使用は、グループ化されたデータを表示するための独自の出力形式を定義する場合に便利です。
- DATA ステップで GROUPFORMAT オプションを使用すると、データセットの作成に使用する BY グループが、グループ化され、フォーマットされたデータをレポートする PROC ステップの BY グループと確実に一致します。GROUPFORMAT は、FIRST.variable と LAST.variable の割り当て方法も決定します。

関連項目

- “BY ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “DATA ステップでの BY グループの処理” (418 ページ)
- “FIRST.と LAST.DATA ステップ変数” (416 ページ)
- “FORMAT プロシジャ” (*Base SAS プロシジャガイド*)

19

WHERE 式の処理

WHERE 式処理の定義

WHERE 式処理
: オブザベーションのサブセットを条件付きで選択できるため、SAS は指定された一連の条件を満たすオブザベーションのみを処理します。たとえば、売上レコードを含む SAS データセットがある場合、売上が$300,000 を超え、$600,000 未満であるオブザベーションのサブセットだけを出力する場合があります。さらに、WHERE 式処理により、要求の効率が向上します。たとえば、インデックスを使用して WHERE 式を最適化できる場合、SAS は要求を実行するためにデータセット内のすべてのオブザベーションを 読み取る必要はありません。

WHERE 式
: 選択したオブザベーションが処理されるために満たさなければならない条件を定義します。次のような単純式と呼ばれる単一の WHERE 式を使用できます。

```
where sales gt 600000;
```

または、次のような複合式と呼ばれる複数の WHERE 式を使用することもできます。

```
where sales gt 600000 and salary lt 100000;
```

WHERE 式の使用場所

SAS では、次の状況で WHERE 式を使用できます。

- DATA ステップと PROC ステップの両方で WHERE ステートメントを使用します。たとえば、次の PRINT プロシジャには WHERE ステートメントが含まれているため、年が 2001 より大きいオブザベーションのみが出力されます。

```
proc print data=employees;
  where startdate > '01jan2001'd;
run;
```

- WHERE=データセットオプション。次の PRINT プロシジャには、WHERE=データセットオプションが含まれています。

```
proc print data=employees (where=(startdate > '01jan2001'd));
run;
```

- SQL プロシジャ、SCL、および SAS/IML ソフトウェアの WHERE 句。たとえば、次の SQL プロシジャには、殺人件数が 7 件を超える州のみを選択する WHERE 句が含まれています。

```
proc sql;
  select state from crime
  where murder > 7;
```

- SAS/FSP ソフトウェアなどのウィンドウ環境の WHERE コマンド:

```
where age > 15
```

- SAS ビュー(DATA ステップビュー、SAS/ACCESS ビュー、PROC SQL ビュー)。定義とともに保存されます。たとえば、次の SQL プロシジャは、データファイル Crime から STAT という名前の SQL ビューを作成し、SQL ビュー定義の WHERE 式を定義します。

```
proc sql;
  create view stat as
  select * from crime
  where murder > 7;
```

場合によっては、WHERE 式の指定に使用するメソッドを組み合わせることができます。つまり、次のように WHERE ステートメントを使用できます。

- WHERE=データセットオプションと組み合わせる
- ウィンドウプロシジャの WHERE=データセットオプションとともに、および WHERE コマンドと組み合わせる
- WHERE 式が保存されている SAS ビュー上

たとえば、データセットをマージするときにメソッドを組み合わせると便利な場合があります。つまり、データのサブセットを作成するときに、各データセットに異なる基準を適用することが必要な場合があります。ただし、メソッドを組み合わせてデータのサブセットを作成する場合は、いくつかの制限があります。たとえば、DATA ステップでは、WHERE ステートメントと WHERE=データセットオプションが同じデータセットに適用される場合、データセットオプションが優先されます。詳細については、WHERE 式の指定に使用しているメソッドのドキュメントを参照してください。

注: デフォルトでは、WHERE 式は追加および変更されたオブザベーションを評価しません。WHERE 式で更新を評価するかどうかを指定するには、WHEREUP=データセットオプションを指定します。"WHEREUP= データセットオプション" (*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。

WHERE 式の構文

WHERE 式の内容

WHERE 式は、オブザベーションを選択するための条件を定義する SAS 式の一種です。WHERE 式は、単一の変数名または定数(固定値)のような単純なものにすることができます。WHERE 式は、SAS 関数にすることも、オブザベーションを選択するための条件を定義する一連のオペランドや演算子にすることもできます。一般に、WHERE 式の構文は次のとおりです。

WHERE *operand* <*operator*> <*operand*>

operand
演算の対象。オペランドには、変数、SAS 関数、または定数を指定できます。"オペランドの指定" (438 ページ)を参照してください。

operator
比較、論理演算、または算術計算を要求する記号。すべての SAS 式演算子は、WHERE 式に対して有効です。これには、算術、比較、論理、最小値と最大値、連結、評価順序を制御するかっこ、および接頭辞の演算子が含まれます。さらに、特別な WHERE 式演算子を使用できます。これらの式演算子には、BETWEEN-AND、CONTAINS、IS NULL または IS MISSING、LIKE、類似、SAME-AND が含まれます。"演算子の指定" (440 ページ)を参照してください。

オペランドの指定

変数

変数は、SAS データセット内の列です。各 SAS 変数には、名前や型(文字または数値)などの属性があります。変数型によって、検索する値を指定する方法が決まります。次に例を示します。

```
where score > 50;
where date >= '01jan2001'd and time >= '9:00't;
where state = 'Texas';
```

WHERE 式では、DATA ステップによって作成された自動変数(FIRST.variable、LAST.variable、_N_、または割り当てステートメントで作成された変数など)を使用することはできません。

他の SAS 式と同様に、数値変数の名前は単独で使用できます。SAS は、0 または欠損している数値を false として扱います。他の値は true となります。次の例では、WHERE 式は、EMPNUM が欠損しておらずゼロではなく、ID が欠損しておらずゼロではないすべての行を返します。

```
where empnum and id;
```

文字変数の名前は単独で使用することもできます。SAS は、文字変数の値が空白でないオブザベーションを選択します。たとえば、次の WHERE 式は、空白に等しくないすべての値を返します。

```
where lastname;
```

SAS 関数

SAS 関数は、計算またはシステム操作から値を返します。ほとんどの関数は指定した引数を使用しますが、一部の関数は動作環境から引数を取得します。WHERE 式で SAS 関数を使用するには、その名前と引数をかっこで囲んで入力します。指定できる関数には次のようなものがあります。

- SUBSTR は部分文字列を抽出します。
- TODAY は現在の日付を返します。
- PUT は、指定された出力形式を使用して指定された値を返します。

次の DATA ステップでは、Name の値が **Mac** で始まり、変数 City の値が **Charleston** または **Atlanta** であるデータセット Customer からのオブザベーションのみを含む SAS データセットを生成します。

```
data testmacs;
  set customer;
  where substr (name,1,3) = 'Mac' and
  (city='Charleston' or city='Atlanta');
```

```
run;
```

OF 構文は一部の SAS 関数で許可されていますが、WHERE 句で指定されている関数を使用する場合には使用できません。次の DATA ステップの例では、OF を RANGE とともに使用できます。

```
data abc;
x1=2;
x2=3;
x3=4;
r=range(of x1-x3);
run;
```

RANGE および OF とともに WHERE 句を使用すると、エラーが SAS ログに書き込まれます。

例のコード 19.1　WHERE 句が OF とともに使用された場合の出力

```
proc print data=abc;
where range(of x1-x3)=6;
                --
                22
                76
ERROR: Syntax error while parsing WHERE clause.
ERROR 22-322: Syntax error, expecting one of the following: !, !!, &, (, *,
**,
+, ',', -, /, <, <=, <>, =, >, >=, ?,
          AND, BETWEEN, CONTAINS, EQ, GE, GT, LE, LIKE, LT, NE, OR, ^=, |,
||, ~=.
ERROR 76-322: Syntax error, statement will be ignored.
run;
```

次は、OF 構文を使用できる SAS 関数の表です。

表 19.1　OF 構文を使用する SAS 関数

CAT	HARMEANZ	RMS
CATS	KURTOSIS	SKEWNESS
CATT	MAX	STD
CATX	MEAN	STDERR
CSS	MIN	SUM
CV	N	USS
GEOMEAN	NMISS	VAR
GEOMEANZ	ORDINAL	
HARMEAN	RANGE	

注: WHERE 式で使用され、インデックスによって最適化できる SAS 関数は、SUBSTR 関数と TRIM 関数です。

SAS 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。

定数

定数とは、数値や引用符で囲まれた文字列などの固定値、つまり検索対象の値です。定数は、SAS データセットから取得された変数の値、または WHERE 式自体の中で作成された値です。定数はリテラルとも呼ばれます。たとえば、定数は便名や都市名などです。定数には、時間、日付、または日時の値を指定することもできます。

値は数値または文字のいずれかです。引用符を使用するかどうかについては、次の規則に注意してください。

- 値が数値の場合は、引用符を使用しません。

  ```
  where price > 200;
  ```

- 値が文字の場合は、引用符を使用します。

  ```
  where lastname eq 'Martin';
  ```

- 単一引用符または二重引用符を使用できますが、混在させないでください。引用符で囲まれた値は、大文字と小文字を含めて完全に一致する必要があります。
- 値に二重引用符が含まれる場合は単一引用符を使用し、値に単一引用符が含まれる場合は二重引用符を使用することが必要な場合があります。

  ```
  where item = '6" decorative pot';
  where name ? "D'Amico";
  ```

- SAS 日付定数は引用符で囲む必要があります。日付値を指定する場合、大文字と小文字は重要ではありません。単一引用符または二重引用符を使用できます。次の式は同等です。

  ```
  where birthday = '24sep1975'd;
  where birthday = '24sep1975"d;
  ```

演算子の指定

算術演算子

算術演算子を使用すると、数学的な演算を実行できます。算術演算子には次のものが含まれます。

表 19.2　算術演算子

記号	定義	例
*	乗算	where bonus = salary * .10;
/	除算	where f = g/h;

記号	定義	例
+	加算	where c = a+b;
-	減算	where f = g-h;
**	指数	where y = a**2;

比較演算子

比較演算子(二項演算子とも呼ばれる)は、変数を値または別の変数と比較します。比較演算子は関係を提案し、その関係が成立するかどうかを SAS に判断させます。たとえば、次の WHERE 式は、数値変数 ZipCode の値 78753 を持つオブザベーションのみにアクセスします。

```
where zipcode eq 78753;
```

次の表に、比較演算子のリストを示します。

表 19.3 比較演算子

記号	同等のニーモニック	定義	例
=	EQ	等しい	where empnum eq 3374;
^=または~=または¬=または <>	NE	等しくない	where status ne full-time;
>	GT	より大きい	where hiredate gt '01jun1982'd;
<	LT	未満	where empnum < 2000;
>=	GE	以上	where empnum >= 3374;
<=	LE	以下	where empnum <= 3374;
	IN	値のリストの 1 つに等しい	where state in ('NC','TX');

文字比較を行う場合、コロン(:)修飾子を使用して、文字列の指定した接頭辞のみを比較できます。たとえば、次の WHERE 式では、等号の後に使用されるコロン修飾子は、変数 LastName の値の最初の文字のみを調べ、文字 **S** で始まる名前を持つオブザベーションを選択するように SAS に指示します。

```
where lastname=: 'S';
```

SQL プロシジャでは、演算子と組み合わせて使用されるコロン修飾子はサポートされていないことに注意してください。かわりに LIKE 演算子を使用できます。

IN 演算子

IN 演算子は比較演算子であり、値のリストの 1 つに等しい文字値と数値を検索します。値のリストはかっこで囲み、各文字値を引用符で囲み、カンマまたは空白で区切る必要があります。

たとえば、ノースカロライナまたはテキサスにあるすべてのサイトが 必要だとします。次のように指定できます。

```
where state = 'NC' or state = 'TX';
```

ただし、リスト内の任意の州を選択する IN 演算子を使用する方が簡単です。

```
where state in ('NC','TX');
```

さらに、NOT 論理演算子を使用してリストを除外できます。

```
where state not in ('CA', 'TN', 'MA');
```

短縮表記を使用して、検索する連続した整数の範囲を指定できます。範囲は、検索するリストの値として構文 M:N を使用して指定します。M は下限、N は上限です。M と N は整数である必要があり、M、N、および M と N の間のすべての 整数が範囲に含まれます。たとえば、次のステートメントは同等です。

- y = x in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
- y = x in (1:10);

完全に制限された範囲条件

完全に制限された範囲条件は、上限と下限の両方を指定する 2 つの比較演算子の間の変数で構成されます。たとえば、次の式は、500-1000 (両端を含む)の範囲内の従業員番号を返します。

```
where 500 <= empnum <= 1000;
```

前述の範囲条件式は次と同等であることに注意してください。

```
where empnum >= 500 and empnum <= 1000;
```

NOT 論理演算子と完全に制限された範囲条件を組み合わせて、範囲外のオブザベーションを選択できます。かっこが必要であることに注意してください。

```
where not (500 <= empnum <= 1000);
```

BETWEEN-AND 演算子

BETWEEN-AND 演算子は、変数の値が値の両端を含む範囲内に収まるオブザベーションを選択する、完全に制限された範囲条件とも見なされます。

範囲の制限を定数または式として指定できます。指定する範囲はすべて 両端を含む範囲であり、範囲の制限のいずれかに等しい値が範囲内に収まります。BETWEEN-AND を使用するための一般的な構文は次のとおりです。

WHERE variable BETWEEN value AND value;

次に例を示します。

```
where empnum between 500 and 1000;
where taxes between salary*0.30 and salary*0.50;
```

NOT 論理演算子と BETWEEN-AND 演算子を組み合わせて、範囲外のオブザベーションを選択できます。

```
where empnum not between 500 and 1000;
```

注: BETWEEN-AND 演算子と完全に制限された範囲条件は、同じ結果を生成します。つまり、次の WHERE 式は同等です。

```
where 500 <= empnum <= 1000;
where empnum between 500 and 1000;
```

CONTAINS 演算子

CONTAINS (?)演算子の最も一般的な使用法は、文字変数の値内で指定された文字セットを検索してオブザベーションを選択することです。変数値内の文字列の位置は関係ありません。ただし、演算子は比較するときに大文字と小文字が区別されます。

次の例では、変数 Company の値 **Mobay** および **Brisbayne** を持つオブザベーションを選択しますが、**Bayview** を含むオブザベーションは選択しません。

```
where company contains 'bay';
where company ? 'bay';
```

NOT 論理演算子と CONTAINS 演算子を組み合わせて、指定した文字列に含まれないオブザベーションを選択できます。

```
where company not contains 'bay';
```

2 つの変数で CONTAINS 演算子を使用することもできます。つまり、ある変数が別の変数に含まれているかどうかを判断できます。2 つの変数を指定する場合は、末尾にスペースが含まれる可能性があることに留意してください。これは TRIM 関数を使用して解決できます。

```
proc sql;
  select *
  from table1 as a, table2 as b
```

```
  where a.fullname contains trim(b.lastname) and
    a.fullname contains trim(b.firstname);
```

さらに、TRIM 関数はマクロ変数を検索するときに役立ちます。

```
proc print;
  where fullname contains trim("&lname");
run;
```

IS NULL または IS MISSING 演算子

IS NULL または IS MISSING 演算子は、変数の値が欠損しているオブザベーションを選択します。演算子は、通常または特殊な欠損値文字の両方を含むオブザベーションを選択し、文字変数と数値変数の両方に使用できます。

```
where idnum is missing;
where name is null;
```

次は文字データと同等です。

```
where name is null;
where name = ' ';
```

また、次は数値データと同等です。このステートメントは、欠損値を特殊な欠損値文字で区別します。

```
where idnum <= .Z;
```

次のように、NOT 論理演算子と IS NULL または IS MISSING を組み合わせて、非欠損値を選択できます。

```
where salary is not missing;
```

LIKE 演算子

LIKE 演算子は、文字変数の値を指定されたパターンと比較することによってオブザベーションを選択します。これはパターンマッチングと呼ばれます。LIKE 演算子では大文字と小文字が区別されます。パターンの指定に使用できる特殊文字が 2 つあります。

パーセント記号(%)
: 任意の数の文字がその位置を占めることができることを指定します。次の WHERE 式は、名前が文字 **N** で始まるすべての従業員を選択します。名前は任意の長さにすることができます。

  ```
  where lastname like 'N%';
  ```

アンダースコア(_)
: アンダースコア 1 文字につき、値内の 1 文字のみと一致します。パターン内で複数の連続したアンダースコア文字を指定したり、同じパターン内でパーセント記号とアンダースコアを指定したりできます。たとえば、さまざまな形式の LIKE 演算子を使用して、次の名のリストから文字値を選択できます。

  - **Diana**

- **Diane**
- **Dianna**
- **Dianthus**
- **Dyan**

次の表は、LIKE 演算子のさまざまな形式を使用してどの名前が選択されるかを示しています。

パターン	選択された名前
like 'D_an'	Dyan
like 'D_an_'	Diana、Diane
like 'D_an__'	Dianna
like 'D_an%'	リスト内のすべての名前

SAS 文字式を使用してパターンを指定することはできますが、SAS 関数を使用する SAS 文字式は使用できません。

NOT 論理演算子と LIKE を組み合わせて、次のように、指定されたパターンを持たない値を選択できます。

```
where frstname not like 'D_an%';
```

%および_文字は LIKE 演算子にとって特別な意味を持つため、値内の%および_文字を検索する場合はエスケープ文字を使用する必要があります。エスケープ文字は、一連の文字の中で、その後に続く文字が別の意味を持つことを示す 1 つの文字です。LIKE 演算子の場合、エスケープ文字は、特殊文字関数を実行するかわりに、変数値内の%および_文字のリテラルインスタンスを検索することを意味します。

たとえば、変数 X に値 **abc**、**a_b**、および **axb** が含まれている場合、エスケープ文字を含む次の LIKE 演算子は、値 **a_b** のみを選択します。エスケープ文字(/)は、パターンが文字 **a** と **b** で囲まれたリテラル'_'を検索することを指定します。エスケープ文字(/)は検索の一部ではありません。

```
where x like 'a/_b' escape '/';
```

エスケープ文字を使用しない場合、次の LIKE 演算子は値 **a_b** および **axb** を選択します。検索パターン内の特殊文字アンダースコアは、アンダースコアを 含む値を含む任意の単一の **b** 文字と一致します。

```
where x like 'a_b';
```

エスケープ文字を指定するには、パターンマッチング式にその文字を含めてから、キーワード ESCAPE の後にエスケープ文字式を続けます。エスケープ文字を含める場合は、パターンマッチング式を引用符で囲む必要があり、列名を含めることはできません。エスケープ文字式は単一文字として評価されます。オペランドは文字または文字列リテラルである必要があります。単一文字の場合は引用符で囲みます。

LIKE '*pattern-matching-expression*' ESCAPE '*escape-character-expression*'

類似演算子

類似(=*)演算子は、指定された単語のスペルバリエーションを含むオブザベーションを選択します。演算子は、Soundex アルゴリズムを使用して変数値とオペランドを比較します。詳細については、*SAS 関数と CALL ルーチン: リファレンス*の SOUNDEX 関数を参照してください。

注: Soundex アルゴリズムは英語に偏っており、英語以外の言語ではあまり役に立たないことに注意してください。

類似演算子は便利ですが、常にすべての可能な値を選択するとは限りません。たとえば、次の Smith に似た名前のリストからオブザベーションを選択するとします。

- **Schmitt**
- **Smith**
- **Smithson**
- **Smitt**
- **Smythe**

次の WHERE 式は、このリストから **Smithson** を除くすべての名前を選択します。

```
where lastname=* 'Smith';
```

NOT 論理演算子と類似演算子を組み合わせて、次のように、指定した単語のスペルバリエーションを含まない値を選択できます。

```
where lastname not =* 'Smith';
```

注: 類似演算子はインデックスを使用して最適化できません。

SAS Viya プラットフォームリリース 2020.1.4 以降、SOUNDSLIKELEN=システムオプションを使用して、類似演算子の実行時に SOUNDEX 関数の結果の比較に使用されるバイト数を指定できるようになりました。詳細については、"SOUNDSLIKELEN システムオプション" (*SAS システムオプション: リファレンス*)を参照してください。

例: 類似演算子によって使用されるバイト数の制御

次の例は、SOUNDSLIKELEN=システムオプションが、類似演算子と比較されるバイトにどのような影響を与えるかを示しています。Test データセットには、7 つの名前と、各名前に対して計算された SOUNDEX 値が含まれています。

```
data test;
  length name $ 12;
  input name $;
  soundexp=soundex(name) ;
datalines;
Schmitt
```

```
Smith
Smithson
Smitt
Smythe
Smoak
Smart
run;

title 'Data to Compare';
proc print data=test;
run;
```

Test データセットのデータ値は次のとおりです。

Data to Compare

Obs	name	soundexp
1	Schmitt	S53
2	Smith	S53
3	Smithson	S5325
4	Smitt	S53
5	Smythe	S53
6	Smoak	S52
7	Smart	S563

SOUNDSLIKELEN=3 の場合、類似演算子は、SOUNDEX 結果の最初の 3 バイトに一致する文字列を返します。

```
title 'SOUNDSLIKELEN=3';
options soundslikelen=3;
proc print data=test;
var name;
where name =* "Smith";
run;
```

SOUNDSLIKELEN=3

Obs	name
1	Schmitt
2	Smith
3	Smithson
4	Smitt
5	Smythe

SOUNDSLIKELEN=MIN の場合、類似演算子は、最初の最小バイトに完全に一致する文字列のみを返します。

SOUNDSLIKELEN=MIN

Obs	name
1	Schmitt
2	Smith
4	Smitt
5	Smythe

SOUNDSLIKELEN=MAX の場合、最大バイトに完全に一致する文字列のみが返されます。

SOUNDSLIKELEN=MAX

Obs	name
1	Schmitt
2	Smith
4	Smitt
5	Smythe

SAME-AND 演算子

SAME-AND 演算子を使用すると、元の条件を再入力することなく、プログラムの後半で既存の WHERE 式に条件を追加できます。この機能は次の場合に役立ちます。

- 対話型 SAS プロシジャ
- コマンドラインに WHERE 式を入力できる全画面 SAS プロシジャ
- あらゆる種類の RUN グループ処理

すでに WHERE 式が定義されており、追加の条件を挿入する場合は、SAME-AND 演算子を使用します。SAME-AND 演算子の形式は次のとおりです。

- where-expression-1;
- ...*SAS statements*...
- WHERE SAME AND where-expression-2;
- ...*SAS statements*...
- WHERE SAME AND where-expression-n;

SAS は、以前に定義された条件に加えて、SAME-AND 演算子の後の条件を満たすオブザベーションを選択します。SAS は、既存のすべての条件を、単一の WHERE 式内の AND 演算子で区切られた条件であるかのように処理します。

次の例は、GPLOT プロシジャの RUN グループ内で SAME-AND 演算子を使用する方法を示しています。SAS データセット YEARS には 3 つの変数があり、2009-2011 年の四半期データが含まれています。

```
proc gplot data=years;
  plot unit*quar=year;
run;

  where year > '01jan2009'd;
run;

  where same and year < '01jan2012'd;
run;
```

次の WHERE 式は、前述のコードと同等です。

```
where year > ''01jan2009'd and year < '01jan2012'd;
```

MIN 演算子および MAX 演算子

MIN 演算子または MAX 演算子を使用して、2 つの量の最小値または最大値を見つけます。最小値または最大値を知りたい 2 つの量で演算子を囲みます。

- MIN 演算子は、2 つの値のうち小さい方を返します。
- MAX 演算子は、2 つの値のうち大きい方を返します。

たとえば、A が B より小さい場合、次は A の値を返します。

```
where x = (a min b);
```

注: 記号表現><はサポートされておらず、<>は"等しくない"と解釈されます。

連結演算子

連結演算子は文字値を連結します。連結演算子は次のように指定します。

- || (2 つの OR 記号)
- !! (2 つの感嘆符)
- ¦¦ (2 つの縦破線)

次に例を示します。

```
where name = 'John'||'Smith';
```

接頭辞演算子

プラス記号(+)とマイナス記号(–)は、接頭辞演算子または算術演算子のいずれかになります。これらは、式の先頭または開きかっこの直前に現れる場合、接頭辞演算子となります。接頭辞演算子は、変数、定数、SAS 関数、かっこ内の式に適用されます。

```
where z = –(x + y);
```

注: NOT 演算子も接頭辞演算子とみなされます。

論理演算子を使用した式の結合

構文

論理演算子(ブール演算子とも呼ばれる) AND、OR、および NOT を使用して、WHERE 式を結合または変更できます。複合 WHERE 式の基本構文は次のとおりです。

WHERE*where-expression-1* AND | OR | NOT *where-expression-n*

AND は、両方の条件を満たすオブザベーションを見つけることによって 2 つの条件を結合します。次に例を示します。

```
where skill eq 'java' and years eq 4;
```

OR は、条件のいずれかまたは両方を満たすオブザベーションを見つけることによって 2 つの条件を結合します。次に例を示します。

```
where skill eq 'java' or years eq 4;
```

NOT は、指定された基準の補集合を見つけることによって条件を変更します。NOT 論理演算子は、SAS および WHERE 式演算子と組み合わせて使用できます。また、NOT 演算子を AND および OR と組み合わせることができます。次に例を示します。

```
where skill not eq 'java' or years not eq 4;
```

論理演算子とそれに相当する記号を次の表に示します。

表 19.4　論理(ブール)演算子

記号	同等のニーモニック
&	AND
!または|または¦	OR
^または~または¬	NOT

複合式の処理

SAS が複合 WHERE 式(複数の条件)に遭遇すると、ソフトウェアはルールに従って各式を評価する順序を決定します。WHERE 式を結合すると、SAS は特定の順序で条件を処理します。

1 NOT 式が最初に処理されます。

2 次に、AND で結合された式が処理されます。

3 最後に、OR で結合された式が処理されます。

かっこを使用した評価順序の制御

SAS は論理演算子を特定の順序で評価しますが、式をかっこ内にネストすることで評価の順序を制御できます。つまり、かっこで囲まれた式は、かっこで囲まれていない式よりも前に処理されます。最も内側のかっこのセット内の式が最初に処理され、次に最も深いものが続き、すべてのかっこが処理されるまで外側に移動します。

たとえば、SAS/GRAPH と SAS/STAT の両方のソフトウェアを備えているカナダのすべてのサイトのリストが必要なため、次の式を発行するとします。

```
where product='GRAPH' or product='STAT' and country='Canada';
```

ただし、結果には、SAS/STAT ソフトウェアをライセンスするカナダのサイトに加えて、SAS/GRAPH ソフトウェアをライセンスするすべてのサイトが含まれます。正しい結果を取得するには、かっこを使用します。これにより、SAS は最初にかっこ内の比較を評価し、いずれかのプロダクトライセンスを持つサイトのリストを提供し、その結果が残りの条件に使用されます。

```
where (product='GRAPH' or product='STAT') and country='Canada';
```

WHERE 処理のパフォーマンスの向上

SAS データセットにインデックスを作成すると、WHERE 処理のパフォーマンスが大幅に向上します。インデックスは、特定のオブザベーションへの直接アクセスを提供するために SAS データファイルに対して作成できるオプションのファイルです。

インデックスを使用せずに WHERE 式を処理するには、SAS が選択基準に一致するオブザベーションを見つけるためにオブザベーションを順番に読み取る必要があります。インデックスがない場合、SAS はまず並べ替えインジケーターをチェックします。並べ替えインジケーターは、前の SORT プロシジャまたは SORTEDBY=データセットオプションからのデータファイルとともに保存されています。並べ替えインジケーターが検証されると、SAS はそれを利用し、WHERE 式を満たす値がもう存在しないことが明らかになるとファイルの読み取りを停止します。たとえば、インデックスを使用せずに年齢で並べ替えたデータセットを考えてみましょう。**where age le 25** という式を処理するために、SAS は 25 より大きいオブザベーションを見つけた後にオブザベーションの読み取りを停止します。SAS はオブザベーションの読み取りをいつ停止するかを決定できますが、インデックスがなければどこから開始すればよいのかが示されないため、SAS は常に最初のオブザベーションから開始し、大量のオブザベーションの読み取りが必要になる可能性があることに注意してください。

インデックスを使用すると、SAS はどのオブザベーションが基準を満たすかを判断できるようになります。これは、WHERE 式の最適化と呼ばれます。ただし、デフォルトでは、SAS はインデックスを使用するか、データセット全体を順番に読み取るかを決定します。SAS がインデックスを使用して WHERE 式を処理する方法の詳細については、“SAS のインデックス” (609 ページ)を参照してください。

データセットのインデックスの作成に加えて、効率的な WHERE 式を作成するためのガイドラインをいくつか示します。

表 19.5　効率的な WHERE 式の構築

ガイドライン	効率的	非効率的
%または_で始まる LIKE 演算子の使用は避けてください。	**where country like 'A %INA';**	**where country like '%INA';**
算術式の使用は避けてください。	**where salary > 48000;**	**where salary > 12*4000;**

条件付きで選択されたデータのセグメントの処理

FIRSTOBS=および OBS=オプションの適用

WHERE 式を使用してオブザベーションのサブセットを条件付きで選択する場合、FIRSTOBS=、OBS=、または両方の処理(データセットオプションおよびシステムオプションとして)を適用して、そのサブセットをセグメント化することもできます。WHERE 式と一緒に使用する場合、

- FIRSTOBS=は、処理を開始するために WHERE 式で選択されたデータのサブセット内のオブザベーション番号を指定します。
- OBS=は、WHERE 式で選択されたデータのサブセットからのオブザベーションの処理をいつ停止するかを指定します。

WHERE 式と一緒に使用する場合、OBS=および FIRSTOBS=に指定される値は、データセット内の物理的なオブザベーション番号ではなく、サブセット内の論理的な番号になります。たとえば、**obs=3** は、データセット内の 3 番目のオブザベーション番号を意味するものではありません。かわりに、WHERE 式によって選択されたデータのサブセット内の 3 番目のオブザベーションを意味します。

データのサブセットへの OBS=および FIRSTOBS=処理の適用は、WHERE ステートメント、WHERE=データセットオプション、および SQL プロシジャの WHERE 句でサポートされています。

別のビューのビューである SAS ビュー(ネストされたビュー)を処理している場合、OBS=および FIRSTOBS=をデータのサブセットに適用すると、予期しない結果が生じる可能性があります。ネストされたビューの場合、OBS=および FIRSTOBS=の処理が各 SAS ビューに適用され、ルート(最下位レベル)ビューから開始され、各 SAS ビューのオブザベーションがフィルタリングされます。その結果、サブセットおよびセグメントの基準を満たすオブザベーションが存在しない可能性があります。“SAS ビューの処理” (453 ページ)を参照してください。

FIRSTOBS=および OBS=をデータのサブセットに適用する

次の SAS プログラムは、データをサブセット化する条件を指定する方法と、処理するデータのサブセットのセグメントを指定する方法を示しています。

```
data A; 1
  do I=1 to 100;
  X=I + 1;
  output;
  end;
run;

proc print data=work.a (firstobs=2 3 obs=4; 4
  where I > 90; 2
run;
```

1 DATA ステップでは、100 個のオブザベーションと 2 つの変数 I と X を含む Work.A という名前のデータセットを作成します。

2 WHERE 式 **I > 90** は、指定された条件を満たすオブザベーションのみを処理するように SAS に指示します。その結果、オブザベーション 91-100 のサブセットが生成されます。

3 FIRSTOBS=データセットオプションは、データのサブセット内の 2 番目のオブザベーション(オブザベーション 92)から処理を開始するように SAS に指示します。

4 OBS=データセットオプションは、データのサブセット内の 4 番目のオブザベーション(オブザベーション 94)に達したときに処理を停止するように SAS に指示します。

PROC PRINT の結果はオブザベーション 92、93、および 94 です。

SAS ビューの処理

次の SAS プログラムは、データセット、そのデータセットの SAS ビューを作成し、次に最初の SAS ビューのデータをサブセット化する 2 番目の SAS ビューを作成します。WHERE ステートメントと OBS=システムオプションの両方が使用されます。

```
data a; 1
  do I=1 to 100;
  X=I + 1;
  output;
  end;
run;

data viewa/view=viewa; 2

  set a;
    Z = X+1;
run;

data viewb/view=viewb; 3

  set viewa;
    where I > 90;
run;

options obs=3; 4

proc print data=work.viewb; 5

run;
```

1 最初の DATA ステップでは、100 個のオブザベーションと 2 つの変数 I と X を含む Work.A という名前のデータセットを作成します。

2 2 番目の DATA ステップでは、100 個のオブザベーションと 3 つの変数 I、X (データセット Work.A から)、および Z (この DATA ステップで割り当て)を含む Work.ViewA という名前の SAS ビューを作成します。

3 3 番目の DATA ステップでは、Work.ViewB という名前の SAS ビューを作成し、WHERE ステートメントでデータのサブセットを作成します。これにより、ビューは 10 個のオブザベーションにアクセスすることになります。

4 OBS=システムオプションは、前の SET ViewA ステートメントに適用され、処理中のデータのサブセット内の 3 番目のオブザベーションに達したときに処理を停止するように SAS に指示します。

5 SAS が PRINT プロシジャを処理すると、次のことが起こります。

   1 まず、SAS は Work.ViewA に *obs=3* を適用し、3 番目のオブザベーションで処理を停止します。

   2 次に、SAS は条件 *I > 90* を処理中の 3 つのオブザベーションに適用します。どのオブザベーションも基準を満たしていません。

   3 PROC PRINT ではオブザベーションがありません。

予期しない結果が生じる可能性を防ぐために、Work.ViewA の作成時に **obs=max** を指定して、SAS がルート(最下位レベル)ビュー内のすべてのオブザベーションを強制的に読み取ることができます。

```
data viewa/view=viewa;
  set a (obs=max);
  Z = X+1;
run;
```

PRINT プロシジャは、オブザベーション 91、92、および 93 を処理します。

WHERE 式とサブセット化 IF ステートメントのどちらを使用するかを決定する

SAS データセットから条件付きでオブザベーションを選択するには、WHERE 式またはサブセット化 IF ステートメントを使用できます。どちらも、SAS がオブザベーションを処理する必要があるかどうかを決定する条件をテストします。ただし、次のような違いがあります。

- サブセット化 IF ステートメントは、DATA ステップでのみ使用できます。サブセット化 IF ステートメントは、オブザベーションがプログラムデータベクトル(PDV)に読み込まれた後の条件をテストします。条件が真の場合、SAS は現在のオブザベーションの処理を続行します。それ以外の場合、オブザベーションは破棄され、処理は次のオブザベーションから続行されます。
- WHERE 式は、DATA ステップと SAS プロシジャの両方で使用できるほか、ウィンドウ環境、SCL プログラム、およびデータセットオプションとしても使用できます。WHERE 式は、オブザベーションが PDV に読み込まれる前に条件をテストします。条件が真の場合、オブザベーションは PDV に読み込まれて処理されます。条件が偽の場合、オブザベーションは PDV に読み込まれず、処理は次のオブザベーションから続行されます。これにより、オブザベーションに多くの変数または非常に長い文字変数(最大 32K バイト)が含まれる場合、大幅な節約が可能になります。さらに、WHERE 式はインデックスを使用して最適化でき、WHERE 式では LIKE や CONTAINS などのより多くの演算子が有効になります。

  注: 一般に、WHERE 式を使用し、処理前に PDV への移動を回避する方が効率的ですが、データセットに変数が非常に少ないオブザベーションが含まれている場合は、PDV への移動が低コストになる可能性があります。ただし、32K バイトの文字データを含む 1 つの変数は、たとえ 1 つの変数とはいえ低コストではありません。

ほとんどの場合、どちらの方法も使用できます。ただし、次の表は、特定の方法を使用する必要があるタスクのリストを示しています。

表 19.6 WHERE 式またはサブセット化 IF ステートメントのいずれかを必要とするタスク

タスク	方法
先行する DATA ステップを使用せずにプロシジャ内で選択を行います。	WHERE 式
インデックス付きデータセットで得られる効率性を活用します。	WHERE 式

タスク	方法
BETWEEN-AND、CONTAINS、IS MISSING または IS NULL、LIKE、SAME-AND、類似などの特殊演算子のグループの 1 つを使用します。	WHERE 式
SAS データセットにすでに存在する変数値以外の値に基づいて選択します。たとえば、生データから読み取られた値、または DATA ステップの過程で計算または割り当てられた値を選択できます。	サブセット化 IF
最初ではなく、DATA ステップ中のどこかの時点で選択を行います。	サブセット化 IF
条件付きで選択を実行します。	サブセット化 IF

20

ループと条件

ループと条件の定義

次の用語は、SAS の条件付きプログラミングと反復プログラミングに関連しています。

Conditional processing
プログラマが指定した条件が真か偽かに応じて異なる計算やアクションを実行する SAS のステートメントまたは式。これらのステートメントのリストについては、“SAS の条件付き処理ステートメントの概要” (459 ページ)を参照してください。

Control flow
プログラム実行の順序とフローを制御する SAS のプログラミング構造。たとえば、条件付き処理ステートメントや DO ループは制御フローステートメントの一種です。SAS の追加の制御フローステートメントのリストについては、“その他の制御フローステートメント” (484 ページ)を参照してください。

詳細については、“特定のオブザベーションのフローの変更” (874 ページ)を参照してください。

DO group
DO ステートメントで始まり、END ステートメントで終わる SAS DATA ステップステートメントのグループ。DO グループステートメントは単位として実行されます。DO グループは、反復 DO ステートメント(DO ループとも呼ばれる)で構成することも、単純な DO ステートメントで構成することもできます。単純な DO ステートメントは、条件付き IF-THEN/ELSE ステートメントとともによく使用されます。

DO loop
特定の条件に達するまで継続的に繰り返される命令。反復 DO ステートメントを使用すると、DO ループを作成できます。詳細については、“DO ループの種類” (462 ページ)を参照してください。

DOLIST syntax
リストの要素を反復する反復 DO ループに基づいた構文の一種。リストは DO ステートメントの start-value として機能し、リスト内のアイテムはカンマで区切られます。DOLIST 構文は、反復 DO WHILE および DO UNTIL ステートメント

でも指定できます。詳細については、“DO ループの種類” (462 ページ)を参照してください。

WHERE expression
WHERE 式は、入力 SAS データセットまたはその他の入力テーブルで処理される行を選択するための条件を定義する SAS 式の一種です。次の WHERE ステートメントは、行を選択するための単一の条件を定義します。

```
where sales > 600000;
```

*複合 WHERE 式*は、論理演算子によって結合された複数の条件で構成されます。

```
where sales > 600000 and salary < 100000;
```

WHERE 式処理により、要求の効率が向上します。たとえば、SAS インデックスを含む WHERE 式を使用する場合、要求を実行するために SAS がデータセット内のすべての行を読み取る必要はありません。

SAS の条件付き処理ステートメントの概要

表 20.1 SAS によるデータの条件付き選択

言語要素	説明	例

CASE WHEN 式(PROC SQL)	指定された条件を満たす結果値を選択します。	proc sql;
		  select Title, length,
		  case
		  when length<120 then 'Short'
		  when length>=120 then 'Long'
		  else 'No Length'
		  end as m_length from movies;
		quit;

DO ループ	DO ステートメントと END ステートメントの間のステートメントを繰り返し実行します。	data simple_do;
		  years=5;
		  if years>0 then
		  do;
		  months=years*12;
		  output;
		  end;

IF ステートメント (DATA ステップ)	条件が真であるかどうかに基づいて条件付きでステートメントを実行します。	data mylib.if_ex;
		  set sashelp.class;
		  if age>14;
		  output;
		run;

言語要素	説明	例

SELECT WHEN ステートメント(DATA ステップ)	条件が真の場合に条件付きでステートメントを実行します。	data class;
		  set sashelp.class;
		  select (Name);
		  when ('Philip');
		  otherwise delete;
		  end;
		run;

WHERE ステートメント(DATA ステップ)	条件付きで行を選択し、SAS が指定された条件を満たす行のみを処理できるようにします。	data class;
		  set sashelp.class;
		  where Age>14;
		run;

WHERE ステートメント(PROC ステップ)	条件付きで行を選択し、SAS プロシジャが指定された条件を満たす行のみを処理できるようにします。
	詳細情報
		proc print data=sashelp.class;
		  where age > 14;
		run;
		
		proc means data=mylib.prdsale;
		  where region = 'EAST';
		run;

WHERE=データセットオプション(DATA ステップ)	入力データセットから読み込む行、または出力データセットに書き出す行を条件付きで選択します。	data class(where=(age>14));
		  set sashelp.class;
		run;
		
		data class;
		  set sashelp.class(where=(age>14));
		run;

WHERE=データセットオプション(PROC ステップ)	条件付きで行を選択し、SAS が指定された条件を満たす行のみを出力できるようにします。
	詳細情報
		proc freq
		  data=sashelp.cars(where=(weight>6000));
		  tables make * model;
		run; quit;

WHERE=句(ODS DOCUMENT プロシジャステートメント)	ODS ドキュメント内の要素の値に基づいて、ODS ドキュメントプロシジャステートメントを条件付きで実行します。詳細情報	ods html file="prog.html";
		proc document name=mylib.start;
		  replay univariate(where=(_name_ = 'Moments'));
		run; quit;
		ods html close;

CASE 式(PROC SQL ステップ)	指定された条件を満たす行を条件付きで選択します。	proc sql;
		  select Name, case
		  when Continent = 'North America' then 'Group A'
		  else 'Need to assign'
		  end as Region
		  from states;

言語要素	説明	例
		run; quit;
WHERE 句(PROC SQL ステップ)	指定された条件を満たす行を条件付きで選択します。	proc sql; select state from crime where murder > 7; run; quit;
定義とともに保存される DATA ステップビュー、PROC SQL ビュー、および SAS/ACCESS	ビューから、指定された条件を満たす行を条件付きで選択します。 詳細情報	proc sql; create view stat as select * from crime where murder > 7; run; quit;

表 20.2 Base SAS プロシジャによるデータの条件付き選択

プロシジャ名	説明	詳細情報
APPEND プロシジャ	ある SAS データセットのオブザベーションを別の SAS データセットの末尾に追加します。	追加されるオブザベーションの条件付き選択
COMPARE プロシジャ		
DATASETS プロシジャ		
FCMP プロシジャ		
MEANS プロシジャ		
PRINT プロシジャ		
RANK プロシジャ		
REPORT プロシジャ		
SORT プロシジャ		
STANDARD プロシジャ		
SUMMARY プロシジャ		
TABULATE プロシジャ		
TRANSPOSE プロシジャ		

表 20.3 CAS アクションによるデータの条件付き選択

CAS 言語要素	説明	例
CASUTIL プロシジャ	変数の値に基づいて条件付きで行を選択します。	“例: PROC CASUTIL” (511 ページ)
テーブルアクションセット内の whereTable パラメーターは、次の CAS アクションに含まれます。 ■ deleteRows アクション ■ loadTable アクション ■ partition アクション ■ update アクション ■ runCode アクション	別のテーブルに保存されている値に基づくフィルターテーブル。	例: テーブル内のインデックス列

DO ループ

DO ループの種類

プログラミングでは、同じステートメントセットを繰り返し実行したり、同じステートメントセットを特定回数実行したりする必要がある場合があります。この種の処理ではループを使用する必要があります。SAS では、DO ステートメントを指定してループを作成します。SAS プログラミングで使用される基本的な DO ループは 4 つあります。

表 20.4 DO ループの種類

種類	説明	例

DO ステートメント	■ DO ステートメントと END ステートメントの間のステートメントを 1 つの単位として実行します。
	■ 多くの場合、IF-THEN/ELSE ステートメント はで使用され、DO グループ内のステートメントは、IF ステートメント条件の値が真か偽かに応じて実行されます。
		data example;
		x=1;
		if x>0 then
		do;
		x=x*10;
		put x=;
		end;
		run;
		例: 単純な反復 DO ループの使用

種類	説明	例

反復 DO ステートメント	■ DO ステートメントと END ステートメントの間のステートメントを、インデックス変数の値に基づいて繰り返し実行します。
	■ 反復 DO ステートメントの形式はさまざまです。
		data example;
		x=0;
		do i=1 to 3;
		x=x+1;
		put x=;
		end;
		run;
		ログ出力
		x=1 i=1
		x=2 i=2
		x=3 i=3
		例: 反復 DO TO 構文を使用して特定の回数を反復する

条件付き DO WHILE ステートメント	■ 条件が真である間、DO ループ内のステートメントを繰り返し実行します。
	■ ループの先頭で条件を評価し、条件が偽の場合に DO WHILE ループが実行されないようにします。
		data do_while;
		x=7;
		do while(x < 10);
		x=x+1;
		put x=;
		end;
		run;
		ログ出力
		x=8
		x=9
		x=10

条件付き DO UNTIL ステートメント	■ 条件が真になるまで、DO ループ内のステートメントを繰り返し実行します。
	■ ループの末尾で条件を評価し、条件が偽であっても DO UNTIL ループが常に少なくとも 1 回実行されるようにします。
		data do_until;
		x=7;
		do until(x > 10);
		x=x+1;
		put x=;
		end;
		run;
		ログ出力
		x=8
		x=9
		x=10
		x=11
		例: 反復 DO WHILE および DO UNTIL ステートメントを使用して、ステートメントのグループを繰り返し実行する

反復 DO ループの形式

前の表にリストされているように、SAS の DO ループの形式の 1 つに反復 DO ループがあります。反復 DO ループにはいくつかの形式があります。

表 20.5　反復 DO ループの形式

構文	説明	例

DO *index-variable* = *start-value*;	単純な **DO** ループ構文
	start-value は index-variable の初期値であり、数値または数値に評価される式です。
		data simple_do;
		x = 5;
		do i= x*2;
		end;
		run;
		例: 単純な反復 DO ループの使用

DO *index-variable* = *start-value* **TO** *stop-value*;	**DO** *start-value* **TO** *stop-value* 構文
	start-value と *stop-value* は数値、または数値に評価される式です。
		data do_to;
		x = 5;
		do i=(x+1) to (x*2);
		x=x+1;
		output;
		end;
		run;
		例: 反復 DO TO 構文を使用して特定の回数を反復する

DO *index-variable* = *start-value* **TO** *stop-value***BY** *increment-value*;	**BY**-*increment* 構文
	start-value と *stop-value* は数値、または数値に評価される式です。
		data do_to_by;
		x = 1;
		do i=0 to 10 by (x*2);
		x=x+1;
		output;
		end;
		run;
		例: 反復 DO TO BY 構文を使用して、特定の間隔で特定の回数を反復する

DO *index-variable* = *start-value-1*,< *start-value-2*>, <*start-value-n*>;	**DOLIST** 構文
	start-value は、カンマで区切られた値のリストです。リスト内の値は、個々の数値、または数値として評価される式にすることができます。
		data do_list;
		do i=5, 10, 20+1;
		output;
		end;
		run;
		例: DOLIST 構文を使用して値のリストを反復処理する

WHERE 式

WHERE ステートメント

SAS DATA ステップで WHERE ステートメントを使用して、特定の条件を満たす SAS データセットから行を選択できます。

```
data cars;
  set sashelp.cars;
  where weight>6000;
run;
```

完全な構文情報については、"WHERE ステートメント" (*SAS DATA ステップステートメント: リファレンス*)を参照してください。

注: デフォルトでは、WHERE 式は追加および変更された行を評価しません。WHERE 式で更新を評価するかどうかを指定するには、WHEREUP=データセットオプションを指定します。"WHEREUP= データセットオプション" (*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。

関連項目

"例: WHERE ステートメントを使用した条件付き行選択" (495 ページ)

WHERE=データセットオプション

WHERE=データセットオプションを使用して、SAS データセットから条件付きで行を選択することもできます。WHERE=データセットオプションは、指定されたデータセットからオブザベーションを選択します。WHERE ステートメントはすべての入力データセットに適用されます。これらの言語要素の比較は次のとおりです。

表 20.6 WHERE ステートメントと WHERE=データセットオプションの比較

言語要素	例	ログ出力

WHERE ステートメント	data snacks;
	  set sashelp.snacks;
	  where QtySold>100;
	run;
		NOTE: There were 5 observations read from the data set SASHELP.SNACKS.
		NOTE: The data set WORK.SNACKS has 5 observations and 6 variables.

言語要素	例	ログ出力
出力データセットの WHERE=データセットオプション	data snacks(where=(QtySold>100)); set sashelp.snacks; run;	NOTE: There were 35770 observations read from the data set SASHELP.SNACKS. NOTE: The data set WORK.SNACKS has 5 observations and 6 variables.
入力データセットの WHERE=データセットオプション	data snacks; set sashelp.snacks(where=(QtySold>100)); run;	NOTE: There were 5 observations read from the data set SASHELP.SNACKS. NOTE: The data set WORK.SNACKS has 5 observations and 6 variables.

WHERE 式の変数

変数型と WHERE 式

変数は、SAS データセットまたはテーブル内のデータの列を参照する名前です。各 SAS 変数には、名前やデータ型などの属性があります。変数のデータ型によって、WHERE 式での操作の実行方法が決まります。たとえば、次の DATA ステップの WHERE 式は、sales が **550** より大きい行を選択します。sales は数値変数であるため、比較演算子(より大きい)は WHERE 式内の数値を比較します。

```
data retail;
  set sashelp.retail;
  where sales > 550;
run;
```

この例では、日付が 1953 年 1 月 1 日より後の行のみを選択する SAS データセットを作成します。この値は SAS 日付定数の例であり、 比較演算子(より大きい(>))と組み合わせて日付値の比較を実行するためにも使用されます。

```
data air;
  set sashelp.air;
  where date>'01jan1953'd;
run;
```

自動 DATA ステップ変数と WHERE 式

WHERE 式では自動 DATA ステップ変数は使用できません。たとえば、WHERE 式の割り当てステートメントで作成された FIRST.変数、LAST.変数、または_N_変数は使用できません。WHERE 式の変数は、読み取りまたは作成するソースデータセット内に存在している必要があります。

スタンドアロン変数と WHERE 式

他の SAS 式と同様に、数値変数の名前は単独で使用できます。SAS は、数値 0 または欠損値を偽として扱い、その他のすべての値を真として扱います。

たとえば、次の DATA ステップでは、WHERE 式は、AgeCHDdiag の数値が欠損またはゼロではなく、DeathCause の文字値が空白または欠損ではない sashelp.heart データセットのすべての行を返します。

```
data heart;
  set sashelp.heart;
  where AgeCHDdiag and DeathCause;
run;
```

WHERE 式の関数

SAS 関数は、計算またはシステム操作から値を返します。WHERE 式で SAS 関数を使用できます。

次の DATA ステップでは、SUBSTR 関数を使用して、name が **Jan** で始まる行のみを含む出力テーブルを生成します。

```
data mylib.class; set sashelp.class; run;

data mylib.J_class;
  set mylib.class;
  where substr (name,1,3) = 'Jan';
run;
proc print data=mylib.J_class;
 run;
```

アウトプット 20.1　WHERE ステートメントで使用される SUBSTR 関数を示す PROC PRINT 出力

Obs	Name	Sex	Age	Height	Weight
1	Jane	F	12	59.8	84.5
2	Janet	F	15	62.5	112.5

次の SAS 関数は、WHERE 式でよく使用されます。

- SUBSTRN 関数 - 部分文字列を抽出します。
- TODAY 関数 - 現在の日付を返します。
- PUT 関数 - 指定された出力形式を使用して指定された値を返します。

> 重要　WHERE 式は OF 演算子を含む関数では使用できません。

一部の SAS 関数では OF 構文が許可されます。たとえば、次の DATA ステップでは、OF を RANGE 関数とともに使用して、リスト **d1–d3** の最大値と最小値の差を返します。これは、OF 演算子を含む関数が WHERE 式に指定されていない限り許可されます。

```
data test;
  n1=1;
  n2=2;
  n3=3;
  num = range(of n1-n3);
run;
```

WHERE 式で OF を使用して同じ関数を指定すると、SAS はログにエラーを返します。

```
ERROR: Syntax error while parsing WHERE clause.
```

WHERE 句で OF を使用するかわりに、OF リストからすべての変数を列挙することができます。

```
where num < range(n1,n2,n3);
```

注: WHERE 式で TRIM 関数と SUBSTR 関数を使用すると、インデックス付きデータのパフォーマンスを向上させることができます。詳細については、“インデックスと WHERE 式” (479 ページ)を参照してください。

関連項目

- “関数と変数リストを使用した OF 演算子” (93 ページ)
- *SAS 関数と CALL ルーチン: リファレンス*

WHERE 式の定数

定数は、数値や引用符で囲まれた文字列などの固定値です。WHERE 式で SAS 定数を指定して、SAS データセットまたはテーブル内の特定の値を見つけることができます。定数値は WHERE 式自体の中で作成されます。定数はリテラルとも呼ばれます。

定数は数値、文字列、または日時値になります。次の表に、さまざまな種類の定数を示します。

表 20.7 WHERE 式で指定できる SAS 定数

定数の種類	説明	例
数値定数	値を引用符で囲みません。	where price > 200;

定数の種類	説明	例
文字定数	値を単一引用符または二重引用符で囲みます。 引用符で囲まれた値が大文字と小文字を含めて完全に一致することを確認します。 値に二重引用符が含まれる場合は単一引用符を使用し、値に単一引用符が含まれる場合は二重引用符を使用します。	where lastname eq 'Martin'; where item = '6" decorative pot'; where name = "D'Amico";

日付定数、時間定数、日時定数	値を単一引用符または二重引用符で囲みます。
	終わり引用符の後に、日付定数の場合は文字 **d**、時間定数の場合は文字 **t**、日時定数の場合は文字 **dt** を指定します。
		where birthday = '24sep1975'd;
		where time>'9:25:19pm't
		where year='2018-05-17T09:15:30'dt;

WHERE 式の演算子

算術演算子

算術演算子を使用すると、数学的な演算を実行できます。算術演算子には次のものが含まれます。

表 20.8　算術演算子

演算子記号	説明	例
*	乗算	where bonus = salary * .10;
/	除算	where f = g / h;
+	加算	where c = a+b;
-	減算	where f = g-h;
**	指数	where y = a**2;

比較演算子

比較演算子(二項演算子とも呼ばれる)は、変数を値または別の変数と比較します。

比較演算子は関係を提案し、その関係が成立するかどうかを SAS に判断させます。

たとえば、次の WHERE 式は、数値変数 zipcode の値が 78753 である行のみにアクセスします。

```
where zipcode eq 78753;
```

次の表は、SAS で使用される比較演算子のリストです。

表 20.9　比較演算子

記号	エイリアス	定義	例
=	EQ	等しい	where empnum eq 3374;
^=または ~=または ¬=または <>	NE	等しくない	where status ne full-time;
>	GT	より大きい	where hiredate gt '01jun1982'd;
<	LT	未満	where empnum < 2000;
>=	GE	以上	where empnum >= 3374;
<=	LE	以下	where empnum <= 3374;
	IN	値のリストの 1 つに等しい	where state in ('NC','TX');

文字比較を行う場合、コロン(:)修飾子を使用して、文字列の指定した接頭辞のみを比較できます。たとえば、次の WHERE 式では、等号の後に使用されるコロン修飾子は、変数 LastName の値の最初の文字のみを調べ、文字 **S** で始まる名前を持つ行を選択するように SAS に指示します。

```
where lastname=: 'S';
```

SQL プロシジャでは、演算子と組み合わせて使用されるコロン修飾子はサポートされていないことに注意してください。LIKE 演算子をかわりに使用できます。

論理演算子

ブール論理演算子 AND、OR、および NOT を使用して、WHERE 式を組み合わせたり変更したりできます。複合 WHERE 式の基本構文は次のとおりです。

WHERE*where-expression-1*AND | OR | NOT*where-expression-n*

論理演算子とそれに相当する記号を次の表に示します。

表 20.10　論理(ブール)演算子

記号	同等のニーモニック	説明	例
&	AND	2 つ以上の式を組み合わせ、すべての条件を満たす行を見つけます。	where skill eq 'java' and years eq 4;
!または|または¦	OR	2 つ以上の式を組み合わせ、条件のいずれかまたはすべてを 満たす行を見つけます。	where skill eq 'java' or years eq 4;
^または~または¬	NOT	指定された基準の補集合を見つけることによって条件を変更します。NOT 論理演算子は、SAS および WHERE 式演算子と組み合わせて使用できます。また、NOT 演算子を AND および OR と組み合わせることができます。	where skill not eq 'java' or years not eq 4;

複合式の評価順序

SAS が複合 WHERE 式(複数の条件)に遭遇すると、ソフトウェアはルールに従って各式を評価する順序を決定します。WHERE 式を組み合わせると、SAS は次の順序で条件を処理します。

1 **NOT** – 最初に処理されます。

2 **AND** – 次に処理されます。

3 **OR** – 最後に処理されます。

かっこを使用した評価順序の制御

SAS 式をかっこで囲むことで、式の評価順序を制御できます。最も内側のかっこのセット内の式が最初に処理され、次の外側のかっこが続き、かっこ内のすべての式が処理されるまで外側に向かって処理されます。

たとえば、 SAS/GRAPH または SAS/STAT のソフトウェアを備えているカナダのすべてのサイトのリストが必要な場合を考えます。かっこを使用せずに次の式を発行します。

```
where product='GRAPH' or product='STAT' and country='Canada';
```

ただし、結果には、SAS/STAT ソフトウェアをライセンスするカナダのサイトに加えて、SAS/GRAPH ソフトウェアをライセンスするすべてのサイトが含まれます。正しい結果を取得するには、かっこを使用します。これにより、SAS は最初にかっこ内の比較を評価し、いずれかのプロダクトライセンスを持つサイトのリストを提供し、その結果が残りの条件に使用されます。

```
where (product='GRAPH' or product='STAT') and country='Canada';
```

IN 演算子

IN 演算子も比較演算子です。値のリスト内のいずれかの値に等しい文字値と数値を検索します。値のリストはかっこで囲み、各文字値を引用符で囲み、カンマまたは空白で区切る必要があります。

たとえば、ノースカロライナまたはテキサスにあるすべてのサイトが 必要だとします。次のように指定できます。

```
where state = 'NC' or state = 'TX';
```

ただし、リスト内の任意の州を選択する IN 演算子を使用する方が簡単です。

```
where state in ('NC','TX');
```

さらに、NOT 論理演算子を使用してリストを除外することもできます。

```
where state not in ('CA', 'TN', 'MA');
```

短縮表記を使用して、検索する連続した整数の範囲を指定できます。範囲は、検索するリストの値として構文 M:N を使用して指定します。M は下限、N は上限です。M と N は整数である必要があり、M、N、および M と N の間のすべての 整数が範囲に含まれます。

たとえば、次のステートメントは同等です。

- y = x in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
- y = x in (1:10);

関連項目

“変数リストの種類” (92 ページ)

間隔比較

間隔比較(または、完全に制限された範囲条件)は、上限と下限の両方を指定する 2 つの比較演算子の間の変数で構成されます。たとえば、次の式は、500-1000 の範囲内の従業員番号を返します。この場合、より小さい(<)比較演算子に等号が付いている場合、この比較は包含間隔と見なされます。

```
where 500 <= empnum <= 1000;
```

前述の範囲条件式は次と同等であることに注意してください。

```
where empnum >= 500 and empnum <= 1000;
```

NOT 論理演算子と完全に制限された範囲条件を組み合わせて、範囲外の行を選択できます。かっこが必要であることに注意してください。

```
where not (500 <= empnum <= 1000);
```

BETWEEN-AND 演算子

BETWEEN-AND 演算子は、変数の値が値の両端を含む範囲内に収まる行を選択する、完全に制限された範囲条件とも見なされます。

範囲の制限を定数または式として指定できます。指定する範囲はすべて 両端を含む範囲であり、範囲の制限のいずれかに等しい値が範囲内に収まります。BETWEEN-AND を使用するための一般的な構文は次のとおりです。

WHERE variable BETWEEN value AND value;

次に例を示します。

```
where empnum between 500 and 1000;
where taxes between salary*0.30 and salary*0.50;
```

NOT 論理演算子と BETWEEN-AND 演算子を組み合わせて、範囲外の行を選択できます。

```
where empnum not between 500 and 1000;
```

注: BETWEEN-AND 演算子と完全に制限された範囲条件は、同じ結果を生成します。つまり、次の WHERE 式は同等です。

```
where 500 <= empnum <= 1000;
where empnum between 500 and 1000;
```

CONTAINS 演算子

CONTAINS (?)演算子の最も一般的な使用法は、文字変数の値内で指定された文字セットを検索して行を選択することです。変数値内の文字列の位置は関係ありません。ただし、演算子は比較するときに大文字と小文字が区別されます。

次の例では、変数 Company の値 **Mobay** および **Brisbayne** を持つ行を選択しますが、**Bayview** を含む行は選択しません。

```
where company contains 'bay';
where company ? 'bay';
```

NOT 論理演算子と CONTAINS 演算子を組み合わせて、指定した文字列に含まれない行を選択できます。

```
where company not contains 'bay';
```

2 つの変数で CONTAINS 演算子を使用して、ある変数が別の変数に含まれているかどうかを判断することもできます。2 つの変数を指定する場合は、末尾にスペースが含まれる可能性があることに留意してください。これは TRIM 関数を使用して解決できます。

```
proc sql;
  select *
  from table1 as a, table2 as b
  where a.fullname contains trim(b.lastname) and
    a.fullname contains trim(b.firstname);
```

さらに、TRIM 関数はマクロ変数を検索するときに役立ちます。

```
proc print;
  where fullname contains trim("&lname");
run;
```

IS NULL または IS MISSING 演算子

IS NULL または IS MISSING 演算子は、変数の値が欠損している行を選択します。演算子は、通常欠損値文字と特殊欠損値文字の両方を含む行を選択し、文字変数と数値変数の両方に使用できます。

```
where idnum is missing;
where name is null;
```

次は文字データと同等です。

```
where name is null;
where name = ' ';
```

また、次は数値データと同等です。このステートメントは、欠損値を特殊な欠損値文字で区別します。

```
where idnum <= .Z;
```

次のように、NOT 論理演算子と IS NULL または IS MISSING を組み合わせて、非欠損値を選択できます。

```
where salary is not missing;
```

LIKE 演算子

LIKE 演算子はパターンマッチングによって行を選択します。つまり、文字変数の値を指定されたパターンと比較して行を選択します。

LIKE 演算子は大文字と小文字を区別し、パターンを指定するために 2 つの特殊文字を使用します。

percent sign (%)
: 任意の数の文字がその位置を占めることができることを指定します。たとえば、次の WHERE 式は、名前が文字 **N** で始まるすべての従業員を選択します。名前は任意の長さにすることができます。

```
where lastname like 'N%';
```

underscore (_)
: アンダースコア 1 文字につき、値内の 1 文字のみと一致します。パターン内で複数の連続したアンダースコア文字を指定したり、同じパターン内でパーセント記号とアンダースコアを指定したりできます。たとえば、さまざまな形式の LIKE 演算子を使用して、次の名のリストから文字値を選択できます。

- **Diana**
- **Diane**
- **Dianna**
- **Dianthus**
- **Dyan**

次の表は、LIKE 演算子のさまざまな形式を使用してどの名前が選択されるかを示しています。

表 20.11　LIKE 演算子を使用して選択された名前

パターン	選択された名前
where frstname like 'D_an'	Dyan
where frstname like 'D_an_'	Diana、Diane
where frstname like 'D_an__'	Diana
where frstname like 'D_an%'	リスト内のすべての名前

SAS 文字式を使用してパターンを指定することはできますが、SAS 関数を使用する SAS 文字式は使用できません。

NOT 論理演算子と LIKE を組み合わせて、次のように、指定されたパターンを持たない値を選択できます。

```
where frstname not like 'D_an%';
```

%および_文字は、LIKE 演算子に対して特別な意味を持ちます。%および_文字を含むパターンを検索する場合は、エスケープ文字を使用する必要があります。

エスケープ文字は、一連の文字の中で、その後に続く文字が別の意味を持つことを示す 1 つの文字です。LIKE 演算子の場合、エスケープ文字は、特殊文字関数を実行するかわりに、変数値内の%および_文字のリテラルインスタンスを検索することを意味します。

たとえば、変数 X に値 **abc**、**a_b**、および **axb** が含まれている場合、エスケープ文字を含む次の LIKE 演算子は、値 **a_b** のみを選択します。エスケープ文字(/)は、パターンが文字 **a** と **b** で囲まれたリテラル'_'を検索することを指定します。エスケープ文字(/)は検索の一部ではありません。

```
where x like 'a/_b' escape '/';
```

エスケープ文字を使用しない場合、次の LIKE 演算子は値 **a_b** および **axb** を選択します。検索パターン内のアンダースコアは、アンダースコア付きの値を含む任意の単一の **b** 文字と一致します。

```
where x like 'a_b';
```

エスケープ文字を指定するには、パターンマッチング式にエスケープ文字を含めてから、キーワード ESCAPE の後にエスケープ文字式を続けます。

```
LIKE 'pattern-matching-expression' ESCAPE 'escape-character-expression'
```

```
WHERE x LIKE a/_b ESCAPE '/';
```

WHERE 式の LIKE 演算子にエスケープ文字を含める場合は、次の点に注意してください。

- パターンマッチング式は引用符で囲まれた文字列である必要があります。
- パターンマッチング式に列名を含めることはできません。
- エスケープ文字式は、1 文字として評価される文字列である必要があります。
- エスケープ文字式が 1 文字の場合は、引用符で囲みます。

類似演算子

類似(=*)演算子は、指定された単語のスペルバリエーションを含む行を選択します。演算子は、Soundex アルゴリズムを使用して変数値とオペランドを比較します。詳細については、*SAS 関数と CALL ルーチン: リファレンス*の SOUNDEX 関数を参照してください。

注: Soundex アルゴリズムは英語に偏っており、英語以外の言語ではあまり役に立ちません。

類似演算子は便利ですが、常にすべての可能な値を選択するとは限りません。たとえば、次の Smith に似た名前のリストから行を選択するとします。

- **Schmitt**
- **Smith**

- **Smithson**
- **Smitt**
- **Smythe**

次の WHERE 式は、このリストから **Smithson** を除くすべての名前を選択します。

```
where lastname=* 'Smith';
```

NOT 論理演算子と類似演算子を組み合わせて、指定した単語のスペルバリエーションを含まない値を選択できます。できないことの例を次に示します。

```
where lastname not =* 'Smith';
```

注: 類似演算子はインデックスを使用して最適化できません。

SAS Viya プラットフォーム 2020.1.4 以降では、SOUNDSLIKELEN システムオプションを使用して、類似演算子の使用時に結果を比較するバイト数を指定できます。詳細については、"SOUNDSLIKELEN システムオプション" (*SAS システムオプション: リファレンス*)を参照してください。

SAME-AND 演算子

SAME-AND 演算子を使用すると、元の条件を再入力することなく、プログラムの後半で既存の WHERE 式に条件を追加できます。この機能は次の場合に役立ちます。

- 対話型 SAS プロシジャ
- コマンドラインに WHERE 式を入力できる全画面 SAS プロシジャ
- あらゆる種類の RUN グループ処理

追加の条件を挿入する場合は、WHERE 式とともに SAME-AND 演算子を使用します。SAME-AND 演算子の形式は次のとおりです。

- where-expression-1;
- *SAS statements*...
- WHERE SAME AND where-expression-2;
- *SAS statements*...
- WHERE SAME AND where-expression-*n*;

SAS は、以前に定義された条件に加えて、SAME-AND 演算子の後の条件を満たす行を選択します。SAS は、既存のすべての条件を、単一の WHERE 式内の AND 演算子で区切られた条件であるかのように処理します。

次の例は、GPLOT プロシジャの RUN グループ内で SAME-AND 演算子を使用する方法を示しています。SAS データセット YEARS には 3 つの変数があり、2009-2011 年の四半期データが含まれています。

```
proc gplot data=years;
  plot unit*quar=year;
run;
  where year > '01jan2009'd;
run;
```

```
      where same and year < '01jan2012'd;
    run;
```

次の WHERE 式は、前述のコードと同等です。

```
where year > ''01jan2009'd and year < '01jan2012'd;
```

MIN 演算子および MAX 演算子

MIN 演算子または MAX 演算子を使用して、2 つの量の最小値または最大値を見つけます。最小値または最大値を知りたい 2 つの量で演算子を囲みます。

- MIN 演算子は、2 つの値のうち小さい方を返します。
- MAX 演算子は、2 つの値のうち大きい方を返します。

たとえば、A が B より小さい場合、次は A の値を返します。

```
where x = (a min b);
```

注: 記号表現><はサポートされておらず、<>は"等しくない"と解釈されます。

連結演算子

連結演算子は文字値を連結します。連結演算子は次のように指定します。

- || (2 つの OR 記号)
- !! (2 つの感嘆符)
- ¦¦ (2 つの縦破線)

次に例を示します。

```
where name = 'John'||'Smith';
```

接頭辞演算子

プラス記号(+)とマイナス記号(–)は、接頭辞演算子または算術演算子のいずれかになります。これらは、式の先頭または開きかっこの直前に現れる場合、接頭辞演算子となります。接頭辞演算子は、変数、定数、SAS 関数、かっこ内の式に適用されます。

```
where z = –(x + y);
```

注: NOT 演算子も接頭辞演算子とみなされます。

効率的な WHERE 式を作成するためのガイドライン

データセットのインデックスの作成に加えて、効率的な WHERE 式を作成するためのガイドラインをいくつか示します。

表 20.12　効率的な WHERE 式の構築

ガイドライン	効率的	非効率的
%または_で始まる LIKE 演算子の使用は避けてください。	where country like 'A%INA';	where country like '%INA';
定数のみを含む算術式の使用は避けてください。	where salary > 48000;	where salary > 12*4000;

インデックスと WHERE 式

インデックスは、SAS が各行を順次読み取る必要なく、特定の行に直接アクセスできるようにするオプションのファイルです。インデックスにより、SAS は値によってオブザベーションを検索できるようになります。SAS データセットにインデックスを作成すると、WHERE 処理のパフォーマンスが大幅に向上します。

たとえば、次の DATA ステップは、変数 Actual の値に基づいて行を選択するように最適化された sashelp.prdsal2 データセットにインデックスを作成します。

```
data prdIndx(index=(Actual));
  set sashelp.prdsal2;
run;
proc print data=myindex(where=(Actual<10));
run;
```

23,000 行を超えるデータセット sashelp.prdsal2 が、インデックスなしで変数 Actual によって並べ替えられていると仮定します。式 **where Actual<10** を処理するために、SAS はデータセットの最初の行から順次行の読み取りを開始し、10 より大きい行が見つかった場合にのみ行の読み取りを停止します。SAS は行の読み取りを停止するタイミングを決定できますが、インデックスがなければ、どこから開始するかはわかりません。したがって、SAS は最初の行から開始します。これには多くの行の読み取りが必要になる場合があります。

インデックスがあると、SAS はどの行が基準を満たすかを判断できます。これは、WHERE 式の最適化と呼ばれます。

注: ただし、デフォルトでは、SAS はインデックスを使用するか、データセット全体を順次読み取るかを決定します。

SAS インデックスの詳細については、次のドキュメントを参照してください。

表 20.13　SAS インデックスのドキュメント

環境	リソース
SAS Compute Server (V9 エンジン)	■ “Using an Index for WHERE Processing” (*SAS V9 LIBNAME Engine: Reference*)
CAS サーバー(CAS エンジン)	■ “インデックス付け” (*SAS Cloud Analytic Servcies: 基本*) ■ “テーブルのインデックス作成” (*SAS Viya プラットフォーム: システムプログラミングガイド*)

WHERE 式を使用したデータセットおよびシステムオプション

WHERE 式を使用してオブザベーションのサブセットを条件付きで選択する場合、FIRSTOBS=および OBS=データセットオプション、または FIRSTOBS=および OBS=システムオプションを指定して、データをさらにサブセット化できます。

次の例では、PROC PRINT ステートメントの FIRSTOBS=および OBS=データセットオプションを使用して、WHERE ステートメントによって返されたデータのサブセットを出力します。

```
data shoes;
  set sashelp.shoes;
  keep Product Sales;
run;
proc print data=shoes(firstobs=2 obs=5);
  title 'Subset of Shoes WHERE Sales<800';
  title2 'Print rows 2 - 5';
  where Sales<800;
run;
```

Subset of Shoes WHERE Sales<800
Print rows 2 - 5

Obs	Product	Sales
176	Sport Shoe	$449
197	Sandal	$325
223	Sport Shoe	$450
297	Sandal	$601

注: データセットオプションの値は 176 と 297 ではなく、2 と 5 であることに注意してください。これは、PRINT 出力が元の Shoes 入力データセットからの物理番号を保持するためです。データセットオプション番号は、フィルタリングされたデータセットの論理行番号を表し、物理行番号を表すものではありません。

別のビューのビューである SAS ビュー(ネストされたビュー)を処理している場合、OBS=および FIRSTOBS=をデータのサブセットに適用すると、予期しない結果が生じる可能性があります。ネストされたビューの場合、OBS=および FIRSTOBS=の処理が各 SAS ビューに適用され、ルート(最下位レベル)ビューから開始され、各 SAS ビューの行がフィルタリングされます。その結果、基準を満たす行がなくなる可能性があります。

関連項目

"例: データセットオプション" (508 ページ)

IF ステートメント

IF ステートメントは、条件が満たされた場合にのみコードを実行します。サブセット化 IF と WHERE ステートメントはどちらも条件をテストして、SAS が行を処理する必要があるかどうかを決定します。ただし、この 2 つのステートメントは同等ではありません。

これら 2 つのステートメントの比較については、"サブセット化 IF ステートメントと WHERE ステートメント" (482 ページ)を参照してください。

IF ステートメントのタイプ

SAS には 2 タイプの IF ステートメントがあります。

表 20.14 IF ステートメントのタイプ

ステートメントタイプ	説明	例
サブセット化 IF ステートメント	指定された式の条件を満たす行のみ処理を続行します。	data if_example; set sashelp.class; if age>14 and weight<115; output; run;
IF-THEN/ELSE ステートメント	特定の条件を満たす行に対して SAS ステートメントを実行します。	data if_then; set sashelp.class; if age>14 and weight<115 then output; else delete;

ステートメントタイプ	説明	例
		run;

サブセット化 IF ステートメントと WHERE ステートメント

SAS データセットから条件付きで行を選択するには、WHERE 式またはサブセット化 IF ステートメントを使用できます。多くの場合、どちらのステートメントを使用しても同じタスクを実行できます。ただし、一部のタスクでは、特定のステートメントが必要になります。次の表は、いずれかのステートメントを使用する必要があるタスクを示しています。

表 20.15　WHERE 式またはサブセット化 IF ステートメントのいずれかを必要とするタスク

タスク	使用ステートメント
先行する DATA ステップを使用せずにプロシジャ内で選択を行います。	WHERE ステートメント
インデックス付きデータセットで得られる効率性を活用します。	WHERE ステートメント
BETWEEN-AND、CONTAINS、IS MISSING または IS NULL、LIKE、SAME-AND、類似などの特殊演算子のグループのいずれかを使用します。	WHERE ステートメント
SAS データセットにすでに存在する変数値以外の値に基づいて選択します。たとえば、生データから読み取られた値、または DATA ステップの過程で計算または割り当てられた値を選択できます。	サブセット化 IF ステートメント
最初ではなく、DATA ステップ中のどこかの時点で選択を行います。	サブセット化 IF ステートメント
条件付きで選択を実行します。	サブセット化 IF ステートメント

表 20.16 IF ステートメントと WHERE 式の動作の比較

WHERE 式	サブセット化 IF ステートメント
WHERE 式は、DATA ステップ、SAS プロシジャ、またはデータセットオプションとして使用できます。	サブセット化 IF ステートメントは、DATA ステップでのみ使用できます。
WHERE 式は、プログラムデータベクトルに読み込まれる*前に*条件をテストします。	サブセット化 IF ステートメントは、行がプログラムデータベクトルに読み込まれた*後に*条件をテストします。
条件が真の場合、SAS は行を PDV に読み込み、行を処理します。 条件が偽の場合、SAS は行を PDV に読み込まず、次の行の処理を続行します。 行に多くの変数または非常に長い文字変数(最大 32KB)が含まれている場合、行が読み取られる*前に*条件をテストすると、大幅な節約が実現できます。	条件が真の場合、SAS は行を処理します。 条件が偽の場合、SAS は PDV から行を削除し、次の行の処理を続行します。 行が読み取られる*前に*条件をテストしないと、特に行に多くの変数または非常に長い文字変数が含まれている場合に、コストが高くなる可能性があります。 データセットに変数が非常に少ない行や短い文字変数が含まれている場合、PDV へのデータの移動は高速になる可能性があります。ただし、32KB の文字データを含む変数は、データが 1 つの変数にのみ含まれている場合でも、移動に時間がかかります。この場合、WHERE 式は、PDV に不要なデータを読み取らないため、一般的にはより効率的です。
WHERE 式はインデックスを使用して最適化できます。	
WHERE 式は、LIKE や CONTAINS などの演算子とともに使用できます。	

関連項目

- “例: IF ステートメント” (503 ページ)
- “その他の制御フローステートメント” (484 ページ)
- “入力バッファーとプログラムバッファーの作成” (868 ページ)。

SELECT WHEN ステートメント

SELECT ステートメントは、複数のステートメントまたはステートメントのグループの 1 つを実行します。次の DATA ステップでは、SELECT ステートメントを使用して、sashelp.holiday 内の holiday 名に指定された値が含まれまる行のみを選択します。

```
data holiday;
 set sashelp.holiday;
  keep Name Cat;
  select (Name);
    when ('CHRISTMAS')  Cat=1;
    when ('MEMORIAL')   Cat=2;
    when ('MLK')        Cat=3;
  otherwise   delete;
end;
run;
proc print;
```

Obs	name	Cat
1	CHRISTMAS	1
2	CHRISTMAS	1
3	MLK	3
4	MEMORIAL	2

関連項目

“例: SELECT WHEN ステートメント” (506 ページ)

その他の制御フローステートメント

“SAS の条件付き処理ステートメントの概要” (459 ページ)で指定された SAS 言語要素に加えて、次のステートメントもプログラム実行のフローを制御します。

表 20.17　その他の SAS 制御フローステートメント

ステートメント	説明

ABORT ステートメント	現在の DATA ステップの実行を停止し、次の DATA または PROC ステップの処理を再開します。ABORT ステートメントで指定されたオプションと操作方法に応じて、SAS プログラムの実行を完全に停止することもできます。
CONTINUE ステートメント	現在の DO ループ反復の実行を(条件に基づいて)停止し、条件に基づいて DO ループの次の反復の実行を再開します。
DELETE ステートメント	現在の行の実行を停止し(行を削除)、DATA ステップの次の反復の実行を再開します。
EOF=オプション(INFILE ステートメント)	入力ファイルの終わりに達すると、SAS プログラムの実行は EOF=で指定されたラベルに続くステートメントにただちに指示されます。
END ステートメント	DO グループまたは SELECT グループ処理の実行を停止します。
ENDSAS ステートメント	SAS プログラムでステートメントが検出されるとすぐに、SAS プログラムの実行を停止し、SAS セッションを終了します。
GOTO ステートメント	SAS プログラムの実行を、GO TO ステートメントで指定されたステートメントラベルにただちに指示します。RETURN ステートメントが続く場合、GO TO は実行を DATA ステップの先頭に戻します。
HEADER=オプション(FILE ステートメント)	PUT ステートメントによって新しい出力ページが開始されるたびに、RETURN ステートメントが検出されるまで、HEADER=オプションで指定されたラベルに続くステートメントが実行されます。RETURN ステートメントが検出されると、SAS は HEADER=オプションがアクテ

	ィブ化されたポイントに制御を戻します。
LEAVE ステートメント	現在の DO ループまたは SELECT グループの実行を停止し、DO ループまたは SELECT グループ外の DATA ステップ内の次のステートメントから再開します。
LINK ステートメント	SAS プログラムの実行を、LINK ステートメントで指定されたステートメントラベルにただちに指示します。RETURN ステートメントが続く場合は、LINK ステートメントに続くステートメントに実行を戻します。
RETURN ステートメント	DATA ステップの特定のポイントでステートメントの実行を停止し、ステップ内の所定のポイントに戻ります。
STOP ステートメント	現在の DATA ステップの実行を停止し、現在の DATA ステップに続くすべてのステートメントの処理を再開します。

例: DO ループ

例: 単純な反復 DO ループの使用

コード例

この例では、反復 DO ステートメントを使用して、変数 balance の値を繰り返し減らし、これらの値を出力データセットに書き込みます。

DATA ステップは、SAS Compute Server 上の SAS で実行されます。

```
data loan;
  balance=10000;
  do payment_number=1 to 10;
```

```
    balance=balance-1000;
    output;
  end;
run;
proc print data=loan;
run;
```

アウトプット 20.2　反復 DO ループを使用した SAS データセットの残高減少を示す PROC PRINT 出力

Obs	balance	payment_number
1	9000	1
2	8000	2
3	7000	3
4	6000	4
5	5000	5
6	4000	6
7	3000	7
8	2000	8
9	1000	9
10	0	10

次の例では、同じ反復 DO ループを使用して CAS で DATA ステップを実行し、CAS 出力テーブルを作成します。

```
cas casauto sessopts=(caslib='casuser');
libname mylib cas;

data mylib.loan;
  balance=10000;
  do payment_number=1 to 10;
    balance=balance-1000;
    output;
  end;
run;
proc print data=mylib.loan;
run;
```

アウトプット 20.3　反復 DO ループを使用した CAS テーブルの残高減少を示す PROC PRINT 出力

Obs	balance	payment_number
1	9000	1
2	8000	2
3	7000	3
4	6000	4
5	5000	5
6	4000	6
7	3000	7
8	2000	8
9	1000	9
10	0	10

注: DATA ステップは CAS で実行されますが、入力テーブルがないため、DATA ステップは CAS の単一スレッドで実行されます。

要点

- 反復 DO ステートメントを使用すると、インデックス変数の値に基づいて、DO ステートメントと END ステートメント間のステートメントのグループを繰り返し実行できます。
- 反復 DO ループの形式は複数あります。
- DO UNTIL および DO WHILE ステートメントを使用すると、SAS データセットまたは入力テーブルから条件付きでデータを選択できます。
- DO ループは、SAS DATA ステップまたは CAS で実行されている DATA ステップで指定できます。構文と動作は両方の環境で同一です。詳細については、"CAS で DATA ステップを実行する際のベストプラクティス" (*SAS Cloud Analytic Services: DATA ステッププログラミング*)を参照してください。

関連項目

- "DO ステートメント: 反復" (*SAS DATA ステップステートメント: リファレンス*)
- "DATA ステップの実行場所の決定と制御" (*SAS Cloud Analytic Services: DATA ステッププログラミング*)
- "処理環境" (*SAS Cloud Analytic Services: DATA ステッププログラミング*)

例: ネストされた反復 DO ループの使用

コード例

次の例は、1 つの DO ループを別の DO ループ内に配置し、毎月複利で年利 7.5%の利息が得られる 1 年間の投資の価値を計算する方法を示しています。

```
data earn;
  Capital=2000;
  do Year=1 to 10;
    do Month=1 to 12;
      Interest=Capital*(.075/12);
      Capital+Interest;
      output;
    end;
  end;
```

```
run;
proc print data=earn; run;
```

アウトプット 20.4　ネストされた DO ループを使用して獲得した複利を示す部分的な PROC PRINT 出力

Obs	Capital	Year	Month	Interest
1	2008.33	1	1	8.3333
2	2016.70	1	2	8.3681
3	2025.10	1	3	8.4029
4	2033.54	1	4	8.4379
5	2042.02	1	5	8.4731
6	2050.52	1	6	8.5084
7	2059.07	1	7	8.5438
8	2067.65	1	8	8.5794
9	2076.26	1	9	8.6152
10	2084.91	1	10	8.6511
11	2093.60	1	11	8.6871
12	2102.32	1	12	8.7233
13	2111.08	2	1	8.7597
14	2119.88	2	2	8.7962
15	2128.71	2	3	8.8328
16	2137.58	2	4	8.8696
17	2146.49	2	5	8.9066
18	2155.43	2	6	8.9437
19	2164.41	2	7	8.9810

要点

- 反復 DO ステートメントは DO ループ内で実行できます。DO ループ内に DO ループを配置することをネストと呼びます。
- ネストされたループは、外側のループ内のデータを通過するたびに、内側のループ内のデータに対して何らかのアクションを繰り返す必要がある場合に便利です。

  たとえば、ファイルを行ごとに読み取り、各行に特定の単語が何回出現するかを数える必要がある場合があります。外側のループは行を読み取り、内側のループは行内の単語をループして単語を検索します。
- 反復 DO ステートメントを使用すると、インデックス変数の値に基づいて、DO と END の間にあるステートメントのグループを繰り返し実行できます。
- 反復 DO ループの形式は複数あります。

関連項目

“DO ステートメント: 反復” (*SAS DATA ステップステートメント: リファレンス*)

例: DOLIST 構文を使用して値のリストを反復処理する

コード例

この例では、DOLIST 構文を使用して値のリストを反復処理します。

```
data do_list;
 x=-5;
 do i=5,            /*a single value*/
    5, 4,          /*multiple values*/
    x + 10,        /*an expression*/
    80 to 90 by 5,  /*a sequence*/
    60 to 40 by x;  /*a sequence with a variable*/
  output;
 end;
run;
proc print data=do_list; run;
```

アウトプット 20.5　DOLIST 構文出力を示す PROC PRINT 出力

Obs	x	i
1	-5	5
2	-5	5
3	-5	4
4	-5	5
5	-5	80
6	-5	85
7	-5	90
8	-5	60
9	-5	55
10	-5	50
11	-5	45
12	-5	40

要点

- DOLIST は、リストの要素を反復処理する反復 DO ループに基づいた構文の一種です。リストは DO ステートメントの start-value として機能し、リスト内のアイテムはカンマで区切られます。

- 反復 DO ステートメントを使用すると、インデックス変数の値に基づいて、DO と END の間にあるステートメントのグループを繰り返し実行できます。

関連項目

- “DO ループの種類” (462 ページ)。
- “DO ステートメント: 反復” (*SAS DATA ステップステートメント: リファレンス*)

例: 反復 DO TO 構文を使用して特定の回数を反復する

コード例

この例では、反復 DO TO-*value* 構文を使用して、x の値を繰り返し増やします。ループでは、OUTPUT ステートメントも指定して、x の各増分値を出力データセットに書き込みます。

```
data do_to;
  x=0;
  do i=0 to 10;
    x=x+1;
    output;
  end;
run;
proc print data=do_to;
run;
```

アウトプット 20.6　DO TO-value 構文を示す PROC PRINT 出力

Obs	x	i
1	1	0
2	2	1
3	3	2
4	4	3
5	5	4
6	6	5
7	7	6
8	8	7
9	9	8
10	10	9
11	11	10

要点

- DO TO-*value* 構文では、TO 値は反復 DO ループ内の index-variable の終了値を指定します。

関連項目

- “DO ステートメント: 反復” (*SAS DATA ステップステートメント: リファレンス*)
- “さまざまな形式の反復 DO ステートメントを使用する” (*SAS DATA ステップステートメント: リファレンス*)

例: 反復 DO TO BY 構文を使用して、特定の間隔で特定の回数を反復する

コード例

この例では、反復 DO TO-*value* BY-*increment* 構文を使用して、x の値を **1** ずつ繰り返し増やします。BY-*increment* 値は、ループが実行されるたびにインデックス変数を **2** ずつ増やすことを指定します。

ループでは、OUTPUT ステートメントが x の各増分値を出力データセットに書き込むことも指定します。

```
data do_to_by;
  x=0;
  do i=0 to 10 by 2;
    x=x+1;
    output;
  end;
run;
proc print data=do_to_by;
run;
```

アウトプット 20.7　反復 DO TO BY 構文を示す PROC PRINT 出力

Obs	x	i
1	1	0
2	2	2
3	3	4
4	4	6
5	5	8
6	6	10

要点

- DO TO-*value* BY-*increment* 構文では、BY 増分値に正または負の数値(または数値を生成する式)を指定して、反復 DO ループ内で index-variable を増やす方法を制御します。
- 反復 DO ステートメントを使用すると、インデックス変数の値に基づいて、DO と END の間にあるステートメントのグループを繰り返し実行できます。

関連項目

- "DO ステートメント: 反復" (*SAS DATA ステップステートメント: リファレンス*)
- "さまざまな形式の反復 DO ステートメントを使用する" (*SAS DATA ステップステートメント: リファレンス*)

例: 反復 DO WHILE および DO UNTIL ステートメントを使用して、ステートメントのグループを繰り返し実行する

コード例

この例の最初の DATA ステップでは、DO ループを使用して、条件が真である*間*、ステートメントのグループを繰り返し実行します。プログラムは、Balance がゼロより大きい*間*、繰り返し反復して減らすことにより、指定されたローン金額に対して行う必要がある支払い回数を計算します。

DATA ステップは、SAS Compute Server 上の SAS で実行されます。

```
data loan;
  balance=10000;
  payment=0;
  do while (balance>0);
    balance=balance-1000;
    payment=payment+1;
    output;
  end;
run;
proc print data=mylib.loan;
run;
```

アウトプット 20.8　DO WHILE ステートメントを使用して SAS データセットの残高減少を示す PROC PRINT 出力

Obs	balance	payment
1	9000	1
2	8000	2
3	7000	3
4	6000	4
5	5000	5
6	4000	6
7	3000	7
8	2000	8
9	1000	9
10	0	10

次の DATA ステップでは、同じデータを別のライブラリ mylib.loan の新しいテーブルにロードしてから、指定されたローンに対して行う必要がある支払い回数を計算します。これは、繰り返し反復し、Balance の値がゼロになるまで減らしていくことによって行われます。

```
libname mylib;

data mylib.loan;
  balance=10000;
  payment=0;
  do until (balance=0);
    balance=balance-1000;
    payment=payment+1;
    output;
  end;
run;
proc print data=mylib.loan;
run;
```

注: DATA ステップは CAS で実行されますが、入力テーブルがないため、DATA ステップは単一スレッドで実行されます。

アウトプット 20.9　DO UNTIL ステートメントを使用してテーブルの残高減少を示す PROC PRINT 出力

Obs	balance	payment
1	9000	1
2	8000	2
3	7000	3
4	6000	4
5	5000	5
6	4000	6
7	3000	7
8	2000	8
9	1000	9
10	0	10

要点

- DO ループは SAS または CAS のいずれかで指定できます。構文と動作は同一です。CAS での DATA ステッププログラミングの詳細については、*SAS Cloud Analytic Services: DATA ステッププログラミング*を参照してください。
- DO UNTIL ステートメントを使用すると、条件が真になるまで DO ループ内のステートメントを繰り返し実行できます。
- DO WHILE ステートメントを使用すると、条件が真の間、DO ループ内のステートメントを繰り返し実行できます。

関連項目

- "DO UNTIL ステートメント" (*SAS DATA ステップステートメント: リファレンス*)
- "DO WHILE ステートメント" (*SAS DATA ステップステートメント: リファレンス*)
- "DO ステートメント: 反復" (*SAS DATA ステップステートメント: リファレンス*)

例: WHERE 処理

例: WHERE ステートメントを使用した条件付き行選択

コード例

この例では、WHERE ステートメントを使用して、sashelp.class データセットから条件付きで行を選択し、選択した行をテーブル mylib.shoes に書き込みます。

最初の DATA ステップは SAS で実行され、データがサブセット化され、Sales が **$500,000** より大きい行のみが選択されます。フィルタリングされたデータは出力テーブル mylib.shoes にロードされます。

2 番目の DATA ステップは、mylib.shoes テーブルで実行され、データをさらにサブセット化します。このデータステップでは、変数 Region の値が **Canada** である行のみを選択します。選択された行は出力テーブル mylib.shoes2 に書き込まれます。

```
libname mylib ;
```

```
data mylib.shoes;
  set sashelp.shoes;
  where Sales>=500000;
run;
proc print data=mylib.shoes;
  var Region Product Sales;
 run;

data mylib.shoes2;
  set mylib.shoes;
  where Region="Canada";
run;

proc print data=mylib.shoes2;
  var Region Product Sales;
run;
```

アウトプット 20.10　Sales が$500,000 より大きい mylib.shoes テーブルの PROC PRINT 出力

Obs	Region	Product	Sales
1	Canada	Men's Dress	$757,798
2	Canada	Slipper	$700,513
3	Canada	Women's Dress	$756,347
4	Central America/Caribbean	Men's Casual	$576,112
5	Middle East	Men's Casual	$1,298,717
6	Western Europe	Women's Casual	$502,636

アウトプット 20.11　Region が Canada である mylib.shoes2 テーブルの PROC PRINT 出力

Obs	Region	Product	Sales
1	Canada	Men's Dress	$757,798
2	Canada	Slipper	$700,513
3	Canada	Women's Dress	$756,347

注: どちらの DATA ステップも、shoes データセットから行のサブセットを作成します。ただし、2 番目の DATA ステップは、分散データに対して複数のスレッドで CAS で実行されるため、処理が高速になります。

要点

- WHERE 式は、SAS データセットまたは出力テーブルで処理される行を選択するための条件を定義する SAS 式の一種です。
- DATA ステップで WHERE ステートメントが指定されている場合、SAS は入力データセットからすべての行を読み取る必要はありません。したがって、WHERE ステートメントは SAS プログラムのパフォーマンスを向上させることができます。WHERE ステートメントを使用すると、SAS は入力データセットからすべての行を読み取る必要がないため、SAS プログラムの効率が向上します。

- WHERE ステートメントは、SAS または CAS で実行されている DATA ステップで指定できます。構文と動作は両方の環境で同一です。CAS での DATA ステッププログラミングの詳細については、*SAS Cloud Analytic Services: DATA ステッププログラミング*を参照してください。
- WHERE ステートメントを IF-THEN ステートメントの一部として使用することはできません。
- WHERE ステートメントには、論理演算子によって結合された複数の WHERE 式を含めることができます。これらの式は複合 WHERE 式と呼ばれます。

関連項目

- *"WHERE ステートメント" (SAS DATA ステップステートメント: リファレンス)*
- *SAS Cloud Analytic Services: DATA ステッププログラミング*

例: 複合 WHERE 式を使用した条件付き行選択

コード例

この例では、複合 WHERE 式を使用して、入力テーブル mylib.shoes からデータのサブセットを選択します。

CASUTIL プロシジャは、まず SAS データセット sashelp.shoes を CAS にロードします。次に、CAS で DATA ステップが実行され、Sales が**$500,000** より大きく、Region が **Canada** である行が条件付きで選択されます。複合 WHERE 式は、AND 演算子によって結合された 2 つの WHERE 式で構成されます。

```
data mylib.shoes;
  set mylib.shoes;
  where Sales>=500000 and Region="Canada";
  keep Region Product Sales;
run;

proc print data=mylib.shoes; run;
```

アウトプット 20.12　複合 WHERE 式を使用した条件付き行選択の PROC PRINT 出力

Obs	Region	Product	Sales
1	Canada	Men's Dress	$757,798
2	Canada	Slipper	$700,513
3	Canada	Women's Dress	$756,347

要点

- *複合 WHERE 式*は、論理演算子によって結合された複数の条件で構成されます。
- SAS は、複数の条件を含む WHERE 式を検出すると、次の順序で条件を処理します。
  1. NOT 式が最初。
  2. AND 式が次。
  3. OR 式が最後。

関連項目

- "WHERE ステートメント" (*SAS DATA ステップステートメント: リファレンス*)
- 複合 WHERE 式

例: 複数の条件に基づいた条件付き行選択

コード例

次の例では、WHERE ステートメントで AND 演算子を使用して、Age と Sex の条件に基づいて行を検索します。

```
data class;
  set sashelp.class;
  where sex="M" and age >= 15;
run;
proc print data=class;
run;
```

出力データセットには、15 歳以上の男性に関する情報が含まれています。

アウトプット 20.13　複数の条件に基づいた条件付き行選択の PROC PRINT 出力

Obs	Name	Sex	Age	Height	Weight
1	Philip	M	16	72.0	150
2	Ronald	M	15	67.0	133
3	William	M	15	66.5	112

次の例では、OR はいずれかの条件を満たす行を検索します。

```
data class;
  set sashelp.class;
  where sex="M" or age>=15;
run;
proc print data=class;
  title 'OR finds all Males and Anyone 15 Years or Older';
run;
```

アウトプット 20.14　複数の条件に基づいた条件付き行選択の PROC PRINT 出力

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Henry	M	14	63.5	102.5
3	James	M	12	57.3	83.0
4	Janet	F	15	62.5	112.5
5	Jeffrey	M	13	62.5	84.0
6	John	M	12	59.0	99.5
7	Mary	F	15	66.5	112.0
8	Philip	M	16	72.0	150.0
9	Robert	M	12	64.8	128.0
10	Ronald	M	15	67.0	133.0
11	Thomas	M	11	57.5	85.0
12	William	M	15	66.5	112.0

次の例では、より小記号(<)は、age と sex の条件を満たす行を検索します。

```
data class;
  set sashelp.class;
  where age < 15 and sex NE "M";
run;
proc print data=class;
  title 'Finds Females
        Older less than 15 Years';
run;
```

アウトプット 20.15　複数の条件に基づいた条件付き行選択の PROC PRINT 出力

Females 15 Years Old and Younger

Obs	Name	Sex	Age	Height	Weight
1	Alice	F	13	56.5	84.0
2	Barbara	F	13	65.3	98.0
3	Carol	F	14	62.8	102.5
4	Jane	F	12	59.8	84.5
5	Joyce	F	11	51.3	50.5
6	Judy	F	14	64.3	90.0
7	Louise	F	12	56.3	77.0

ブール演算子によって結合された式を SAS が処理する順序は、出力に影響します。デフォルトの演算順序では、最初に NOT 式、次に AND 式、最後に OR 式が処理されます。

```
data class;
  set sashelp.class;
  where age>15 or height<60 and sex="F";
run;
proc print data=class;
  title 'age > 15 OR height < 60 AND sex = F';
run;
```

アウトプット 20.16　複数のブール演算子を使用した複数の条件に基づく条件付き行選択の PROC PRINT 出力

Obs	Name	Sex	Age	Height	Weight
1	Alice	F	13	56.5	84.0
2	Jane	F	12	59.8	84.5
3	Joyce	F	11	51.3	50.5
4	Louise	F	12	56.3	77.0
5	Philip	M	16	72.0	150.0

評価の順序を制御するには、かっこを使用します。これは、かっこを使用して OR 式が最初に評価され、次に AND 式が評価されることを指定する以外は、同じ例です。

```
data class;
  set sashelp.class;
  where (age>15 or height<60) and sex="F";
run;
proc print data=class;
  title '(age > 15 OR height < 60) AND sex = F';
run;
```

アウトプット 20.17　行の条件付き選択と演算順序の制御の PROC PRINT 出力

Obs	Name	Sex	Age	Height	Weight
1	Alice	F	13	56.5	84.0
2	Jane	F	12	59.8	84.5
3	Joyce	F	11	51.3	50.5
4	Louise	F	12	56.3	77.0

要点

- 複数の条件に基づいて行を条件付きで選択するには、ブール演算子 AND、OR および NOT を使用します。
- SAS は、複数の条件を含む WHERE 式を検出すると、次の順序で条件を処理します。
  1. NOT 式が最初。
  2. AND 式が次。
  3. OR 式が最後。

例: WHERE=データセットオプションを使用した条件付き行選択

コード例

この例では、WHERE=データセットオプションを使用して、sashelp.shoes データセットから条件付きで行を選択します。

```
data sales;
  set sashelp.shoes(where=(Region="Canada" and Sales<2000));
run;
proc print data=sales; run;
```

Obs	Region	Product	Subsidiary	Stores	Sales	Inventory	Returns
1	Canada	Sandal	Toronto	2	$1,190	$6,763	$38

CAS サーバーで処理が行われるように DATA ステップを実行するには、SET ステートメントで入力テーブルに WHERE=データセットオプションを指定します。WHERE=データセットオプションは、入力 CAS テーブル(SET ステートメント内)で指定した場合にのみ CAS での処理がサポートされます。出力 CAS テーブルで WHERE=データセットオプションを指定すると、次の例に示すようにエラーが発生しません。

```
/* Specify the WHERE= data set option on the output CAS
  table (unsupported for CAS DATA step processing) */

data mylib.sales(where=(Region="Canada" and Sales<2000));
  set mylib.shoes;
run;
proc print data=mylib.sales; run;
```

ただし、次のログ出力が示すように、DATA ステップは CAS ではなく SAS Compute Server 上の SAS で実行されます。

```
101  data mylib.sales(where=(Region="Canada" and Sales<2000));
102    set mylib.shoes;
103  run;
NOTE: There were 395 observations read from the data set mylib.SHOES.
NOTE: The data set mylib.SALES has 1 observations and 7 variables.
```

非常に大きなテーブルの場合は、CAS で DATA ステップを実行する方が効率的です。

要点

- SAS Compute Server 上の SAS で処理する場合、入力データセット(SET ステートメント)または出力データセット(DATA ステートメント)のいずれかに WHERE=データセットオプションを指定できます。
- WHERE=データセットオプションは、CAS サーバーでの処理で SET ステートメント(入力テーブル)のみがサポートされています。
- SET および MODIFY ステートメント内で、WHERE=データセットオプションと POINT=オプションを一緒に使用することはできません。

関連項目

- *“WHERE= データセットオプション” (SAS データセットオプション: リファレンス)*
- *“例: サポートされていない言語要素のために SAS Compute Server 上で実行される DATA ステップ” (SAS Cloud Analytic Services: DATA ステッププログラミング)*

例: インデックスを使用した WHERE 処理のパフォーマンス向上

コード例

次の例では、最初の DATA ステップで出力データセット mybaseball が作成され、index=オプションで team 変数に単一インデックスが追加されます。2 番目の DATA ステップでは、データセットを読み取り、チーム名が **Atlanta** である行のみを処理対象として選択します。

```
data mybaseball(index=(team));
  set sashelp.baseball;
```

```
run;

data mybaseball;
  set sashelp.baseball;
  where Team="Atlanta";
  keep Name Team Position;
run;
proc print data=mybaseball; run;
```

要点

- SAS データセットにインデックスを作成すると、WHERE 処理のパフォーマンスが大幅に向上します。インデックスは、特定の行への直接アクセスを提供するために SAS データセットに対して作成できるオプションのファイルです。
- データセットを作成するときに DATA ステップでインデックスを作成するには、INDEX=データセットオプションを使用します。

関連項目

"Indexes" (*SAS V9 LIBNAME Engine: Reference*)

例: IF ステートメント

例: サブセット化 IF ステートメントを使用したデータのサブセット化

コード例

この例では、サブセット化 IF ステートメントを使用して入力 SAS データセットのデータをサブセット化し、行を出力データセットに書き出す方法を示します。DATA ステップは、条件 Age=13 が真の場合に行に対して SET ステートメントを実行します。したがって、プログラムは、Age が **13** に等しい 3 行のみを選択します。

```
data class;
  set sashelp.class;
  if Age = 13;
run;
proc print data=class; run;
```

アウトプット 20.18　サブセット化 IF ステートメントを使用してデータをフィルタリングする方法を示す PROC PRINT 出力

Obs	Name	Sex	Age	Height	Weight
1	Alice	F	13	56.5	84
2	Barbara	F	13	65.3	98
3	Jeffrey	M	13	62.5	84

次の DATA ステップは、変数 Age の値が欠損している行に対して SET ステートメントを実行します。入力データには Age の欠損データが含まれていないため、SAS は IF ステートメントを **false** と評価し、出力データセットに行は書き込まれません。

```
data class;
  set sashelp.class;
  if Age = .;
run;
proc print data=class; run;
```

例のコード 20.1　サブセット化 IF ステートメントを使用してデータをサブセット化した場合のログ出力

```
82   proc print data=class; run;
NOTE: No observations in data set WORK.CLASS.
```

要点

- サブセット化 IF ステートメントでは、入力 SAS データソースからデータをフィルタリングし、指定した条件を満たす行のみを書き出します。
- 式が偽の場合(値が 0 または欠損値である場合)、その行に対してそれ以降のステートメントは処理されず、現在の行は出力に書き込まれず、DATA ステップの残りのプログラムステートメントは実行されません。

関連項目

- サブセット化 IF ステートメント.
- “IF-THEN/ELSE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)

例: IF ステートメントを使用した特定の行の条件付き選択

コード例

この例では、IF-THEN/ELSE ステートメントを使用して、Region が **Canada** で Product が **Slipper** である行を条件付きで選択する方法を示します。

```
data slipper;
  set sashelp.shoes;
  format commission comma6.;
  if Region = "Canada" and Product="Slipper"
    then commission = (Sales - Returns) * .10;
    else delete;
  keep Region Product Sales Returns Commission;
run;
proc print data=slipper;
```

アウトプット 20.19　IF-THEN/ELSE ステートメントを使用してデータを条件付きで選択する方法を示す PROC PRINT 出力

Obs	Region	Product	Sales	Returns	commission
1	Canada	Slipper	$5,676	$253	542
2	Canada	Slipper	$135,305	$3,395	13,191
3	Canada	Slipper	$30,905	$1,397	2,951
4	Canada	Slipper	$80,352	$1,908	7,844
5	Canada	Slipper	$700,513	$21,247	67,927

要点

- IF-THEN/ELSE ステートメントは、特定の条件を満たすオブザベーションに対して SAS ステートメントを実行します。THEN 句が実行されない場合、オプションの ELSE ステートメントが代替アクションを指示します。
- SAS では、IF-THEN ステートメントの式を評価し、非ゼロ、ゼロ、または欠損のいずれかの結果を生成します。
- 結果が非ゼロおよび非欠損の場合、式は真となり、結果がゼロまたは欠損の場合、式は偽となります。
- 相互に排他的な条件が長く続く場合は、一連の IF-THEN ステートメントではなく、SELECT WHEN ステートメントを使用します。

- IF 句に指定した条件に一致するオブザベーションまたはレコードのみの処理を続行するには、THEN 句を指定せずにサブセット化 IF ステートメントを使用します。

関連項目

- “IF ステートメント: サブセット化” (*SAS DATA ステップステートメント: リファレンス*)
- “IF-THEN/ELSE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “SELECT ステートメント” (*SAS DATA ステップステートメント: リファレンス*)

例: SELECT WHEN ステートメント

コード例

この例では、DATA ステップ SELECT ステートメントを使用して SAS データセットをサブセット化する方法を示します。

```
data shoes;
  set sashelp.shoes(where=(Region='Canada' or Region='Pacific' or Region='Asia'));
  keep Region Sales Product;
  select(Region);
    when('Canada') sales = sales * .10;
    when('Pacific') sales = sales * .09;
    when('Asia') sales = sales * .07;
  end;
run;
proc print data=shoes; run;
```

アウトプット 20.20　SELECT ステートメントを使用して SAS データセットから条件付きで行を選択する方法を示す PROC PRINT 出力の一部

Obs	Region	Product	Sales
1	Asia	Boot	$140
2	Asia	Men's Dress	$212
3	Asia	Sandal	$226
4	Asia	Slipper	$211
5	Asia	Women's Casual	$377
6	Asia	Boot	$4,250
7	Asia	Men's Casual	$823
8	Asia	Men's Dress	$8,143
9	Asia	Sandal	$348
10	Asia	Slipper	$10,431
11	Asia	Sport Shoe	$86
12	Asia	Women's Casual	$1,431
13	Asia	Women's Dress	$5,476
14	Asia	Sport Shoe	$81
15	Canada	Boot	$1,772
16	Canada	Men's Dress	$1,278
17	Canada	Sandal	$289
18	Canada	Slipper	$568
19	Canada	Sport Shoe	$975
20	Canada	Women's Dress	$1,260
21	Canada	Boot	$4,021
22	Cana	Men's Casual	$5,193

要点

- SELECT WHEN ステートメントを使用すると、単一のカテゴリ変数の値に基づいて、ステートメントのグループ内の 1 つ以上のステートメントを条件付きで処理できます。
- SELECT ステートメントと END ステートメントの間にあるステートメントのグループは、SELECT グループと呼ばれます。SELECT グループには、特定の条件が真の場合に実行される SAS ステートメントを識別する WHEN ステートメントが含まれます。
- SELECT ステートメントを指定する場合、END ステートメントと少なくとも 1 つの WHEN ステートメントが必要です。
- 相互に排他的な条件が長く続く場合は、SELECT グループを使用すると、一連の IF-THEN ステートメントを使用するよりも効率的です。
- 多数の条件を使用すると、CPU 時間が削減されるため、SELECT グループは IF-THEN/ELSE ステートメントよりも効率的になります。また、SELECT グループを使用すると、プログラムが読みやすく、デバッグしやすくなります。
- オプションの OTHERWISE ステートメントは、WHEN 条件が満たされない場合に実行するステートメントを指定します。たとえば、データに欠損データまたは無効なデータが含まれている場合などです。

- OTHERWISE ステートメントで使用される null ステートメントでは、SELECT WHEN ステートメントによって SAS がエラーメッセージを発行することが防止されます。

関連項目

- “SELECT ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “IF ステートメント: サブセット化” (*SAS DATA ステップステートメント: リファレンス*)
- “IF-THEN/ELSE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)

例: データセットオプション

コード例

次の例では、WHERE ステートメントで FIRSTOBS=および OBS=データセットオプションを使用して、条件付きで処理するデータのセグメントを指定する方法を示します。

この例では、DATA ステップによって、10 行と 2 つの変数(i と x)を含む Example という名前のデータセットが作成されます。

```
data example;
  do i=1 to 10;
    x=i + 1;
    output;
  end;
run;

proc print data=example; run;
```

アウトプット 20.21 サブセット化されるデータを示す PROC PRINT 出力

Obs	i	x
1	1	2
2	2	3
3	3	4
4	4	5
5	5	6
6	6	7
7	7	8
8	8	9
9	9	10
10	10	11

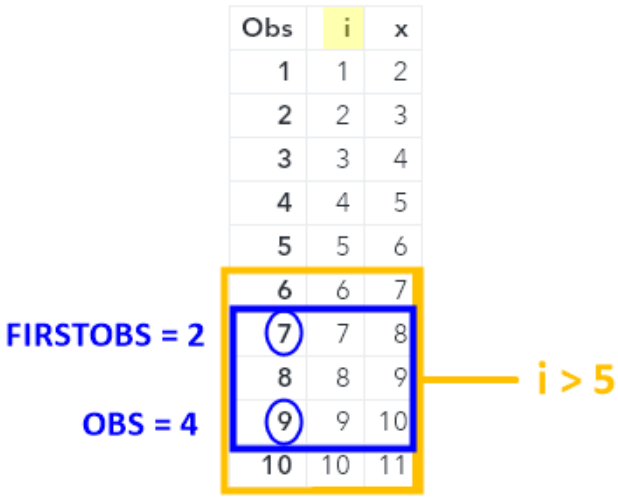
次の PROC ステップには、WHERE ステートメントとデータセットオプション FIRSTOBS=および OBS=という 2 つの別個のサブセット化アクションが含まれています。WHERE ステートメントは、最初に元の入力データセット example に対して処理され、次に 2 つのデータセットオプションが WHERE ステートメントの結果に対して処理されます。

WHERE **i > 5** は、元の入力データセットから i が 5 より大きい行のみを選択するように SAS に指示します。これは元のデータセットのサブセットで、元の入力データセットの行 6 から 10 までの 5 行のみが含まれます。

次に、SAS は結果の 5 行に対してデータセットオプション FIRSTOBS=2 および OBS=4 を処理し、WHERE 結果の 2 行目から 4 行目までを出力します。PRINT プロシジャは、元の入力データセットから行 7、8、および 9 を出力します。

```
proc print data=example (firstobs=2 obs=4);
  where i > 5;
run;
```

アウトプット 20.22 PROC PRINT は i>5 の行を選択し、そのサブセットの行 2 から 4 までを選択する



PROC PRINT の結果は、行 7、8、および 9 です。

アウトプット 20.23　行 7、8、および 9 を含むデータセットの例を示す PROC PRINT 出力

Obs	i	x
7	7	8
8	8	9
9	9	10

要点

- FIRSTOBS=は SAS データセット内で SAS が最初に処理するオブザベーションを指定し、OBS=は SAS がデータセット内で最後に処理するオブザベーションを指定します。
- DATA ステップで使用される場合、これらのオプションは入力(読み取り)処理でのみ有効です。つまり、SET ステートメントと INFILE ステートメントで有効です。
- これらのオプションは、CAS で実行されている DATA ステップでは有効ではありません。
- WHERE 式と一緒に使用する場合、OBS=および FIRSTOBS=に指定される値は、データセット内の物理的なオブザベーション番号ではなく、サブセット内の論理的な番号になります。たとえば、**obs=3** は、データセット内の 3 番目のオブザベーション番号を意味するものではありません。かわりに、WHERE 式によって選択されたデータのサブセット内の 3 番目のオブザベーションを意味します。

関連項目

- “FIRSTOBS= データセットオプション” (*SAS データセットオプション: リファレンス*)
- “OBS= データセットオプション” (*SAS データセットオプション: リファレンス*)

例: PROC CASUTIL

コード例

この例では、3 つの PROC CASUTIL ステートメントを使用します。LOAD ステートメントはデータセット sashelp.cars を CAS にロードし、その名前として cars を指定します。PARTITION ステートメントは、cars テーブルを読み取り、2 つの変数の値に基づいて行を条件付きで選択し、carsWhere という名前の新しい出力テーブルを作成します。ALTERTABLE ステートメントは、carsWhere テーブルを更新し、元のテーブルから 3 つの列のみが含まれるように指定します。

```
cas casauto sessopts=(caslib='casuser');
libname mylib cas;

proc casutil;
  load data=sashelp.cars
    casout='cars' replace;
  partition casdata='cars'
    casout='carsWhere' replace
    where='MSRP>90000 and Make="Porsche"';
  altertable casdata="carsWhere"
    keep={"make", "model", "MSRP"};
quit;
proc print data=mylib.carsWhere;
```

PRINT プロシジャでは、次の結果が生成されます。

Make	Model	MSRP
Porsche	911 GT2 2dr	$192,465

要点

- LOAD ステートメント(PROC CASUTIL)は、caslib のデータソース内のファイルまたはクライアント側のファイルからデータを読み取り、SAS Cloud Analytic Services のメモリにロードします。
- WHERE パラメーター(LOAD ステートメント)は、データからオブザベーションを選択するための条件を指定します。

関連項目

“CASUTIL プロシジャ” (*SAS Cloud Analytic Services: ユーザーガイド*)

21

データの結合

データ結合の定義

SAS データセットを結合するとは、複数の入力データセットからデータを読み取り、このセクションで概説する方法の 1 つを使用してそれらを結合することを意味します。SAS データセットを読み取る方法については、“SAS データセットの読み取りと作成の方法” (301 ページ)を参照してください。

次の用語はこの章全体で使用されており、ここではデータ結合のコンテキストで定義されています。

common variable
: 結合されるすべての入力データセットに存在する変数。共通変数を BY 変数として指定できますが、必ずしもそうする必要はありません。変数は、名前に大文字または小文字が使用されているかどうかに関係なく、同じ名前を持つ場合、同じ、または複数のデータセットに共通であると認識されます。たとえば、SAS は、あるデータセットの変数 name を別のデータセットの変数 NAME と同一であると見なします。共通変数は、名前が同じであるだけでなく、データ型も同じである必要があります。詳細については、“例: 共通変数が異なるデータ型を持つデータの準備” (541 ページ)を参照してください。

BY variable
: 複数のデータセットを 1 つに結合するときに、その値が行をどのようにマージするかの基礎として機能する変数(複数可)。BY 変数は、SAS の BY ステートメント

で指定されます。BY 変数は、データを結合するコンテキストでは常に*共通変数*です。BY 変数は、データを結合するときに MERGE、UPDATE、SET、MODIFY のいずれかのステートメントの BY ステートメントで指定される変数です。詳細については、“例: 結合するデータを検査する” (536 ページ)および“例: データに重複 BY 値が含まれる場合に一意の BY 値を作成する” (539 ページ)を参照してください。

BY value
BY 変数が特定の行またはオブザベーションに持つ値。

BY group
BY ステートメントで指定された変数に対して同じ値を持つ行のグループ。BY ステートメントで複数の変数が指定されている場合、BY グループは、それらの変数の値の一意の組み合わせを持つ行のグループです。

data relationships
1 対 1、1 対多(または多対 1)、および多対多の 3 つの方法のいずれかで存在するデータセットまたはテーブル間の関連付け。データセットは、物理レベルまたは論理レベルで共通データによって関連付けられます。たとえば、ビジネスデータベースでは、共通の値を共有する Employee_ID 変数を通じて従業員データと部門データを関連付けることができます。別のデータセットには、部分値が行番号によって別のデータセットに論理的に関連付けられている数値シーケンスが含まれる場合があります。データリレーションシップと、それらがデータの結合にどのように関係するかの詳細については、“データリレーションシップ”を参照してください。

matched rows
値が選択された基準に一致する場合に、出力用に選択される入力データセット内の行。一致基準は、行番号(またはデータセット内の位置)に基づくことも、ユーザーが BY ステートメントで指定した変数の値に基づくこともできます。

unmatched rows
値が指定された基準に一致しないため、出力用に選択されなかった入力データセット内の行。一致基準は、行番号(またはデータセット内の位置)に基づくことも、BY ステートメントで指定する変数の値に基づくこともできます。

SAS データセットを結合する方法の概要

表 21.1 データセットを結合する方法の概要

種類	方法	ステートメントまたはプロシジャ	簡略化されたコード例
連結	SET ステートメントを使用した連結	SET ステートメント	data concat; set data1 data2; run;
	PROC SQL を使用した連結	SQL プロシジャ	proc sql; select * from data1 union all corresponding

種類	方法	ステートメントまたはプロシジャ	簡略化されたコード例
			select * from data2; quit;
	PROC APPEND を使用した追加	APPEND プロシジャ	proc append base=data1 data=data2; run;
インターリーブ	インターリーブ	SET ステートメントと BY ステートメント	data interleav; set data1 data2; by year; run;
マージ	マッチマージ	MERGE ステートメントと BY ステートメント	data matchmerged; merge data1 data2; by year; run;
	1 対 1 マージ	MERGE ステートメント	data merged; merge data1 data2; run;
	1 対 1 読み取り	SET ステートメント	data merged; set data1; set data2; run;

	ハッシュテーブルマージ	HASH オブジェクトと FIND メソッド	declare hash h(dataset:'data1');
			rc = h.definekey('key');
			rc = h.definedata('data1');
			h.definedone();
			set data2;
			h.find();
			run;

	SQL プロシジャ	PROC SQL ステートメント	proc sql; select * from data1, data2; by year; where data1.key=data2.key; run;
更新	更新	UPDATE ステートメントと BY ステートメント	data master; update master trans; by year; run;
	変更	MODIFY ステートメントと BY ステートメント	data master; modify master trans; by year; run;

	ハッシュテーブルの更新	HASH オブジェクトと REPLACE メソッド	declare hash h(dataset:'data1');
			rc = h.definekey('key');
			rc = h.definedata('data1');
			h.definedone();
			set data2;
			rc = h.find();
			if rc = 0 then h.replace();
			run;

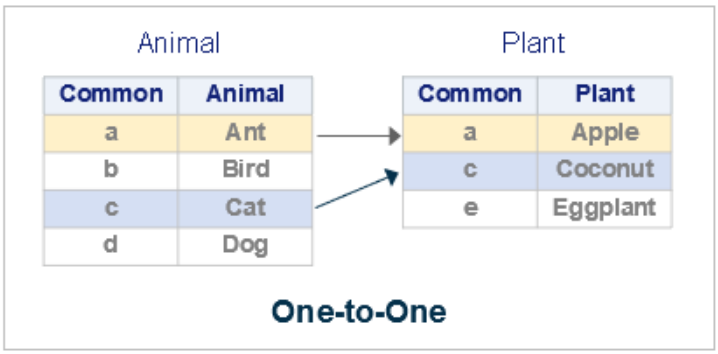
データの結合

データリレーションシップ

次の 4 つのカテゴリは、データセット間での行の関係によって特徴付けられます。すべての関連データは、これらのカテゴリのいずれかに分類されます。データ内の既存のリレーションシップを特定できる必要があります。これは、入力データがどのように処理されて望ましい結果が得られるかを理解するために不可欠な知識であるためです。

1 対 1 リレーションシップ

1 対 1 リレーションシップでは、1 つのデータセットの 1 行は 2 番目のデータセットの 1 行のみに関連付けられ、2 番目のデータセットの 1 行は最初のデータセットの 1 行のみに関連付けられます。このリレーションシップは、選択した 1 つ以上の変数の値に基づいています。1 対 1 リレーションシップは、各データセット内に選択した変数の重複値が存在しないことを意味します。選択した複数の変数を操作する場合、このリレーションシップは、値の各組み合わせが各データセット内で 1 回しか出現しないことを意味します。

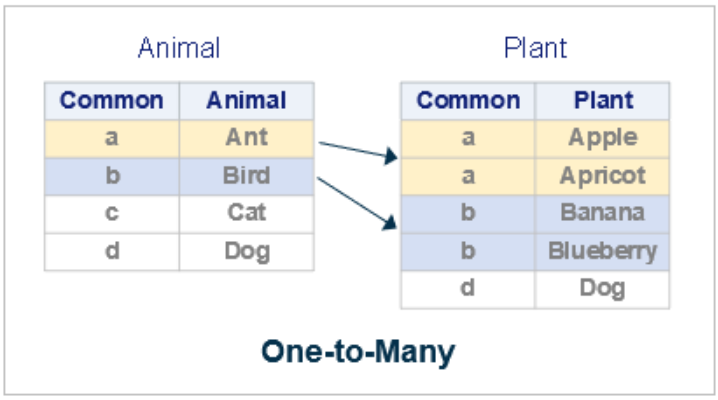


図では、データセット Animal と Plant の行は、変数 Common の一致する値によって関連付けられます。変数 Common の値は、両方のデータセットで一意です。

1 対多リレーションシップ

入力データセット間の 1 対多リレーションシップは、一方のデータセットには、選択された変数の特定の値を含む行が最大 1 行しか含まれないが、もう一方の入力データセットには各値の複数または重複が含まれる可能性があることを意味します。

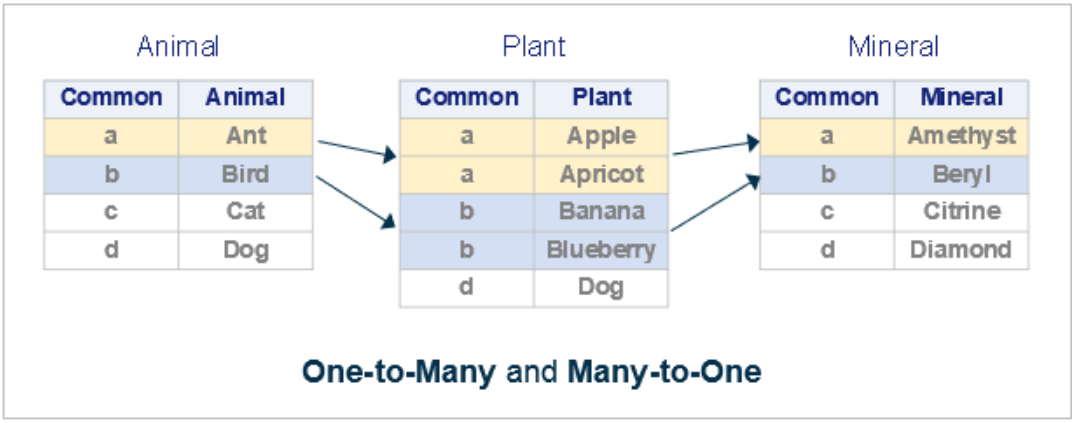
選択した複数の変数を操作する場合、このリレーションシップは、値の各組み合わせが 1 つのデータセット内で 1 回しか出現しないことを意味します。この組み合わせは、他のデータセットで複数回出現する可能性があります。入力データセットが処理される順序によって、リレーションシップが 1 対多であるか多対 1 であるかが決まります。



図では、データセット Animal と Plant の行は、変数 Common の共通の値によって関連付けられます。変数 Common の値は、データセット Animal では一意ですが、データセット Plant では一意ではありません。

1 対多および多対 1 のリレーションシップ

入力データセット間の 1 対多または多対多のリレーションシップは、一方のデータセットには、選択された変数の特定の値を含む行が最大 1 行しか含まれないが、もう一方の入力データセットには各値が複数回出現する可能性があることを意味します。選択した複数の変数を操作する場合、このリレーションシップは、値の各組み合わせが 1 つのデータセット内で 1 回しか出現しないことを意味します。ただし、この組み合わせは、他のデータセットで複数回出現する可能性があります。入力データセットが処理される順序によって、リレーションシップが 1 対多であるか多対 1 であるかが決まります。

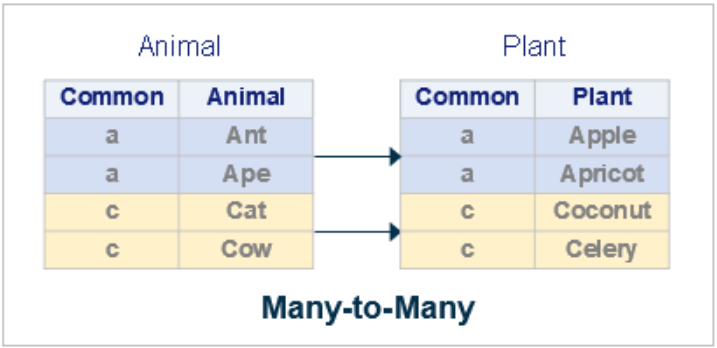


図では、データセット Animal、Plant、および Mineral の行は、変数 Common の共通の値によって関連付けられています。Common の値は、データセット Animal および Mineral では一意ですが、Plant では一意ではありません。データセット Animal と

Plant の行の間には 1 対多リレーションシップが存在し、データセット Plant と Mineral の行の間には多対一リレーションシップが存在します。

多対多リレーションシップ

多対多のカテゴリは、各入力データセットの複数の行を 1 つ以上の共通変数の値に基づいて関連付けることができることを意味します。



図では、データセット Animal と Plant の行は、変数 Common の共通の値によって関連付けられます。変数 Common の値は、どちらのデータセットでも一意ではありません。これらのデータセット内の値 **a** と **c** の行間には、多対多リレーションシップが存在します。

連結

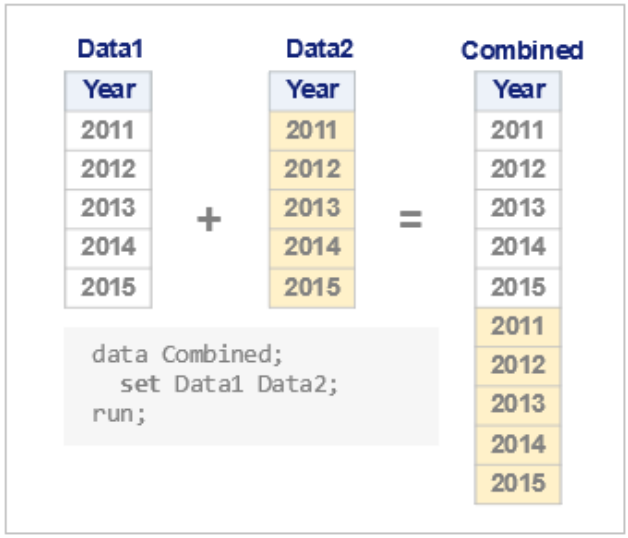
定義

連結では、2 つ以上の SAS データセットを 1 つの新しい出力データセットに次々と結合します。

構文

SET<*data-set-1*> <*data-set-2*><... *data-set-n*>;

2 つのデータセットの連結



この図では、Data2 の行が Data1 の行に追加されています。

詳細

- SET ステートメントを使用して連結すると、入力データセットが SET ステートメントにリストされている順序で順次処理されます。
- 連結では、入力データセットに同じ変数が含まれている必要はありません。
- 連結すると、すべての入力データセットのすべての行とすべての 入力データセットのすべての列を含む出力が作成されます。
- 連結すると、すべての入力データセットに存在しない変数の値が欠損に設定されます。

関連項目

- “例: データの連結” (550 ページ)
- “SAS データセットの連結” (*SAS DATA ステップステートメント: リファレンス*)

インターリーブ

定義

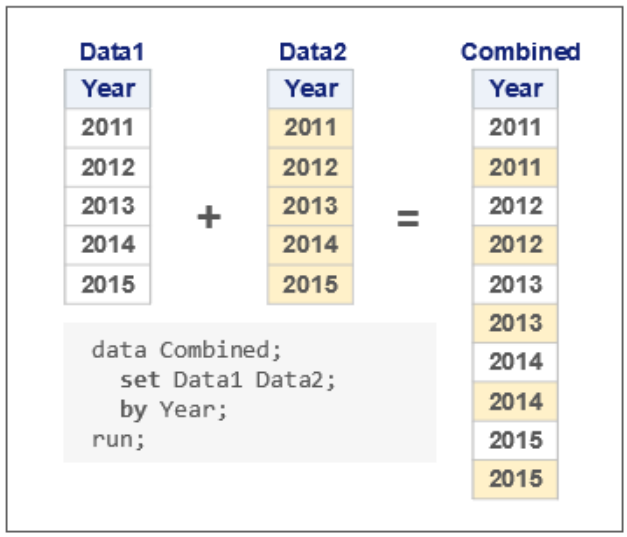
インターリーブでは、共通の BY 変数の値に基づいて行を相互に散在させることにより、2 つ以上の SAS データセットを 1 つのデータセットに結合します。一部のプログラミング言語では、マージという用語は行をインターリーブすることを意味します。ただし、SAS データセット内でインターリーブされている行はマージされません。これらは、元のデータセットから BY 変数の値の順序でコピーされます。

構文

SET<*data-set-1*> <*data-set-2*><...*data-set-n*>;

BY *variable-1* <...*variable-n*>;

2 つのデータセットのインターリーブ



この図では、Data2 の行が Data1 の行の間に散在しています。

詳細

- インターリーブには、行を一致させる 1 つ以上の共通変数を指定する BY ステートメントとともに SET ステートメントが必要です。
- 入力データセットは、BY 変数に基づいてインデックス付けされるか、並べ替えられる必要があります。
- インターリーブでは、元の入力データセット内の位置によって各 BY グループ内で順序付けされた行が返されます。

関連項目

- “例: データのインターリーブ” (558 ページ)
- “SAS データセットのインタリーブ” (*SAS DATA ステップステートメント: リファレンス*)

マッチマージ

定義

マッチマージは、BY 変数の値に基づいて、2 つ以上の入力データセットの行を出力データセットの 1 つの行に結合します。

構文

MERGE *data-set-1* <*data-set-2*><...*data-set-n*>;

BY *variable-1* <...*variable-n*>;

2 つのデータセットのマッチマージ

Data1

Year	X
2011	x1
2012	x2
2013	x3
2014	x4
2015	x5

+

Data2

Year	Y
2011	y1
2012	y2
2013	y3
2014	y4
2015	y5

=

Combined

Year	X	Y
2011	x1	y1
2012	x2	y2
2013	x3	y3
2014	x4	y4
2015	x5	y5

```
data Combined;
 merge Data1 Data2;
 by Year;
run;
```

この図では、BY 変数 Year の値に基づいて、Data2 の行が Data1 の行とマージされています。

詳細

- マッチマージには、行を一致させる 1 つ以上の共通変数を指定する BY ステートメントとともに MERGE ステートメントが必要です。
- マッチマージでは、入力データセットにインデックスが含まれているか、BY 変数の値によって並べ替えられていることが必要です。
- BY ステートメントの変数は、すべての入力データセットに共通である必要があります。
- DATA ステップ内の各 MERGE ステートメントに付随できる BY ステートメントは 1 つだけです。
- MERGE、UPDATE、および MODIFY ステートメントを使用して、SAS データセット内の行を結合および更新できます。これらの方法の比較については、“更新と変更の比較” (529 ページ)を参照してください。

関連項目

- “例: データのマッチマージ” (565 ページ)
- “マッチマージ” (*SAS DATA ステップステートメント: リファレンス*)

1 対 1 マージ

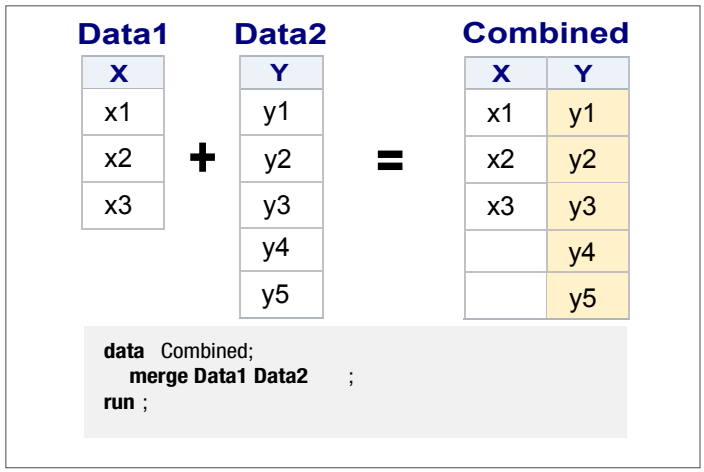
定義

データを 1 対 1 でマージすると、複数の入力データセットの行が新しい出力データセットの 1 つの行に結合されます。行は、入力データセット内の位置に基づいて結合されます。最初の入力データセットの最初の行は、2 番目の入力データセットの最初の行と結合され、以下同様に続きます。

構文

MERGE *data-set-1* <*data-set-2*><...*data-set-n*>;

2 つのデータセットの 1 対 1 マージ



この図では、Data2 の行が行番号に基づいて Data1 の行とマージされています。

詳細

- 1 対 1 マージには、BY ステートメントを使用しない MERGE ステートメントが必要です。マージの基礎となるキー変数はありません。かわりに、行は行番号によって暗黙的にマージされます。
- 入力データセットに共通変数が含まれている場合、SAS は最初に読み取られたデータセットの値を最後に読み取られたデータセットの値に置き換えます。
- SAS は、すべての入力データセットからすべての行を読み取るまで、行の読み取りを停止しません。これと比較すると、1 対 1 読み取りでは、SAS は最小の入力データセットから最後の行を読み取ったときに行の読み取りを停止します。
- 出力データセットには、すべての入力データセットのすべての変数が含まれます。出力データセットの行数は、最大の入力データセットの行数と等しくなります。

関連項目

- “例: データの 1 対 1 マージ” (582 ページ)
- “1 対 1 のマージ” (*SAS DATA ステップステートメント: リファレンス*)
- 例 1: 1 対 1 マージ
- “1 対 1 読み取りと 1 対 1 マージ” (531 ページ)

1 対 1 読み取り

定義

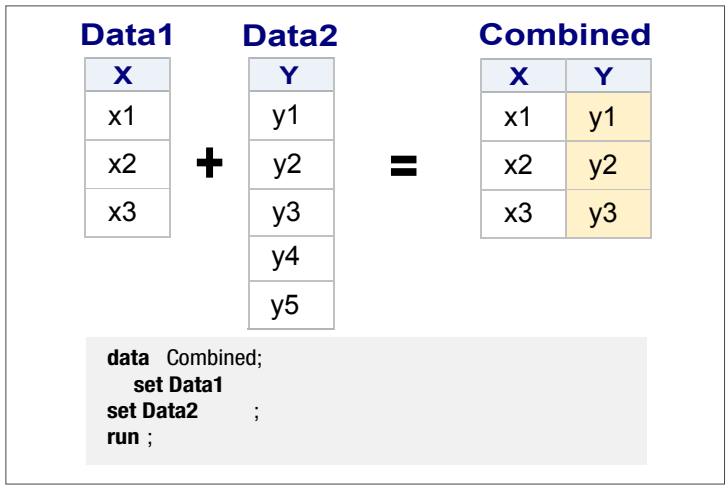
1 対 1 読み取りでは、複数の入力データセットの行が新しいデータセットの 1 つの行に結合されます。行は、データセット内の位置に基づいて一致させます。最初のデータセットの最初の行は、2 番目のデータセットの最初の行と結合され、以下同様に続きます。

構文

SET<*data-set-1*>;

SET <*data-set-2*>;

2 つのデータセットの 1 対 1 読み取り



この図では、Data2 の行が行番号に基づいて Data1 の行とマージされています。

詳細

- 1 対 1 読み取りには、BY ステートメントを使用しない複数の SET ステートメントが必要です。データセットをマージするためのキー BY 変数がありません。1 対 1 マージと同様に、行は BY 変数の値ではなく行番号によって暗黙的にマージされます。
- 入力データセットに共通変数が含まれている場合、SAS は、最初に読み取られたデータセットの共通変数の値を、最後に読み取られたデータセットの値に置き換えます。行をマージするためのキー BY 変数がないため、行番号によって暗黙的に結合されます。
- SAS は、*最小*の入力データセットから最後の行を読み取った後、入力データセットからの行の読み取りを停止します。これと比較すると、1 対 1 マージでは、SAS はすべての入力データセットからすべての行を読み取るまで読み取りを停止しません。
- 出力データセットには、すべての入力データセットのすべての変数が含まれます。出力データセットの行数は、最小の入力データセットの行数と等しくなります。

関連項目

- “例: データの 1 対 1 結合” (590 ページ)
- “マスターファイルに重複するオブザベーションが含まれる場合、テーブルルックアップを実行する” (*SAS DATA ステップステートメント: リファレンス*)
- “テーブルルックアップの実行” (*SAS DATA ステップステートメント: リファレンス*)

■ “1 対 1 読み取りと 1 対 1 マージ” (531 ページ)

ハッシュテーブルマージ

定義

DATA ステップハッシュオブジェクトは、ルックアップキーに基づいてデータを効率的に保存、検索、取得できるようにする一時的なインメモリデータ構造です。ハッシュオブジェクトは DECLARE ステートメントによって初期化されます。FIND、ADD、OUTPUT などのメソッドは、ハッシュオブジェクトを操作します。ハッシュテーブルを使用したインメモリ処理は、他の DATA ステップ方法よりも高速です。

構文

DECLAREobjecthash-table-name

```
data hashmerge(keep=Name Sex Age);
 if 0 then set sashelp.class;
 declare hash hmerge(dataset:'sashelp.class', hashexp:6);
   rc1 = hmerge.defineKey('Name');
   rc2 = hmerge.defineData('Sex', 'Age');
   rc3 = hmerge.defineDone();
    do until(done);
      set sashelp.classfit end=done;
      rc4 = hmerge.find();
      if rc4 = 0 then output hashmerge;
     end;
 stop;
run;
```

この例では、ハッシュオブジェクトは、Sashelp.Class データセットからの Name、Sex、および Age 変数とともにロードされます。Sashelp.Classfit データセットの変数 Name は、Sashelp.Class ハッシュオブジェクト内で一致するアイテムを見つけるためのルックアップキーとして使用されます。

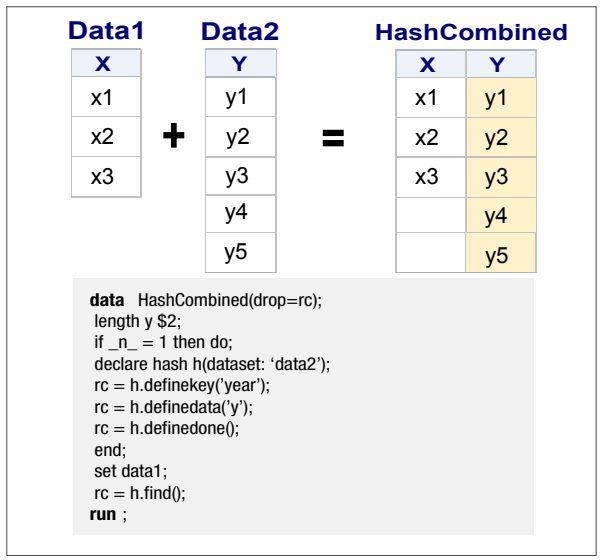
定義

DATA ステップハッシュオブジェクトは、ルックアップキーに基づいてデータを効率的に保存、検索、取得できるようにする一時的なインメモリデータ構造です。ハッシュオブジェクトは、DATA ステップの間のみ使用できます。ハッシュオブジェクトは DECLARE ステートメントによって初期化されます。FIND、ADD、OUTPUT などのメソッドは、ハッシュオブジェクトを操作します。ハッシュオブジェクトの内容は、ハッシュ OUTPUT メソッドを使用してデータテーブルに書き込むことができます。ハッシュテーブルを使用したインメモリ処理は、通常、他の DATA ステップ方法よりも高速です。

構文

DECLAREobjecthash-table-name

2 つのデータセットのハッシュテーブルマージ



この例では、DATA ステートメントによって書き込まれたデータセットには、読み取られているデータセットからのオブザベーションと、キー変数で一致したオブザベーションの DEFINEDATA メソッドで列挙されたデータ変数が含まれています。

詳細

- ハッシュオブジェクトは、DATA ステップの間のみ使用できます。
- OUTPUT メソッドを使用して、ハッシュオブジェクトの内容をデータテーブルに出力できます。
- DATA ステートメントによって書き込まれるデータセットには、読み取られているデータセットからのオブザベーションと、キー変数に一致するオブザベーションに対して DEFINEDATA メソッドで指定されたデータ変数が含まれます。
- ハッシュテーブルを使用してデータセットをマージする場合、並べ替えは必要ありません。
- ハッシュテーブルをメモリ内に保持できる限り、ハッシュテーブルマージにより、大きなテーブルの高速処理が可能になります。
- 両方のデータセットのキーに同じ変数名を使用する必要はありません。一致はさまざまな変数に対して実行できます。

関連項目

- “例: ハッシュテーブルを使用したデータのマージ” (592 ページ)
- *SAS コンポーネントオブジェクト: リファレンス*

更新

定義

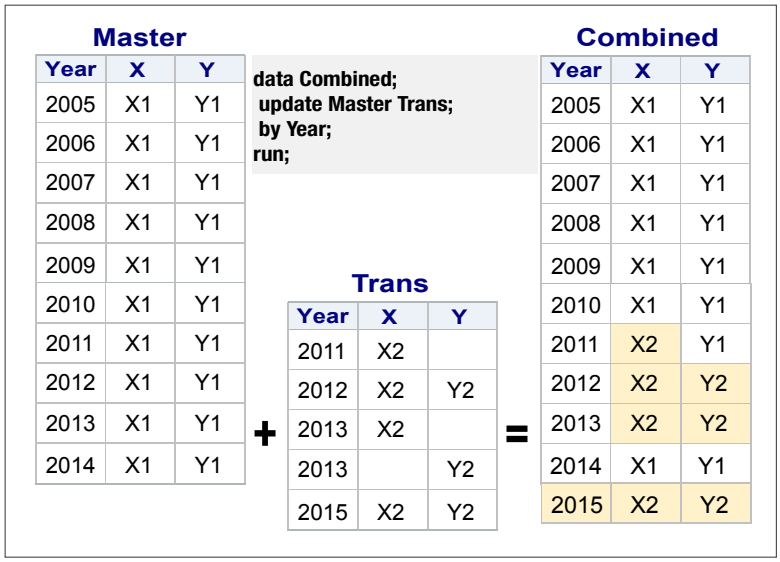
UPDATE ステートメントは、*トランザクションデータセット*の行を使用して、*マスターデータセット*内の対応する行の値を変更することにより、新しい更新されたデータセットを作成します。マスターデータセットには元の情報が含まれ、トランザクションデータセットにはマスターデータセットに適用する必要がある新しい情報が含まれます。

構文

UPDATE*master-data-set transaction-data-set* <(*data-set-options*)>

BY *by-variable*;

トランザクションデータセットからのマスターデータセットの更新



この図では、Trans データセットの値に基づいて Master データセットの行を更新することにより、新しい出力データセット Combined が作成されます。

詳細

- UPDATE ステートメントには、行を一致させる 1 つ以上の共通変数を指定する BY ステートメントが必要です。
- 入力データセットは、BY 変数に基づいてインデックス付けされるか、並べ替えられる必要があります。
- BY 変数の値は、マスターデータセット内のオブザベーションごとに一意である必要があります。マスターデータセットに BY 変数の同じ値を持つ複数の行が含まれている場合、最初のオブザベーションが更新され、BY グループ内の残りのオブザベーションは無視されます。SAS は、DATA ステップの実行時に警告メッセージをログに書き込みます。
- トランザクションデータセットには、同じ BY 値を持つ複数のオブザベーションを含めることができます。複数のトランザクション行はすべて、出力ファイルに書き込まれる前にマスターオブザベーションに適用されます。
- 更新すると、新しい出力データセットが作成されます。変更とは異なり、既存のマスター入力データセットは更新されません。
- 通常、マスターデータセットとトランザクションデータセットには同じ変数が含まれます。ただし、処理時間を短縮するために、更新される変数のみを含むトランザクションデータセットを作成できます。トランザクションデータセットには、出力データセットに追加される新しい変数を含めることもできます。
- MERGE、UPDATE、および MODIFY ステートメントを使用して、SAS データセット内の行を結合および更新できます。これらの方法の比較については、“更新と変更の比較” (529 ページ)を参照してください。

関連項目

- “例: UPDATE ステートメントを使用したデータの更新” (596 ページ)
- 表 21.2 (529 ページ)
- PROC SQL を使用した別のテーブルの値によるテーブルの更新

変更

定義

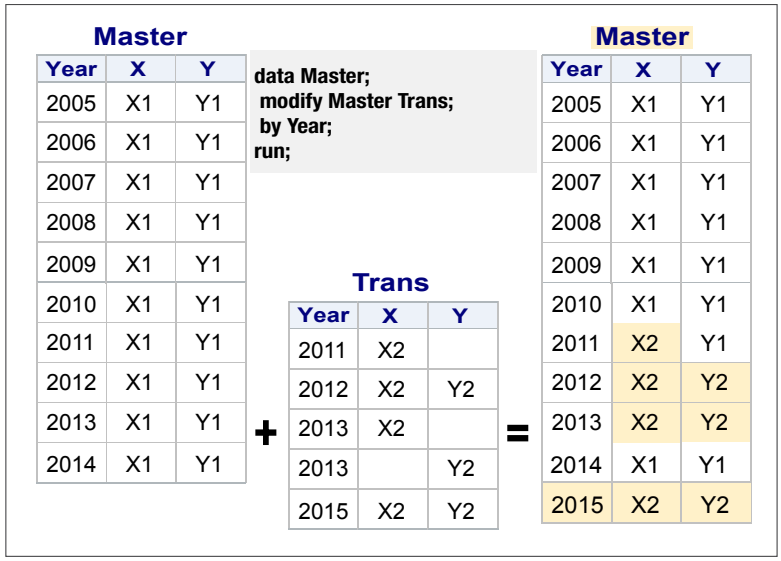
MODIFY ステートメントは、*マスター*データセット内の 1 つ以上の BY 変数の値を、*トランザクション*データセット内の同じ変数と一致させます。既存のマスターデータセットが変更されます。新しいデータセットは作成されません。

トランザクションデータセットからのマスターデータセットの変更

構文

MODIFY *master-data-set transaction-data-set* <(*data-set-options*)>

BY *by-variable*;



この図では、Master データセットの行は、Trans データセットで指定された BY 変数 Year の値に基づいて変更されます。

詳細

- MODIFY ステートメントでは、入力データセットが BY ステートメントで指定された共通変数を共有する必要があります。
- 変更には、入力データセットを並べ替えたりインデックスを付けたりする必要はありません。
- マスターデータセットとトランザクションデータセットの両方に、BY 変数の値が重複する行を含めることができます。
- BY 変数の重複値がマスターデータセットに存在する場合、最初に出現した値のみが更新されます。
- トランザクションデータセットに BY 変数の重複値が存在する場合、最後のトランザクションの値のみがマスターオブザベーションに表示されるように、重複は重ねて適用されます。
- 変更しても、新しい変数を作成したり、変数の属性を変更したりすることはできません。
- MERGE、UPDATE、および MODIFY ステートメントはすべて、SAS データセット内の行を結合および更新できます。これらの方法の比較については、“更新と変更の比較” (529 ページ)を参照してください。

関連項目

- “例: オブザベーションを追加してデータセットを変更する” (603 ページ)
- “更新と変更の比較” (529 ページ)

方法の比較

PROC SQL とマッチマージ

DATA ステップで MERGE ステートメントと BY ステートメントを使用するか、SQL プロシジャを使用することによって、テーブルを作成および変更できます。DATA ステップのマッチマージと PROC SQL の詳細については、“テーブルとビューの作成と更新” (*SAS SQL プロシジャユーザーガイド*)を参照してください。

マージされるデータセット間で共通変数の値が異なる場合、それらは"不一致行"または"不一致レコード"と呼ばれることがよくあります。欠損値も不一致行になります。

DATA ステップと PROC SQL では、不一致行の処理が異なります。

PROC SQL 内部結合:

```
proc sql number;
  select *
  from inventory, Sales
  where inventory.PartNumber=
      Sales.PartNumber;
quit;
```

デフォルトでは、一致行のみが出力に含まれます。

DATA ステップのマッチマージ:

```
data merged;
  merge Inventory Sales; by PartNumber;
run;
```

デフォルトでは、一致行と不一致行の両方が出力に含まれます。

SQL Inner Join Inventory and Sales

Row	PartNumber	PartName	PartNumber	PartName	SalesPerson
1	BV1E	timer	BV1E	timer	JN
2	BV1E	timer	BV1E	timer	KM
3	K89R	seal	K89R	seal	SJ
4	LK43	filter	LK43	filter	KM
5	LK43	filter	LK43	filter	JN
6	M4J7	sander	M4J7	sander	KM
7	NCF3	valve	NCF3	valve	JN

Only matched observations are included

DATA Step Match-Merge Inventory and Sales

Obs	PartNumber	PartName	SalesPerson
1	BC85	clamp	
2	BV1E	timer	JN
3	BV1E	timer	KM
4	JD03	switch	
5	K89R	seal	SJ
6	KJ66	cutter	
7	LK43	filter	KM
8	LK43	filter	JN
9	M4J7	sander	KM
10	MN21	brace	
11	NCF3	valve	JN
12	UYN7	rod	

Matched and unmatched observations are included

マッチマージ内の黄色のハイライトは、SQL 出力に含まれない不一致行を示します。DATA ステップと PROC SQL を比較する別の例については、“PROC SQL と SAS DATA ステップとの比較” (*SAS SQL プロシジャユーザーガイド*)を参照してください。

更新と変更の比較

表 21.2　テーブルの更新とテーブルの変更の比較

特性	更新	変更
構文	UPDATE ステートメント data <*output-data-set*>; UPDATE *master-data-set*; *trans-data-set* BY *variable-name*;	形式 1: MODIFY ステートメント一致アクセス) data <*output-data-set*>; MODIFY *master-data-set*; *trans-data-set* BY *variable-name*;
いつ使用するか	■ トランザクションデータセットの値に基づいて変更または更新する必要があるマスターデータセットがあります。 ■ マスターデータセット内のほとんどのデータを処理したいと考えています。	■ 追加の新しい出力データセットを作成せずに、別のデータセットの値に基づいてマスターデータセットを変更したいと考えています。MODIFY は、元のマスターデータセットの更新バージョンを作成します。 ■ マスターデータセットのごく一部のみを処理または変更したいと考えています。
処理できるデータセットの数	UPDATE および MODIFY を BY ステートメントとともに指定すると、正確に 2 つのデータセットの行を結合できます。	UPDATE および MODIFY を BY ステートメントとともに指定すると、正確に 2 つのデータセットの行を結合できます。
ディスク領域	“更新”では、データセットの更新されたコピーが生成されるため、より多くのディスク領域が必要になります。	“変更”では、新しいテーブルを作成せずに既存のテーブルを更新するため、ディスク領域が節約されます。
BY ステートメントの要件	BY ステートメントが必要です。	入力データセットが 1 つ以上の共通変数によってなんらかの方法で関連付けられている場合、MODIFY ステートメントを使用して別のトランザクションデータセットの値に基づいてマスターデータを更新する場合、BY ステートメントが必要です。*
一意の BY またはキー値の要件	■ トランザクションデータセットでは、BY 変数の重複値が許可されます。複数のトランザクション行はすべて、出力ファイルに書き込まれる前にマスターオブザベーションに適用されます。 ■ マスターデータセットでは、BY 変数の一意の値が必要です。BY 変数の重複値がマスターデータセットに存在する場合、BY グループ内の	■ BY 変数の重複値は、マスターデータセットまたはトランザクションデータセットのいずれかで許可されます。 ■ マスターデータセット内に重複が存在する場合、生成された WHERE ステートメントは常にマスターデータセット内の最初の出現箇所を検索するため、最初の出現箇所のみが更新されます。

特性	更新	変更
	その値を持つ最初のオブザベーションのみが更新され、SAS は警告を発行します。	■ トランザクションデータセットに重複が存在する場合、重複をすべてマスターオブザベーションに追加する累積ステートメントを作成しない限り、重複は重ねて適用されます。累積ステートメントを使用しないと、重複内の値が相互に上書きされるため、最後のトランザクションの値のみがマスターオブザベーションの結果となります。
並べ替えまたはインデックスの要件	■ 入力データセットには並べ替えまたはインデックス付けが必要です。 ■ 入力データセットにインデックスが存在する場合、UPDATE ステートメントは更新されたデータセット上でインデックスを維持しません。インデックスを再構築する必要があります。	■ どのデータセットにも並べ替えやインデックス付けは必要ありません。 ■ BY ステートメントを MODIFY ステートメントとともに使用すると、動的な WHERE 処理がトリガーされるため、マスターデータセットもトランザクションデータセットも並べ替えやインデックス付けは必要ありません。 ■ 入力データセットにインデックスが存在する場合、MODIFY ステートメントは更新されたデータセット上でインデックスを維持します。
入力データセットまたはマスターデータセットの欠損値の処理	■ デフォルトでは、MODIFY および UPDATE は、マスターデータセットの値をトランザクションデータセットの欠損値で上書きしません。 ■ トランザクションデータセットの欠損値によってマスターデータセットの既存の値が置き換えられるようにするには、UPDATEMODE=オプションで NOMISSINGCHECK を指定します。例については、例: 欠損値を含むデータセットを更新する (600 ページ)を参照してください。	
変数への変更許可	はい。 新しい変数の追加、変数の削除、変数の変更、およびデータセットのディスクリプター情報に影響を与えるその他の変更を実行できます。	いいえ。 新しい変数の追加、変数の削除、変数の変更、またはデータセットのディスクリプター情報に影響を与える変更を実行することはできません。
エラーチェック機能	いいえ。 対応するマスターレコードのないトランザクションはデフォルトでデータセットに追加されるため、更新時にエラーチェックは必要ありません。	はい。 エラーチェック機能には、自動変数_IORC_および SYSRC 自動呼び出しマクロの使用が含まれます。詳細については、“自動変数_IORC_と SYSRC 自動呼び出しマクロ” (*SAS DATA ステップステートメント: リファレンス*)を参照してください。
データセットの一貫性	UPDATE ステートメントはデータのコピーに対して機能するため、データの損失は発生しません。	タスクの異常終了により、データが部分的にしか更新されない可能性があります。

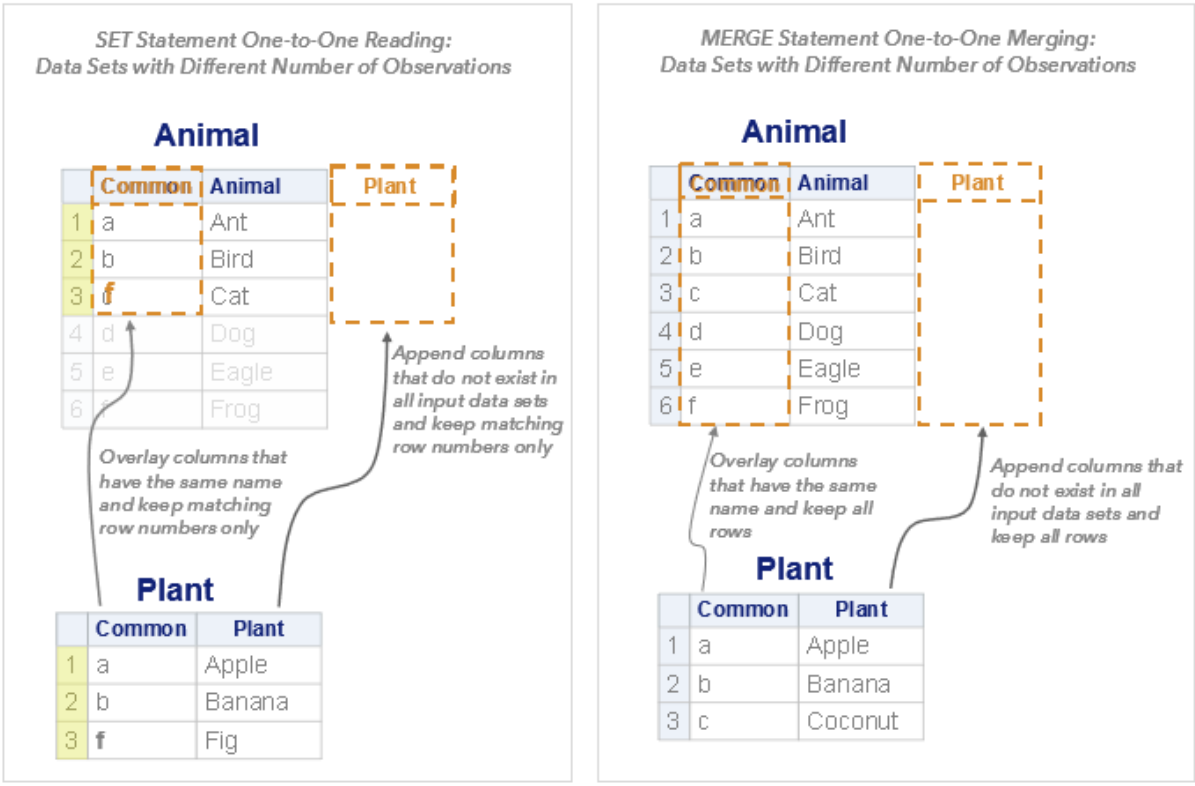
* UPDATE ステートメントとは異なり、MODIFY ステートメントは他の形式で使用して、単一の入力データセットを変更できます。このような場合、構文形式 2 から 4 および BY ステートメントは必要ありません。ただし、このトピック(データの結合)の目的では、MODIFY ステートメントには形式 1 バージョンの構文が必要ですが、これには BY ステートメントが必要です。

1 対 1 読み取りと 1 対 1 マージ

表 21.3　1 対 1 読み取りと 1 対 1 マージ

特性	1 対 1 読み取り	1 対 1 マージ
構文	SET ステートメント(DATA ステップ) data <*output-data-set*>; SET *input-data-set-1*; SET *input-data-set-2*;	MERGE ステートメント(DATA ステップ) data <*output-data-set*>; MERGE *input-data-set-1* *input-data-set-2*;
行の一致方法	1 対 1 読み取りと 1 対 1 マージの両方で、行は行番号によって一致させます。たとえば、データセット 1 の行 1 はデータセット 2 の行 1 と一致させ、データセット 1 の行 2 はデータセット 2 の行 2 と一致させます。	
“unmatched rows”の処理	一致行のみが出力用に選択されます。つまり、両方のデータセットで同じ行番号値を持つ行のみが、マージされた出力に含まれます。これは、異なる行数を持つデータセットをマージすると、最小の入力データセットと同じ行数を持つマージされたデータセットが生成されることを意味します。	一致行と不一致行の両方が出力用に選択されます。つまり、入力データセットの行数が異なる場合でも、すべてのデータセットのすべての行が出力に含まれます。
共通変数の処理	共通変数の値は上書きされます。つまり、SET または MERGE ステートメントで指定された最後のデータセットの共通変数の値は、前のデータセットの値を上書きします。これは、データセットの 1 対 1 読み取りと 1 対 1 マージの両方に当てはまります。	
並べ替え要件	なし。これらの方法のいずれにも並べ替え要件はありません。	

図 21.1　1 対 1 読み取りと 1 対 1 マージ



データセットを結合するための SAS 言語要素の比較

表 21.4　データセットを結合するための SAS 言語要素の比較

言語要素	目的	アクセス方式		BY ステートメントと一緒に使用
		順次	直接	
BY	■ BY ステートメントは、MERGE、MODIFY、SET、UPDATE ステートメントの操作を制御し、グループの作成と順序付けを行います。 ■ BY ステートメントを使用すると、等しい値を持つ変数を含む行を処理できます。	NA	NA	NA

言語要素	目的	アクセス方式		BY ステートメントと一緒に使用
		順次	直接	
MERGE ステートメント	■ MERGE ステートメントは、2 つ以上のデータセットから行を読み取り、1 つの行に結合します。 ■ BY ステートメントとともに使用される場合、このタイプのマージはマッチマージと呼ばれます。 ■ BY ステートメントなしで使用される場合、このタイプのマージは 1 対 1 マージと呼ばれます。 ■ MERGE ステートメントを BY ステートメントとともに使用する場合は、BY 変数の値に基づいてデータを並べ替えるか、インデックスを付ける必要があります。	X		X
MODIFY	■ 変更では、(新しいデータセットを作成する更新とは対照的に)新しいデータセットを作成するのではなく、既存のマスターデータセットを更新することによって、SAS データセット内の行を処理します。 ■ 並べ替えは必須ではありませんが、パフォーマンスのために推奨されます。	X	X	X
SET	■ SET ステートメントは、1 つ以上の SAS データセットから任意の行を読み取ります。 ■ データセット行のインターリーブは、単一の SET ステートメントを BY ステートメントとともに使用し、SET ステートメントで複数の入力データセットを指定する場合に実行されます。	X	X	X

言語要素	目的	アクセス方式		BY ステートメントと一緒に使用
		順次	直接	
	■ データセットの連結(追加)は、BY ステートメントが使用されないことを除いて、インターリーブと同じ方法で実行されます。単一の SET ステートメントが指定され、複数のデータセットが指定されます。 ■ 1 対 1 読み取りは、入力データセットごとに 1 つずつ複数の SET ステートメントが使用され、BY ステートメントが使用されない場合に実行されます。 ■ SET ステートメントには、KEY=オプションおよび POINT=オプションを指定できます。これらを使用すると、出力用に特定の行を直接選択できます。			
UPDATE	■ “更新”では、トランザクションをマスターデータセット内の行に適用します。 ■ 更新では、既存のマスターデータセットを変更して行は更新されません。既存のマスターデータセットの更新されたコピーが生成されます。 ■ 更新するには、マスターデータセットとトランザクションデータセットの両方のデータが BY 変数の値によってインデックス付けされるか、並べ替えられる必要があります。	X		X
PROC APPEND	■ PROC APPEND は、ある SAS データセットの行を	X		

言語要素	目的	アクセス方式		BY ステートメントと一緒に使用
		順次	直接	
	別の SAS データセットの末尾に追加します。			
PROC SQL	■ PROC SQL は、1 つ以上の SAS データセットから任意の行を読み取ります。 ■ PROC SQL は、最大 32 の SAS データセットから行を読み取り、それらを単一の行に結合します。 ■ PROC SQL は、新しいデータセットを作成せずに、既存の SAS データセット内の行を操作します。 ■ PROC SQL は、入力データセット内の変数のデカルト積を生成できます。アクセス方式は内部オプティマイザーによって選択されます。	X	X	X
ハッシュテーブル	■ ハッシュテーブルマージでは、データセット内の変数の値がハッシュテーブル内のキーの値と一致する場合に、データセットが結合されます。	X	X	

例: データの準備

例: 結合するデータを検査する

コード例

この例では、CONTENTS、 SORT、および PRINT プロシジャを使用して、3 つのデータセット内のデータを結合する前に検査します。

次の表は、結合される 3 つの入力データセット Inventory、Sales、および Sales2019 を示しています。

Inventory

Part Number	Part Name
K89R	seal
M4J7	sander
LK43	filter
MN21	brace
BC85	clamp
NCF3	valve
KJ66	cutter
UYN7	rod
JD03	switch
BV1E	timer

Sales

Part Number	Part Name	Sales Person
NCF3	valve	JN
BV1E	timer	JN
LK43	filter	KM
K89R	seal	SJ
LK43	filter	JN
M4J7	sander	KM
BV1E	timer	KM

Sales2019

Part Number	LastSoldDate
BC85	10/15/2019
BV1E	10/23/2019
KJ66	11/11/2019
LK43	09/12/2019
MN21	09/13/2019
NCF3	07/24/2019
UYN7	12/11/2019

```
proc contents data=Inventory; run;          /* 1 */
proc contents data=Sales; run;
proc contents data=Sales2019; run;
quit;
proc sort data=inventory; by PartNumber; run;  /* 2 */
proc sort data=Sales; by PartNumber; run;
proc sort data=Sales2019; by PartNumber; run;

proc print data=inventory; run              /* 3 */;
proc print data=Sales; run;
proc print data=Sales2019; run;
```

1 入力データセットに関する説明情報を表示して、それらが共通変数を共有しているかどうかを判断します。 PROC CONTENTS 出力で、共通変数 PartNumber が同じ属性を持ち、各入力データセット内の同じデータを表すことを確認します。

2 入力データセットを共通変数の値で並べ替えます。変数 partNumber は 3 つの入力データセットすべてに共通であるため、データをマージするための BY 変数として使用できます。

3 データセットを出力して、共通変数の値を検査し、データセット間のリレーションシップを判断します。PROC PRINT 出力は、データセット Inventory と Sales の間に 1 対多リレーションシップ、データセット Sales と Sales2019 の間に a 多対 1 リレーションシップ、データセット Inventory と Sales2019 の間に 1 対 1 リレーションシップがあることを示しています。

図 21.2 Inventory、Sales、および Sales2019 データセットを比較する部分的な PROC CONTENTS 出力

The CONTENTS Procedure
WORK.INVENTORY

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
2	partName	Char	8
1	partNumber	Char	8

The CONTENTS Procedure
WORK.SALES

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
2	partName	Char	8
1	partNumber	Char	8
3	salesPerson	Char	8

The CONTENTS Procedure
WORK.SALES2019

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format
2	lastSoldDate	Num	8	MMDDYY10.
1	partNumber	Char	8	

図 21.3 1 対多および多対 1 のリレーションシップを示す Inventory、Sales、および Sales2019 データセットの PROC PRINT 出力

WORK.INVENTORY

Obs	partNumber	partName
1	BC85	clamp
2	BV1E	timer
3	JD03	switch
4	K89R	seal
5	KJ66	cutter
6	LK43	filter
7	M4J7	sander
8	MN21	brace
9	NCF3	valve
10	UYN7	rod

WORK.SALES

Obs	partNumber	partName	salesPerson
1	BV1E	timer	JN
2	BV1E	timer	KM
3	K89R	seal	SJ
4	LK43	filter	KM
5	LK43	filter	JN
6	M4J7	sander	KM
7	NCF3	valve	JN

WORK.SALES2019

Obs	partNumber	lastSoldDate
1	BC85	10/15/2019
2	BV1E	10/23/2019
3	KJ66	11/11/2019
4	LK43	09/12/2019
5	MN21	09/13/2019
6	NCF3	07/24/2019
7	UYN7	12/11/2019

注: Inventory データセットに存在し、Sales2019 データセットには存在しない BY 変数値があります。たとえば、Inventory データセットにある PartNumber JD03 は Sales2019 データセットには存在しません。マージされるデータセット間で BY 変数の値が異なる場合、それらは"不一致オブザベーション"または"不一致レコード"と呼ばれることがよくあります。欠損値も、不一致オブザベーションと見なされます。マージ方法が異なれば、不一致オブザベーションの処理方法も異なります。詳細については、“方法の比較” (528 ページ)および “PROC SQL とマッチマージ” (528 ページ)を参照してください。

要点

- PRINT プロシジャを使用すると、結合するデータセットの構造と内容を調べることができます。
- CONTENTS プロシジャでは、変数情報、出力形式、オブザベーションの数、ファイルサイズ、データに関するその他の要約情報など、データセットに関する説明情報が表示されます。
- 1 対多または多対 1 のリレーションシップを持つデータセットをマージする場合は、PROC SQL、DATA ステップ MERGE BY (マッチマージ) (570 ページ)、MODIFY または UPDATE ステートメント、ハッシュテーブルマージ (592 ページ)のいずれかを使用できます。
- DATA ステップの SET、MERGE、MODIFY、UPDATE ステートメントを使用してデータセットを結合する場合、共通変数の属性が一致する必要があります。詳細については、“例: 共通変数が異なるデータ型を持つデータの準備” (541 ページ) “例: 共通変数の長さが異なるデータの準備” (544 ページ)、および “例: 共通変数が異なる出力形式とラベルを持つデータの準備” (547 ページ)を参照してください。

関連項目

- “COMPARE プロシジャ” (*Base SAS プロシジャガイド*)
- “例: 複数の入力データセット内の共通変数を見つける” (538 ページ)

例: 複数の入力データセット内の共通変数を見つける

コード例

この例では、PROC SQL は SAS セッション Dictionary テーブルのメタデータを使用して、3 つの入力データセット(Inventory、Sales、および Sales2019)に共通する変数を決定します。

```
proc sql;
  title "Variables Common to Inventory, Sales, and Sales2019";
  create table commonvars as                          /* 1 */
    select memname, upcase(name) as name
        from dictionary.columns
        where libname='WORK' and
```

```
                memname in ('INVENTORY', 'SALES', 'SALES2019');
  select name                                          /* 2 */
        from commonvars
        group by name
        having count(*)=(select count(distinct(memname)) from commonvars);
quit;
```

1 work.inventory、work.sales、および work.sales2019 の変数を含むテーブル commonvars を作成します。

2 commonvars テーブルから一意の変数名を識別します。

Variables Common to Inventory, Sales, and Sales2019

name
PARTNUMBER

要点

- COMPARE プロシジャを使用して、最大 2 つのデータセットを比較することもできます。
- データセット全体で共通変数を識別することは、データを連結または追加するのではなく、一致する値に基づいてデータをマージする場合に役立ちます。PROC COMPARE と PROC SQL を使用すると、2 つのデータセット間の共通変数を識別できます。PROC SQL を使用すると、複数のデータセットの共通変数を識別できます。

関連項目

- "COMPARE プロシジャ"

例: データに重複 BY 値が含まれる場合に一意の BY 値を作成する

コード例

この例では、sales データセットに対して FREQ プロシジャを使用して、Sales データセット内の変数 partNumber の重複値をチェックします。次に、CATX 関数で_N_ DATA ステップ自動変数を使用して、重複値の一意の行 ID を作成します。

```
proc sort data=sales;                    /* 1 */
  by partNumber;
run;
proc print data=sales; run;

proc freq data = Sales noprint;                  /* 2 */
  tables partNumber / out = SalesDupes
                (keep = partNumber Count
                 where = (Count > 1));
run;

proc print data=SalesDupes;run;

data SalesUnique;                          /* 3 */
  set Sales;
  uniqueID = catx('.',partNumber,_n_);
run;

proc print data=SalesUnique;                    /* 4 */
  var uniqueID partNumber partName salesPerson;
run;
```

1 BY 変数として使用される変数 partNumber によってデータを並べ替えます。

2 PROC FREQ を使用して、partNumber に重複値があるかどうかを確認します。BY 変数 partNumber および Count のみを含むデータセット SalesDupes を作成します。Count は PROC FREQ によって自動的に生成される変数で、Sales データセット内の partNumber の各値の度数を表します。partNumber の値は、その値が Sales に複数回出現する(Count > 1)場合にのみ SalesDupes に書き込まれます。

3 BY 変数 partNumber の一意の値を含むデータセットを作成します。各値に一意の番号を追加して、partNumber の値を変更して一意の値を作成します。CATX 関数は、partNumber の各値に_N_の値を追加し、新しい値を uniqueID に格納します。

4 PROC PRINT は SalesUnique の値を表示します。

アウトプット 21.1　Sales データセットで見つかった共通変数 PartNumber の重複値を示す PROC FREQ の出力

Obs	partNumber	COUNT
1	BV1E	2
2	LK43	2

アウトプット 21.2　共通変数 PartNumber の一意の値を含む新しい変数を示す PROC PRINT 出力

Obs	uniqueID	partNumber	partName	salesPerson
1	BV1E.1	BV1E	timer	JN
2	BV1E.2	BV1E	timer	KM
3	K89R.3	K89R	seal	SJ
4	LK43.4	LK43	filter	KM
5	LK43.5	LK43	filter	JN
6	M4J7.6	M4J7	sander	KM
7	NCF3.7	NCF3	valve	JN

要点

- さまざまな結合方法の比較、およびどの方法で一意の BY 値が必要になるかについては、“方法の比較” (528 ページ)を参照してください。

関連項目

-
- CATX 関数の詳細については、“CATX 関数” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。
- “データリレーションシップ” (517 ページ)
- “データの結合” (517 ページ)
- 表 21.2 (529 ページ)
- PROC FREQ によって生成される自動変数の詳細については、FREQ プロシジャの出力データセットのドキュメントを参照してください。

例: 共通変数が異なるデータ型を持つデータの準備

コード例

この例では、2 つのデータセットに、異なるデータ型を持つ共通変数が含まれています。

この例では、入力データセット CarsSmall と Sashelp.Cars データセットのサブセットを使用します。

```
data cars;                          /* 1 */
  set sashelp.cars;
run;

proc contents data=cars; run;              /* 2 */
proc contents data=CarsSmall; run;

data combineCars;
  merge cars CarsSmallNum;                 /* 3 */
  by make model;
  keep Make DriveTrain Model MakeModelDrive Weight;
run;
```

1 Sashelp.Cars データセットから例の入力データを作成します。

2 共通変数を識別するために、各入力データセットに関する説明情報を表示します。CONTENTS プロシジャ出力は、共通変数 Weight が CarsSmall データセットでは CHAR 型変数であり、Cars データセットでは NUMERIC 型変数であることを示しています。

3 共通変数 Weight の値によってデータセットをマージします。入力データセットごとに型属性が異なるため、SAS は変数型に互換性がないことを示すエラーメッセージをログに出力します。

図 21.4 データセット Vehicles と VehiclesSmall の共通変数のデータ型の違いを示す PROC CONTENTS 出力の比較

WORK.CARSSMALL

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
4	DriveTrain	Char	5
6	Length	Num	8
1	Make	Char	9
7	MakeModelDrive	Char	58
2	Model	Char	29
3	Type	Char	6
5	Weight	Char	4

WORK.CARS

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
1	Make	Char	13		
2	Model	Char	40		
4	Origin	Char	6		
3	Type	Char	8		
13	Weight	Num	8		Weight (LBS)
14	Wheelbase	Num	8		Wheelbase (IN)

Sort Information

Sortedby	Make Type
Validated	YES
Character Set	ANSI

例のコード 21.1 入力データセット内の BY 変数の型属性が異なることを示すログエラー

```
ERROR: Variable WeightLBS has been defined as both character and numeric.
1274  run;
```

この問題を解決するには、新しい DATA ステップで INPUT 関数を使用して、carsSmall データセットに共通変数 Weight を再作成します。

```
data CarsSmallNum;                /* 1 */
  set CarsSmall;
  weightNum=input(weight, 8.);
  drop weight;
  rename WeightNum=weight;
run;

proc sort data=cars; by make model; run;    /* 2 */
proc sort data=CarsSmallNum; by Make Model; run;

data combineCars;
  merge cars CarsSmallNum;              /* 3 */
  by make model;
  keep Make DriveTrain Model MakeModelDrive Weight;
run;
proc contents data=combineCars; run;       /* 4 */
proc print data=combineCars; run;
```

1 INPUT 関数は、変数 Weight を数値データ型に変換し、その値を新しい変数 WeightNum に格納します。CarsSmall の元の Weight 文字変数は削除され、新しい数値変数 WeightNum の名前が Weight に変更されます。変数の名前を変更すると、Cars データセットと CarsSmall データセットに、データセットをマージするための共通変数が確保されます。

2 Cars データセットと CarsSmallNum データセットは、マージできるように同じ変数で並べ替えられます。

3 Cars および CarsSmallNum データセットは、make と model ごとにマージされます。各データセットの Weight 変数のデータ型が一致するため、データセットは正常にマージされ、combineCars データセットが作成されます。

4 CarsSmall データセットの一部のみが図 21.5 に表示されています。

図 21.5 共通変数の型属性を正規化した後の、マッチマージされたデータセットの部分的な PROC PRINT 出力

Cars by Make Model

Obs	Make	Model	DriveTrain	Weight	MakeModelDrive
1	Acura	3.5 RL 4dr	Front	3880	
2	Acura	3.5 RL w/Navigation 4dr	Front	3893	
3	Acura	MDX	All	4451	
4	Acura	NSX coupe 2dr manual S	Rear	3153	
5	Acura	RSX Type S 2dr	Front	2778	
6	Acura	TL 4dr	Front	3575	
90	Chevrolet	Venture LS	Front	3699	
91	Chevrolet	Aveo 4dr	Front	2370	Chevrolet Aveo 4dr - Front wheel drive
92	Chevrolet	Aveo LS 4dr hatch	Front	2348	Chevrolet Aveo LS 4dr hatch - Front wheel drive
93	Chrysler	300M 4dr	Front	3581	
94	Chrysler	300M Special Edition 4dr	Front	3650	

要点

- データセットを結合する場合、入力データセット内の同じ名前を持つ変数は共通変数と呼ばれます。1 つの入力データセット内の共通変数は、必ずしも別のデータセット内の同じ変数と同じ属性を共有するわけではありません。
- 同じデータを持ちながら属性が異なる共通変数は、データセットを結合するときに問題を引き起こす可能性があります。
- 型属性が異なる場合、SAS は DATA ステップの処理を停止し、変数に互換性がないことを示すエラーメッセージを発行します。このエラーを修正するには、DATA ステップを使用して変数を再作成する必要があります。
- 長さ属性が異なる場合、SAS は SET、MERGE、または UPDATE ステートメントで指定された最初のデータセットから長さを取得します。

関連項目

- “文字値から数値への変換” (*SAS 関数と CALL ルーチン: リファレンス*)
- “数値から文字値への変換” (*SAS 関数と CALL ルーチン: リファレンス*)

例: 共通変数の長さが異なるデータの準備

コード例

この例では、MERGE ステートメントと BY ステートメントを使用して、データセット quarter1、quarter2、quarter3、quarter4 が 1 つの出力データセットにマッチマージされます。データセットは事前に並べ替えられ、共通変数 account に基づいてマージされます。

この例では、Quarter1、Quarter2、Quarter3、Quarter4 のデータセットを使用します。

```
proc print data=quarter1; run;        /* 1 */
proc print data=quarter2; run;
proc print data=quarter3; run;
proc print data=quarter4; run;

data yearly;
  merge quarter1 quarter2 quarter3 quarter4;  /* 2 */
  by Account;
run;
```

```
data yearly;                          /* 3 */
  length Mileage 6;
  merge quarter1 quarter2 quarter3 quarter4;
  by Account;
run;
proc contents data=yearly; run;       /* 4 */
```

1 入力データセットに関する説明情報を表示して、共通変数の属性が各入力データセットで同じかどうかを比較します。PROC CONTENT 出力は、共通変数 mileage の長さが、quarter1 データセットでは 4 バイト、quarter2 データセットでは 8 バイト、quarter3 と quarter4 データセットでは 6 バイトであることを示しています。

2 共通変数 mileage によってデータセットをマージします。SAS がゼロ以外のリターンコードを発行し、ログ出力に警告を出力することに注意してください。

3 MERGE ステートメントを指定する前に、LENGTH ステートメントで適切な長さを指定して、mileage 変数の長さを変更します。SET、MERGE、および UPDATE ステートメントを使用する場合は、LENGTH ステートメントもその前に配置する必要があります。

4 マージされた出力データセットの説明情報を表示します。

Quarter1

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
2	account	Num	8
1	mileage	Num	4

Quarter2

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
2	account	Num	8
1	mileage	Num	8

Quarter3

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
2	account	Num	8
1	mileage	Num	6

Quarter4

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
2	account	Num	8
1	mileage	Num	6

例のコード 21.2　異なる長さの変数を含むデータセットのマッチマージのログ出力

```
WARNING: Multiple lengths were specified for the variable mileage by input data set(s).
This can cause truncation of data.
```

Yearly

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
2	account	Num	8
1	mileage	Num	4

Yearly2

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
2	account	Num	8
1	mileage	Num	6

ATTRIB ステートメントを使用して、変数の長さ属性を変更することもできます。

```
data yearly;
  merge quarter1 quarter2 quarter3 quarter4;
  by Account;
  attrib Mileage
     length = 6;
run;
```

データの切り捨てが予想される場合(たとえば、文字値の末尾から意味のない空白を削除する場合)、警告が予想され、SAS がゼロ以外のリターンコードを発行しないようにする必要があります。この場合、VARLENCHK=システムオプションを NOWARN に設定することで、この警告をオフにすることができます。

要点

- データセットを結合する場合、入力データセット内の同じ名前を持つ変数は共通変数と呼ばれます。1 つの入力データセット内の共通変数は、必ずしも別のデータセット内の同じ変数と同じ属性を共有するわけではありません。
- 同じデータを持ちながら属性が異なる共通変数は、データセットを結合するときに問題を引き起こす可能性があります。
- データセットを結合するときに、共通変数の長さが異なる場合、SAS は SET、MERGE、または UPDATE ステートメントで指定された最初のデータセットから長さを取得し、警告メッセージ (545 ページ)を SAS ログに出力します。

関連項目

- “LENGTH ステートメント” (*SAS DATA ステップステートメント: リファレンス*)

例: 共通変数が異なる出力形式とラベルを持つデータの準備

コード例

この例では、BY 変数 Name を指定した MERGE ステートメントを使用して、データセット class と classfit がマッチマージされます。

ATTRIB ステートメントは、最終出力データセット内の共通変数の長さ、ラベル、および出力形式を制御するために使用されます。

この例では、Class および Classfit データセットを使用します。

データセットをマージする前に、各データセットに対して PROC CONTENTS を実行して共通変数を識別し、共通変数の属性間の違いを調べます。

```
proc contents data=class; run;          /* 1 */
proc contents data=classfit; run;

proc sort data=class; by name; run;           /* 2 */
proc sort data=classfit; by name; run;

data merged;
  merge class classfit; by Name;
  attrib Weight
     label = "Weight";
  attrib Height Weight Predict format=comma8.2;   /* 3 */
run;
proc print data=merged;                     /* 4 */
run;
proc contents data=merged; run;
```

1 入力データセットに関する説明情報を表示して、それらが共通変数を共有しているかどうかを判断し、それらの属性を比較します。PROC CONTENTS 出力は、共通変数 Height と Weight が各入力データセットで異なるラベルを持つことを示しています。

2 入力データセットを BY 変数の値で並べ替えます。

3 ATTRIB ステートメントを指定して属性を変更します。

4 マージされた出力データセットのディスクリプター情報を出力します。

アウトプット 21.3　出力形式とラベルの違いを示すデータセット Class と Classfit の PROC CONTENTS 出力

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format
2	Age	Num	8	
3	Height	Num	8	COMMA8.
1	Name	Char	8	
4	Weight	Num	8	COMMA8.

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
2	Age	Num	8		
3	Height	Num	8		
1	Name	Char	8		
5	Predict	Num	8	COMMA8.1	
4	Weight	Num	8	COMMA8.2	Weight in Pounds

アウトプット 21.4　マージされたデータセットの出力形式とラベルが異なる場合のデフォルトの動作の PROC CONTENTS 出力

Obs	Name	Age	Height	Weight in Pounds	Predict
1	Alice	13	57	84	.
2	Barbara	13	65	98	.
3	Carol	14	63	103	.
4	Jane	12	60	85	.
5	Janet	15	63	113	100.7
6	Joyce	11	51	51	.
7	Judy	14	64	90	.
8	Louise	12	56	77	.
9	Mary	15	67	112	116.3
10	Philip	16	72	112	137.7
11	Ronald	15	67	133	118.2
12	William	15	67	150	116.3

アウトプット 21.5　ATTRIB ステートメントがマージされた出力データセットの出力形式とラベルをどのように変更するかを示す PROC CONTENTS 出力

Obs	Name	Age	Height	Weight	Predict
1	Alice	13	56.50	84.00	.
2	Barbara	13	65.30	98.00	.
3	Carol	14	62.80	102.50	.
4	Jane	12	59.80	84.50	.
5	Janet	15	62.50	112.50	100.66
6	Joyce	11	51.30	50.50	.
7	Judy	14	64.30	90.00	.
8	Louise	12	56.30	77.00	.
9	Mary	15	66.50	112.00	116.26
10	Philip	16	72.00	112.00	137.70
11	Ronald	15	67.00	133.00	118.21
12	William	15	66.50	150.00	116.26

PROC CONTENTS Output for Data Set Merged

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
2	Age	Num	8		
3	Height	Num	8	COMMA8.2	
1	Name	Char	8		
5	Predict	Num	8	COMMA8.2	
4	Weight	Num	8	COMMA8.2	Weight

要点

- 入力データセット内の共通変数のラベル、出力形式、または入力形式属性が異なる場合、SAS は SET、MERGE、MODIFY、UPDATE ステートメントにリストされている最初のデータセットから属性を取得します。
- 明示的に指定したラベル、出力形式、または入力形式はデフォルトをオーバーライドします。すべてのデータセットに明示的に指定された属性が含まれている場合、SET または MERGE ステートメントにリストされている最初のデータセットで指定された属性が他の属性をオーバーライドします。
- DATA ステップで ATTRIB ステートメントを使用すると、新しい出力データセットに必要な属性が含まれることを確認できます。
- VLABEL 関数と VLABELX 関数を使用して変数の属性を変更することもできます。

関連項目

- "ATTRIB ステートメント" (*SAS DATA ステップステートメント: リファレンス*)
- "VLABEL 関数" (*SAS 関数と CALL ルーチン: リファレンス*)
- "VLABELX 関数" (*SAS 関数と CALL ルーチン: リファレンス*)

例: 共通変数が異なるデータを表すデータの準備準備する

コード例

この例では、マージするデータセットに同じ名前を持つ変数が含まれていますが、それらは完全に異なるデータを表します。この意味では、それらは共通変数であることを意図したものではありません。

この問題を解決するには、入力データセットの 1 つにある変数の名前を変更します。ここでは、RENAME=データセットオプションを使用して、CarsSmall 入力データセット内の変数 Weight の名前を変更します。

```
data vehicles(rename=(weight=weightLBS));
   set vehicles;
run;
```

要点

- 入力データセット内の共通変数のラベル、出力形式、または入力形式属性が異なる場合、SAS は SET、MERGE、MODIFY、UPDATE ステートメントにリストされている最初のデータセットから属性を取得します。
- 明示的に指定したラベル、出力形式、または入力形式はデフォルトをオーバーライドします。すべてのデータセットに明示的に指定された属性が含まれている場合、SET または MERGE ステートメントにリストされている最初のデータセットで指定された属性が他の属性をオーバーライドします。
- DATA ステップで ATTRIB ステートメントを使用すると、新しい出力データセットに必要な属性が含まれることを確認できます。
- “VLABEL 関数” (*SAS 関数と CALL ルーチン: リファレンス*)と“VLABELX 関数” (*SAS 関数と CALL ルーチン: リファレンス*)を使用して変数の属性を変更することもできます。

関連項目

- “RENAME ステートメント” (*SAS DATA ステップステートメント: リファレンス*)

例: データの連結

例: SET ステートメントを使用した連結

コード例

この例では、SET ステートメントを使用して 2 つのデータセットを 1 つの出力データセットに連結します。入力データセットは 1 つの共通変数(common)を共有します。連結では、データセットがなんらかの変数を共有したり、データがなんらかの形で関連したりする必要はありません。連結では、1 つのデータセットをそのまま他のデータセットの後に配置するだけです。

次の表は、この例で使用される入力データセット animal と plant を示しています。

animal	plant

OBS	common	animal
1	a	Ant
2	b	Bird
3	c	Cat
4	d	Dog
5	e	Eagle
6	f	Frog

OBS	common	plant
1	a	Apple
2	b	Banana
3	c	Coconut
4	d	Dewberry
5	e	Eggplant
6	f	Fig

```
data concatenate;
  set animal plant;          /* 1 */
run;
proc print data=concatenate;     /* 2 */
run;
```

1 SET ステートメントは、両方の入力データセットからすべてのオブザベーションを順次読み取り、読み取られた最初のデータセットのオブザベーションを 2 番目に読み取られたデータセットに追加します。結果は、concatenate という名前の新しい出力データセットに格納されます。その結果、animal データセットからのオブザベーションが最初に出力に表示され、次に plant データセットからのオブザベーションが表示されます。

2 結果(アウトプット 21.6 (551 ページ))を出力します。

アウトプット 21.6　SET ステートメントを使用して連結データセットを表示する PROC PRINT 出力

Obs	common	animal	plant
1	a	Ant	
2	b	Bird	
3	c	Cat	
4	d	Dog	
5	e	Eagle	
6	f	Frog	
7	a		Apple
8	b		Banana
9	c		Coconut
10	d		Dewberry
11	e		Eggplant
12	f		Fig

出力データセット内のオブザベーションの数は 12 であり、これは両方の入力データセットからのオブザベーションの合計であることに注意してください。

要点

- 連結とは、2 つ以上のデータセットを 1 つのデータセットに次々に結合することです。
- 出力データセットでは、SET ステートメントで最初にリストされているデータセットからのオブザベーションの後に、SET ステートメントで 2 番目にリストされているデータセットからのオブザベーションが続き、その後も同様に続きます。
- 出力データセットには、両方の入力データセットのすべての変数が含まれます。あるデータセットには存在するが別のデータセットには存在しない変数の値は、欠損として設定されます。
- 通常は、同じ変数を持つ SAS データセットを連結します。

関連項目

- 連結 (519 ページ)(データセット)
- “SET ステートメント” (*SAS DATA ステップステートメント: リファレンス*)

例: PROC SQL を使用した連結

コード例

この例では、SQL プロシジャを使用して 2 つのデータセットを連結して、新しい出力 SAS データセットと SQL テーブルを作成します。連結中、SQL は両方の入力データセットからすべての行を読み取り、combined という名前の新しい出力データセットを作成します。

次の表は、この例で使用される入力データセット animal と plant を示しています。

animal

OBS	common	animal
1	a	Ant
2	b	Bird
3	c	Cat
4	d	Dog
5	e	Eagle
6	f	Frog

plant

OBS	common	plant
1	a	Apple
2	b	Banana
3	c	Coconut
4	d	Dewberry
5	e	Eggplant
6	f	Fig

```
proc sql;
   create table combined as       /* 1 */
    select * from animal          /* 2 */
    outer union corresponding     /* 3 */
    select * from plant;          /* 4 */
quit;
```

1 Create ステートメントを使用して、テーブル combined を作成します。

2 animal のすべての変数を含めます。

3 OUTER UNION CORRESPONDING ステートメントを使用して、両方のテーブルのすべての行を含め、共通変数をオーバーレイします。

4 plant のすべての変数を含めます。

アウトプット 21.7　PROC SQL を使用した animal と plant の連結データセット

common	animal	plant
a	Ant	
b	Bird	
c	Cat	
d	Dog	
e	Eagle	
f	Frog	
a		Apple
b		Banana
c		Coconut
d		Dewberry
e		Eggplant
f		Fig

結果の出力は、SAS データセットと SQL テーブルで構成されます。どちらのテーブルにも、それぞれ 12 行(オブザベーション)があります。出力の行の合計数は、結合されたデータセットの行の合計と等しくなります。あるデータセットには存在するが別のデータセットには存在しない変数の値は、欠損として設定されます。

出力データセットでは、plant データセットからのオブザベーションが animal データセットからのオブザベーションに追加されます。animal データセットは SET ステートメントの最初にあるため、出力でも最初に表示されます。出力データセットには、両方の入力データセットのすべての変数が含まれます。

要点

- 連結とは、2 つ以上のデータセットを 1 つのデータセットに次々に結合することです。
- 連結とは、2 つ以上のデータセットを 1 つのデータセットに次々に結合することです。

関連項目

- “クエリの結果の連結(OUTER UNION)” (*SAS SQL プロシジャユーザーガイド*)
- “和結合” (*SAS SQL プロシジャユーザーガイド*)
- “OUTER UNION 演算子” (*SAS SQL プロシジャユーザーガイド*)
- “PROC SQL と SAS DATA ステップとの比較” (*SAS SQL プロシジャユーザーガイド*)

例: PROC APPEND を使用したファイルの追加

コード例

この例では、APPEND プロシジャを使用して、オブザベーションを Year2 データセットから Year1 データセットの末尾に追加します。処理中、プロシジャは Year2 データセットから行のみを読み取ります。次に、プロシジャは、Year2 データセットからのオブザベーションを追加することによって Year1 データセットを更新します。このプロシジャでは、新しい出力データセットは生成されません。既存の Year1 データセットを更新するだけです。

次の表は、この例で使用される入力データセット Year1 と Year2 を示しています。

表 21.5　結合する入力データ

Year1

OBS	Date
1	2009
2	2010
3	2011
4	2012

Year2

OBS	Date
1	2010
2	2011
3	2012
4	2013
5	2014

```
proc append base=Year1 data=Year2;
run;
proc print data=Year1;
  title "Year1";
run;
```

アウトプット 21.8　連結テーブル(PROC APPEND)

Year1

Obs	date
1	2009
2	2010
3	2011
4	2012
5	2010
6	2011
7	2012
8	2013
9	2014

Appended rows from Year2

Year1 データセットには、両方のデータセットのすべてのオブザベーションが含まれます。

注: PROC APPEND を使用して、順次ライブラリ内の SAS データセットにオブザベーションを追加することはできません。

要点

- SET ステートメントを使用してデータセットを連結すると、SAS はすべての入力データセットのすべての行を読み取り、新しい出力データセットを作成します。APPEND プロシジャを使用すると、SAS は DATA=オプションで指定されたデータセット内の行のみを読み取り、新しいデータセットは作成しません。2 つの方法の選択に関する詳細については、"SET ステートメントと APPEND ステートメント間の選択" (*Base SAS プロシジャガイド*)を参照してください。
- DATA=データセットに BASE=データセットにない変数が含まれている場合は、APPEND ステートメントで FORCE オプションを指定する必要があります。詳細については、"変数が異なるデータセットへの追加" (*Base SAS プロシジャガイド*)を参照してください。
- 追加の処理が必要ない場合は、DATA ステップを使用してデータセットを連結するよりも、PROC APPEND または PROC DATASETS の APPEND ステートメントを使用する方が効率的です。
- 一般的に、1 つ以上の共通変数を共有する SAS データセットを連結します。

関連項目

- "2 つの SAS データセットを連結する" (*Base SAS プロシジャガイド*)
- "APPEND プロシジャ" (*Base SAS プロシジャガイド*)
- "APPEND ステートメント" (*Base SAS プロシジャガイド*)

例: 入力データセットの変数が欠損している場合にログにメッセージを追加する

コード例

この例では、DATA ステートメントの OPEN=DEFER オプションにより、読み取られる最初の入力データセットに 1 つ以上の入力データセットの変数が存在しない場合、SAS ログに注記が書き込まれます。

次の表は、この例で使用される入力データセットを示しています。Table2 で定義されている変数 predict と lowermean が Table1 には存在しないことに注目してください。

OPEN=DEFER オプションを指定せずにデータセットを連結し、結果を出力すると、出力には、すべての入力データセットに含まれていない変数の値が欠損として表示される典型的な連結が表示されます。

```
data concat;
  set table1 table2;
run;
proc print data=concat;
  title "Concatenate Table1 and Table2";
run;
```

Concatenate Table1 and Table2

Obs	Name	Sex	Age	Height	Weight	predict	lowermean
1	Alfred	M	14	69.0	112.5	.	.
2	Carol	F	14	62.8	102.5	.	.
3	Henry	M	14	63.5	102.5	.	.
4	Judy	F	14	64.3	90.0	.	.
5	Alfred	M	14	69.0	112.5	126.006	116.942
6	Carol	F	14	62.8	102.5	101.832	96.375
7	Henry	M	14	63.5	102.5	104.562	98.982
8	Judy	F	14	64.3	90.0	107.681	101.842

この出力を、OPEN=DEFER オプションを使用した場合の出力と比較します。最初の SET ステートメントにリストされているデータセットの変数だけが最終出力データセットに表示されることに注意してください。

```
data concat2;
  set table1 table2 open=defer;
run;
proc print data=concat2;
  title "Concatenate with OPEN=DEFER";
```

```
run;
```

Concatenate with OPEN=DEFER

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Carol	F	14	62.8	102.5
3	Henry	M	14	63.5	102.5
4	Judy	F	14	64.3	90.0
5	Alfred	M	14	69.0	112.5
6	Carol	F	14	62.8	102.5
7	Henry	M	14	63.5	102.5
8	Judy	F	14	64.3	90.0

ログに次のメッセージが表示されます。

アウトプット 21.9　OPEN=DEFER 使用時に欠損している変数を示す部分的なログ出力

```
NOTE: There were 4 observations read from the data set WORK.TABLE1.
NOTE: For OPEN=DEFER processing, all variables processed should be specified by
the first data
      set listed in the SET statement.
NOTE: Variable predict, found on WORK.TABLE2, is being ignored. NOTE: Variable
lowermean, found on WORK.TABLE2, is being ignored.
NOTE: There were 4 observations read from the data set WORK.TABLE2.
NOTE: The data set WORK.CONCAT2 has 8 observations and 5 variables.
```

要点

- ほとんどの場合、後続のデータセットで定義された変数のセットが最初のデータセットで定義された変数と異なる場合、SAS はログに警告メッセージを出力しますが、実行は停止しません。
- SET ステートメントを使用してデータセットを連結すると、SAS はすべての入力データセットのすべての行を読み取り、新しい出力データセットを作成します。APPEND プロシジャを使用すると、SAS は DATA=オプションで指定されたデータセット内の行のみを読み取り、新しいデータセットは作成しません。2 つの方法の選択に関する詳細については、“SET ステートメントと APPEND ステートメント間の選択” (*Base SAS プロシジャガイド*)を参照してください。
- DATA=データセットに BASE=データセットにない変数が含まれている場合は、APPEND ステートメントで FORCE オプションを指定する必要があります。詳細については、“変数が異なるデータセットへの追加” (*Base SAS プロシジャガイド*)を参照してください。

- 追加の処理が必要ない場合は、DATA ステップを使用してデータセットを連結するよりも、PROC APPEND または PROC DATASETS の APPEND ステートメントを使用する方が効率的です。
- 通常は、同じ変数を持つ SAS データセットを連結します。

関連項目

- “SET ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “2 つの SAS データセットを連結する” (*Base SAS プロシジャガイド*)
- “APPEND プロシジャ” (*Base SAS プロシジャガイド*)
- FORCE オプション
- “属性が異なる変数を含むデータセットへの追加” (*Base SAS プロシジャガイド*)

例: データのインターリーブ

例: データセットのインターリーブ

コード例

次のプログラムは、入力データセットを作成し、各データセットを共通変数で並べ替え、データセットをインターリーブして、結果を出力します。共通変数 common は、BY 変数として指定されます。

次の入力データセット animal と plant がこの例で使用されます。

animal

OBS	common	animal
1	a	Ant
2	b	Bird
3	c	Cat
4	d	Dog
5	e	Eagle
6	f	Frog

plant

OBS	common	plant
1	a	Apple
2	b	Banana
3	c	Coconut
4	d	Dewberry
5	e	Eggplant
6	f	Fig

次のプログラムは、まず SORT プロシジャを使用して両方の入力データセットを並べ替え、データセットをインターリーブして、結果を出力します。

```
proc sort data=animal; by common; run;
proc sort data=plant; by common; run;

data interleave;
  set animal plant;
  by common;
run;
proc print data=interleave; run;
```

入力データセットは、DATA ステップの BY ステートメントで指定された同じ変数によって並べ替えられることに注意してください。

出力データセットには、すべてのデータセットのすべての 変数と、DATA ステップによって作成された変数が含まれます。あるデータセットには存在するが別のデータセットには存在しない変数の値は、欠損として設定されます。出力データセット内のオブザベーションの数は 12 で、これは両方のデータセットからのオブザベーションの合計です。

アウトプット 21.10　インターリーブデータセットの PROC PRINT 出力

Obs	common	animal	plant
1	a	Ant	
2	a		Apple
3	b	Bird	
4	b		Banana
5	c	Cat	
6	c		Coconut
7	d	Dog	
8	d		Dewberry
9	e	Eagle	
10	e		Eggplant
11	f	Frog	
12	f		Fig

要点

- 入力データセットは、まず BY 変数の値によってインデックス付けされるか、並べ替えられる必要があります。
- インターリーブされたデータセット内のオブザベーションは結合されず、BY 変数の値の順序で元のデータセットからコピーされます。
- 入力データセットは、SET ステートメントにリストされている順序で順次処理されます。
- 出力データセット内のオブザベーションは、BY 変数の値によって配置されます。

関連項目

- “インターリーブ” (520 ページ)データセット
- “SET ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “BY ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “SAS データセットの結合” (*SAS DATA ステップステートメント: リファレンス*)

例: BY 変数の重複値によるインターリーブ

コード例

次のプログラムは、入力データセットを作成し、各データセットを共通変数で並べ替え、データセットをインターリーブして、結果を出力します。

プログラムはまず、SORT プロシジャを使用して入力データセットを並べ替えます。このプロシジャは、共通変数 common の値によって各データセット内の行をグループ化し、並べ替えます。両方の入力データセットで共有 BY 変数に重複値があることに注意してください。

次の表は、入力データセットの animalDupes と plantDupes を示しており、各データセットで重複値が強調表示されています。

animalDupes

OBS	common	animal
1	a	Ant
2	a	Ape
3	b	Bird
4	c	Cat
5	d	Dog
6	e	Eagle

plantDupes

OBS	common	plant
1	a	Apple
2	b	Banana
3	c	Coconut
4	c	Celery
5	d	Dewberry
6	e	Eggplant

```
proc sort data=animalDupes; by common; run;
proc sort data=plantDupes; by common; run;

data interleave;
  set animalDupes plantDupes;
  by common;
run;

proc print data=interleave; run;
```

出力データセットには、両方の入力データセットのすべての変数が含まれます。あるデータセットには存在するが別のデータセットには存在しない変数の値は、欠損として設定されます。出力データセット内のオブザベーションの数は 12 で、これは入力データセットからのオブザベーションの合計です。オブザベーションは、元のデータセットに出現する順序で出力データセットに書き込まれます。

アウトプット 21.11　BY 変数の値が重複しているインターリーブデータセットの PROC PRINT 出力

Obs	common	animal	plant
1	a	Ant	
2	a	Ape	
3	a		Apple
4	b	Bird	
5	b		Banana
6	c	Cat	
7	c		Coconut
8	c		Celery
9	d	Dog	
10	d		Dewberry
11	e	Eagle	
12	e		Eggplant

入力データセット animalDupes からのオブザベーションが最初にリストされ、次に 2 番目の入力データセット plantDupes からのオブザベーションが続くことに注意してください。これは、DATA ステップがデータセットを SET ステートメントにリストされている順序で順次処理するためです。たとえば、データセット plantDupes が SET ステートメントの最初にリストされるように入力データセットの順序を変更すると、plantDupes からのオブザベーションが出力データセットの最初にリストされます。

```
data interleave;
  set plantDupes animalDupes; by common;
run;
proc print data=interleave; run;
```

アウトプット 21.12　SET ステートメントの順序が変更されたインターリーブデータセットの PROC PRINT 出力

Obs	common	plant	animal
1	a	Apple	
2	a		Ant
3	a		Ape
4	b	Banana	
5	b		Bird
6	c	Coconut	
7	c	Celery	
8	c		Cat
9	d	Dewberry	
10	d		Dog
11	e	Eggplant	
12	e		Eagle

要点

- 入力データセットは、まず BY 変数の値によってインデックス付けされるか、並べ替えられる必要があります。
- インターリーブされたデータセット内のオブザベーションは結合されず、BY 変数の値の順序で元のデータセットからコピーされます。
- 入力データセットは、SET ステートメントにリストされている順序で順次処理されます。
- 出力データセット内のオブザベーションは、BY 変数の値によって配置されます。

関連項目

- “インターリーブ” (520 ページ)データセット
- “SET ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “BY ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “SAS データセットの結合” (*SAS DATA ステップステートメント: リファレンス*)

例: 共通 BY 変数の異なる値によるインターリーブ

コード例

次のプログラムは、入力データセットを作成し、各データセットを共通変数で並べ替え、データセットをインターリーブして、結果を出力します。

次の表は、この例で使用されている入力データセットの animalDupes と plantMissing2 を示しており、異なる BY 値が強調表示されています。

animalDupes

OBS	common	animal
1	a	Ant
2	a	Ape
3	b	Bird
4	c	Cat
5	d	Dog
6	e	Eagle

plantMissing2

OBS	common	plant
1	a	Apple
2	b	Banana
3	c	Coconut
4	e	Eggplant
5	f	Fig

どちらの入力データセットにも、もう一方のデータセットには存在しない変数 common の値が含まれています。たとえば、animalDupes データセットの値"d"は、plantMissing2 データセットには存在しません。plantMissing2 データセットの値"f"は、animalDupes データセットには存在しません。

```
proc sort data=animalDupes; by common; run;   /* 1 */
proc sort data=plantMissing2; by common; run;

data interleave;                               /* 2 */
  set animalDupes plantMissing2;
  by common;
run;

proc print data=interleave; run;
```

1 入力データセット animalDupes および plantMissing2 を作成します。各入力データセットには変数 common が含まれており、SORT プロシジャは BY 変数 common の値の順にオブザベーションを並べ替えます。

2 DATA ステップは、common の値に基づいてデータセットをインターリーブします。

出力データセットには、両方の入力データセットのすべての変数が含まれます。あるデータセットには存在するが別のデータセットには存在しない変数の値は、欠損として設定されます。出力データセット内のオブザベーションの数は 11 で、これは入力データセットからのオブザベーションの合計です。オブザベーションは、元のデータセットに出現する順序で出力データセットに書き込まれます。

アウトプット 21.13　BY 値が異なるインターリーブデータセットの PROC PRINT 出力

Obs	common	animal	plant
1	a	Ant	
2	a	Ape	
3	a		Apple
4	b	Bird	
5	b		Banana
6	c	Cat	
7	c		Coconut
8	d	Dog	
9	e	Eagle	
10	e		Eggplant
11	f		Fig

要点

- 入力データセットは、まず BY 変数の値によってインデックス付けされるか、並べ替えられる必要があります。
- インターリーブされたデータセット内のオブザベーションは結合されず、BY 変数の値の順序で元のデータセットからコピーされます。
- 入力データセットは、SET ステートメントにリストされている順序で順次処理されます。
- 出力データセット内のオブザベーションは、BY 変数の値によって配置されます。

関連項目

- “インターリーブ” (520 ページ)データセット
- “SET ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “BY ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “SAS データセットの結合” (*SAS DATA ステップステートメント: リファレンス*)

例: データのマッチマージ

例: BY 変数に基づくオブザベーションのマッチマージ

コード例

この例では、MERGE ステートメントを BY ステートメントと組み合わせて使用し、2 つのデータセットをマージします。最初のデータセットのオブザベーションは、共通 BY 変数の値に基づいて、2 番目のデータセットのオブザベーションとマージされます。

入力データセット animal と plant には両方とも変数 common が含まれており、この例では BY 変数として指定されています。

この例では、どちらの入力データセットにも BY 変数の重複値がないため、1 対 1 マージが示されています。

次の表は、この例で使用される入力データセット animal と plant を示しています。

表 21.6 入力データセット

animal

OBS	common	animal
1	a	Ant
2	b	Bird
3	c	Cat
4	d	Dog
5	e	Eagle
6	f	Frog

plant

OBS	common	plant
1	a	Apple
2	b	Banana
3	c	Coconut
4	d	Dewberry
5	e	Eggplant
6	f	Fig

次のプログラムは、まず SORT プロシジャを使用して両方の入力データセットを並べ替え、次に共通 BY 変数 common によってデータセットをマッチマージします。PROC PRINT は結果を出力するために使用されます。

```
proc sort data=animal; by common; run;
proc sort data=plant; by common; run;

data matchmerge;
  merge animal plant;
  by common;
```

```
run;
proc print data=matchmerge; run;
```

アウトプット 21.14　BY 変数に基づくオブザベーションのマッチマージの PROC PRINT 出力

Obs	common	animal	plant
1	a	Ant	Apple
2	b	Bird	Banana
3	c	Cat	Coconut
4	d	Dog	Dewberry
5	e	Eagle	Eggplant
6	f	Frog	Fig

要点

- マッチマージは、1 つ以上の共通変数を持つデータセットをマージし、共通変数の値に基づいてデータセットをマージするために使用されます。
- 入力データセットは、マージする前に BY 変数でインデックス付けまたは並べ替えの必要があります。
- SAS のマッチマージは、BY 変数の値がすべて一致し、重複 BY 変数がない場合、PROC SQL の*内部結合*に相当します。詳細については、“内部結合” (*SAS SQL プロシジャユーザーガイド*)を参照してください。
- SAS は、値が欠損(または不一致)の場合でも、プログラムデータベクトル内のすべての変数の値を保持します。
- SAS が 1 つのデータセット内の BY グループから最後のオブザベーションを読み取ると、その BY グループのすべてのオブザベーションがすべてのデータセットから読み取られるまで、SAS はそのデータセットに一意のすべての変数の値をプログラムデータベクトルに保持します。最終データセット内のオブザベーションの合計数は、いずれかのデータセットの BY グループ内のオブザベーションの最大数の合計です。

関連項目

- “マッチマージ” (521 ページ)データセット
- “MERGE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “データリレーションシップ” (517 ページ)
- “DATA ステップのマッチマージと PROC SQL 結合の比較” (*SAS SQL プロシジャユーザーガイド*)

例: BY 変数ではない共通変数を持つオブザベーションのマッチマージ

コード例

この例では、MERGE ステートメントを BY ステートメントと組み合わせて使用し、2 つのデータセットを 1 対多マージでマージします。最初のデータセットのオブザベーションは、共通 BY 変数の値に基づいて、2 番目のデータセットのオブザベーションとマージされます。入力データセットの 1 つに BY 変数の重複値があるため、これは 1 対多マージです。

この例では、共通変数が BY 変数として指定されていない場合にその変数の値に何が起こるかを示します。

この例で使用される入力データセット one と many には、両方とも変数 ID と state が含まれています。変数 ID は BY 変数であり、その値は one データセット内で一意です。ただし、その値は many データセット内で一意ではありません。many データセット内の ID の値には複数のオブザベーションがあります。変数 state は両方の入力データセットに共通ですが、BY 変数として指定されていません。

マージの目的は、many データセット内の state の誤った略語を、データセット one に示されている正しい 2 文字の略語に置き換えることです。

次の表は、この例で使用される one および many データセットを示しています。

表 21.7 入力データセット

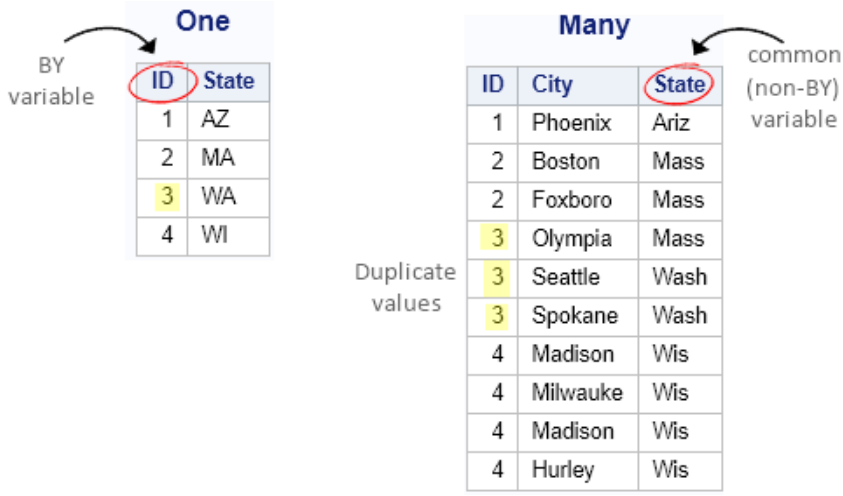
one

ID	state
1	AZ
2	MA
3	WA
4	WI

many

ID	city	state
1	Phoenix	Ariz
2	Boston	Mass
2	Foxboro	Mass
3	Olympia	Mass
3	Seattle	Wash
3	Spokane	Wash
4	Madison	Wis
4	Milwaukee	Wis
4	Madison	Wis
4	Hurley	Wis

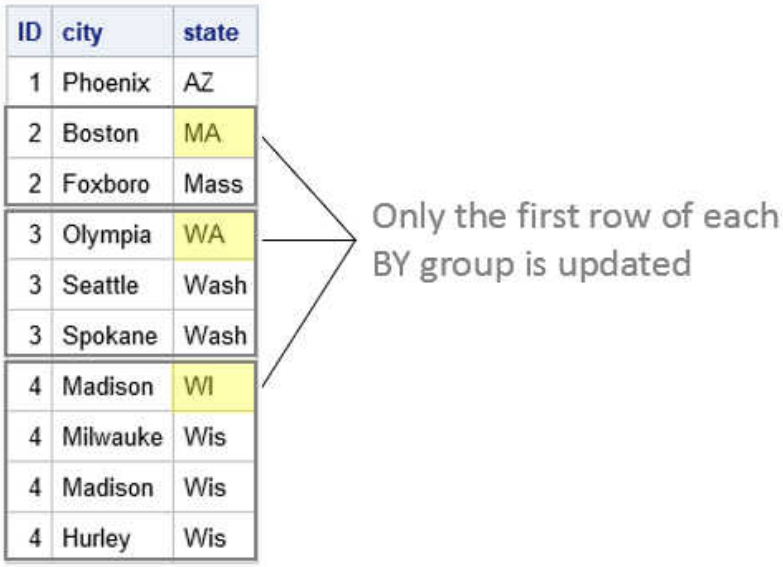
アウトプット 21.15　共通 BY 変数と共通非 BY 変数を含む入力データセットを示す PROC PRINT 出力



これらのデータセットがマージされると、データセット one の共通変数の値がデータセット many の値を上書きします。これは、one データセットが MERGE ステートメントの 2 番目にリストされているためです。ただし、同じ BY グループに対する MERGE ステートメントの後続の反復では、one データセットは再度読み取られません。したがって、one データセットの値は、BY グループの残りの値を置き換えません。これは、各 BY グループの最初の値は置き換えられますが、残りの値は置き換えられないことを意味します。

```
data three;
  merge many one;
  by ID;
run;
proc print data=three noobs; run;
title;
```

アウトプット 21.16　BY 変数ではない共通変数を持つオブザベーションのマッチマージの PROC PRINT 出力



BY グループ内のすべてのオブザベーションの state の値を置き換えるには、many データセットの共通(非 BY)変数を削除するか、名前を変更する必要があります。

```
data solution;
 merge many(drop=state) one;
 by ID;
run;
proc print data=solution noobs; run;
```

アウトプット 21.17　BY 変数ではない共通変数を使用した 1 対多マージのソリューションの PROC PRINT 出力

ID	city	state
1	Phoenix	AZ
2	Boston	MA
2	Foxboro	MA
3	Olympia	WA
3	Seattle	WA
3	Spokane	WA
4	Madison	WI
4	Milwauke	WI
4	Madison	WI
4	Hurley	WI

SAS は、マージの最初の反復時にデータセット one から値を読み取ります。データセットから読み取られた変数は BY グループ全体で自動的に保持されるため、BY グループ state のすべてのオブザベーションにはデータセット one の値が含まれます。

要点

- 1 対多マッチマージで最初に一致するオブザベーションについては、MERGE ステートメントで指定された最後のデータセット内の共通変数の値が、前のデータセットの値を上書きします。ただし、同じ BY グループに対する MERGE ステートメントの後続の反復では、最初のデータセットは再度読み取られません。したがって、共通 BY 変数の残りの値は、最後に読み取られたデータセットからではなく、元のデータセットから取得されます。
- マッチマージは、1 つ以上の共通変数を持つデータセットをマージし、共通変数の値に基づいてデータセットをマージするために使用されます。
- 入力データセットは、マージする前に BY 変数でインデックス付けまたは並べ替えの必要があります。
- SAS のマッチマージは、BY 変数の値がすべて一致し、重複 BY 変数がない場合、PROC SQL の*内部結合*に相当します。詳細については、“内部結合” (*SAS SQL プロシジャユーザーガイド*)を参照してください。

- SAS は、値が欠損(または不一致)の場合でも、プログラムデータベクトル内のすべての変数の値を保持します。

関連項目

- “マッチマージ” (521 ページ)データセット
- “MERGE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “データリレーションシップ” (517 ページ)
- “DATA ステップのマッチマージと PROC SQL 結合の比較” (*SAS SQL プロシジャユーザーガイド*)

例: BY 変数の重複値を持つオブザベーションのマッチマージ

コード例

この例では、MERGE ステートメントを BY ステートメントと組み合わせて使用し、2 つのデータセットをマージします。最初のデータセットのオブザベーションは、共通 BY 変数の値に基づいて、2 番目のデータセットのオブザベーションとマージされます。入力データセットには BY 変数(common)の重複値が含まれているため、これは多対 1 マージの例です。

次の表は、この例で使用される入力データセット animalDupes と plantDupes を示しています。

animalDupes

OBS	common	animal1
1	a	Ant
2	a	Ape
3	b	Bird
4	c	Cat
5	d	Dog
6	e	Eagle

plantDupes

OBS	common	plant1
1	a	Apple
2	b	Banana
3	c	Coconut
4	c	Celery
5	d	Dewberry
6	e	Eggplant

次のプログラムは、まず SORT プロシジャを使用して両方の入力データセットを並べ替え、次に共通変数 common によってデータセットをマッチマージします。PROC PRINT は結果を出力するために使用されます。

```
proc sort data=animalDupes; by common; run;
proc sort data=plantDupes; by common; run;
```

```
data matchmerge;
  merge animalDupes plantDupes;
  by common;
run;
proc print data=matchmerge; run;
```

アウトプット 21.18　BY 変数の重複値を持つオブザベーションのマッチマージ

Obs	common	animal	plant
1	a	Ant	Apple
2	a	Ape	Apple
3	b	Bird	Banana
4	c	Cat	Coconut
5	c	Cat	Celery
6	d	Dog	Dewberry
7	e	Eagle	Eggplant

要点

- マッチマージは、1 つ以上の共通変数を持つデータセットをマージし、共通変数の値に基づいてデータセットをマージするために使用されます。
- 入力データセットは、マージする前に BY 変数でインデックス付けまたは並べ替えの必要があります。
- SAS のマッチマージは、BY 変数の値がすべて一致し、重複 BY 変数がない場合、PROC SQL の*内部結合*に相当します。詳細については、“内部結合” (*SAS SQL プロシジャユーザーガイド*)を参照してください。
- SAS は、値が欠損(または不一致)の場合でも、プログラムデータベクトル内のすべての変数の値を保持します。
- SAS が 1 つのデータセット内の BY グループから最後のオブザベーションを読み取ると、その BY グループのすべてのオブザベーションがすべてのデータセットから読み取られるまで、SAS はそのデータセットに一意のすべての変数の値をプログラムデータベクトルに保持します。最終データセット内のオブザベーションの合計数は、いずれかのデータセットの BY グループ内のオブザベーションの最大数の合計です。

- データセット内に等しくないオブザベーションメンバーがある場合、共通変数は、それらの値を提供するデータセットから値を取得します。データセットの一意の変数は、BY グループの最後までその値を保持します。

関連項目

- “マッチマージ” (521 ページ)データセット
- “MERGE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “データリレーションシップ” (517 ページ)
- “DATA ステップのマッチマージと PROC SQL 結合の比較” (*SAS SQL プロシジャユーザーガイド*)

例: BY 変数の異なる値を持つオブザベーションのマッチマージ

コード例

この例では、MERGE ステートメントを BY ステートメントと組み合わせて使用し、2 つの入力データセットに対してマッチマージを実行します。1 つのデータセットからのオブザベーションは、共通変数によって別のデータセットからのオブザベーションとマージされます。

2 つの入力データセットでは共通変数の値が異なるため、オブザベーションが不一致になります。不一致オブザベーションとは、1 つの入力データセットのオブザベーションに、もう 1 つの入力データセットの BY 変数と同じ値が含まれていないことを意味します。

次の表は、この例で使用される入力データセット animalMissing と plantMissing2 を示しています。

animalMissing

OBS	common	animal1
1	a	Ant
2	c	Cat
3	d	Dog
4	e	Eagle

plantMissing2

OBS	common	plant
1	a	Apple
2	b	Banana
3	c	Coconut
4	e	Eggplant
5	f	Fig

入力データセットでは、データセット animalMissing には共通変数の値"b"または"f"が含まれていませんが、データセット plantMissing2 には含まれています。データセット plantMissing2 には共通変数の値"d"が含まれていませんが、データセット animalMissing には含まれています。

次のプログラムは、まず SORT プロシジャを使用して両方の入力データセットを並べ替え、次に共通変数 common によってデータセットをマッチマージします。PROC PRINT は結果を出力するために使用されます。

```
proc sort data=animalMissing; by common; run;
proc sort data=plantMissing2; by common; run;

data matchmerge;
  merge animalMissing plantMissing2;
  by common;
run;

proc print data=matchmerge; run;
```

1 つのデータセットで値が欠損している場合でも、SAS が最終出力で両方の入力データセットのすべての変数の値を保持する方法に注目してください。

アウトプット 21.19　不一致オブザベーションとのマッチマージの PROC PRINT 出力

Obs	common	animal	plant
1	a	Ant	Apple
2	b		Banana
3	c	Cat	Coconut
4	d	Dog	
5	e	Eagle	Eggplant
6	f		Fig

要点

- マッチマージは、1 つ以上の共通変数を持つデータセットをマージし、共通変数の値に基づいてデータセットをマージするために使用されます。
- 入力データセットは、マージする前に BY 変数でインデックス付けまたは並べ替えの必要があります。
- SAS のマッチマージは、BY 変数の値がすべて一致し、重複 BY 変数がない場合、PROC SQL の*内部結合*に相当します。詳細については、“内部結合” (*SAS SQL プロシジャユーザーガイド*)を参照してください。
- SAS は、値が欠損(または不一致)の場合でも、プログラムデータベクトル内のすべての変数の値を保持します。
- SAS が 1 つのデータセット内の BY グループから最後のオブザベーションを読み取ると、その BY グループのすべてのオブザベーションがすべてのデータセットから読み取られるまで、SAS はそのデータセットに一意のすべての変数の値をプログラ

ムデータベクトルに保持します。最終データセット内のオブザベーションの合計数は、いずれかのデータセットの BY グループ内のオブザベーションの最大数の合計です。

関連項目

- “マッチマージ” (521 ページ)データセット
- “MERGE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “データリレーションシップ” (517 ページ)
- “DATA ステップのマッチマージと PROC SQL 結合の比較” (*SAS SQL プロシジャユーザーガイド*)

例: 不一致オブザベーションのマッチマージおよび削除

コード例

この例では、MERGE ステートメントを BY ステートメントと組み合わせて使用し、2 つのデータセットをマージします。最初のデータセットのオブザベーションは、共通変数の値に基づいて、2 番目のデータセットのオブザベーションとマージされます。

次の例では、MERGE ステートメントと BY ステートメントを使用して、不一致オブザベーションを持つ 2 つのデータセットを結合します。この例は、出力データセットから不一致オブザベーションを削除するために IN=データセットオプションが使用されていることを除き、“例: BY 変数の異なる値を持つオブザベーションのマッチマージ” (572 ページ)と同じです。不一致オブザベーションとは、共有 BY 変数の値が両方の入力データセットで等しくないオブザベーションを指します。

次の表は、この例で使用される入力データセット animalMissing と plantMissing2 を示しています。

animalMissing

OBS	common	animal
1	a	Ant
2	c	Cat
3	d	Dog
4	e	Eagle

plantMissing2

OBS	common	plant
1	a	Apple
2	b	Banana
3	c	Coconut
4	e	Eggplant
5	f	Fig

入力データセットでは、データセット animalMissing には共通変数の値 **b** または **f** が含まれていませんが、データセット plant には含まれています。データセット plantMissing2 には共通変数の値 **d** が含まれていませんが、データセット animal には含まれています。

データセット animalMissing および plantMissing2 には、BY 変数 common のすべての値が含まれているわけではありません。

次のプログラムは、まず SORT プロシジャを使用して両方の入力データセットを並べ替え、次に共通変数 common によってデータセットをマッチマージします。PROC PRINT は結果を出力するために使用されます。

```
proc sort data=animalMissing; by common; run;
proc sort data=plantMissing2; by common; run;

data matchmerge;
  merge animalMissing plantMissing2;
  by common;
run;
proc print data=matchmerge; run;
```

アウトプット 21.20　不一致オブザベーションとのマッチマージ

Obs	common	animal	plant
1	a	Ant	Apple
2	b		Banana
3	c	Cat	Coconut
4	d	Dog	
5	e	Eagle	Eggplant
6	f		Fig

次の例では、プログラムは 2 つのデータセットをマッチマージし、入力データセットで IN=データセットオプションを使用して、出力データセットから不一致オブザベーションを削除します。

IN=データセットオプションはブール値変数であり、データセットが出力の現在のオブザベーションに寄与する場合は値が 1 になり、データセットが出力の現在のオブザベーションに寄与しない場合は値が 0 になります。

```
data matchmerge2;
  merge animalMissing(in=i) plantMissing2(in=j);
  by common;
  if (i=1) and (j=1);
run;
proc print data=matchmerge2; run;
```

アウトプット 21.21　不一致オブザベーションを削除する IN=データセットオプションを使用したマッチマージの PROC PRINT 出力

Obs	common	animal	plant
1	a	Ant	Apple
2	c	Cat	Coconut
3	e	Eagle	Eggplant

要点

- マッチマージは、1 つ以上の共通変数を持つデータセットをマージし、共通変数の値に基づいてデータセットをマージするために使用されます。
- 入力データセットは、マージする前に BY 変数でインデックス付けまたは並べ替えの必要があります。
- SAS のマッチマージは、BY 変数の値がすべて一致し、重複 BY 変数がない場合、PROC SQL の*内部結合*に相当します。詳細については、“内部結合” (*SAS SQL プロシジャユーザーガイド*)を参照してください。
- SAS は、値が欠損(または不一致)の場合でも、プログラムデータベクトル内のすべての変数の値を保持します。
- SAS が 1 つのデータセット内の BY グループから最後のオブザベーションを読み取ると、その BY グループのすべてのオブザベーションがすべてのデータセットから読み取られるまで、SAS はそのデータセットに一意のすべての変数の値をプログラムデータベクトルに保持します。最終データセット内のオブザベーションの合計数は、いずれかのデータセットの BY グループ内のオブザベーションの最大数の合計です。

関連項目

- “マッチマージ” (521 ページ)データセット
- “MERGE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “データリレーションシップ” (517 ページ)
- “DATA ステップのマッチマージと PROC SQL 結合の比較” (*SAS SQL プロシジャユーザーガイド*)

例: 複数のデータセットに重複値があるデータセットのマッチマージ

コード例

この例では、MERGE ステートメントを BY ステートメントと組み合わせて使用し、2 つの入力データセットに対してマッチマージを実行します。1 つのデータセットのオブザベーションは、共通 BY 変数に基づいて別のデータセットのオブザベーションとマージされます。

データセット fruit は、共通変数 ID の値に基づいてデータセット color とマージされます。

fruit

OBS	ID	fruit
1	a	apple
2	c	apricot
3	d	banana
4	e	bluebarry
5	c	cantaloupe
6	c	coconut
7	c	cherry
8	c	crabapple
9	c	cranberry

color

OBS	ID	color
1	a	amber
2	b	brown
3	b	blue
4	b	black
5	b	beige
6	b	bronze
7	c	cocoa
8	c	cream

データセット fruit には重複値(ID の **a** と **c**)があることに注意してください。データセット color には重複値(ID の **b** と **c**)があります。

注: この例では、データセットが BY 変数 ID に基づいて事前に並べ替えまたはインデックス付けされていると想定されています。

```
proc print data=fruit; title 'Fruit'; run;
proc print data=fruit; title 'Color'; run;

data merged;
  merge fruit color;
  by id;
run;

proc print data=merged;
   title 'Merged by ID';
run;
```

アウトプット 21.22　複数の入力データセットに重複が存在する場合に共通変数でオブザベーションをマージするための PROC PRINT 出力

Fruit

Obs	id	fruit
1	a	apple
2	a	apricot
3	b	banana
4	b	blueberry
5	c	cantaloupe
6	c	coconut
7	c	cherry
8	c	crabapple
9	c	cranberry

Color

Obs	id	color
1	a	amber
2	b	brown
3	b	blue
4	b	black
5	b	beige
6	b	bronze
7	c	cocoa
8	c	cream

Merged by ID

Obs	id	fruit	color
1	a	apple	amber
2	a	apricot	amber
3	b	banana	brown
4	b	blueberry	blue
5	b	blueberry	black
6	b	blueberry	beige
7	b	blueberry	bronze
8	c	cantaloupe	cocoa
9	c	coconut	cream
10	c	cherry	cream
11	c	crabapple	cream
12	c	cranberry	cream

BY グループ **c** の変数 fruit の値が異なることに注目してください。BY グループの 2 番目のオブザベーションがデータセット color から読み取られると、PDV 内の値 **cocoa** が **cream** で上書きされます。値 **cream** は BY グループの残りの部分を引き継ぎます。BY ステートメントを MERGE ステートメントと一緒に使用すると、BY グループが変更されるまで変数が欠損に再初期化されないため、値が引き継がれます。

要点

- マッチマージは、1 つ以上の共通変数を持つデータセットをマージし、共通変数の値に基づいてデータセットをマージすることを目的としています。
- 入力データセットは、マージする前に BY 変数でインデックス付けまたは並べ替えの必要があります。
- SAS のマッチマージは、BY 変数の値がすべて一致し、重複 BY 変数がない場合、PROC SQL の*内部結合*に相当します。詳細については、“内部結合” (*SAS SQL プロシジャユーザーガイド*)を参照してください。
- SAS は、値が欠損(または不一致)の場合でも、プログラムデータベクトル内のすべての変数の値を保持します。
- SAS が 1 つのデータセット内の BY グループから最後のオブザベーションを読み取ると、その BY グループのすべてのオブザベーションがすべてのデータセットから読み取られるまで、SAS はそのデータセットに一意のすべての変数の値をプログラムデータベクトルに保持します。最終データセット内のオブザベーションの合計数は、いずれかのデータセットの BY グループ内のオブザベーションの最大数の合計です。

関連項目

- “マッチマージ” (521 ページ)データセット
- “MERGE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “データリレーションシップ” (517 ページ)
- “DATA ステップのマッチマージと PROC SQL 結合の比較” (*SAS SQL プロシジャユーザーガイド*)

例: 共通変数に欠損値がある 1 対多のマッチマージ

コード例

この例では、MERGE ステートメントを BY ステートメントと組み合わせて使用し、1 つのデータセットに欠損データが含まれている場合に、2 つの入力データセットに対してマッチマージを実行します。この例では、欠損値が非欠損値と同じように扱われる方法を示しています。

次の表は、この例で使用される入力データセットを示しています。データセット one には、各 ID の最初の値の後に age の欠損値があることに注意してください。

one

OBS	ID	age	score
1	1	8	90
2	1	.	100
3	1	.	95
4	2	9	80
5	2	.	100

two

OBS	ID	name	age
1	1	Sarah	11
2	2	John	10

これら 2 つのデータセットを ID で単純にマージすると、マージされた出力データセット内の変数 age の値が欠損します。

```
proc print data=one; title 'One'; run;
proc print data=two; title 'Two'; run;

data merge1;
  merge one two;
  by id;
run;
proc print data=merge1; title 'Merged by ID'; run;
```

One

Obs	id	age	score
1	1	8	90
2	1	.	100
3	1	.	95
4	2	9	80
5	2	.	100

Two

Obs	id	name	age
1	1	Sarah	11
2	2	John	10

Merged by ID

Obs	id	age	score	name
1	1	11	90	Sarah
2	1	.	100	Sarah
3	1	.	95	Sarah
4	2	10	80	John
5	2	.	100	John

各データセットの最初のオブザベーションが結合された後、SAS は BY グループの次のオブザベーションのためにデータセット one を読み取ります。PDV 内の値は、age の欠損値も含めて、それらの値で上書きされます。BY グループが完了しているため、データセット two から再度読み取ることはありません。したがって、次のオブザベーションには age の欠損値が含まれています。

データセット two の age の値を取得して欠損値を置き換えるには、IF ステートメントを使用して欠損値をチェックします。age の値が欠損している場合は、一時変数 temp_age が作成され、BY グループの最後の非欠損値が含まれます。age が欠損している場合は、temp_age の値が age の値として使用されます。

```
data merge2 (drop=temp_age);
  merge one two;
  by id;
  retain temp_age;
  if first.id then temp_age = .;
  if age = . then age = temp_age;
  else temp_age = age;
run;
proc print; title 'Merged by ID with Age Retained'; run;
```

One

Obs	id	age	score
1	1	8	90
2	1	.	100
3	1	.	95
4	2	9	80
5	2	.	100

Two

Obs	id	name	age
1	1	Sarah	11
2	2	John	10

Merged by ID with Age Retained

Obs	id	age	score	name
1	1	11	90	Sarah
2	1	11	100	Sarah
3	1	11	95	Sarah
4	2	10	80	John
5	2	10	100	John

要点

- マッチマージは、1 つ以上の共通変数を持つデータセットをマージし、共通変数の値に基づいてデータセットをマージすることを目的としています。
- 入力データセットは、マージする前に BY 変数でインデックス付けまたは並べ替えの必要があります。
- SAS のマッチマージは、BY 変数の値がすべて一致し、重複 BY 変数がない場合、PROC SQL の*内部結合*に相当します。詳細については、“内部結合” (*SAS SQL プロシジャユーザーガイド*)を参照してください。
- SAS は、値が欠損(または不一致)の場合でも、プログラムデータベクトル内のすべての変数の値を保持します。
- SAS が 1 つのデータセット内の BY グループから最後のオブザベーションを読み取ると、その BY グループのすべてのオブザベーションがすべてのデータセットから読み取られるまで、SAS はそのデータセットに一意のすべての変数の値をプログラムデータベクトルに保持します。最終データセット内のオブザベーションの合計数は、いずれかのデータセットの BY グループ内のオブザベーションの最大数の合計です。

関連項目

- “マッチマージ” (521 ページ)データセット
- “MERGE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “データリレーションシップ” (517 ページ)

- “DATA ステップのマッチマージと PROC SQL 結合の比較” (*SAS SQL プロシジャ ユーザーガイド*)

例: データの 1 対 1 マージ

例: オブザベーション数が等しいデータセットのマージ

コード例

この例では、BY ステートメントなしで MERGE ステートメントを使用して、行数が等しい 2 つのデータセットの 1 対 1 マージを実行します。入力データセット animal と plantG に含まれている共通変数の名前は common です。共有変数の値は、6 行目を除いて同じです。6 行目では、値 **f** が animal データセットで **g** が plantG データセットです。

次の表は、この例で使用される入力データセット animal と plantG を示しています。

animal

OBS	common	animal
1	a	Ant
2	b	Bird
3	c	Cat
4	d	Dog
5	e	Eagle
6	f	Frog

plantG

OBS	common	plant
1	a	Apple
2	b	Banana
3	c	Coconut
4	d	Dewberry
5	e	Eggplant
6	g	Fig

次のプログラムはこれらのデータセットをマージし、結果を出力します。

```
data merged;
   merge animal plantG;
run;

proc print data=merged; run;
```

アウトプット 21.23 オブザベーション数が等しいデータセットの 1 対 1 マージの PROC PRINT 出力

Obs	common	animal	plant
1	a	Ant	Apple
2	b	Bird	Banana
3	c	Cat	Coconut
4	d	Dog	Dewberry
5	e	Eagle	Eggplant
6	g	Frog	Fig

出力データセットには、両方の入力データセットのすべての変数が含まれます。出力データセットのオブザベーション 6 の変数 common の値が g であることに注意してください。データセット plantG の共通変数 common の値は、animal データセットの common の値を置き換えます。

要点

- MERGE ステートメントは入力データセット内の共通変数を認識しますが、BY ステートメントがなければ、変数値に基づいてオブザベーションをマージしません。むしろ、変数の値に関係なく、行番号に基づいて暗黙的にオブザベーションを一致させます。
- 入力データセット内の列名が同じで、BY ステートメントが指定されていない場合、マージによって共通列の値が上書きされます。MERGE ステートメントで最後に指定されたデータセット内の共通変数の値は、以前に指定されたデータセット内の値を上書きします。すべての入力データセットで共有されない列名(変数)は、新しい列として追加されます。
- DATA ステップは、最初のデータセットから最初のオブザベーションを読み取り、次に 2 番目のデータセットから最初のオブザベーションを読み取り、以下同様に続きます。
- 結果の出力データセットには、オブザベーションの数が同じかどうかに関係なく、すべての入力データセットのすべてのオブザベーションが含まれます。
- MERGENOBY システムオプションを使用すると、1 対 1 マージを実行する際のログメッセージングを制御できます。

関連項目

- “MERGE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “データリレーションシップ” (517 ページ)

例: オブザベーション数が等しくないデータセットのマージ

コード例

この例では、BY ステートメントなしで MERGE ステートメントを使用して、行数が等しくない 2 つのデータセットの 1 対 1 マージを実行します。

次の表は、この例で使用される入力データセット animal と plantMissing を示しています。

animal

OBS	common	animal
1	a	Ant
2	b	Bird
3	c	Cat
4	d	Dog
5	e	Eagle
6	f	Frog

plantMissing

OBS	common	plant
1	a	Apple
2	b	Banana
3	c	Coconut

データセット animal と plantmissing には両方とも変数 common が含まれており、オブザベーションは common の値によって配置されます。plantmissing データセットのオブザベーションは animal データセットよりも少ないです。

次のプログラムはこれらの等しくないデータセットをマージし、結果を出力します。

```
data animal;
input common $ animal $;
datalines;
a Ant
b Bird
c Cat
d Dog
e Eagle
f Frog
;

data plantMissing;
input common $ plant $;
datalines;
a Apple
b Banana
c Coconut
;

data merged;
   merge animal plantmissing;
```

```
run;
proc print data=merged; run;
```

アウトプット 21.24　オブザベーション数が等しくないデータセットのマージの PROC PRINT 出力

Obs	common	animal	plant
1	a	Ant	Apple
2	b	Bird	Banana
3	c	Cat	Coconut
4	d	Dog	
5	e	Eagle	
6	f	Frog	

前述のプログラムを、SET ステートメントを使用して同じデータセットを 1 対 1 で読み取る場合と比較します。

```
data combine;
  set animal;
  set plantMissing;
run;
proc print data=combine; run;
```

アウトプット 21.25　SET ステートメントを使用した、オブザベーション数が等しくないデータセットの結合の PROC PRINT 出力

Obs	Common	Animal	Plant
1	a	Ant	Apple
2	b	Bird	Banana
3	c	Cat	Coconut

DATA ステップは、オブザベーションの数が最も少ないデータセットの最後のオブザベーションを読み取った後、出力するオブザベーションの選択を停止します。したがって、結果の出力データセット内のオブザベーションの数は、最小の元のデータセット内のオブザベーションの数になります。この例では、DATA ステップは、plantMissing の最後のオブザベーションを読み取った時点でオブザベーションの選択を停止します。

要点

- MERGE ステートメントは入力データセット内の共通変数を認識しますが、BY ステートメントがなければ、変数値に基づいてオブザベーションをマージしません。むしろ、変数の値に関係なく、行番号に基づいて暗黙的にオブザベーションを一致させます。

- 入力データセット内の列名が同じで、BY ステートメントが指定されていない場合、マージによって共通列の値が上書きされます。MERGE ステートメントで最後に指定されたデータセット内の共通変数の値は、以前に指定されたデータセット内の値を上書きします。すべての入力データセットで共有されない列名(変数)は、新しい列として追加されます。
- DATA ステップは、最初のデータセットから最初のオブザベーションを読み取り、次に 2 番目のデータセットから最初のオブザベーションを読み取り、以下同様に続きます。
- 結果の出力データセットには、オブザベーションの数が同じかどうかに関係なく、すべての入力データセットのすべてのオブザベーションが含まれます。
- MERGENOBY システムオプションを使用すると、1 対 1 マージを実行する際のログメッセージングを制御できます。

関連項目

- “MERGE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “MERGE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “データリレーションシップ” (517 ページ)

例: 共通変数の重複値を持つデータセットのマージ

コード例

次の例は、共通変数の重複値を含むデータセットをマージすると、望ましくない結果が生じる可能性があることを示しています。この場合、BY 変数を指定せずにマージが行われます。このタイプのマージは、1 対 1 リレーションシップを持つデータセットにのみ使用してください。この例の入力データセットには 1 対 1 リレーションシップがありません。

次の表は、この例で使用される入力データセット animalDupes と plantDupes を示しています。

animalDupes

OBS	common	animal
1	a	Ant
2	a	Ape
3	b	Bird
4	c	Cat
5	d	Dog
6	e	Eagle

plantDupes

OBS	common	plant
1	a	Apple
2	b	Banana
3	c	Coconut
4	c	Celery
5	d	Dewberry
6	e	Eggplant

データセット animalDupes と plantDupes には変数 common が含まれており、各データセットには common の重複値を持つオブザベーションが含まれています。

次のプログラムは、データセットをマージし、結果を出力します。

```
/* This program illustrates undesirable results. */
data animalDupes;
input common $ animal $;
datalines;
a Ant
a Ape
b Bird
c Cat
d Dog
e Eagle
;

data plantDupes;
input common $ plant $;
datalines;
a Apple
b Banana
c Coconut
c Celery
d Dewberry
e Eggplant
;

data merged;
  merge animalDupes plantDupes;
run;
proc print data=merged; run;
```

アウトプット 21.26　共通変数の重複値を持つデータセットをマージする場合の望ましくない結果に対する PROC PRINT 出力

Obs	common	animal	plant
1	a	Ant	Apple
2	b	Ape	Banana
3	c	Bird	Coconut
4	c	Cat	Celery
5	d	Dog	Dewberry
6	e	Eagle	Eggplant

この方法は、1 対 1 リレーションシップを持つデータに対して期待どおりに機能します。この例のデータのリレーションシップは 1 対多と多対 1 の両方のリレーションシップを持っているため、望ましくない結果が生じる可能性があります。

要点

- MERGE ステートメントは入力データセット内の共通変数を認識しますが、BY ステートメントがなければ、変数値に基づいてオブザベーションをマージしません。むしろ、変数の値に関係なく、行番号に基づいて暗黙的にオブザベーションを一致させます。
- 入力データセット内の列名が同じで、BY ステートメントが指定されていない場合、マージによって共通列の値が上書きされます。MERGE ステートメントで最後に指定されたデータセット内の共通変数の値は、以前に指定されたデータセット内の値を上書きします。すべての 入力データセットで共有されない列名(変数)は、新しい列として追加されます。
- DATA ステップは、最初のデータセットから最初のオブザベーションを読み取り、次に 2 番目のデータセットから最初のオブザベーションを読み取り、以下同様に続きます。
- 結果の出力データセットには、オブザベーションの数が同じかどうかに関係なく、すべての 入力データセットのすべてのオブザベーションが 含まれます。
- MERGENOBY システムオプションを使用すると、1 対 1 マージを実行する際のログメッセージングを制御できます。

関連項目

- “MERGE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “データリレーションシップ” (517 ページ)

例: 共通変数の値が異なるデータセットのマージ

コード例

次の例は、共通変数の値が異なるデータセットを 1 対 1 のマージ使用して結合した場合に得られる望ましくない結果を示しています。

この例では、データセット animalMissing と plantMissing2 の変数 common の値が異なります。

次の表は、この例で使用される入力データセット animalMissing と plantMissing2 を示しています。

animalMissing	plantMissing2

OBS	common	animal
1	a	Ant
2	c	Cat
3	d	Dog
4	e	Eagle

OBS	common	plant
1	a	Apple
2	b	Banana
3	c	Coconut
4	e	Eggplant
5	f	Fig

次のプログラムはデータセット merged を生成し、結果を出力します。

```
data merged;
  merge animalMissing plantMissing2;
run;
proc print data=merged; run;
```

アウトプット 21.27　共通変数の値が異なるデータセットの 1 対 1 マージによる望ましくない結果を示す PROC PRINT 出力

Obs	common	animal	plant
1	a	Ant	Apple
2	b	Cat	Banana
3	c	Dog	Coconut
4	e	Eagle	Eggplant
5	f		Fig

要点

- MERGE ステートメントは入力データセット内の共通変数を認識しますが、BY ステートメントがなければ、変数値に基づいてオブザベーションをマージしません。むしろ、変数の値に関係なく、行番号に基づいて暗黙的にオブザベーションを一致させます。
- 入力データセット内の列名が同じで、BY ステートメントが指定されていない場合、マージによって共通列の値が上書きされます。MERGE ステートメントで最後に指定されたデータセット内の共通変数の値は、以前に指定されたデータセット内の値を上書きします。すべての入力データセットで共有されない列名(変数)は、新しい列として追加されます。
- DATA ステップは、最初のデータセットから最初のオブザベーションを読み取り、次に 2 番目のデータセットから最初のオブザベーションを読み取り、以下同様に続きます。
- 結果の出力データセットには、オブザベーションの数が同じかどうかに関係なく、すべての入力データセットのすべてのオブザベーションが含まれます。

- MERGENOBY システムオプションを使用すると、1 対 1 マージを実行する際のログメッセージングを制御できます。

関連項目

- “MERGE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “データリレーションシップ” (517 ページ)

例: データの 1 対 1 結合

例: オブザベーション数が等しいデータセットの結合

コード例

この例では、2 つの SET ステートメントを使用して、1 つのデータセットからのオブザベーションを別のデータセットからのオブザベーションと結合します。入力データセット animal および plantG には、common.という名前の共通変数が含まれています。共有変数の値は、6 行目を除いて同じです。6 行目の値は、animal データセットでは f、plantG データセットでは g です。

次の表は、この例で使用される入力データセット animal と plantG を示しています。

animal

OBS	common	animal
1	a	Ant
2	b	Bird
3	c	Cat
4	d	Dog
5	e	Eagle
6	f	Frog

plantG

OBS	common	plant
1	a	Apple
2	b	Banana
3	c	Coconut
4	d	Dewberry
5	e	Eggplant
6	g	Fig

次のプログラムは、データセットを結合し、結果を出力します。

```
data combine;
  set animal;
```

```
  set plantG;
run;

proc print data=combine; run;
```

アウトプット 21.28　SET ステートメントを使用した、オブザベーション数が等しいデータセットの結合の PROC PRINT 出力

Obs	common	animal	plant
1	a	Ant	Apple
2	b	Bird	Banana
3	c	Cat	Coconut
4	d	Dog	Dewberry
5	e	Eagle	Eggplant
6	g	Frog	Fig

SET ステートメントは共通変数の値を一致させることによってオブザベーションをマージしないため、類似の名前を持つ変数に異なる値を持つデータセットでこの方法を使用すると、望ましくない結果が発生する可能性があります。

この例では、animal データセットの 6 行目に、類似の名前を持つ変数 common の値 f があります。plantG データセットの 6 行目の common の値は g です。データセット plantG は最後の SET ステートメントで指定されているため、plantG データセットの common の値は animal データセットの common の値を上書きします。

要点

- SET ステートメントで最後に指定されたデータセットの共通変数の値は、以前のデータセットの共通変数の値を置き換えます。
- DATA ステップは、最初のデータセットから最初のオブザベーションを読み取り、次に 2 番目のデータセットから最初のオブザベーションを読み取り、以下同様に続きます。
- 結果の出力データセットには、すべての入力データセットのすべての変数が含まれます。
- DATA ステップは、オブザベーションの数が最も少ないデータセットの最後のオブザベーションを読み取った後、出力するオブザベーションの選択を停止します。したがって、結果の出力データセット内のオブザベーションの数は、最小の元のデータセット内のオブザベーションの数になります。
- SET ステートメントを使用してオブザベーションの数が等しくないデータセットを結合するには、POINT=オプションを使用して、共通変数によってオブザベーションに直接アクセスして一致させます。

関連項目

- KEY=データセットオプション
- “1 オブザベーションと複数オブザベーションを結合する” (*SAS DATA ステップステートメント: リファレンス*)
- “マスターファイルに重複するオブザベーションが含まれる場合、テーブルルックアップを実行する” (*SAS DATA ステップステートメント: リファレンス*)

例: ハッシュテーブルを使用したデータのマージ

例: ハッシュテーブルを使用してデータセットを 1 対多または多対多でマージする

コード例

この例では、ハッシュテーブルを使用して、共通変数を持つ 2 つのデータセットをマージします。ハッシュテーブルは、メモリにロードされた SAS データセットから作成され、それを作成した DATA ステップで使用できます。ハッシュテーブル内の共通変数は一意であり、内部メモリテーブルへの非常に高速なルックアップを提供するキーとして使用されます。

2 番目のデータセットはベースデータセットとして使用されます。DATA ステップは、ベースデータセットからオブザベーションを読み取り、共通変数を使用してハッシュテーブル内の一致を検索します。一致オブザベーションは、DATA ステートメントで指定された出力データセットに書き出されます。

次の表は、入力データセット product_list と supplier を示しており、各データセットで共通変数 Supplier_ID が強調表示されています。

product_list

Product_Id	Product_Name	Supplier_ID
240200100101	Grandslam Staff Tour Mhl Golf Gloves	3808
210200100017	Sweatshirt Children's O-Neck	3298
240400200022	Aftm 95 Vf Long Bg-65 White	1280
230100100017	Men's Jacket Rem	50

```
210200300006 Fleece Cuff Pant Kid'S            1303
210200500002 Children's Mitten                 772
210200700016 Strap Pants BBO                   798
210201000050 Kid Children's T-Shirt            2963
210200100009 Kids Sweat Round Neck,Large Logo  3298
210201000067 Logo Coord.Children's Sweatshirt  2963
220100100019 Fit Racing Cap                    1303
220100100025 Knit Hat                          1303
220100300001 Fleece Jacket Compass             772
220200200036 Soft Astro Men's Running Shoes    1747
230100100015 Men's Jacket Caians               50
230100500004 Backpack Flag, 6,5x9 Cm.          316
210200500006 Rain Suit, Plain w/backpack Jacket 772
230100500006 Collapsible Water Can             316
224040020000 Bat 5-Ply                         3808
220200200035 Soft Alta Plus Women's Indoor Shoes 1747
240400200066 Memhis 350,Yellow Medium, 6-pack  1280
240200100081 Extreme Distance 90 3-pack        3808
```

supplier

```
Supplier_ID Supplier_Name            Supplier_Address    Country
50          Scandinavian Clothing A/S  Kr. Augusts Gate 13 NO
316         Prime Sports Ltd         9 Carlisle Place    GB
755         Top Sports               Jernbanegade 45     DK
772         AllSeasons Outdoor Clothing 553 Cliffview Dr  US
798         Sportico                 C. Barquillo 1      ES
1280        British Sports Ltd       85 Station Street   GB
1303        Eclipse Inc              1218 Carriole Ct    US
1684        Magnifico Sports         Rua Costa Pinto 2   PT
1747        Pro Sportswear Inc       2434 Edgebrook Dr   US
3298        A Team Sports            2687 Julie Ann Ct   US
3808        Carolina Sports          3860 Grand Ave      US
```

このプログラムでは、product_list データセットがベースデータセットとして使用されます。DECLARE ステートメントは、supplier データセットのインメモリの場所を指定します。DEFINEKEY メソッドは、データセットを結合するために使用される一意のキー変数を識別します。DEFINEDATA メソッドは、メモリにロードする追加の変数をリストします。

注: ベースデータセットにはキーの重複が含まれています。オブザベーションは一度に 1 つずつ読み取られて処理され、重複値は処理に影響しません。

```
data supplier_info;
  drop rc;
  length Supplier_Name $40 Supplier_Address $ 45 Country $ 2;       /* 1 */
  if _N_=1 then do;
    declare hash S(dataset:'work.supplier');                        /* 2 */
    S.definekey('Supplier_ID');
    S.definedata('Supplier_Name',
                 'Supplier_Address','Country');
    S.definedone();
    call missing(Supplier_Name,
```

```
                Supplier_Address,Country);                  /* 3 */
  end;
  set work.product_list;                                /* 4 */
  rc=S.find();                                        /* 5 */
run;
proc print data=supplier_info;
  var Product_ID Supplier_ID Supplier_Name
     Supplier_Address Country;
  title "Product Information";
run;
title;
```

1 LENGTH ステートメントを含めて、Supplier_Name、Supplier_Address、および Country が PDV で定義されていることを確認します。

2 DATA ステップの最初の反復で、ハッシュオブジェクト S を宣言します。Supplier_ID をハッシュオブジェクトキーとして割り当てます。work.supplier データセットの Supplier_Name、Supplier_Address、および Country の値を含めます。

3 Supplier_Name、Supplier_Address、および Country には初期値が明示的に割り当てられていないため、SAS は変数が初期化されていないという NOTE をログに書き込みます。CALL MISSING は、ログに書き込まれる NOTE を抑制します。

4 product_list データセットからオブザベーションを読み取ります。

5 FIND メソッドは、product_list の Supplier_ID がハッシュオブジェクト内のレコードの 1 つの Supplier_ID キーと一致するかどうかを確認するために呼び出されます。一致する場合(rc=0)、オブザベーションは supplier_info データセットに書き込まれます。

Product Information

Obs	Product_Id	Supplier_ID	Supplier_Name	Supplier_Address	Country
1	240200100101	3808	Carolina Sports	3860 Grand Ave	US
2	210200100017	3298	A Team Sports	2687 Julie Ann Ct	US
3	240400200022	1280	British Sports Ltd	85 Station Street	GB
4	230100100017	50	Scandinavian Clothing A/S	Kr. Augusts Gate 13	NO
5	210200300006	1303	Eclipse Inc	1218 Carriole Ct	US
6	210200500002	772	AllSeasons Outdoor Clothing	553 Cliffview Dr	US
7	210200700016	798	Sportico	C. Barquillo 1	ES
8	210200100009	3298	A Team Sports	2687 Julie Ann Ct	US
9	220100100019	1303	Eclipse Inc	1218 Carriole Ct	US
10	220100100025	1303	Eclipse Inc	1218 Carriole Ct	US
11	220100300001	772	AllSeasons Outdoor Clothing	553 Cliffview Dr	US
12	220200200036	1747	Pro Sportswear Inc	2434 Edgebrook Dr	US
13	230100100015	50	Scandinavian Clothing A/S	Kr. Augusts Gate 13	NO
14	230100500004	316	Prime Sports Ltd	9 Carlisle Place	GB
15	210200500006	772	AllSeasons Outdoor Clothing	553 Cliffview Dr	US
16	230100500006	316	Prime Sports Ltd	9 Carlisle Place	GB
17	224040020000	3808	Carolina Sports	3860 Grand Ave	US
18	220200200035	1747	Pro Sportswear Inc	2434 Edgebrook Dr	US
19	240400200066	1280	British Sports Ltd	85 Station Street	GB
20	240200100081	3808	Carolina Sports	3860 Grand Ave	US

要点

- SAS ハッシュテーブルには行(ハッシュエントリ)と列(ハッシュ変数)が含まれます。
- 各ハッシュエントリには、少なくとも 1 つのキー列と 1 つのデータ列が必要です。値はハードコーディングすることも、SAS データセットからロードすることもできます。
- ハッシュテーブルは完全にメモリ内に常駐するため、操作が高速になります。データを事前に並べ替える必要はありません。
- ハッシュテーブルは一時的なものであり、DATA ステップの実行が停止すると存在しなくなります。したがって、後続のステップで再利用することはできません。ただし、そのコンテンツは SAS データセットまたは外部データベースに保存できます。
- ハッシュオブジェクトのサイズは動的に変更されます。

関連項目

- “OUTPUT メソッド” (*SAS コンポーネントオブジェクト: リファレンス*)
- SAS®ハッシュオブジェクトの基礎
- 重複キーエントリを持つ SAS®ハッシュオブジェクトの使用

例: データの更新

例: UPDATE ステートメントを使用したデータの更新

コード例

この例では、UPDATE ステートメントを使用して、トランザクションデータセットの新しい値に基づいてマスターデータセットを更新します。データセット master には、共有変数 common と plant の元の値が含まれています。トランザクションデータセット plantNew には、共有変数 plant の新しい値が含まれています。目標は、plant の新しい値でマスターデータセットを更新することです。具体的には、5 行目の plant の値 **Eggplant** を、plantNew データセットの値 **Escarole** に置き換える必要があります。

次の表は、入力データセット master とトランザクションデータセット plantNew を示しています。

master

OBS	common	animal	plant
1	a	Ant	Apple
2	b	Bird	Banana
3	c	Cat	Coconut
4	d	Dog	Dewberry
5	e	Eagle	Eggplant
6	f	Frog	Fig

plantNew

OBS	common	plant
1	a	Apricot
2	b	Barley
3	c	Cactus
4	d	Date
5	e	Escarole
6	f	Fennel

プログラムはまず 2 つの入力データセットを作成し、次に BY 変数 common の値に基づいてマスターデータセットを更新します。PRINT プロシジャは結果を出力します:

```
data master2;
  update master plantNew;
  by common;
run;
proc print data=master2; run;
```

アウトプット 21.29　UPDATE ステートメントを使用してデータを更新する場合の PROC PRINT 出力

Obs	common	animal	plant
1	a	Ant	Apricot
2	b	Bird	Barley
3	c	Cat	Cactus
4	d	Dog	Date
5	e	Eagle	Escarole
6	f	Frog	Fennel

要点

- UPDATE ステートメントを使用すると、別のデータセット(トランザクションデータセット)の値に基づいて、1 つのデータセット(マスターデータセット)の値を更新できます。UPDATE ステートメントは新しい出力データセットを作成します。
- BY ステートメントは必須であり、入力データセットにはインデックスが含まれているか、BY 変数の値によって並べ替えられている必要があります。
- UPDATE ステートメントは出力を生成するときに新しいファイルを作成するため、更新を実行するときに変数を追加、削除、または名前変更できます。
- デフォルトでは、トランザクションデータセット内の欠損値は、マスターデータセット内の既存の値を置き換えません。NOMISSINGCHECK を UPDATEMODE=オプションで指定することにより、トランザクションデータセット内の欠損値がマスターデータセット内の値を置き換えるようにこの動作を変更できます。
- トランザクションデータセットに BY 変数の重複値が含まれている場合、トランザクションデータセットの値がマスターデータセットの値を置き換えます。
- トランザクションデータセット内のオブザベーションに対応するオブザベーションがマスターデータセット内にない場合、SAS はマスター出力データセットにオブザベーションを追加します。オブザベーションは BY 変数の値に基づいて一致させます。
- マスターデータセット内のオブザベーションに変更を加える必要がない場合は、そのオブザベーションをトランザクションデータセットに含める必要はありません。

関連項目

- “更新” (525 ページ)
- 表 21.2 (529 ページ)
- “UPDATE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “SORT プロシジャ” (*Base SAS プロシジャガイド*)
- *SAS DATA ステップステートメント: リファレンス*の UPDATEMODE=

例: BY 変数の重複値を持つデータセットの更新

コード例

この例では、UPDATE ステートメントを使用して、トランザクションデータセットの新しい値に基づいてマスターデータセットを更新します。トランザクションデータセットには、オブザベーション 4 と 5 の common 変数の重複値が含まれています。データセットは共通変数 plant も共有しています。この例では、BY 変数に重複値がある場合に、BY 変数以外の共通変数に何が起こるかを示します。

次の表は、マスター入力データセット master とトランザクションデータセット plantNewDupes を示しています。

master

Obs	common	animal	plant
1	a	Ant	Apple
2	b	Bird	Banana
3	c	Cat	Coconut
4	d	Dog	Dewberry
5	e	Eagle	Eggplant
6	f	Frog	Fig

plantNewDupes

Obs	common	plant
1	a	Apricot
2	b	Barley
3	c	Cactus
4	d	Date
5	d	Dill
6	e	Escarole
7	f	Fennel

次のプログラムは、まず 2 つの入力データセットを作成し、次に BY 変数 common の値に基づいてマスターデータセットを更新します。これはトランザクションデータセットに BY 変数 common の重複値が含まれているためです。BY 変数として指定されていないため、他の共通変数 plant の値はトランザクションデータセット内の値に置き換えられます。マスターデータセット内の値 Dewberry は、トランザクションデータセット内の plant の最後の値である Dill に置き換えられます。PRINT プロシジャは結果を出力します:

```
data master;
```

```
  update master plantNewDupes;
  by common;
run;
proc print data=master; run;
```

注意
BY 変数の値は、マスターデータセット内のオブザベーションごとに一意である必要があります。マスターデータセットに BY 変数の重複値が含まれている場合、その変数を含む最初のオブザベーションのみが更新され、その変数を含む後続のオブザベーションは無視されます。SAS は、DATA ステップの実行時に警告メッセージをログに書き込みます。

アウトプット 21.30　BY 変数の値が重複しているデータセットを更新する場合の PROC PRINT 出力

Obs	common	animal	plant
1	a	Ant	Apricot
2	b	Bird	Barley
3	c	Cat	Cactus
4	d	Dog	Dill
5	e	Eagle	Escarole
6	f	Frog	Fennel

要点

- UPDATE ステートメントを使用すると、別のデータセット(トランザクションデータセット)の値に基づいて、1 つのデータセット(マスターデータセット)の値を更新できます。UPDATE ステートメントは新しい出力データセットを作成します。
- BY ステートメントは必須であり、入力データセットにはインデックスが含まれているか、BY 変数の値によって並べ替えられている必要があります。
- UPDATE ステートメントは出力を生成するときに新しいファイルを作成するため、更新を実行するときに変数を追加、削除、または名前変更できます。
- デフォルトでは、トランザクションデータセット内の欠損値は、マスターデータセット内の既存の値を置き換えません。NOMISSINGCHECK を UPDATEMODE=オプションで指定することにより、トランザクションデータセット内の欠損値がマスターデータセット内の値を置き換えるようにこの動作を変更できます。
- トランザクションデータセットに BY 変数の重複値が含まれている場合、トランザクションデータセットの値がマスターデータセットの値を置き換えます。
- トランザクションデータセット内のオブザベーションに対応するオブザベーションがマスターデータセット内にない場合、SAS はマスター出力データセットにオブザベーションを追加します。オブザベーションは BY 変数の値に基づいて一致させます。
- マスターデータセット内のオブザベーションに変更を加える必要がない場合は、そのオブザベーションをトランザクションデータセットに含める必要はありません。

関連項目

- “更新” (525 ページ)
- 表 21.2 (529 ページ)
- “UPDATE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “SORT プロシジャ” (*Base SAS プロシジャガイド*)
- *SAS DATA ステップステートメント: リファレンス*の UPDATEMODE=

例: BY 変数の欠損値と異なる値を持つデータセットの更新

コード例

この例では、UPDATE ステートメントを BY ステートメントと組み合わせて使用し、トランザクションデータセットの新しい値に基づいてマスターデータセットを更新します。

マスターデータセット master には、最初のオブザベーションの変数 plant に欠損値が含まれています。共通 BY 変数(common)のすべての値が含まれているわけではありません。

トランザクションデータセット minerals には、新しい変数(**mineral**)、BY 変数 common の新しい値、およびいくつかのオブザベーションの欠損値が含まれています。

次の表は、入力データセット master と minerals を示しています。

master

OBS	common	animal	plant
1	a	Ant	.
2	c	Cat	Coconut
3	d	Dog	Dewberry
4	e	Eagle	Eggplant
5	f	Frog	Fig

minerals

OBS	common	plant	mineral
1	a	Apricot	Amethyst
2	b	Barley	Beryl
3	c	Cactus	.
4	e	.	.
5	f	Fennel	.
6	g	Grape	Garnet

次のプログラムは、minerals データセットの値に基づいて master データセットを更新し、結果を出力します。

```
data master;
  update master minerals;
  by common;
run;

proc print data=master; run;
```

アウトプット 21.31　不一致オブザベーション、欠損値、および新しい変数を処理するために UPDATE を使用する場合の PROC PRINT 出力

Obs	common	animal	plant	mineral
1	a	Ant	Apricot	Amethyst
2	b		Barley	Beryl
3	c	Cat	Cactus	
4	d	Dog	Dewberry	
5	e	Eagle	Eggplant	
6	f	Frog	Fennel	
7	g		Grape	Garnet

更新されたマスターデータセットについては、次の点に注意してください。

- 変数 mineral はマスター出力データセットに追加され、一部のオブザベーションでは欠損に設定されています。
- トランザクションデータセット内のオブザベーション 2 と 6 の値には、マスターデータセット内に対応する値がありません。それらは新しいオブザベーションとしてマスターデータセットに追加されます。
- オブザベーション 4 の plant の値は、トランザクションデータセットでは欠損しているにもかかわらず、欠損に変更されません。
- 新しいデータセット内の 3 つのオブザベーションには、変数 plant の更新された値が含まれています。

トランザクションデータセットで欠損している値をマスター出力データセットで欠損として更新する場合は、UPDATE ステートメントで UPDATEMODE=オプションを指定します。

```
data master;
  update master minerals updatemode=nomissingcheck;
  by common;
run;
proc print data=master;
  title "Updated Data Set master";
  title2 "With Values Updated to Missing";
run;
```

次の PROC PRINT 出力では、オブザベーション 5 の plant の値がトランザクションデータセットで欠損しており、UPDATEMODE=NOMISSINGCHECK オプションが有効になっているため、欠損に設定されています。

アウトプット 21.32　値が欠損に更新された更新されたマスターデータセットの PROC PRINT 出力

Updated Data Set master
With Values Updated to Missing

Obs	common	animal	plant	mineral
1	a	Ant	Apricot	Amethyst
2	b		Barley	Beryl
3	c	Cat	Cactus	
4	d	Dog	Dewberry	
5	e	Eagle		
6	f	Frog	Fennel	
7	g		Grape	Garnet

要点

- UPDATE ステートメントを使用すると、別のデータセット(トランザクションデータセット)の値に基づいて、1 つのデータセット(マスターデータセット)の値を更新できます。UPDATE ステートメントは新しい出力データセットを作成します。
- BY ステートメントは必須であり、入力データセットにはインデックスが含まれているか、BY 変数の値によって並べ替えられている必要があります。
- UPDATE ステートメントは出力を生成するときに新しいファイルを作成するため、更新を実行するときに変数を追加、削除、または名前変更できます。
- デフォルトでは、トランザクションデータセット内の欠損値は、マスターデータセット内の既存の値を置き換えません。NOMISSINGCHECK を UPDATEMODE=オプションで指定することにより、トランザクションデータセット内の欠損値がマスターデータセット内の値を置き換えるようにこの動作を変更できます。
- トランザクションデータセットに BY 変数の重複値が含まれている場合、トランザクションデータセットの値がマスターデータセットの値を置き換えます。
- トランザクションデータセット内のオブザベーションに対応するオブザベーションがマスターデータセット内にない場合、SAS はマスター出力データセットにオブザベーションを追加します。オブザベーションは BY 変数の値に基づいて一致させます。
- マスターデータセット内のオブザベーションに変更を加える必要がない場合は、そのオブザベーションをトランザクションデータセットに含める必要はありません。

関連項目

- “更新” (525 ページ)

- 表 21.2 (529 ページ)
- “UPDATE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “SORT プロシジャ” (*Base SAS プロシジャガイド*)
- *SAS DATA ステップステートメント: リファレンス*の UPDATEMODE=

例: データの変更

例: オブザベーションを追加してデータセットを変更する

コード例

この例では、MODIFY ステートメントを使用して、トランザクションデータセットに含まれる値に基づいてマスターデータセットを更新します。トランザクションデータセット内のオブザベーションは、共通変数 partNumber の値を一致させることによって、マスターデータセット内のオブザベーションと一致させます。

この例のデータは、工具や金物を保管する倉庫の在庫を表しています。各工具は部品番号によって一意に識別されます。マスターデータセット Inventory には、倉庫の在庫の記録が保持されます。倉庫に新しいアイテムの出荷が届いたときに、変更を反映するように更新されます。InventoryAdd データセットは *トランザクション*データセットです。トランザクションデータセットには、在庫に追加される新しいアイテム(新しい種類の工具)に関する情報が含まれます。また、トランザクションデータセットは、在庫(newStock)を既存のアイテムに追加し、既存のアイテムの価格(newPrice)を変更します。

次の表は、マスター入力データセット Inventory とトランザクションデータセット InventoryAdd を示しています。

Inventory

partNumber	partName	stock	price	receivedDate
K89R	seal	34	245.00	07jul2015
M4J7	sander	98	45.88	20jun2015
LK43	filter	121	10.99	19may2016
MN21	brace	43	27.87	10aug2016
BC85	clamp	80	9.55	16aug2016
NCF3	valve	198	24.50	20mar2016
KJ66	cutter	6	19.77	18jun2016
UYN7	rod	211	11.55	09sep2016
JD03	switch	383	13.99	09jan2017
BV1E	timer	26	34.50	03aug2017

InventoryAdd

partNumber	partName	newStock	newPrice
AA11	hammer	55	32.26
BB22	wrench	21	17.35
BV1E	timer	30	36.50
CC33	socket	7	22.19
K89R	seal	6	247.50
KJ66	cutter	10	24.50

この例を開始するには、まず比較のために Inventory および InventoryAdd データセットを並べ替えて出力します。

```
proc sort data=Inventory; by partNumber; run;
proc sort data=InventoryAdd; by partNumber; run;
proc print data=Inventory; title "Inventory"; run;
proc print data=InventoryAdd; title "InventoryAdd"; run;
```

注: MODIFY ステートメントを使用してデータセットを変更する場合、SORT プロシジャは必要ありません。この例のデータセットは、2 つのデータセット間の違いをよりわかりやすく示すために並べ替えられています。

アウトプット 21.33　partNumber で並べ替えられたマスター Inventory データセットの PROC PRINT 出力

Inventory

Obs	partNumber	partName	stock	price	receivedDate
1	BC85	clamp	80	$9.55	08/16/2017
2	BV1E	timer	26	$34.50	08/03/2018
3	JD03	switch	383	$13.99	01/09/2018
4	K89R	seal	34	$245.00	07/07/2018
5	KJ66	cutter	6	$19.77	06/18/2018
6	LK43	filter	121	$10.99	05/19/2018
7	M4J7	sander	98	$45.88	06/20/2018
8	MN21	brace	43	$27.87	08/10/2018
9	NCF3	valve	198	$24.50	03/20/2017
10	UYN7	rod	211	$11.55	09/09/2017

アウトプット 21.34　partNumber で並べ替えられたトランザクションデータセット InventoryAdd の PROC PRINT 出力

InventoryAdd

Obs	partNumber	partName	newStock	newPrice
1	AA11	hammer	55	$32.26
2	BB22	wrench	21	$17.35
3	BV1E	timer	30	$36.50
4	CC33	socket	7	$22.19
5	K89R	seal	6	$247.50
6	KJ66	cutter	10	$24.50

トランザクションデータセット InventoryAdd のオブザベーション 1、2、および 4 がマスターデータセット Inventory には存在しないことに注意してください。また、トランザクションデータセット内の変数 newStock および newPrice の値に新しい値が含まれていることにも注意してください。

ここで、トランザクションデータセットの新しい情報に基づいてマスターデータセットを変更し、変数 partNumber の一意の値によってオブザベーションを一致させます。

```
data Inventory;
  modify Inventory InventoryAdd;                    /* 1 */
    by partNumber;
    select (_iorc_);                                /* 2 */
  /*** The observation exists in the master data set   */
       when (%sysrc(_sok))do;                       /* 3 */
         stock = stock + newStock;
         price=newPrice;
         receivedDate = today();
         replace;                                   /* 4 */
       end;
  /*** The observation does not exist in the master data set*/
       when (%sysrc(_dsenmr)) do;                   /* 5 */
         stock=newStock;
         price=newPrice;
         receivedDate=today();
         output;                                    /* 6 */
         _error_=0;
       end;
     otherwise do;                                  /* 7 */
       put "An unexpected I/O error has occurred."
       _error_ = 0;
       stop;
     end;
  end;
run;
proc sort data=Inventory;
  by partNumber;
run;
proc print data=Inventory;
```

```
  title "Modified Inventory Data Set Sorted by partNumber";
run;
quit;
```

1 MODIFY ステートメントは、マスターデータセットとトランザクションデータセットからデータをロードします。BY ステートメントは、変数 partNumber の一意の値に基づいて各データセットのオブザベーションを一致させます。

2 トランザクションデータセットの partNumber に一致するものがマスターデータセットの partNumber に見つかった場合、_IORC_自動変数は自動的に_SOK のコードに設定されます。

3 %SYSRC 自動呼び出しマクロは、_IORC_の値が_SOK であるかどうかをチェックします。値が_SOK の場合、SELECT ステートメントは最初の DO ステートメントブロックを実行します。トランザクションデータセット内のオブザベーションはマスターデータセット内のオブザベーションと一致するため、オブザベーションの値は置換によって更新できます。

4 REPLACE ステートメントは、マスターデータセットのオブザベーションをトランザクションデータセットのオブザベーションで置き換えることによって、マスターデータセットを更新します。REPLACE ステートメントは、オブザベーション 4、7、および 8 (出力では青で強調表示されています)を stock と price の新しい値で更新します。stock 値は、トランザクションデータセット内の newStock の値に基づいて更新されます。price 値は、トランザクションデータセット内の newPrice の値に基づいて更新されます。これらのオブザベーションの receivedDate 値は、過去に受信した既存のアイテムであるため更新されません。

5 トランザクションデータセットの partNumber とマスターデータセットの partNumber の一致が見つからない場合、_IORC_自動変数は自動的にコード _DSENMR に設定されます。これは、一致が見つからなかったことを意味します。%SYSRC 自動呼び出しマクロは、_IORC_の値が_DSENMR であるかどうかをチェックします。値が_DSENMR の場合、SELECT ステートメントは 2 番目の DO ブロックを実行します。トランザクションデータセット内のオブザベーションはマスターデータセットに存在しないため、値を単純に置き換えることはできません。オブザベーション全体が作成され、マスターデータセットに追加されます。

6 OUTPUT ステートメントは、新しいオブザベーションをマスターデータセットに書き込みます。OUTPUT ステートメントは、オブザベーション 1、2、および 5 をマスターデータセットに追加します(出力で黄色で強調表示されたオブザベーションを参照)。これらのオブザベーションの receivedDate 値は、TODAY 関数の戻り値に基づいて更新されます。

7 どちらの条件も満たされない場合、OTHERWISE ステートメントは最後の DO ブロックを実行し、PUT ステートメントはエラーメッセージをログに書き込みます。

次の出力では、トランザクションデータセットに、**hammer**、**wrench**、**socket** という 3 つの新しいアイテムが含まれています。一部のオブザベーションはマスターデータセットに存在せず、トランザクションデータセットから追加されるため、明示的な OUTPUT ステートメントが必要です。マスターデータセットにすでに存在するオブザベーションについては、これらのオブザベーションの値を更新するために REPLACE ステートメントが必要です。

プログラムは、OUTPUT ステートメントを使用してオブザベーション 1、2、および 5 をマスターデータセットに追加し、REPLACE ステートメントを使用してオブザベーション 4、7、および 8 を stock と price の新しい値で更新します。

アウトプット 21.35　partNumber で並べ替えられた変更済み Inventory マスター データセットの PROC PRINT 出力

Modified Inventory Data Set Sorted by partNumber

Obs	partNumber	partName	stock	price	receivedDate
1	AA11	hammer	55	$32.26	01/25/2019
2	BB22	wrench	21	$17.35	01/25/2019
3	BC85	clamp	80	$9.55	08/16/2017
4	BV1E	timer	56	$36.50	01/25/2019
5	CC33	socket	7	$22.19	01/25/2019
6	JD03	switch	383	$13.99	01/09/2018
7	K89R	seal	40	$247.50	01/25/2019
8	KJ66	cutter	16	$24.50	01/25/2019
9	LK43	filter	121	$10.99	05/19/2018
10	M4J7	sander	98	$45.88	06/20/2018
11	MN21	brace	43	$27.87	08/10/2018
12	NCF3	valve	198	$24.50	03/20/2017
13	UYN7	rod	211	$11.55	09/09/2017

New observation　Update to existing observation

注: DATA ステップで OUTPUT または REPLACE を使用すると、オブザベーションのデフォルトの置換がオーバーライドされます。これらのステートメントを DATA ステップで使用する場合は、実行する各アクションを明示的にプログラムする必要があります。

要点

- MODIFY ステートメントは、新しい出力データセットを作成せずに既存のデータセットを更新します。MODIFY ステートメントとは異なり、SET、MERGE、および UPDATE ステートメントは新しい出力データセットを作成します。詳細については、表 21.2 (529 ページ)および表 21.4 (532 ページ)を参照してください。
- MODIFY では、変数の追加、削除、名前変更、変更はできません。
- マスターデータセットとトランザクションデータセットの両方に、BY 変数の値が重複するオブザベーションを含めることができます。MODIFY は重複を次のように処理します。
  - マスターデータセットに重複値が存在する場合、最初に出現した値のみが更新されます。

- トランザクションデータセットに重複が存在する場合、重複した変数の最後の値によって、前の重複値が上書きされます。累積ステートメントを指定して重複した変数をすべてマスターオブザベーションに追加すると、重複値は上書きされません。

関連項目

- “変更” (526 ページ)
- 表 21.2 (529 ページ)
- “%SYSRC 自動呼び出しマクロ” (*SAS マクロ言語: リファレンス*)
- “SELECT ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “REPLACE ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
- “OUTPUT ステートメント” (*SAS DATA ステップステートメント: リファレンス*)

22

インデックスの使用

SAS のインデックス

SAS インデックスは、SAS データセットのオプションコンポーネントであり、これにより SAS はデータセット内のオブザベーションに迅速かつ効率的にアクセスできるようになります。SAS インデックスの目的は、WHERE 句処理を最適化し、BY グループ処理を容易にすることです。

SAS インデックスの詳細については、次の表にリストされているドキュメントを参照してください。

表 22.1　SAS エンジンのインデックス

エンジン	ドキュメント	使用上の注意
V9 エンジン	“Indexes” (*SAS V9 LIBNAME Engine: Reference*)	リンクされたドキュメントの例では、SAS インデックスを作成する方法を示しています。 インデックスを使用して、WHERE または BY 処理を最適化できます。
SPD Engine	“Features That Boost Processing Performance” (*SAS Scalable Performance Data Engine: Reference*)	V9 エンジンと同じ言語要素を使用して、SAS インデックスを作成または使用できます。 インデックスは、V9 エンジンとは異なる方法で保存されます。SPD Engine の追加の言語要素により、インデックスの使用をより詳細に制御できるようになります。さらに、エンジンは BY 処理の自動並べ替えを提供するため、並べ替えられていないデータセットにはインデックスは必要ありません。

エンジン	ドキュメント	使用上の注意
CAS エンジン	“インデックス付け” (*SAS Cloud Analytic Servcies: 基本*)	インデックスを作成するには、いくつかの CAS アクションのいずれかを使用します。構文は V9 エンジンとは異なります。 特に非常に大きなテーブルでは、インデックスを使用して WHERE 処理を向上させることができます。
SAS/ACCESS エンジン	support.sas.com の SAS/ACCESS ソフトウェアドキュメント	SAS/ACCESS エンジンは SAS インデックスを作成または使用しません。エンジンに既存の DBMS インデックスを使用するように要求したり、DBMS 列をキー変数として使用するように要求したりできます。適切な DBMS インデックスが見つかった場合、結合 WHERE 句が DBMS に渡されるため、非常に効率的です。インデックスが見つからないか適切でない場合は、すべてのデータが結合プロセスのために SAS に転送されます。

23

配列の使用

CASL の配列

CAS 言語(CASL)での配列の作成と使用の詳細については、“CASL 配列” (*SAS Cloud Analytic Services: CASL プログラマガイド*)を参照してください。

SAS プログラミング言語における配列の定義

配列
: SAS DATA ステップ内の SAS 変数の一時的なグループ。特定の順序で配置され、*配列名*によって識別されます。配列は、現在の DATA ステップの間のみ存在します。*配列名*は変数ではありません。これは、その配列を同じ DATA ステップ内の他の配列から区別するだけです。

注: SAS の配列は、他の多くのプログラミング言語の配列とは異なります。SAS では、配列はデータ構造ではありません。配列は、変数のグループを一時的に定義する便利な方法にすぎません。

配列処理
: 一連の関連する変数に対して同じタスクを実行する方法。

配列参照
: 配列内の 1 つ以上の変数値にアクセスする方法。詳細については、“配列の参照” (615 ページ)を参照してください。

1 次元配列
: 出力を 1 次元の行形式で表す配列。詳細については、“1 次元配列” (615 ページ)を参照してください。

多次元配列
: 出力を列や行など 2 次元以上で表す、より複雑な配列。詳細については、“2 次元配列” (616 ページ)を参照してください。

基本的な配列処理には次のステップが含まれます。

- 変数を配列にグループ化する
- アクション用に配列から変数を選択する
- 配列上でアクションを繰り返す

配列の作成

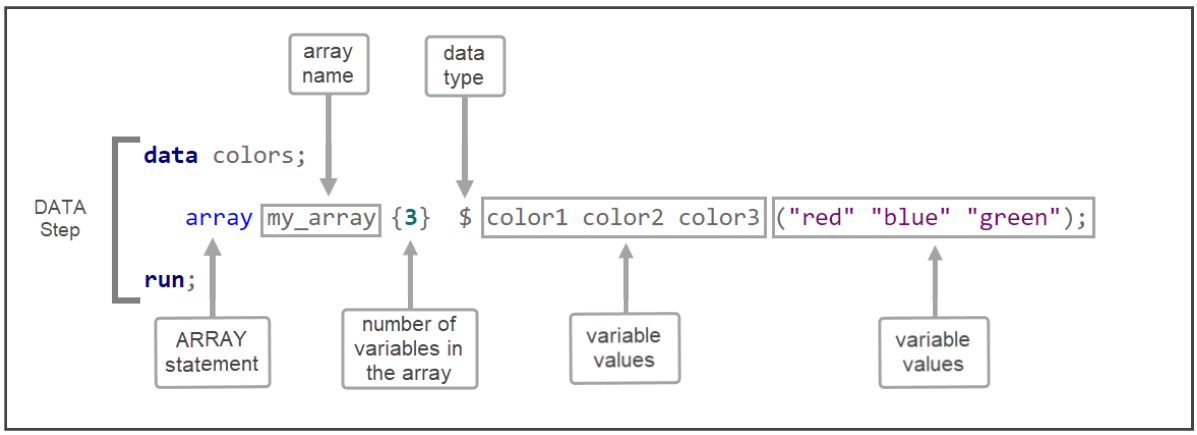
構文

SAS DATA ステップで配列を作成するには、ARRAY ステートメントを使用します。次の例では、3 つの数値変数を含む my_array という名前の配列を作成し、それらの値を定義します。

```
data nums;
  array my_array{3} num1 num2 num3 (10 20 30);
run;
```

次の図は、DATA ステップの ARRAY ステートメントの部分を示しています。

図 23.1　ARRAY ステートメントの部分



ARRAY ステートメントの形式は次のとおりです。

ARRAY *array-name* {*number-of-elements*} <$> <*length*> <*array-elements*> <(*initial-value-list*)>;

array-name
　変数のグループを識別する SAS 名です。

number-of-elements
　グループ内の変数の数です。この値は丸かっこ()、中かっこ{}、または角かっこ[]のいずれかで囲む必要があります。

$
　配列内の要素が文字要素であることを指定します。

length
　以前に長さが割り当てられていない配列内の要素の長さを指定します。

array-elements
グループ内の変数の名前のリストです。特定の配列内で定義されているすべての変数は、同じ型(すべて文字かすべて数値)である必要があります。

initial-value-list
配列内の対応する要素の初期値のリストです。

完全な情報については、"ARRAY ステートメント" (*SAS DATA ステップステートメント: リファレンス*)を参照してください。

同じ DATA ステップで以前に定義された配列を参照するには、配列参照ステートメントを使用します。配列参照の形式は次のとおりです。

array-name {*subscript*}

説明

array-name
同じ DATA ステップ内の ARRAY ステートメントで以前に定義された配列の名前です。

subscript
添字を指定します。添字には、数値定数、値が数値である変数の名前、SAS 数値式、アスタリスク(*)のいずれかを指定できます。

注: SAS の添字は、デフォルトでは 1 ベースであり、他のプログラミング言語のように 0 ベースではありません。

完全な情報については、*SAS DATA ステップステートメント: リファレンス*の配列参照ステートメントを参照してください。

単純な配列の作成

次の ARRAY ステートメントは、Reference、Usage、および Introduction の 3 つの変数を含む Books という名前の配列を作成します。

```
array books{3} Reference Usage Introduction;
```

配列を定義すると、SAS は各配列要素に *array-name*{*subscript*}という形式の *array reference* を割り当てます。ここで、*subscript* はリスト内の変数の位置です。次の表に、前の ARRAY ステートメントの配列参照割り当てを示します。

表 23.1 配列 Books の配列参照割り当て

変数	配列参照
Reference	books{1}
Usage	books{2}
Introduction	books{3}

後の DATA ステップで、配列内の変数を処理する場合は、名前または配列参照で変数を参照できます。たとえば、Reference と Books{1}という名前は同等です。

配列の参照

配列を参照する前に、配列の作成に使用したのと同じ DATA ステップに ARRAY ステートメントを指定する必要があります。配列を参照するための規則は次のとおりです。

- SAS 式を記述できる場所ならどこでも配列を参照できます。
- 一部の SAS 関数の引数として配列を参照できます。
- 配列を参照するには、中かっこ、角かっこ、または丸かっこで囲まれた添字を使用して配列を参照します。
- INPUT ステートメントまたは PUT ステートメント、あるいは関数の引数内の配列内のすべての変数を参照するには、特殊な配列アスタリスク(*)を使用します。

  注: _TEMPORARY_配列ではアスタリスクを使用できません。

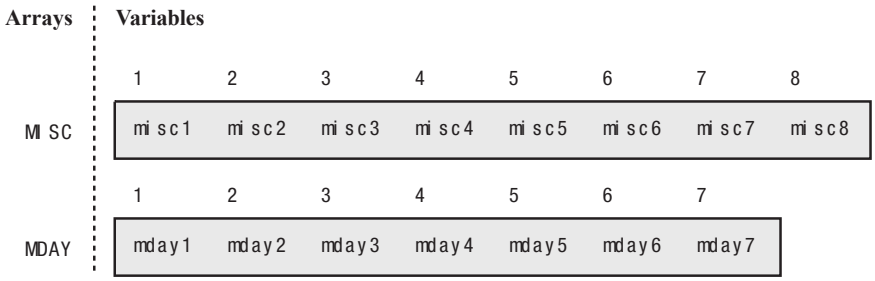
配列定義は、DATA ステップの間のみ有効です。複数の DATA ステップで同じ配列を使用する場合は、各ステップで配列を再定義する必要があります。ただし、後の DATA ステップでマクロ変数を使用すると、同じ変数で配列を再定義できます。次の例に示すように、マクロ変数は、必要な変数名を保存するのに役立ちます。

```
%let list=NC SC GA VA;

data one;
  array state{*} &list;
  ... more SAS statements ...
run;

data two;
  array state{*} &list;
  ... more SAS statements ...
run;
```

1 次元配列

次の図は、2 つの 1 次元配列、Misc と Mday の概念的な表現です。

図 23.2　1 次元配列



Misc には、変数 Misc1 から Misc8 までの 8 つの要素が含まれます。これらの変数のデータを参照するには、Misc{*n*}の形式を使用します。ここで、*n* は配列内の要素番号です。たとえば、Misc{6}は配列の 6 番目の要素です。

Mday には、変数 Mday1 から Mday7 までの 7 つの要素が含まれます。Mday{3}は配列の 3 番目の要素です。

2 次元配列

概念図

次の図は、2 次元配列 Expenses の概念的な表現です。

図 23.3　2 次元配列の例

First Dimension		Second Dimension							
Expense Categories		Days of the Week							Total
		1	2	3	4	5	6	7	8
Hotel	1	hotel1	hotel2	hotel3	hotel4	hotel5	hotel6	hotel7	hotel8
Phone	2	phone1	phone2	phone3	phone4	phone5	phone6	phone7	phone8
Pers. Auto	3	peraut1	peraut2	peraut3	peraut4	peraut5	peraut6	peraut7	peraut8
Rental Car	4	carrnt1	carrnt2	carrnt3	carrnt4	carrnt5	carrnt6	carrnt7	carrnt8
Airfare	5	airlin1	airlin2	airlin3	airlin4	airlin5	airlin6	airlin7	airlin8
Dues	6	dues1	dues2	dues3	dues4	dues5	dues6	dues7	dues8
Registration Fees	7	regfee1	regfee2	regfee3	regfee4	regfee5	regfee6	regfee7	regfee8
Other	8	other1	other2	other3	other4	other5	other6	other7	other8
Tips (non-meal)	9	tips1	tips2	tips3	tips4	tips5	tips6	tips7	tips8
Meals	10	meals1	meals2	meals3	meals4	meals5	meals6	meals7	meals8

Expenses 配列には、それぞれ 8 つの変数からなる 10 のグループが含まれています。10 のグループ(経費カテゴリ)が配列の 1 次元目を構成し、8 つの変数(曜日)が 2 次元目を構成します。配列変数内のデータを参照するには、Expenses{*m,n*}の形式を使用します。ここで、*m* は配列の 1 次元目の要素番号、*n* は配列の 2 次元目の要素番号です。Expenses{6,4}は、4 日目の会費の値を参照します(変数は Dues4)。

多次元配列の作成

多次元配列を作成するには、配列名の後に各次元の要素の数を{*n*, ... }の形式で配置します。多次元配列の各次元には *n* が必要です。

右から左に、右端の次元が列を表します。次の次元は行を表します。左端の各位置はより高い次元を表します。次の ARRAY ステートメントは、2 行 5 列の 2 次元配列を定義します。配列には、2 つの都市(c1 と c2)からの 5 つの温度測定値(t1 から t5)の 10 個の変数が含まれています。

```
array temprg{2,5} c1t1-c1t5 c2t1-c2t5;
```

SAS は、配列の左上隅から順にすべての行を埋めることによって、変数を多次元配列に配置します(行優先順序と呼ばれます)。変数は次のような配置になっていると考えることができます。

```
c1t1 c1t2 c1t3 c1t4 c1t5
c2t1 c2t2 c2t3 c2t4 c2t5
```

後で配列参照を使用して配列の要素を参照するには、配列名と添字を使用できます。次の表に、前の例の配列参照の一部を示します。

表 23.2　配列 TEMPRG の配列参照

変数	配列参照
c1t1	temprg{1,1}
c1t2	temprg{1,2}
c2t2	temprg{2,2}
c2t5	temprg{2,5}

多次元配列でのネストされた DO ループの使用

多次元配列は通常、ネストされた DO ループ内で処理されます。たとえば、次は 2 次元配列を処理する形式の 1 つです。

DO *index-variable-1*=1 **TO** *number-of-rows*;
    DO *index-variable-2*=1 **TO** *number-of-columns*;

```
    ... more SAS statements ...
  END;
END;
```

配列参照では、2 つ以上のインデックス変数を添字として使用して、配列の 2 つ以上の次元を参照できます。次の形式を使用します。

array-name {*index-variable-1*, ...,*index-variable-n*}

次の例では、2 つの都市(c1 と c2)からの 5 つの温度測定値(t1 から t5)を含む 10 個の変数を含む配列を作成します。DATA ステップには 2 つの DO ループが含まれています。

- 外側の DO ループ(DO I=1 TO 2)は内側の DO ループを 2 回処理します。
- 内側の DO ループ(DO J=1 TO 5)は、1 行内のすべての変数に ROUND 関数を適用します。

DO ループの反復ごとに、SAS は I と J の現在の値に対応する配列要素の値を置き換えます。

```
options linesize=80 pagesize=60;

data temps;
  array temprg{2,5} c1t1-c1t5 c2t1-c2t5;
  input c1t1-c1t5 /
        c2t1-c2t5;
  do i=1 to 2;
    do j=1 to 5;
      temprg{i,j}=round(temprg{i,j});
    end;
  end;
  datalines;
89.5 65.4 75.3 77.7 89.3
73.7 87.3 89.9 98.2 35.6
75.8 82.1 98.2 93.5 67.7
101.3 86.5 59.2 35.6 75.7
;

proc print data=temps;
  title 'Temperature Measures for Two Cities';
run;
```

次のデータセット Temps には、最も近い整数に丸められた変数の値が含まれています。

アウトプット 23.1　多次元配列の使用

Temperature Measures for Two Cities

Obs	c1t1	c1t2	c1t3	c1t4	c1t5	c2t1	c2t2	c2t3	c2t4	c2t5	i	j
1	90	65	75	78	89	74	87	90	98	36	3	6
2	76	82	98	94	68	101	87	59	36	76	3	6

前の例では、DIM 関数を使用して同じ結果を生成することもできます。

```
do
i=1 to dim1(temprg);
  do j=1 to dim2(temprg);
    temprg{i,j}=round(temprg{i,j});
  end;
end;
```

DIM1(TEMPRG)の値は 2 です。DIM2(TEMPRG)の値は 5 です。

基本的な配列処理のバリエーション

配列内の要素の数を効率的に決定する

反復 DO ステートメントの DIM 関数は、次元の下限が 1 の場合、1 次元配列の要素数、または多次元配列の指定された次元の要素数を返します。DIM 関数を使用すると、配列内の要素の数を変更するたびに反復 DO グループの上限が変更されることを回避できます。

DIM 関数の形式は次のとおりです。

DIM*n*(*array-name*)

ここで、*n* は指定された次元で、デフォルト値は 1 です。

アスタリスクを使用して配列内の要素の数を指定する場合は、DIM 関数を使用することもできます。DIM 関数の例をいくつか示します。

- do i=1 to dim(days);
- do i=1 to dim4(days) by 2;

DO WHILE および DO UNTIL 式

配列は、DO WHILE または DO UNTIL 式で配列参照を使用する反復 DO ループで処理されることがよくあります。この例では、反復 DO ループは、Trend という名前の配列の要素を処理します。

```
data test;
  array trend{5} x1-x5;
  input x1-x5 y;
  do i=1 to 5 while(trend{i}<y);
  ... more SAS statements ...
  end;
  datalines;
... data lines ...
```

;

変数リストを使用して配列を迅速に定義する

SAS は、変数リスト名として使用するために、次の 3 つの名前を予約しています。

- _CHARACTER_
- _NUMERIC_
- _ALL_

これらの変数リスト名を使用して、同じ DATA ステップで以前に定義された変数を参照できます。_CHARACTER_変数は文字値のみをリストします。_NUMERIC_変数は数値のみをリストします。_ALL_変数には、以前に変数を定義した方法に応じて、すべての文字またはすべての数値がリストされます。

たとえば、次の INPUT ステートメントは、$8.入力形式を使用して変数 X1 から X3 を文字値として読み取り、変数 X4 から X5 を数値変数として読み取ります。次の ARRAY ステートメントは、変数リスト_CHARACTER_を使用して、配列に文字変数のみを含めます。アスタリスクは、SAS が配列内の変数を数えることによって添字を決定することを示します。

```
input (X1-X3) ($8.) X4-X5;
array item {*} _character_;
```

プログラムで_NUMERIC_変数を使用できます(たとえば、通貨を変換する必要がある場合)。このアプリケーションでは、変数名を知る必要はありません。すべての値を新しい通貨に変換するだけで済みます。

変数リストの詳細については、*“ARRAY ステートメント” (SAS DATA ステップステートメント: リファレンス)*を参照してください。

配列範囲の指定

上限と下限の特定

通常、ARRAY ステートメントでは、配列の各次元の添字の範囲は 1 から *n* までです。ここで、*n* はその次元の要素の数です。したがって、1 は配列のその次元の下限であり、*n* は上限です。たとえば、次の配列では、下限は 1、上限は 4 です。

```
array new{4} Jackson Poulenc Andrew Parson;
```

次の ARRAY ステートメントでは、1 次元目の範囲は 1 から 2 で、2 次元目の範囲は 1 から 5 です。

```
array test{2,5} test1-test10;
```

範囲制限ありの配列の次元は次の形式になります。

{<*lower-1*:>*upper-1*<,...<*lower-n*:>*upper-n*>}

したがって、前の ARRAY ステートメントを次のように記述することもできます。

```
array new{1:4} Jackson Poulenc Andrew Parson;
array test{1:2,1:5} test1-test10;
```

ほとんどの配列では、1 が便利な下限であるため、下限を指定する必要はありません。ただし、配列の次元に 1 以外の開始点がある場合は、下限と上限の両方を指定すると便利です。

次の例では、10 個の変数に Year76 から Year85 という名前が付けられています。次の ARRAY ステートメントは、変数を First と Second という名前の 2 つの配列に配置します。

```
array first{10} Year76-Year85;
array second{76:85} Year76-Year85;
```

最初の ARRAY ステートメントでは、要素 first{4}は変数 Year79、first{7}は Year82 などとなります。2 番目の ARRAY ステートメントでは、要素 second{79}は Year79、要素 second{82}は Year82 です。

DO グループの配列名 Second を処理するには、次のように DO ループの範囲が配列の範囲と一致することを確認します。

```
do i=76 to 85;
  if second{i}=9 then second{i}=.;
end;
```

配列範囲の決定: LBOUND 関数と HBOUND 関数

LBOUND 関数と HBOUND 関数を使用して、配列範囲を決定できます。LBOUND 関数は、1 次元配列の下限、または多次元配列の指定された次元の下限を返します。HBOUND 関数は、1 次元配列の上限、または多次元配列の指定された次元の上限を返します。

LBOUND 関数と HBOUND 関数の形式は次のとおりです。

LBOUND*n*(*array-name*)

HBOUND*n*(*array-name*)

説明

n
　指定された次元であり、デフォルト値は 1 です。

LBOUND 関数と HBOUND 関数を使用して、Second という名前の配列の要素を処理する反復 DO ループの開始値と終了値を指定できます。

```
do i=lbound{second} to hbound{second};
  if second{i}=9 then second{i}=.;
end;
```

この例では、反復 DO ステートメントのインデックス変数の範囲は 76 から 85 です。

DIM 関数のかわりに HBOUND 関数を使用する場合

次の ARRAY ステートメントは、下限 72、上限 76 の合計 5 つの要素を含む配列を定義します。これは 1972 年から 1976 年までのカレンダー年を表します。

```
array years{72:76} first second third fourth fifth;
```

YEARS という名前の配列を反復 DO ループで処理するには、次のように DO ループの範囲が配列の範囲と一致していることを確認してください。

```
do i=lbound(years) to hbound(years);
  if years{i}=99 then years{i}=.;
end;
```

LBOUND(YEARS)の値は 72 です。HBOUND(YEARS)の値は 76 です。

この例では、DIM 関数は、配列 YEARS 内の要素の総数である 5 の値を返します。したがって、配列の上限に HBOUND 関数のかわりに DIM 関数を使用した場合、DO ループ内のステートメントは実行されません。

2 次元配列での範囲の指定

次のリストには、X60 から X99 という名前の 40 個の変数が含まれています。これらは 1960 年から 1999 年を表します。

```
X60  X61  X62  X63  X64  X65  X66  X67  X68  X69
X70  X71  X72  X73  X74  X75  X76  X77  X78  X79
X80  X81  X82  X83  X84  X85  X86  X87  X88  X89
X90  X91  X92  X93  X94  X95  X96  X97  X98  X99
```

次の ARRAY ステートメントは、配列内の変数を 10 年ごとに配列します。行の範囲は 6 から 9、列の範囲は 0 から 9 です。

```
array X{6:9,0:9} X60-X99;
```

配列 X では、変数 X63 は要素 X{6,3}であり、変数 X89 は要素 X{8,9}です。反復 DO ループで配列 X を処理するには、次のいずれかの方法を使用します。

- 方法 1:

  ```
  do i=6 to 9;
    do j=0 to 9;
      if X{i,j}=0 then X{i,j}=.;
    end;
  end;
  ```

- 方法 2:

  ```
  do i=lbound1(X) to hbound1(X);
    do j=lbound2(X) to hbound2(X);
      if X{i,j}=0 then X{i,j}=.;
  ```

```
  end;
end;
```

どちらの例も、変数 X60 から X99 の 0 の値をすべて欠損に変更します。最初の例では、DO グループの範囲を明示的に設定します。2 番目の例では、LBOUND 関数と HBOUND 関数を使用して、配列の各次元の範囲を返します。

例

例: 配列内での文字変数の使用

ARRAY ステートメントでは、文字変数とその長さを指定できます。次の例では、変数を 2 つの配列、NAMES と CAPITALS にグループ化します。

```
options linesize=80 pagesize=60;

data text;
  array names{*} $ n1-n5;          /* 1 */
  array capitals{*} $ c1-c5;
  input names{*};                  /* 2 */
    do i=1 to 5;
      capitals{i}=upcase(names{i}); /* 3 */
    end;
  datalines;
smithers michaels gonzalez hurth frank
;

proc print data=text;
  title 'Names Changed from Lowercase to Uppercase';
run;
```

1 ドル記号($)は、要素を文字変数として作成するように SAS に指示します。変数がすでに文字変数として宣言されている場合、配列内にドル記号は必要ありません。

2 INPUT ステートメントは、配列 NAMES 内のすべての変数を読み取ります。

3 DO ループ内のステートメントは UPCASE 関数を使用して、配列 NAMES 内の変数の値を大文字に変更します。次に、ステートメントは、CAPITALS 配列の変数に大文字の値を保存します。

次の出力は、TEXT データセットを示しています。

アウトプット 23.2　配列内での文字変数の使用

Names Changed from Lowercase to Uppercase

Obs	n1	n2	n3	n4	n5	c1	c2	c3	c4	c5	i
1	smithers	michaels	gonzalez	hurth	frank	SMITHERS	MICHAELS	GONZALEZ	HURTH	FRANK	6

例: 配列の要素に初期値を割り当てる

この例では、配列 Test に変数を作成し、初期値 90、80、および 70 を割り当てます。Score という名前の別の配列に値を読み取り、Score の各要素を Test の対応する要素と比較します。

```
options linesize=80 pagesize=60;

data score1(drop=i);
  array test{3} t1-t3 (90 80 70);     /* 1 */
  array score{3} s1-s3;
  input id score{*};                  /* 2 */
  do i=1 to 3;
    if score{i}>=test{i} then         /* 3 */
      do;
        NewScore=score{i};
        output;                       /* 4 */
      end;
  end;
  datalines;
1234  99 60 82
5678  80 85 75
;

proc print noobs data=score1;
  title 'Data Set SCORE1';
run;
```

1 初期値 90、80、および 70 を Test 配列に割り当て、変数 s1、s2、および s3 を Score 配列に割り当てます。

2 INPUT ステートメントは、ID という名前の変数の値を読み取り、次に Score 配列内のすべての 変数の値を読み取ります。

3 Score の要素の値が Test の要素の値以上である場合、変数 NewScore に要素 Score の値が割り当てられます。

4 OUTPUT ステートメントは、オブザベーションを SAS データセットに書き込みます。

次の出力は、Score1 データセットを示しています。

アウトプット 23.3　配列の要素に初期値を割り当てる

Data Set SCORE1

t1	t2	t3	s1	s2	s3	id	NewScore
90	80	70	99	60	82	1234	99
90	80	70	99	60	82	1234	82
90	80	70	80	85	75	5678	85
90	80	70	80	85	75	5678	75

例: 現在の DATA ステップで一時的に使用する配列の作成

配列の要素が DATA ステップの実行中にのみ必要な定数である場合、配列グループから変数を省略し、かわりに一時配列要素を使用できます。一時データ要素は、配列名と次元によって参照します。これらは変数のように動作しますが、一時配列要素には名前がなく、出力データセットには表示されません。一時配列要素は、DATA ステップの次の反復の開始時に欠損状態にリセットされるのではなく、自動的に保持されます。

一時配列を作成するには、_TEMPORARY_引数を使用します。次の例では、Test という名前の一時配列を作成します。

```
options linesize=80 pagesize=60;

data score2(drop=i);
    array test{3} _temporary_ (90 80 70);
    array score{3} s1-s3;
    input id score{*};
      do i=1 to 3;
        if score{i}>=test{i} then
          do;
            NewScore=score{i};
            output;
          end;
      end;
    datalines;
  1234  99 60 82
  5678  80 85 75
  ;

  proc print noobs data=score2;
    title 'Data Set SCORE2';
  run;
```

次の出力は、Score2 データセットを示しています。

アウトプット 23.4　_TEMPORARY_配列の使用

Data Set SCORE2

s1	s2	s3	id	NewScore
99	60	82	1234	99
99	60	82	1234	82
80	85	75	5678	85
80	85	75	5678	75

例: すべての数値変数に対するアクションの実行

この例では、配列 Test 内のすべての数値変数を 3 で乗算します。

```
options nodate pageno=1 linesize=80 pagesize=60;

data sales;
  infile datalines;
  input Value1 Value2 Value3 Value4;
  datalines;
11 56 58 61
22 51 57 61
22 49 53 58
;
data convert(drop=i);
  set sales;
  array test{*} _numeric_;
  do i=1 to dim(test);
    test{i} = (test{i}*3);
  end;
run;

proc print data=convert;
  title 'Data Set CONVERT';
run;
```

次の出力は、CONVERT データセットを示しています。

アウトプット 23.5　_NUMERIC_変数リストを使用した場合の出力

Data Set CONVERT

Obs	Value1	Value2	Value3	Value4
1	33	168	174	183
2	66	153	171	183
3	66	147	159	174

24

エラーのデバッグ

SAS におけるエラーの種類の定義

SAS は、SAS 処理のコンパイルフェーズと実行フェーズの両方でエラー処理を実行します。SAS ログ内の処理メッセージを理解し、コードを修正することで、SAS プログラムをデバッグできます。DATA ステップデバッガーを使用すると、実行中に DATA ステップの論理エラーを検出できます。

SAS は 5 種類のエラーを認識します。

表 24.1　エラーの種類の概要

エラーの種類	このエラーがいつ発生するか	エラーがいつ検出されるか
構文エラー	プログラミングステートメントが SAS 言語のルールに準拠していない場合	コンパイル時
意味エラー	言語要素は正しいが、その要素が特定の使用法では有効ではない可能性がある場合	コンパイルまたは実行時
実行時エラー	SAS がプログラムを実行しようとして実行に失敗した場合	実行時
データエラー	データ値が無効な場合	実行時
マクロ関連エラー	マクロ機能を誤って使用した場合	マクロのコンパイル時または実行時、DATA、または PROC ステップのコンパイル時または実行時

プログラムをデバッグする方法

SAS のデバッグ方法とツール

表 24.2　プログラムをデバッグする方法の概要

デバッグ方法またはツール
“AUTOCORRECT システムオプション” (*SAS システムオプション: リファレンス*)
“BYERR システムオプション” (*SAS システムオプション: リファレンス*)

デバッグ方法またはツール
_ERROR_自動 DATA ステップ変数
“CHKPTCLEAN システムオプション” (*SAS システムオプション: リファレンス*)
“DKRICOND= システムオプション” (*SAS システムオプション: リファレンス*)
“DSNFERR システムオプション” (*SAS システムオプション: リファレンス*)
ERRORBYABEND システムオプション
“ERRORCHECK= システムオプション” (*SAS システムオプション: リファレンス*)
ERROR ステートメント
“INVALIDDATA= システムオプション” (*SAS システムオプション: リファレンス*)
“LABELCHKPT システムオプション” (*SAS システムオプション: リファレンス*)
MSGLEVEL=システムオプション
“WORKTERM システムオプション” (*SAS システムオプション: リファレンス*)
“WORKINIT システムオプション” (*SAS システムオプション: リファレンス*)
“PRINTMSGLIST システムオプション” (*SAS システムオプション: リファレンス*)
“SOURCE システムオプション” (*SAS システムオプション: リファレンス*)
“SOURCE2 システムオプション” (*SAS システムオプション: リファレンス*)
“SYNTAXCHECK システムオプション” (*SAS システムオプション: リファレンス*)システムオプション(バッチモード)
CLEANUP システムオプション
STEPCHKPT システムオプション
SYSRC 自動呼び出しマクロ
“VARINITCHK= システムオプション” (*SAS システムオプション: リファレンス*)
“VNFERR システムオプション” (*SAS システムオプション: リファレンス*)

エラーチェックツール

SET ステートメントまたは MODIFY ステートメントを、 KEY=オプションを指定して使用する場合、次のツールを使用してエラーをチェックできます。

- _IORC_自動変数
- SYSRC 自動呼び出しマクロ

  エラー状態に対応する値を返します。

_IORC_は、KEY=を指定した MODIFY ステートメントまたは SET ステートメントを使用すると自動的に作成されます。_IORC_の値は、KEY=を指定して最後に実行された MODIFY または SET ステートメントからの I/O 操作のステータスを示す数値のリターンコードです。この変数の値をチェックすると、異常な I/O 状態を検出し、アプリケーションを異常終了させるかわりに特定のコードパスに実行を誘導することができます。たとえば、KEY=変数値が 2 つのオブザベーション間で一致する場合、それらを結合してオブザベーションを出力することができます。ただし、一致しない場合は、ログに注記のみを書き込むことができます。

_IORC_自動変数の値は内部的なものであり、変更される可能性があるため、将来の_IORC_値の変更からコードを保護しながら特定の I/O 状態をテストできるように SYSRC マクロが作成されました。SYSRC を使用する場合、次の表にリストされているニーモニックのいずれかを指定して、_IORC_の値を確認できます。

表 24.3　DATA ステップ処理における_IORC_の最も一般的なニーモニック値

ニーモニック値	リターンコードの意味	リターンコードがいつ発生するか
_DSENMR	トランザクションデータセットのオブザベーションがマスターデータセットに存在しません。	BY を指定した MODIFY が使用され、一致が発生しない場合。
_DSEMTR	同じ BY 変数値を持つ複数のトランザクションデータセットオブザベーションがマスターデータセットに存在しません。	BY を指定した MODIFY が使用され、同じ BY 値を持つ連続したオブザベーションが最初のデータセットで一致を見つけられない場合。この状況では、一致が見つからなかった最初のオブザベーションは _DSENMR を返します。後続のオブザベーションは _DSEMTR を返します。
_DSENOM	マスターデータセットに一致するオブザベーションが見つかりませんでした。	KEY=を指定した SET または MODIFY で一致が見つらない場合。

ニーモニック値	リターンコードの意味	リターンコードがいつ発生するか
_SENOCHN	出力操作が失敗しました。	MODIFY ステートメントの KEY=オプションに重複値が含まれている場合。
_SOK	I/O 操作が成功しました。	一致が見つかった場合。

バッチ処理のデバッグ

チェックポイントモードと再起動モード

チェックポイントモードと再起動モードを一緒に使用すると、完了前に終了したバッチプログラムを、すでに完了したステップまたはラベル付きコードセクションを再実行せずに再サブミットできる環境が作成されます。実行は、障害が発生したときに実行されていた DATA または PROC ステップ、あるいはラベル付きコードセクションのいずれかから再開されます。

ラベル付きコードセクションは、DATA または PROC ステップの外側の *label:*で始まり、DATA または PROC ステップの外側の次の *label:*の前にある RUN ステートメントで終わる SAS コードです。ラベルは一意である必要があります。一方のデータが他方のデータに依存しているためにグループ化する必要がある可能性のある DATA ステップまたは PROC ステップをグループ化する場合は、ラベル付きコードセクションの使用を検討してください。

次のプログラム例には、ラベル付きコードセクションが 2 つあります。最初のラベル付きコードセクションは、ラベル readSortData:で始まり、proc sort data=mylib.mydata;の run;ステートメントで終わります。2 番目のラベル付きコードセクションは、ラベル report:で始まり、proc report data=mylib.mydata;の run;ステートメントで終わります。

例のコード 24.1　2 つのラベル付きコードセクションを含むプログラム

```
readSortData:
data mylib.mydata;
...more sas code...
run;

proc sort data=mylib.mydata;
...more sas code...
run;

report:
proc report data=mylib.mydata;
...more sas code...;
run;
endReadSortReport:
```

注: チェックポイントモードおよび再起動モードでの *label:*の使用は、DATA または PROC ステートメントの外部でのみ有効です。ラベル付きコードセクションのチェックポイントモードと再起動モードは、DATA ステップまたはマクロ内のラベルには無効です。

チェックポイントモードと再起動モードは、DATA ステップと PROC ステップ、またはラベル付きコードセクションのいずれかに対して有効にできますが、両方を同時に有効にすることはできません。チェックポイントモードと再起動モードを段階的に使用するには、ステップチェックポイントモードとステップ再起動モードを使用します。コードセクションのグループに基づいてチェックポイントモードと再起動モードを使用するには、ラベルチェックポイントモードとラベル再起動モードを使用します。各コードグループは一意のラベルによって識別されます。ラベルを使用する場合、SAS プログラム内のすべてのステップはラベル付きコードセクションに属している必要があります。

チェックポイントモードが有効になっている場合、SAS は DATA および PROC ステップまたはラベル付きコードセクションに関する情報をチェックポイントライブラリに記録します。バッチプログラムが途中で終了した場合は、再起動モードでプログラムを再サブミットして実行を完了できます。再起動モードでは、グローバルステートメントが再実行され、マクロ定義が再コンパイルされ、マクロが再実行されます。SAS はチェックポイントライブラリ内のデータを読み取り、どのステップまたはラベル付きコードセクションが完了したかを判断します。プログラムの実行は、障害が発生したときに実行されていたステップまたはラベルから再開されます。

チェックポイント/再起動データには、完了した DATA および PROC ステップまたはラベル付きコードセクションと、完了しなかったステップまたはラベル付きコードセクションに関する情報のみが含まれます。チェックポイント/再起動データには、次の情報は含まれません。

- マクロ変数とマクロ定義に関する情報
- SAS データセットに関する情報
- 完了しなかったステップまたはラベル付きコードセクションで処理された可能性のある情報

ベストプラクティスとして、ラベル付きコードセクションを使用する場合は、プログラムの最後にラベルを追加します。プログラムが正常に完了すると、ラベルがチェックポイント/再起動データに記録されます。プログラムが再起動モードで再度サブミットされると、SAS はプログラムがすでに正常に完了したことを認識します。

DATA または PROC ステップを再実行する必要がある場合は、ステップの直前にグローバルステートメント CHECKPOINT EXECUTE_ALWAYS ステートメントを追加して、チェックポイント/再起動データを考慮せずに常にステップを実行するように SAS に指示することができます。これは、ステートメントに続くステップにのみ適用されます。

SAS でバッチプログラムを起動するときに、システムオプションを使用して、DATA ステップと PROC ステップのチェックポイントモードと再起動モードを有効にします。

- STEPCHKPT はチェックポイントモードを有効にし、SAS にチェックポイント/再起動データを記録するよう指示します。
- STEPCHKPTLIB システムオプションは、ユーザー指定のチェックポイント/再起動ライブラリを識別します。

- STEPRESTART システムオプションは、再起動モードを有効にし、チェックポイント/再起動ライブラリによって示された DATA または PROC ステップで実行が再開されることを保証します。

ラベル付きコードのチェックポイントモードと再起動モードの有効化

SAS でバッチプログラムを起動するときに、次のシステムオプションを使用して、ラベル付きコードセクションのチェックポイントモードと再起動モードを有効にします。

- LABELCHKPT システムオプションは、ラベル付きコードセクションのチェックポイントモードを有効にし、SAS にチェックポイント/再起動データを記録するように指示します。
- LABELCHKPTLIB システムオプションは、ユーザー指定のチェックポイント/再起動ライブラリを識別します。
- LABELRESTART システムオプションは、再起動モードを有効にし、チェックポイント/再起動ライブラリによって示されたラベル付きコードセクションで実行が再開されることを保証します。

Work ライブラリをチェックポイント/再起動ライブラリとして使用する場合、バッチプログラムが正常に実行された後に CHKPTCLEAN システムオプションを使用して Work ライブラリ内のファイルを消去することができます。

チェックポイントモードと再起動モードの使用要件

チェックポイントモードと再起動モードが正常に機能するには、バッチプログラム内の DATA ステップと PROC ステップ、またはラベル付きコードセクションの数と順序が SAS 起動間で変更されないようにする必要があります。SAS の起動時に ERRORABEND および ERRORCHECK システムオプションを指定すると、有効なチェックポイント/再起動データを維持するために、ほとんどのエラー状態で SAS が終了します。

チェックポイント/再起動ライブラリはユーザー指定のライブラリにすることができます。ライブラリが指定されていない場合は、チェックポイント/再起動データは Work ライブラリに保存されます。チェックポイント/再起動データがユーザー指定のライブラリに保存されるか、Work ライブラリに保存されるかに関係なく、常に NOWORKTERM および NOWORKINIT システムオプションを使用して SAS を起動します。SAS は、Work ライブラリの名前を SAS ログに書き込みます。

注: STEPCHKPT システムオプションまたは"LABELCHKPT システムオプション" (*SAS システムオプション: リファレンス*)オプションを使用する場合は、チェックポイント/再起動ライブラリを割り当てることを検討してください。サイトで CLEANWORK ユーティリティを 定期的に実行するように設定している場合、Work ライブラリ内のデータが失われる可能性があります。

ラベル付きコードセクションのラベルは一意である必要があります。SAS が重複ラベルのラベルに対して再起動モードに入ると、SAS は最初のラベルから開始します。重複ラベル間のコードが不必要に再実行される可能性があります。

チェックポイントモードと再起動モードの設定と実行

チェックポイントモードと再起動モードを設定するには、バッチプログラムに次の変更を加えます。

- バッチプログラムがサブミットされるたびに実行する DATA および PROC ステップの前に CHECKPOINT EXECUTE_ALWAYS ステートメントを追加します。
- チェックポイント/再起動ライブラリがユーザー定義のライブラリである場合は、チェックポイント/再起動ライブラリ参照名を定義する LIBNAME ステートメントをバッチプログラムの最初のステートメントとして追加する必要があります。Work ライブラリをチェックポイントライブラリとして使用する場合、LIBNAME ステートメントは必要ありません。

バッチプログラムを変更したら、適切なシステムオプションを使用してプログラムを起動します。

- Work ライブラリに保存されているチェックポイント/再起動データの場合は、次のシステムオプションを指定するバッチ SAS セッションを開始します。
  - SYSIN は、動作環境で必要な場合に、バッチプログラムの名前を指定します。
  - STEPCHKPT または LABELCHKPT はチェックポイントモードを有効にします。
  - NOWORKTERM は、SAS の終了時に Work ライブラリを保存します。
  - NOWORKINIT は、SAS の起動時に Work ライブラリを初期化しません。
  - ERRORCHECK STRICT は、LIBNAME、FILENAME、%INCLUDE、LOCK ステートメントでエラーが発生した場合に、SAS を構文チェックモードにします。
  - ERRORABEND は、ほとんどのエラーで SAS が終了するかどうかを指定します。
  - CHKPTCLEAN は、バッチプログラムが正常に実行された場合に、Work ライブラリ内のファイルを消去し、Work ライブラリを削除するかどうかを指定します。

  次の SAS コマンドは、Work ライブラリをチェックポイント/再起動ライブラリとして使用して、バッチプログラムをチェックポイントモードで開始します。

  ```
  sas -sysin 'u/mysas/myprogram.sas' -stepchkpt -noworkterm -noworkinit -errorcheck strict -errorabend -chkptclean
  ```

- ユーザー指定のライブラリに保存されているチェックポイント/再起動データの場合は、次のシステムオプションを含むバッチ SAS セッションを開始します。
  - SYSIN は、動作環境で必要な場合に、バッチプログラムの名前を指定します。
  - STEPCHKPT または LABELCHKPT はチェックポイントモードを有効にします。

- STEPCHKPTLIB または LABELCHKPTLIB は、SAS がチェックポイント/再起動データを保存するライブラリのライブラリ参照名を指定します。
- NOWORKTERM は、SAS の終了時に Work ライブラリを保存します。
- NOWORKINIT は、SAS の起動時に Work ライブラリを初期化しません。
- ERRORCHECK STRICT は、LIBNAME、FILENAME、%INCLUDE、LOCK ステートメントでエラーが発生した場合に、SAS を構文チェックモードにします。
- ERRORABEND は、ほとんどのエラーで SAS が終了するかどうかを指定します。

次の SAS コマンドは、ユーザー指定のチェックポイント/再起動ライブラリを使用して、バッチプログラムをチェックポイントモードで開始します。

```
sas -sysin 'u/mysas/myprogram.sas' -labelchkpt -labelchkptlib mylibref -noworkterm -noworkinit -errorcheck strict -er
```

この場合、MyProgram.sas の最初のステートメントは、**MyLibref** ライブラリ参照名を定義する LIBNAME ステートメントです。

バッチプログラムの再起動

Work ライブラリに保存されているチェックポイント/再起動データを使用してバッチ SAS セッションを再サブミットするには、SAS の起動時に次のシステムオプションを含めます。

- SYSIN は、動作環境で必要な場合に、バッチプログラムの名前を指定します。
- STEPCHKPT または LABELCHKPT はチェックポイントモードを続行します。
- STEPRESTART または LABELRESTART は再起動モードを有効にし、SAS にチェックポイント/再起動データを使用するように指示します。
- NOWORKINIT は、前回の SAS セッションの Work ライブラリを使用して SAS を起動します。
- NOWORKTERM は、SAS の終了時に Work ライブラリを保存します。
- ERRORCHECK STRICT は、LIBNAME、FILENAME、%INCLUDE、LOCK ステートメントでエラーが発生した場合に、SAS を構文チェックモードにします。
- ERRORABEND は、ほとんどのエラーで SAS が終了するかどうかを指定します。
- CHKPTCLEAN は、バッチプログラムが正常に実行された場合に、Work ライブラリ内のファイルを消去するかどうかを指定します。

次の SAS コマンドは、チェックポイント/再起動データが Work ライブラリに保存されたバッチプログラムを再サブミットします。

```
sas -sysin 'u/mysas/mysasprogram.sas' -stepchkpt -steprestart -noworkinit -noworkterm -errorcheck strict -errorabend -ch
```

NOWORKTERM システムオプションと、STEPCHKPT または LABELCHKPT システムオプションのいずれかを指定すると、バッチプログラムが再起動した後もチェックポイントモードは引き続き有効になります。

ユーザー指定のライブラリに保存されているチェックポイント/再起動データを使用してバッチ SAS セッションを再サブミットするには、SAS の起動時に次のシステムオプションを含めます。

- SYSIN は、動作環境で必要な場合に、バッチプログラムの名前を指定します。
- STEPCHKPT または LABELCHKPT はチェックポイントモードを続行します。
- STEPRESTART または LABELRESTART は再起動モードを有効にし、SAS にチェックポイント/再起動データを使用するように指示します。
- STEPCHKPTLIB または LABELCHKPTLIB は、チェックポイント/再起動ライブラリのライブラリ参照名を指定します。
- NOWORKTERM は、SAS の終了時に Work ライブラリを保存します。
- NOWORKINIT は、SAS の起動時に Work ライブラリを初期化しません。
- ERRORCHECK STRICT は、LIBNAME、FILENAME、%INCLUDE、LOCK ステートメントでエラーが発生した場合に、SAS を構文チェックモードにします。
- ERRORABEND は、ほとんどのエラーで SAS が終了するかどうかを指定します。

次の SAS コマンドは、チェックポイント/再起動データがユーザー指定のライブラリに保存されたバッチプログラムを再サブミットします。

```
sas -sysin 'u/mysas/mysasprogram.sas' -labelchkpt -labelrestart -labelchklib -noworkterm -noworkinit mylibref -errorchecl
```

構文チェックモード

構文チェックモードの概要

構文チェックモードでは、SAS は構文エラーのある DATA ステップを検出したときに構文エラーにフラグを設定したり、処理を停止したりできます。SAS は、プログラムがデータセットを作成する場合にのみ構文チェックモードに入ることができます。たとえば、DATA _NULL_ステートメントを使用する場合、プログラムはデータセットを作成しないため、SAS は構文チェックモードに入ることができません。

構文チェックモードでは、SAS は内部的に OBS=オプションを 0 に設定し、REPLACE/NOREPLACE オプションを NOREPLACE に設定します。これらのオプションが有効な場合、SAS は次のように動作します。

- DATA ステップまたは PROC ステップの残りのステートメントを読み取る
- ステートメントが有効な SAS ステートメントであるかどうかをチェックする
- グローバルステートメントを実行する
- SAS ログにエラーを書き込む
- プログラムステートメントで指定された出力データセットのディスクリプター部分を作成する
- SAS が作成する新しいデータセットにはオブザベーションを書き込まない
- プログラム内の後続の DATA ステップやプロシジャのほとんどを実行しない(例外には PROC DATASETS と PROC CONTENTS が含まれる)

注: SAS が構文チェックモードに入った後に作成されたデータセットは、同じ名前の既存のデータセットを置き換えることはありません。

構文チェックが有効になっている場合、SAS は DATA ステップで構文エラーまたは意味エラーを検出したポイントに下線を付け、エラーを番号で識別します。その後、SAS は構文チェックモードに入り、プログラムの実行が完了するまでこのモードのままになります。SAS が構文チェックモードに入ると、すべての DATA ステップステートメントと PROC ステップステートメントが検証されます。

構文チェックモードの有効化

SYNTTAXCHECK は、バッチモードまたは非対話モードでのデフォルト値です。

構文チェックモードを無効にするには、NOSYNTAXCHECK および NODMSSYNCHK システムオプションを使用します。

SYNTAXCHECK システムオプションまたは DMSSYNCHK システムオプションを含む OPTIONS ステートメントを、それを適用するステップの前に配置できます。OPTIONS ステートメントをステップ内に配置すると、SYNTAXCHECK または DMSSYNCHK は次のステップの開始まで有効になりません。

例 1: 予期しない状態が発生した場合のルーティング実行

概要

この例では、予期しない状態によって DATA ステップが終了しないようにする方法を示します。目標は、トランザクションデータセットからの新しい情報を使用してマスターデータセットを更新することです。このアプリケーションでは、どちらのデータセットでも共通変数に重複値が存在しないことを前提としています。

注: このプログラムは、マスターデータセットとトランザクションデータセットに共通変数に同じ値を持つ連続したオブザベーションが含まれていない場合にのみ、期待どおりに動作します。重複が存在する場合の KEY=を指定した MODIFY の動作の説明については、*SAS DATA ステップステートメント: リファレンス*の MODIFY ステートメントを参照してください。

入力データセット

Transaction データセットには、Master の情報に対する 2 つの更新と、追加する必要がある PartNumber 値 6 に関する新しいオブザベーションの 3 つのオブザベーションが含まれています。Master は PartNumber に基づいてインデックス付けされます。Master または Transaction に PartNumber の重複値はありません。Master および Transaction 入力データセットを次に示します。

	Master			Transaction	
OBS	PartNumber	Quantity	OBS	PartNumber	AddQuantity
1	1	10	1	4	14
2	2	20	2	6	16
3	3	30	3	2	12
4	4	40			
5	5	50			

元のプログラム

目標は、Transaction データセットの情報を使用して Master データセットを更新することです。プログラムは Transaction を順次読み取ります。Master は、MODIFY ステートメントと KEY=オプションを使用して、順次ではなく直接読み取られます。KEY=変数である PartNumber の値が一致するオブザベーションのみが Master から読み取られます。

```
data master; 1
  set transaction; 2
  modify master key=PartNumber; 3
  Quantity = Quantity + AddQuantity; 4
run;
```

1 更新のために Master データセットを開きます。

2 Transaction データセットからオブザベーションを読み取ります。

3 PartNumber の値に基づいて、Master データセットからのオブザベーションを一致させます。

4 Transaction データセットから新しい値を追加して、Quantity に関する情報を更新します。

結果ログ

このプログラムは 1 つのオブザベーションを正しく更新しましたが、PartNumber 値 6 に一致するものが見つからなかったため停止しました。次の行が SAS ログに書き込まれます。

```
ERROR: No matching observation was found in Master data set.
PartNumber=6 AddQuantity=16 Quantity=70 _ERROR_=1
_IORC_=1230015 _N_=2
NOTE: The SAS System stopped processing this step because
    of errors.
NOTE: The data set WORK.MASTER has been updated.  There were
    1 observations rewritten, 0 observations added and 0
    observations deleted.
```

結果データセット

Master ファイルが正しく更新されませんでした。更新された Master には 5 つのオブザベーションがあります。1 つのオブザベーションは正しく更新されましたが、新しいオブザベーションは追加されず、2 番目の更新は行われませんでした。誤って更新された Master データセットを次に示しています。

Master

OBS	PartNumber	Quantity
1	1	10
2	2	20
3	3	30
4	4	54
5	5	50

改訂プログラム

目標は、Master に 2 つの更新と 1 つの追加を適用することです。このアクションにより、Transaction の PartNumber 値 6 に一致するものが Master に見つからない場合に、DATA ステップが停止することがなくなります。エラーチェックを追加することで、この DATA ステップは正常に完了し、Master の正しく修正されたバージョンが生成されます。このプログラムは、SELECT グループ内の_IORC_自動変数と SYSRC 自動呼び出しマクロを使用して、_IORC_変数の値をチェックします。一致が見つかった場合、プログラムは適切なコードを実行します。

```
data master;  1
  set transaction;  2
  modify master key=PartNumber;  3

select(_iorc_);  4
    when(%sysrc(_sok)) do;
      Quantity = Quantity + AddQuantity;
      replace;
    end;
    when(%sysrc(_dsenom)) do;
      Quantity = AddQuantity;
      _error_ = 0;
      output;
    end;
    otherwise do;
```

```
        put 'ERROR: Unexpected value for _IORC_= ' _iorc_;
        put 'Program terminating. DATA step iteration # ' _n_;
        put _all_;
        stop;
      end;
    end;
run;
```

1 更新のために Master データセットを開きます。

2 Transaction データセットからオブザベーションを読み取ります。

3 PartNumber の値に基づいて、Master データセットからのオブザベーションを一致させます。

4 PartNumber に一致する値が Master に見つかったかどうかに基づいて、適切なアクションを実行します。Transaction からの新しい値を追加して Quantity を更新します。SELECT グループは、実行を正しいコードに指示します。一致が発生した場合(_SOK)、Quantity を更新し、Master 内の元のオブザベーションを置き換えます。一致がない場合(_DSENOM)、Quantity を Transaction の AddQuantity 量に設定し、新しいオブザベーションを追加します。プログラムデータベクトルのコンテンツを SAS ログに書き込むエラー状態を防ぐために、_ERROR_は 0 にリセットされます。予期しない状態が発生した場合、メッセージとプログラムデータベクトルのコンテンツをログに書き込み、DATA ステップを停止します。

結果ログ

DATA ステップはエラーなしで実行され、オブザベーションは適切に更新および追加されました。次の行が SAS ログに書き込まれます。

```
NOTE: The data set WORK.MASTER has been updated.  There were
      2 observations rewritten, 1 observations added and 0
      observations deleted.
```

正しく更新された Master データセット

Master には、PartNumber 値 2 および 4 の更新された数量と、PartNumber 値 6 の新しいオブザベーションが含まれています。正しく更新された Master データセットを次に示しています。

Master

OBS	PartNumber	Quantity
1	1	10
2	2	32
3	3	30
4	4	54
5	5	50
6	6	16

例 2: KEY=を使用するすべてのステートメントでエラーチェックを使用する

概要

この例は、データを読み取るときに KEY=オプションを使用するすべてのステートメントでエラーチェックを使用することがいかに重要であるかを示しています。

入力データセット

Master データセットと Description データセットは両方とも PartNumber でインデックス付けされます。Order データセットには、単一の注文内のすべての部品の値が含まれます。Order のみに PartNumber 値 8 が含まれます。Master、Order、および Description 入力データセットを次に示します。

Master

OBS	PartNumber	
1	1	10
2	2	20
3	3	30
4	4	40
5	5	50

ORDER

Quantity	OBS	PartNumber
1	2	
2	4	
3	1	
4	3	
5	8	
6	5	
7	6	

Description

OBS	PartNumber	PartDescription
1	4	Nuts
2	3	Bolts
3	2	Screws
4	6	Washers

論理エラーのある元のプログラム

目標は、Master と Description の 2 つの入力データセットのどちらにも見つからない部品を除き、単一の注文内の各部品の説明と在庫数を含むデータセットを作成することです。トランザクションデータセットには、単一の注文内のすべての部品の

部品番号が含まれます。1 つのデータセットは部品の説明を取得するために読み取られ、もう 1 つのデータセットは在庫数量を取得するために読み取られます。

プログラムは、Order データセットを順次読み取り、次に KEY=オプションを指定した SET を使用して、Master データセットと Description データセットを直接読み取ります。この読み取りは、PartNumber のキー値に基づいています。一致が発生すると、Order 内の PartNumber の各値に必要なすべての情報を含むオブザベーションが書き込まれます。この最初の解決策の試みでは、KEY=を使用してデータセットを読み取る 2 つの SET ステートメントのうちの 1 つに対してのみエラーチェックを使用します。

```
data combine;  1
  length PartDescription $ 15;
  set order;  2
  set description key=PartNumber;  2
  set master key=PartNumber;  2
  select(_iorc_);  3
      when(%sysrc(_sok)) do;
        output;
      end;
      when(%sysrc(_dsenom)) do;
        PartDescription = 'No description';
        _error_ = 0;
        output;
      end;
      otherwise do;
        put 'ERROR: Unexpected value for _IORC_= ' _iorc_;
        put 'Program terminating.';
        put _all_;
        stop;
      end;
  end;
run;
```

1 Combine データセットを作成します。

2 Order データセットからオブザベーションを読み取ります。キー変数である PartNumber の一致する値に基づいて、Description および Master データセットからオブザベーションを読み取ります。Description からオブザベーションが読み取られた後、エラーチェックは行われないことに注意してください。

3 PartNumber に一致する値が Master または Description に見つかったかどうかに基づいて、適切なアクションを実行します。(このロジックは、この SELECT グループが KEY=オプションを含む先行する SET ステートメントの両方に対してエラーチェックを実行するという誤った想定に基づいています。実際には最新のものに対してのみエラーチェックを実行します。)SELECT グループは、実行を正しいコードに指示します。一致が発生した場合(_SOK)、Master から読み取られているオブザベーション内の PartNumber の値は、Order の現在の PartNumber 値と一致します。そこで、オブザベーションを出力します。一致しない場合(_DSENOM)、Master 内のオブザベーションには PartNumber の現在の値が含まれていないため、PartDescription の値を適切に設定してオブザベーションを出力します。プログラムデータベクトルのコンテンツを SAS ログに書き込むエラー状態を防ぐために、_ERROR_は 0 にリセットされます。予期しない状態が発生した場合、メッセージとプログラムデータベクトルのコンテンツをログに書き込み、DATA ステップを停止します。

結果ログ

このプログラムは出力データセットを作成しますが、1 つのエラーで実行されます。次の行が SAS ログに書き込まれます。

```
PartNumber=1 PartDescription=Nuts Quantity=10 _ERROR_=1
_IORC_=0 _N_=3
PartNumber=5 PartDescription=No description Quantity=50
_ERROR_=1 _IORC_=0 _N_=6
NOTE: The data set WORK.COMBINE has 7 observations and 3 variables.
```

結果データセット

誤って作成された Combine データセットを次に示しています。オブザベーション 5 はこのデータセットに含まれるべきではありません。PartNumber 値 8 は Master にも Description にも存在しないため、Quantity をリストするべきではありません。また、オブザベーション 3 と 7 には、それぞれオブザベーション 2 と 6 の説明が含まれています。

Combine

OBS	PartNumber	PartDescription	Quantity
1	2	Screws	20
2	4	Nuts	40
3	1	Nuts	10
4	3	Bolts	30
5	8	No description	30
6	5	No description	50
7	6	No description	50

改訂プログラム

正確な出力データセットを作成するために、この例では、KEY=オプションを使用する両方の SET ステートメントでエラーチェックを実行します。

```
data combine(drop=Foundes);  1
  length PartDescription $ 15;
  set order;  2
  Foundes = 0;  3
  set description key=PartNumber;  4
  select(_iorc_);  5
    when(%sysrc(_sok)) do;
      Foundes = 1;
    end;
    when(%sysrc(_dsenom)) do;
      PartDescription = 'No description';
      _error_ = 0;
```

```
    end;
    otherwise do;
      put 'ERROR: Unexpected value for _IORC_= ' _iorc_;
      put 'Program terminating. Data set accessed is Description';
      put _all_;
      _error_ = 0;
      stop;
    end;
  end;
set master key=PartNumber;  6
select(_iorc_); 7
    when(%sysrc(_sok)) do;
      output;
    end;
    when(%sysrc(_dsenom)) do;
      if not Foundes then do;
        _error_ = 0;
        put 'WARNING: PartNumber ' PartNumber 'is not in'
            ' Description or Master.';
      end;
      else do;
        Quantity = 0;
        _error_ = 0;
        output;
      end;
    end;
    otherwise do;
      put 'ERROR: Unexpected value for _IORC_= ' _iorc_;
      put 'Program terminating. Data set accessed is Master';
      put _all_;
      _error_ = 0;
      stop;
    end;
  end;     /* ends the SELECT group */
run;
```

1 Combine データセットを作成します。

2 Order データセットからオブザベーションを読み取ります。

3 変数 Foundes を作成し、その値を後で使用して、PartNumber 値が Description データセット内で一致するかどうかを示すことができるようにします。

4 PartNumber をキー変数として使用して、Description データセットからオブザベーションを読み取ります。

5 PartNumber に一致する値が Description に見つかったかどうかに基づいて、適切なアクションを実行します。SELECT グループは、_IORC_の値に基づいて、正しいコードに実行を指示します。一致が発生した場合(_SOK)、Description から読み取られているオブザベーション内の PartNumber の値は、Order の現在の値と一致します。Foundes は、Description が現在のオブザベーションに寄与したことを示すために 1 に設定されます。一致しない場合(_DSENOM)、Description 内のオブザベーションには PartNumber の現在の値が含まれていないため、説明は適切に設定されます。プログラムデータベクトルのコンテンツを SAS ログに書き込むエラー状態を防ぐために、_ERROR_は 0 にリセットされます。その他の_IORC_値は予期しない状態が満たされたことを示しており、メッセージがログに書き込まれ、DATA ステップが停止されます。

6 PartNumber をキー変数として使用して、Master データセットからオブザベーションを読み取ります。

7 PartNumber に一致する値が Master に見つかったかどうかに基づいて、適切なアクションを実行します。Order と Master の現在の PartNumber 値の間に一致が見つかった場合(_SOK)、オブザベーションを書き込みます。Master で一致が見つからない場合(_DSENOM)、Foundes の値をテストします。Foundes が真でない場合は、Description にも値が見つからなかったため、ログにメッセージを書き込みますが、オブザベーションは書き込みません。ただし、Foundes が真の場合、値は Description には含まれますが、Master には含まれません。したがって、オブザベーションを書き込みますが、Quantity を 0 に設定します。この場合も、予期しない状態が発生した場合は、メッセージを書き込んで DATA ステップを停止します。

結果ログ

DATA ステップはエラーなしで実行されました。6 つのオブザベーションが正しく作成され、次のメッセージがログに書き込まれました。

```
WARNING: PartNumber 8 is not in Description or Master.
NOTE: The data set WORK.COMBINE has 6 observations
      and 3 variables.
```

正しく作成された Combine データセット

正しく更新された Combine データセットを次に示しています。Combine には PartNumber 値が 8 のオブザベーションが含まれていないことに注意してください。この値は Master にも Description にも存在しません。

Combine

OBS	PartNumber	PartDescription	Quantity
1	2	Screws	20
2	4	Nuts	40
3	1	No description	10
4	3	Bolts	30
5	5	No description	50
6	6	Washers	0

システムオプションを使用してエラー処理を制御

次のシステムオプションを使用して、プログラム内のエラー処理(エラーの解決)を制御できます。

BYERR
SORT プロシジャが_NULL_データセットを処理しようとしたときに SAS がエラーを生成するかどうかを指定します。

CHKPTCLEAN
チェックポイントモードまたはリセットモードで、バッチプログラムが正常に実行された場合に Work ディレクトリ内のファイルを消去するかどうかを指定します。

DKRICOND=
エラー DROP=、KEEP=、および RENAME=データセットオプションの処理中に入力データセットから変数が欠損している場合にレポートするエラー検出のレベルを指定します。

DKROCOND=
DROP=、KEEP=、および RENAME=データセットオプションの処理中に出力データセットから変数が欠損している場合にレポートするエラー検出のレベルを指定します。

DSNFERR
SAS データセットが見つからない場合に、SAS がエラーメッセージを発行するかどうかを指定します。

ERRORABEND
SAS がエラーに応答して終了するかどうかを指定します。

ERRORCHECK=
LIBNAME、FILENAME、%INCLUDE、LOCK ステートメントでエラーが見つかった場合に SAS が構文チェックモードに入るかどうかを指定します。

ERRORS=
SAS が完全なエラーメッセージを発行するオブザベーションの最大数を指定します。

FMTERR
変数の出力形式が見つからない場合に、SAS がエラーを生成するか、処理を続行するかを指定します。

INVALIDDATA=
無効な数値データが検出された場合に SAS が変数に割り当てる値を指定します。

LABELCHKPT
ラベル付きコードセクションを含むバッチプログラムに対して SAS チェックポイント/再起動データを記録するかどうかを指定します。

LABELCHKPTLIB
ラベル付きコードセクションのチェックポイント/再起動データが保存されるライブラリのライブラリ参照名を指定します。

LABELRESTART
ラベル付きコードセクションのチェックポイント/再起動データを使用してバッチプログラムを実行するかどうかを指定します。

MERROR
マクロのような名前がマクロキーワードと一致しない場合に SAS が警告メッセージを発行するかどうかを指定します。

QUOTELENMAX
引用符で囲まれた文字列が許容される最大長を超えた場合に、SAS が SAS ログに警告メッセージを書き込むかどうかを指定します。

SERROR
マクロ変数参照がマクロ変数と一致しない場合に SAS が警告メッセージを発行するかどうかを指定します。

STEPCHKPT
バッチプログラムに対してチェックポイント/再起動データを記録するかどうかを指定します。

STEPCHKPTLIB=
チェックポイント/再起動データが保存されるライブラリのライブラリ参照名を指定します。

STEPRESTART
チェックポイント/再起動データを使用してバッチプログラムを実行するかどうかを指定します。

VARINITCHK=
変数が初期化されていない場合に DATA ステップの処理を停止するか続行するかを指定します。SAS ログに書き込まれるメッセージの種類を指定することもできます。

VNFERR
BY 変数が 1 つのデータセットに存在するが別のデータセットには存在しない場合に、SAS がエラーまたは警告を発行するかどうかを指定します。SAS は、SET、MERGE、UPDATE、MODIFY のいずれかのステートメントを処理するときにのみ、これらのエラーまたは警告を発行します。

SAS システムオプションの詳細については、*SAS システムオプション: リファレンス* を参照してください。

その他のエラーチェックオプション

プログラミングエラーを特定するには、次の方法を使用できます。

- MODIFY ステートメントまたは SET ステートメントの KEY=データセットオプションを使用するとき、SAS が作成する_IORC_自動変数(および関連する IORCMSG 関数)
- SAS がログに書き込む同一エラーの数を制限する ERRORS=システムオプション
- データセットまたは外部ファイルアクセス関数がエラー状態に遭遇したときに情報を返す SYSRC および SYSMSG 関数
- リターンコードを受け取るための SYSRC 自動マクロ変数
- メモリ不足やコンポーネントシステムの障害などの主要なシステムエラーを検出するための SYSERR 自動マクロ変数
- ログ制御オプション:

  MSGLEVEL=
  SAS ログに書き込まれるメッセージの詳細レベルを制御します。

  PRINTMSGLIST
  SAS ログへのメッセージの拡張リストの出力を制御します。

SOURCE
SAS がソーステートメントを SAS ログに書き込むかどうかを制御します。

SOURCE2
SAS が%INCLUDE によって含まれるソーステートメントを SAS ログに書き込むかどうかを制御します。

構文エラーのデバッグ

プログラムステートメントが SAS 言語のルールに準拠していない場合、構文エラーが発生します。構文エラーの例をいくつか示します。

- SAS キーワードのスペルミス
- 引用符の不一致
- セミコロンの欠落
- 無効なステートメントオプション
- 無効なデータセットオプション

SAS は構文エラーが発生すると、コードの意図を解釈してエラーを修正しようとします。その後、SAS はこれらの仮定に基づいてプログラムの処理を続行します。SAS がエラーを修正できない場合は、ログにエラーメッセージを出力します。SAS で構文エラーを修正しない場合は、NOAUTOCORRECT システムオプションを設定できます。詳細については、“AUTOCORRECT システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

次の例では、DATA ステートメントにスペルミスがあるため、SAS はログに警告メッセージを出力します。SAS はスペルミスのある単語を解釈できるため、プログラムが実行され、出力が生成されます。

```
date temp;  /* 1  */
  x=1;
run;

proc print data=temp;
run;
```

1 data のスペルミスが date です。

例のコード 24.1　SAS ログ: 構文エラー(キーワードのスペルミス)

```
39   date temp;
     ----
     14
WARNING 14-169: Assuming the symbol DATA was misspelled as date.

40       x=1;
41   run;
NOTE: The data set WORK.TEMP has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

42
43   proc print data=temp;
44   run;
NOTE: There were 1 observations read from the data set WORK.TEMP.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

45   proc printto; run;
```

一部のエラーは、SAS がログに出力するメッセージによって完全に説明されます。SAS ではエラーが発生した場所を正確に検出できないことがあるため、その他のエラーメッセージは解釈が容易ではありません。たとえば、SAS ステートメントをセミコロンで終了しなかった場合、SAS はエラーが発生した時点でエラーを検出するとは限りません。SAS ステートメントの自由形式の性質(どこからでも開始および終了できる)により、この問題が発生する可能性があります。次の例では、DATA ステートメントの末尾のセミコロンがありません。SAS はログに ERROR という単語を出力し、エラーの可能性のある場所を特定し、エラーの説明を出力し、DATA ステップの処理を停止します。

```
data temp
  x=1;
run;

proc print data=temp;
run;
```

例のコード 24.2　SAS ログ: 構文エラー(セミコロンの欠落)

```
67   data temp
68      x=1;
        -
        22
        76
ERROR 22-322: Syntax error, expecting one of the following: a name,
           a quoted string, (, /, ;, _DATA_, _LAST_, _NULL_.

ERROR 76-322: Syntax error, statement will be ignored.

69   run;
NOTE: The SAS System stopped processing this step because of errors.
NOTE: DATA statement used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds

70
71   proc print data=temp;
72   run;
NOTE: There were 1 observations read from the data set WORK.TEMP.
NOTE: PROCEDURE PRINT used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds

73   proc printto; run;
```

後続のステップが実行されるかどうかは、SAS を実行するために使用する方法と動作環境によって異なります。

注: これらの行をコードに追加することで、コメントタグの不一致、引用符の不一致、セミコロンの欠落を修正することができます。

```
/* '; * "; */;
quit;
run;
```

意味エラー

意味エラーは、SAS ステートメント内の要素の形式は正しいが、その要素がその使用法に対して有効でない場合に発生します。意味エラーはコンパイル時に検出され、SAS が構文チェックモードに入る原因となる可能性があります。(構文チェックモードの説明については、“構文チェックモード” (638 ページ)を参照してください。)

意味エラーの例には次のものがあります。

- 関数の引数の数を間違えて指定する
- 文字変数のみが有効な場合に数値変数名を使用する
- 配列への無効な参照を使用する
- 変数が初期化されていない

次の例では、SAS はコンパイル時に配列 all への無効な参照を検出します。

```
data _null_;
  array all{*} x1-x5;
  all=3;
  datalines;
1 1.5
. 3
2 4.5
3 2 7
3 . .
;

run;
```

例のコード 24.3　SAS ログ: 意味エラー(配列への無効な参照)

```
81   data _null_;
82     array all{*} x1-x5;
ERROR: invalid reference to the array all.
83     all=3;
84     datalines;
NOTE: The SAS System stopped processing this step because of errors.
NOTE: DATA statement used (Total process time):
      real time          0.15 seconds
      cpu time           0.01 seconds

90   ;
91
92   run;
93   proc printto; run;
```

コンパイル時に発生する意味エラーの別の例を次に示します。この DATA ステップでは、ライブラリ参照名 SomeLib は LIBNAME ステートメントで事前に割り当てられていません。

```
data test;
  set somelib.old;
run;
```

例のコード 24.4　SAS ログ: 意味エラー(ライブラリ参照名が事前に割り当てられていない)

```
101  data test;
ERROR: Libname SomeLib is not assigned.
102    set somelib.old;
103  run;
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.TEST may be incomplete.  When this step was
         stopped there were 0 observations and 0 variables.
```

実行時に意味エラーが発生すると、SAS は"NOTE: SAS went to a new line when input statement reached past the end of a line."のようなメッセージを書き込みます。この注記は、FLOWOVER が使用され、INPUT ステートメント内のすべての変数を完全に読み取ることができない場合に SAS ログに書き込まれます。

初期化されていない変数の検出は、別の意味エラーです。デフォルトでは、SAS はエラーを報告しませんが、SAS ログに注記を書き込みます。変数が初期化されておらず、システムオプション VARINITCHK=ERROR が発行されている場合、SAS は DATA ステップの処理を停止し、SAS ログにエラーメッセージを書き込みます。

実行時エラー

定義

実行時エラーは、SAS がデータ値を処理するプログラムを実行するときに発生します。ほとんどの実行時エラーでは、SAS ログに警告メッセージまたは注記が生成されますが、プログラムは実行を継続できます。実行時エラーの場所は通常、注記またはエラーメッセージ内の行番号と列番号として示されます。

注: SAS を非対話モードで実行すると、より重大なエラーにより SAS が構文チェックモードに入り、プログラムの処理が停止する可能性があります。

一般的な実行時エラーには次のようなものがあります。

- 関数への無効な引数
- 無効な数学演算(たとえば、0 による除算)
- BY グループ処理の順序が間違っているオブザベーション
- 配列の存在しないメンバーへの参照(配列の添え字が範囲外の場合に発生)
- INFILE および FILE ステートメントでの SAS データセットおよびその他のファイルの開け閉めのエラー
- データ行と一致しない INPUT ステートメント(たとえば、変数の列を間違ってリストしたり、変数が文字変数であることを示していない INPUT ステートメント)

リソース不足状態

実行時エラーは、ディスクがいっぱいになったり、SAS プロシジャを完了するためのメモリが不足したりするなど、リソース不足状態に陥った場合にも発生する可能性があります。これらの状態が発生すると、SAS は現在使用できるリソースを見つけようとします。たとえば、SAS は、リソース不足状態で次のアクションを実行するためのアクセス許可をユーザーに求める場合があります。

- 不要になった可能性のある一時データセットを削除します。
- マクロ変数が保存されているメモリを解放します。

例

次の例では、SAS が 2 番目のオブザベーションのデータ値を使用して割り当てステートメントで除算演算を実行するときに、実行時エラーが発生します。0 による除算は無効な数学演算であり、実行時エラーが発生します。

```
data inventory;
  input Item $ 1-14 TotalCost 15-20
      UnitsOnHand 21-23;
  UnitCost=TotalCost/UnitsOnHand;
  datalines;
Hammers      440   55
Nylon cord    35    0
Ceiling fans  1155  30
;

proc print data=inventory;
  format TotalCost dollar8.2 UnitCost dollar8.2;
run;
```

例のコード 24.5　SAS ログ: 実行時エラー(0 による除算)

```
115  data inventory;
116    input Item $ 1-14 TotalCost 15-20
117          UnitsOnHand 21-23;
118    UnitCost=TotalCost/UnitsOnHand;
119    datalines;
NOTE: Division by zero detected at line 118 column 22.
RULE:      ----+----1----+----2----+----3----+----4----+----5---
121        Nylon cord    35    0
Item=Nylon cord TotalCost=35 UnitsOnHand=0 UnitCost=. _ERROR_=1
_N_=2
NOTE: Mathematical operations could not be performed at the
      following places. The results of the operations have been
      set to missing values.
      Each place is given by:
      (Number of times) at (Line):(Column).
      1 at 118:22
NOTE: The data set WORK.INVENTORY has 3 observations and 4
      variables.
NOTE: DATA statement used (Total process time):
      real time          0.03 seconds
      cpu time           0.00 seconds

123  ;
124
125  proc print data=inventory;
126    format TotalCost dollar8.2 UnitCost dollar8.2;
127  run;

NOTE: Writing HTML Body file: sashtml1.htm
NOTE: There were 3 observations read from the data set
      WORK.INVENTORY.
NOTE: PROCEDURE PRINT used (Total process time):
      real time          0.56 seconds
      cpu time           0.01 seconds
```

アウトプット 24.1　SAS 出力: 実行時エラー(0 による除算)

The SAS System

Obs	Item	TotalCost	UnitsOnHand	UnitCost
1	Hammers	$440.00	55	$8.00
2	Nylon cord	$35.00	0	.
3	Ceiling fans	$1155.00	30	$38.50

SAS はステップ全体を実行し、出力内の変数 UnitCost に欠損値を割り当て、次の内容を SAS ログに書き込みます。

- エラーについて記述する注記
- 入力バッファーに保存されている値
- エラーが発生したときのプログラムデータベクトルのコンテンツ
- エラーを説明する注記

プログラムデータベクトルにリストされる値には、_N_および_ERROR_自動変数が含まれることに注意してください。これらの自動変数は各オブザベーションに一時的に割り当てられ、データセットには保存されません。

次の実行時エラーの例では、プログラムが配列を処理し、SAS が範囲外の配列の添字の値を検出します。SAS はログにエラーメッセージを出力し、処理を停止します。

```
data test;
  array all{*} x1-x3;
  input I measure;
  if measure > 0 then
    all{I} = measure;
  datalines;
1 1.5
. 3
2 4.5
;

proc print data=test;
run;
```

例のコード 24.6　SAS ログ: 実行時エラー(添字が範囲外)

```
163  data test;
164     array all{*} x1-x3;
165     input I measure;
166     if measure > 0 then
167        all{I} = measure;
168     datalines;
ERROR: Array subscript out of range at line 167 column 7.
RULE:      ----+----1----+----2----+----3----+----4----+----5---
170        . 3
x1=. x2=. x3=. I=. measure=3 _ERROR_=1 _N_=2
NOTE: The SAS System stopped processing this step because of
      errors.
WARNING: The data set WORK.TEST may be incomplete.  When this
         step was stopped there were 1 observations and 5
         variables.
WARNING: Data set WORK.TEST was not replaced because this step
         was stopped.
NOTE: DATA statement used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds

172  ;
173
174  proc print data=test;
175  run;
NOTE: No variables in data set WORK.TEST.
NOTE: PROCEDURE PRINT used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds

176  proc printto; run;
```

データエラー

定義

データエラーは、一部のデータ値がプログラムで指定した SAS ステートメントに適していない場合に発生します。たとえば、変数を数値として定義したが、データ値が実際には文字である場合、SAS はデータエラーを生成します。SAS はプログラム実行中にデータエラーを検出し、プログラムの実行を継続して、次の操作を実行します。

- 無効なデータの注記を SAS ログに書き込みます。
- 無効な値を含む入力行とカラム番号を SAS ログに出力します。印字不可能な文字は 16 進数で表示されます。カラム番号を判別しやすくするために、SAS は入力行の上に罫線を出力します。
- 罫線の下にオブザベーション結果を出力します。

- 現在のオブザベーションに対して自動変数_ERROR_を 1 に設定します。

この例では、Number 変数の文字値により、プログラム実行中にデータエラーが発生します。

```
data age;
  input Name $ Number;
  datalines;
Sue 35
Joe xx
Steve 22
;

proc print data=age;
run;
```

SAS ログには、プログラムの 8 行目、位置 5-6 にエラーがあることが示されています。

例のコード 24.7　SAS ログ: データエラー

```
234  data age;
235    input Name $ Number;
236    datalines;
NOTE: Invalid data for Number in line 238 5-6.
RULE:      ----+----1----+----2----+----3----+----4----+----5---
238        Joe xx
Name=Joe Number=. _ERROR_=1 _N_=2
NOTE: The data set WORK.AGE has 3 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time          0.01 seconds
      cpu time           0.00 seconds

240  ;
241
242  proc print data=age;
243  run;

NOTE: Writing HTML Body file: sashtml2.htm
NOTE: There were 3 observations read from the data set WORK.AGE.
NOTE: PROCEDURE PRINT used (Total process time):
      real time          0.07 seconds
      cpu time           0.04 seconds
```

アウトプット 24.2　SAS 出力: データエラー

The SAS System

Obs	Name	Number
1	Sue	35
2	Joe	.
3	Steve	22

INVALIDDATA=システムオプションを使用すると、プログラムで無効なデータが発生したときに変数に値を割り当てることもできます。詳細については、*SAS システ*

*ムオプション: リファレンス*の INVALIDDATA=システムオプションを参照してください。

エラーレポートのフォーマット修飾子

INPUT ステートメントは、エラーレポートに?および??フォーマット修飾子を使用します。フォーマット修飾子は、SAS ログに書き込まれる情報の量を制御します。?と??の両方の修飾子は無効なデータメッセージを抑制します。ただし、??修飾子は自動変数_ERROR_も 0 に設定します。たとえば、次の 2 つのステートメントセットは同等です。

- input x ?? 10-12;
- input x ? 10-12;
  error=0;

どちらの場合でも、SAS は X の無効な値を欠損値に設定します。

マクロ関連エラー

マクロ関連エラーにはいくつかの種類があります。

- マクロ機能自体を使用するときに生成されるマクロコンパイル時およびマクロ実行時のエラー
- マクロ機能によって生成された SAS コードのエラー

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。

例

例 1: 予期しない状態が発生した場合のルート実行

コード例

この例では、予期しない状態によって DATA ステップが終了しないようにする方法を示します。目標は、トランザクションデータセットからの新しい情報を使用してマスターデータセットを更新することです。このアプリケーションでは、どちらのデータセットでも共通変数に重複値が存在しないことを前提としています。

注: このプログラムは、マスターデータセットとトランザクションデータセットに共通変数に同じ値を持つ連続したオブザベーションが含まれていない場合にのみ、期待どおりに動作します。重複が存在する場合の KEY=を指定した MODIFY の動作の説明については、*SAS DATA ステップステートメント: リファレンス*の MODIFY ステートメントを参照してください。

Transaction データセットには、Master の情報に対する 2 つの更新と、追加する必要がある PartNumber 値 6 に関する新しいオブザベーションの 3 つのオブザベーションが含まれています。Master は PartNumber に基づいてインデックス付けされます。Master または Transaction に PartNumber の重複値はありません。Master および Transaction 入力データセットを次に示します。

	Master			Transaction	
OBS	PartNumber	Quantity	OBS	PartNumber	AddQuantity
1	1	10	1	4	14
2	2	20	2	6	16
3	3	30	3	2	12
4	4	40			
5	5	50			

目標は、Transaction データセットの情報を使用して Master データセットを更新することです。プログラムは Transaction を順次読み取ります。Master は、MODIFY ステートメントと KEY=オプションを使用して、順次ではなく直接読み取られます。KEY=変数である PartNumber の値が一致するオブザベーションのみが Master から読み取られます。

```
data master;                        /* 1 */
  set transaction;                  /* 2 */
  modify master key=PartNumber;     /* 3 */
  Quantity = Quantity + AddQuantity;  /* 4 */
run;
```

1 更新のために Master データセットを開きます。

2 Transaction データセットからオブザベーションを読み取ります。

3 PartNumber の値に基づいて、Master データセットからのオブザベーションを一致させます。

4 Transaction データセットから新しい値を追加して、Quantity に関する情報を更新します。

このプログラムは 1 つのオブザベーションを正しく更新しましたが、PartNumber 値 6 に一致するものが見つからなかったため停止しました。次の行が SAS ログに書き込まれます。

```
ERROR: No matching observation was found in Master data set.
PartNumber=6 AddQuantity=16 Quantity=70 _ERROR_=1
_IORC_=1230015 _N_=2
NOTE: The SAS System stopped processing this step because
    of errors.
NOTE: The data set WORK.MASTER has been updated.  There were
    1 observations rewritten, 0 observations added and 0
    observations deleted.
```

結果データセット: Master ファイルが正しく更新されませんでした。更新された Master には 5 つのオブザベーションがあります。1 つのオブザベーションは正しく更新されましたが、新しいオブザベーションは追加されず、2 番目の更新は行われませんでした。誤って更新された Master データセットを次に示しています。

Master

OBS	PartNumber	Quantity
1	1	10
2	2	20
3	3	30
4	4	54
5	5	50

目標は、Master に 2 つの更新と 1 つの追加を適用することです。このアクションにより、Transaction の PartNumber 値 6 に一致するものが Master に見つからない場合に、DATA ステップが停止することがなくなります。エラーチェックを追加することで、この DATA ステップは正常に完了し、Master の正しく修正されたバージョンが生成されます。このプログラムは、SELECT グループ内の_IORC_自動変数と SYSRC 自動呼び出しマクロを使用して、_IORC_変数の値をチェックします。一致が見つかった場合、プログラムは適切なコードを実行します。

```
data master;                        /* 1 */
  set transaction;                  /* 2 */
  modify master key=PartNumber;     /* 3 */

select(_iorc_);                     /* 4 */
    when(%sysrc(_sok)) do;
      Quantity = Quantity + AddQuantity;
      replace;
    end;
    when(%sysrc(_dsenom)) do;
      Quantity = AddQuantity;
      _error_ = 0;
      output;
    end;
    otherwise do;
      put 'ERROR: Unexpected value for _IORC_= ' _iorc_;
      put 'Program terminating. DATA step iteration # ' _n_;
      put _all_;
      stop;
    end;
  end;
run;
```

1 更新のために Master データセットを開きます。

2 Transaction データセットからオブザベーションを読み取ります。

3 PartNumber の値に基づいて、Master データセットからのオブザベーションを一致させます。

4 PartNumber に一致する値が Master に見つかったかどうかに基づいて、適切なアクションを実行します。Transaction からの新しい値を追加して Quantity を更新します。SELECT グループは、実行を正しいコードに指示します。一致が発生した場合(_SOK)、Quantity を更新し、Master 内の元のオブザベーションを置き換えます。一致がない場合(_DSENOM)、Quantity を Transaction の AddQuantity 量に設定し、新しいオブザベーションを追加します。プログラムデ

ータベクトルのコンテンツを SAS ログに書き込むエラー状態を防ぐために、_ERROR_は 0 にリセットされます。予期しない状態が発生した場合、メッセージとプログラムデータベクトルのコンテンツをログに書き込み、DATA ステップを停止します。

例: KEY=を使用するステートメントでエラーチェックを使用する

概要

データの読み取り時に KEY=オプションを使用するすべてのステートメントでエラーチェックを使用することが重要です。

入力データセット

Master データセットと Description データセットは両方とも PartNumber でインデックス付けされます。Order データセットには、単一の注文内のすべての部品の値が含まれます。Order のみに PartNumber 値 8 が含まれます。Master、Order、および Description 入力データセットを次に示します。

Master

OBS	PartNumber	Quantity
1	1	10
2	2	20
3	3	30
4	4	40
5	5	50

ORDER

OBS	PartNumber
1	2
2	4
3	1
4	3
5	8
6	5
7	6

Description

OBS	PartNumber	PartDescription
1	4	Nuts
2	3	Bolts
3	2	Screws
4	6	Washers

論理エラーのある元のプログラム

目標は、Master と Description の 2 つの入力データセットのどちらにも見つからない部品を除き、単一の注文内の各部品の説明と在庫数を含むデータセットを作成することです。トランザクションデータセットには、単一の注文内のすべての部品の部品番号が含まれます。1 つのデータセットは部品の説明を取得するために読み取られ、もう 1 つのデータセットは在庫数量を取得するために読み取られます。

プログラムは、Order データセットを順次読み取り、次に KEY=オプションを指定した SET を使用して、Master データセットと Description データセットを直接読み取ります。この読み取りは、PartNumber のキー値に基づいています。一致が発生すると、Order 内の PartNumber の各値に必要なすべての情報を含むオブザベーションが書き込まれます。この最初の解決策の試みでは、KEY=を使用してデータセットを読み取る 2 つの SET ステートメントのうちの 1 つに対してのみエラーチェックを使用します。

```
data combine;                    /* 1 */
  length PartDescription $ 15;
  set order;                     /* 2 */
  set description key=PartNumber;  /* 2 */
  set master key=PartNumber;       /* 2 */
  select(_iorc_);                /* 3 */
      when(%sysrc(_sok)) do;
        output;
      end;
      when(%sysrc(_dsenom)) do;
        PartDescription = 'No description';
        _error_ = 0;
        output;
      end;
      otherwise do;
        put 'ERROR: Unexpected value for _IORC_= ' _iorc_;
        put 'Program terminating.';
        put _all_;
        stop;
      end;
  end;
run;
```

1 Combine データセットを作成します。

2 Order データセットからオブザベーションを読み取ります。キー変数である PartNumber の一致する値に基づいて、Description および Master データセットからオブザベーションを読み取ります。Description からオブザベーションが読み取られた後、エラーチェックは行われないことに注意してください。

3 PartNumber に一致する値が Master または Description に見つかったかどうかに基づいて、適切なアクションを実行します。(このロジックは、この SELECT グループが KEY=オプションを含む先行する SET ステートメントの両方に対してエラーチェックを実行するという誤った想定に基づいています。実際には最新のものに対してのみエラーチェックを実行します。)SELECT グループは、実行を正しいコードに指示します。一致が発生した場合(_SOK)、Master から読み取られているオブザベーション内の PartNumber の値は、Order の現在の

PartNumber 値と一致します。そこで、オブザベーションを出力します。一致しない場合(_DSENOM)、Master 内のオブザベーションには PartNumber の現在の値が含まれていないため、PartDescription の値を適切に設定してオブザベーションを出力します。プログラムデータベクトルのコンテンツを SAS ログに書き込むエラー状態を防ぐために、_ERROR_は 0 にリセットされます。予期しない状態が発生した場合、メッセージとプログラムデータベクトルのコンテンツをログに書き込み、DATA ステップを停止します。

このプログラムは出力データセットを作成しますが、1 つのエラーで実行されます。次の行が SAS ログに書き込まれます。

```
PartNumber=1 PartDescription=Nuts Quantity=10 _ERROR_=1
  _IORC_=0 _N_=3
  PartNumber=5 PartDescription=No description Quantity=50
  _ERROR_=1 _IORC_=0 _N_=6
  NOTE: The data set WORK.COMBINE has 7 observations and 3 variables.
```

結果データセット

誤って作成された Combine データセットを次に示しています。オブザベーション 5 はこのデータセットに含まれるべきではありません。PartNumber 値 8 は Master にも Description にも存在しないため、Quantity をリストするべきではありません。また、オブザベーション 3 と 7 には、それぞれオブザベーション 2 と 6 の説明が含まれています。

Combine

OBS	PartNumber	PartDescription	Quantity
1	2	Screws	20
2	4	Nuts	40
3	1	Nuts	10
4	3	Bolts	30
5	8	No description	30
6	5	No description	50
7	6	No description	50

改訂プログラム

正確な出力データセットを作成するために、この例では、KEY=オプションを使用する両方の SET ステートメントでエラーチェックを実行します。

```
data combine(drop=Foundes);       /* 1 */
  length PartDescription $ 15;
  set order;                    /* 2 */
  Foundes = 0;                    /* 3 */
  set description key=PartNumber;  /* 4 */
  select(_iorc_);                /* 5 */
    when(%sysrc(_sok)) do;
      Foundes = 1;
```

```
    end;
    when(%sysrc(_dsenom)) do;
      PartDescription = 'No description';
      _error_ = 0;
    end;
    otherwise do;
      put 'ERROR: Unexpected value for _IORC_= ' _iorc_;
      put 'Program terminating. Data set accessed is Description';
      put _all_;
      _error_ = 0;
      stop;
    end;
  end;
set master key=PartNumber;          /* 6 */
select(_iorc_);                     /* 7 */
    when(%sysrc(_sok)) do;
      output;
    end;
    when(%sysrc(_dsenom)) do;
      if not Foundes then do;
        _error_ = 0;
        put 'WARNING: PartNumber ' PartNumber 'is not in'
            ' Description or Master.';
      end;
      else do;
        Quantity = 0;
        _error_ = 0;
        output;
      end;
    end;
    otherwise do;
      put 'ERROR: Unexpected value for _IORC_= ' _iorc_;
      put 'Program terminating. Data set accessed is Master';
      put _all_;
      _error_ = 0;
      stop;
    end;
  end;     /* ends the SELECT group */
run;
```

1 Combine データセットを作成します。

2 Order データセットからオブザベーションを読み取ります。

3 変数 Foundes を作成し、その値を後で使用して、PartNumber 値が Description データセット内で一致するかどうかを示すことができるようにします。

4 PartNumber をキー変数として使用して、Description データセットからオブザベーションを読み取ります。

5 PartNumber に一致する値が Description に見つかったかどうかに基づいて、適切なアクションを実行します。SELECT グループは、_IORC_の値に基づいて、正しいコードに実行を指示します。一致が発生した場合(_SOK)、Description から読み取られているオブザベーション内の PartNumber の値は、Order の現在の値と一致します。Foundes は、Description が現在のオブザベーションに寄与したことを示すために 1 に設定されます。一致しない場合(_DSENOM)、Description 内のオブザベーションには PartNumber の現在の値が含まれていないため、説明は適切に設定されます。プログラムデータベクトルのコンテンツ

を SAS ログに書き込むエラー状態を防ぐために、_ERROR_は 0 にリセットされます。その他の_IORC_値は予期しない状態が満たされたことを示しており、メッセージがログに書き込まれ、DATA ステップが停止されます。

6 PartNumber をキー変数として使用して、Master データセットからオブザベーションを読み取ります。

7 PartNumber に一致する値が Master に見つかったかどうかに基づいて、適切なアクションを実行します。Order と Master の現在の PartNumber 値の間に一致が見つかった場合(_SOK)、オブザベーションを書き込みます。Master で一致が見つからない場合(_DSENOM)、Foundes の値をテストします。Foundes が真でない場合は、Description にも値が見つからなかったため、ログにメッセージを書き込みますが、オブザベーションは書き込みません。ただし、Foundes が真の場合、値は Description には含まれますが、Master には含まれません。したがって、オブザベーションを書き込みますが、Quantity を 0 に設定します。この場合も、予期しない状態が発生した場合は、メッセージを書き込んで DATA ステップを停止します。

DATA ステップはエラーなしで実行されました。6 つのオブザベーションが正しく作成され、次のメッセージがログに書き込まれました。

例のコード 24.8　ログ出力

```
WARNING: PartNumber 8 is not in Description or Master.
NOTE: The data set WORK.COMBINE has 6 observations
      and 3 variables.
```

正しく作成された Combine データセット

正しく更新された Combine データセットを次に示しています。Combine には PartNumber 値が 8 のオブザベーションが含まれていないことに注意してください。この値は Master にも Description にも存在しません。

Combine

OBS	PartNumber	PartDescription	Quantity
1	2	Screws	20
2	4	Nuts	40
3	1	No description	10
4	3	Bolts	30
5	5	No description	50
6	6	Washers	0

例: 複数のエラーを処理

コード例

エラーの種類と重要度、SAS の実行方法、および動作環境に応じて、SAS はプログラム処理を停止するか、エラーにフラグを設定して処理を続行します。SAS は、特定の種類のエラーを検出した後も、プロシジャ内の個々のステートメントのチェックを続行します。場合によっては、SAS は 1 つのステートメントで複数のエラーを検出し、特定の状況に対してさらに多くのエラーメッセージを発行することがあります。これは、エラーを含むステートメントが出力 SAS データセットを作成する場合に発生する可能性があります。

```
data temporary;
  Item1=4;
run;

proc print data=temporary;
  var Item1 Item2 Item3;
run;
```

例のコード 24.9　SAS ログ: 複数のプログラムエラー

```
273  data temporary;
274    Item1=4;
275  run;
NOTE: The data set WORK.TEMPORARY has 1 observations and 1
    variables.
NOTE: DATA statement used (Total process time):
    real time          0.01 seconds
    cpu time           0.01 seconds

276
277  proc print data=temporary;
ERROR: Variable ITEM2 not found.
ERROR: Variable ITEM3 not found.
278    var Item1 Item2 Item3;
279  run;
NOTE: The SAS System stopped processing this step because of
    errors.
NOTE: PROCEDURE PRINT used (Total process time):
    real time          0.52 seconds
    cpu time           0.00 seconds

280  proc printto; run;
```

SAS は、変数 Item2 と変数 Item3 の 2 つのエラーメッセージを表示します。

25

システムパフォーマンスの最適化

システムパフォーマンスの最適化の定義

パフォーマンス統計量
: 個々の DATA ステップまたは PROC ステップの処理に使用される入出力操作 (I/O)、メモリ、および CPU 時間の合計の測定値です。SAS システムオプションを使用してこれらの統計量を取得すると、ジョブの初期パフォーマンスを測定し、パフォーマンスを向上させる方法を決定するのに役立ちます。

システムパフォーマンス
: システムが SAS プログラムを処理するために使用する I/O、メモリ、および CPU 時間の総量によって測定されます。次のセクションで説明する手法を使用すると、これら 3 つの重要なリソースの使用量を削減または再割り当てして、システムパフォーマンスを向上させることができます。すべての状況にすべての手法を活用できるわけではありませんが、特定の状況に最も適した手法を選択することはできます。

パフォーマンス統計量の収集と解釈

FULLSTIMER および STIMER システムオプションの使用

FULLSTIMER および STIMER システムオプションは、SAS ログ内のパフォーマンス統計量の出力を制御します。これらのオプションは、動作環境に応じて異なる結果を生成します。これらのオプションに対して SAS が生成する出力の詳細については、使用している動作環境の SAS ドキュメントを参照してください。

次の出力は、UNIX 動作環境で生成された、SAS ログ内の FULLSTIMER 出力の例を示しています。

例のコード 25.1　UNIX 動作環境で FULLSTIMER オプションを使用した結果のサンプル

```
NOTE: DATA statement used:
      real time           0.19 seconds
      user cpu time       0.06 seconds
      system cpu time     0.01 seconds
      Memory                       460k
      Semaphores    exclusive 194 shared 9 contended 0
      SAS Task context switches       1   splits 0
```

STIMER オプションは、FULLSTIMER 統計量のサブセットを報告します。次の例は、SAS ログの STIMER 出力を示しています。

例のコード 25.2　UNIX 動作環境で STIMER オプションを使用した結果のサンプル

```
NOTE: DATA statement used:
      real time           1.16 seconds
      cpu time            0.09 seconds
```

FULLSTIMER および STIMER 統計量の解釈

STIMER および FULLSTIMER オプションによって、処理時間(経過時間)や CPU 時間を含む、いくつかのタイプのリソース使用状況統計量がレポートされます。処理時間は、ジョブまたはステップの実行にかかったクロックタイムを表します。それはシステムの容量と現在の負荷に大きく依存します。特定のリソースを共有するユーザーが増えると、利用できるリソースが減ります。CPU 時間は、ジョブの実行に CPU が必要とする実際の処理時間を表します(容量および負荷係数は除く)。リソースを長く待つ必要がある場合、CPU 時間は増加しませんが、処理時間は増加します。プログラムの効率性の唯一の基準として処理時間を使用することはお勧めできません。その理由は、システムの容量と負荷の要求を常に制御できるわけではないためです。システムパフォーマンスのより正確な評価は CPU 時間であり、プログラムを変更して効率を高めると、CPU 時間はより予測どおりに減少します。

FULLSTIMER によってレポートされる統計量は、I/O、メモリ、CPU 時間という 3 つの重要なコンピューターリソースに関連しています。多くの状況では、これら 3 つのリソースのいずれかの使用を減らすと、通常、特定のジョブのスループットが向上し、使用される処理時間が短縮されます。ただし、次のセクションで説明するように例外があります。

I/O を最適化する手法

I/O を最適化する手法の概要

I/O は、パフォーマンスを最適化するための最も重要な要素の 1 つです。ほとんどの SAS ジョブは、特定のデータセットを読み取り、さまざまなデータ分析およびデータ操作タスクを実行する繰り返しサイクルで構成されます。SAS ジョブのパフォーマンスを向上させるには、SAS がディスクまたはテープデバイスにアクセスする回数を減らす必要があります。

これを行うには、次の方法で必要な変数とオブザベーションのみを処理するように SAS プログラムを変更します。

- WHERE 処理の使用

- DROP および KEEP ステートメントの使用
- LENGTH ステートメントの使用
- OBS=および FIRSTOBS=データセットオプションの使用

次の方法でプログラムを変更して、内部でデータを処理する回数を減らすこともできます。

- SAS データセットの作成
- インデックスの使用
- SAS ビューを介したデータへのアクセス
- 効率的なエンジン使用
- 変数属性を変更する場合の PROC DATASETS の使用
- 数値を文字として保存
- メモリ使用量を最適化する手法の使用

次の方法でデバイスがアクセスされるたびにより多くのデータを処理することで、データアクセスの数を減らすことができます。

- ALIGNSASIOFILES、BUFNO=、BUFSIZE=、CATCACHE=、COMPRESS=、DATAPAGESIZE=、STRIPESIZE=、UBUFNO=、UBUFSIZE=システムオプションを設定する
- SASFILE グローバルステートメントを使用して SAS データセットを開き、データセット全体をメモリ内に保持するのに十分なバッファーを割り当てる

SAS DATA ステップビューを使用する場合、次の方法でパフォーマンスを向上させることができます。

- VBUFSIZE=システムオプションの指定
- OBSBUF=データセットオプションの指定

注: 場合によっては、複数の方法を使用して、SAS ジョブをさらに効率化できる場合があります。

WHERE 処理の使用

プロシジャ内で WHERE ステートメントを使用すると、サブセット化 IF ステートメントを使用して DATA ステップと同じタスクを実行できる場合があります。WHERE ステートメントを使用すると、不必要なオブザベーションが処理されないため、特定の分析を実行するときに余分な DATA ステップの処理を排除できます。

たとえば、次の DATA ステップではデータセット Seatbelt を作成します。このデータセットには、Seatbelt の値が **YES** である Auto.Survey データセットのオブザベーションのみが含まれています。その後、新しいデータセットが出力されます。

```
libname auto 'SAS-library';
data seatbelt;
  set auto.survey;
```

```
  if seatbelt='yes';
run;

proc print data=seatbelt;
run;
```

ただし、次の例のように、PRINT プロシジャで WHERE ステートメントを使用すると、データセットを作成せずに PROC PRINT ステップから同じ出力を取得できます。

```
proc print data=auto.survey;
  where seatbelt='yes';
run;
```

WHERE ステートメントを使用すると、データを処理する回数が減るため、リソースを節約できます。この例では、DATA ステップを削除することで、使用する時間とメモリを削減できる可能性があります。また、中間データセットがないため、使用する I/O も少なくなります。生データを読み取る DATA ステップでは WHERE ステートメントを使用できないことに注意してください。

達成できる節約の程度は、データセットのサイズなどの多くの要因によって異なります。プログラムをテストして、最も効率的な解決策を決定することをお勧めします。

詳細については、“WHERE 式” (465 ページ)を参照してください。

DROP および KEEP ステートメントの使用

効率を向上させるもう 1 つの方法は、DROP ステートメントと KEEP ステートメントを使用してオブザベーションのサイズを減らすことです。一時データセットを作成し、必要な変数のみを含めると、データの処理に必要な I/O 操作の回数を減らすことができます。詳細については、“DROP ステートメント” (*SAS DATA ステップステートメント: リファレンス*)および “KEEP ステートメント” (*SAS DATA ステップステートメント: リファレンス*)を参照してください。

LENGTH ステートメントの使用

LENGTH ステートメントを使用して、オブザベーションのサイズを減らすこともできます。各変数に必要な記憶領域のみを含めると、データの処理に必要な I/O 操作の回数を減らすことができます。ただし、数値変数の長さを変更する前に、“LENGTH ステートメント” (*SAS DATA ステップステートメント: リファレンス*)を参照してください。詳細については、“LENGTH ステートメント” (*SAS DATA ステップステートメント: リファレンス*)を参照してください。

OBS=および FIRSTOBS=データセットオプションの使用

OBS=および FIRSTOBS=データセットオプションを使用して、処理されるオブザベーションの数を減らすこともできます。一時データセットを作成し、必要なオブザベーションのみを含めると、データの処理に必要な I/O 操作の回数を減らすことができます。詳細については、*"FIRSTOBS= データセットオプション" (SAS データセットオプション: リファレンス)*および*"OBS= データセットオプション" (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。

SAS データセットの作成

同じ生データを繰り返し処理する場合は、通常、SAS データセットを作成する方が効率的です。SAS は、生データファイルを処理するよりも効率的に SAS データセットを処理できます。

詳細については、"生データの読み取り" (346 ページ)を参照してください。

インデックスの使用

インデックスは、特定のオブザベーションへの直接アクセスを提供するために SAS データファイルに対して作成できるオプションのファイルです。インデックスは、特定の変数の値を昇順で保存し、データファイル内のオブザベーション内のそれらの値の位置に関する情報を含みます。つまり、インデックスを使用すると、インデックス付き変数の値によってオブザベーションを見つけることができます。

インデックスがない場合、SAS はデータファイルに保存されている順序でオブザベーションにシーケンシャルアクセスします。インデックスを使用すると、SAS はオブザベーションに直接アクセスします。したがって、インデックスを作成して使用すると、オブザベーションに迅速にアクセスできるようになります。

一般に、SAS は次の状況でインデックスを使用してパフォーマンスを向上させることができます。

- WHERE 処理の場合、インデックスを使用すると、データのサブセットへのより高速かつ効率的なアクセスが可能になります。
- BY 処理の場合、インデックスは、SORT プロシジャを使用せずに、インデックス順(値の昇順)でオブザベーションを返します。
- SET および MODIFY ステートメントの場合、KEY=オプションを使用すると、DATA ステップでインデックスを指定して、データファイル内の特定のオブザベーションを取得できます。

注: パフォーマンスを向上させるためにインデックスが存在します。ただし、インデックスは他のリソースを犠牲にして一部のリソースを節約します。したがって、インデックスの作成、使用、保守に関連するコストを考慮する必要があります。インデックスの詳細と、インデックスを作成するかどうかの決定については、22 章, “インデックスの使用”を参照してください。

SAS ビューを介したデータへのアクセス

SQL プロシジャまたは DATA ステップを使用して、データの SAS ビューを作成できます。SAS ビューは、より少ないステートメントでデータをサブセット化する、保存された命令セットです。また、SAS ビューを使用すると、新しいデータセットを作成せずに複数のデータセットからデータをグループ化できるため、処理時間とディスク領域の両方を節約できます。詳細については、16 章, “SAS ビュー”および *Base SAS プロシジャガイド*を参照してください。

SAS ビューを使用したシステムパフォーマンスの最適化については、“SAS DATA ステップビューの VBUFSIZE=および OBSBUF=の設定” (678 ページ)を参照してください。

効率的なエンジン使用

LIBNAME ステートメントでエンジンを指定しない場合、SAS はどのエンジンを SAS ライブラリに関連付けるかを決定するために追加の処理ステップを実行する必要があります。SAS は、使用するエンジンを決定するのに十分な情報が得られるまで、ディレクトリ内のすべてのファイルを調べる必要があります。たとえば、次のステートメントは、ライブラリ参照名 Fruits に特定のエンジンを使用するように SAS に明示的に指示しているため、効率的です。

```
/* Engine specified. */

libname fruits v9 'SAS-library';
```

次のステートメントでは、エンジンを明示的に指定していません。出力では、生成されるエンジンタイプの混合に関する注記に注目してください。

```
/* Engine not specified. */

libname fruits 'SAS-library';
```

例のコード 25.3　LIBNAME ステートメントからの SAS ログ出力

```
NOTE: Directory for library FRUITS contains files of mixed engine types.
NOTE: Libref FRUITS was successfully assigned as follows:
      Engine:      V9
      Physical Name: SAS-library
```

SAS エンジンの詳細については、12 章, “SAS エンジン”を参照してください。

I/O パフォーマンスを向上させるためのシステムオプションの設定

次の SAS システムオプションは、SAS ファイルに必要なディスクアクセスの回数を減らすのに役立ちますが、メモリ使用量と SAS データセットサイズが増加する可能性があります。

ALIGNSASIOFILES
: SAS データセットは、ヘッダーとそれに続く 1 ページ以上のデータで構成されます。通常、ヘッダーは Windows では 1K、UNIX では 8K です。
ALIGNSASIOFILES システムオプションは、データページがより効率的な I/O を可能にする境界に合わせて配置されるように、ヘッダーをデータページと同じサイズにするように強制します。ページサイズは BUFSIZE=オプションを使用して設定します。

  詳細については、*"ALIGNSASIOFILES システムオプション" (SAS システムオプション: リファレンス)*および動作環境の SAS ドキュメントを参照してください。

BUFNO=
: SAS は、BUFNO=オプションを使用して、SAS データセットを処理するときに開いているページバッファーの数を調整します。このオプションの値を増やすと、SAS がより少ないパスでより多くのデータを読み取れるようになり、アプリケーションのパフォーマンスが向上します。ただし、メモリ使用量は増加します。このオプションのさまざまな値を試して、ニーズに最適な値を決定してください。

  注: CBUFNO=システムオプションを使用して、開いている各 SAS カタログに割り当てる追加のページバッファーの数を制御することもできます。

  詳細については、*"BUFNO= システムオプション" (SAS システムオプション: リファレンス)*および動作環境の SAS ドキュメントを参照してください。

BUFSIZE=
: BASE エンジンはデータセットを作成するときに、BUFSIZE=オプションを使用してデータセットの永久ページサイズを設定します。ページサイズは、1 回の I/O 操作で 1 つのバッファーに転送できるデータの量です。BUFSIZE=のデフォルト値は、動作環境によって決まります。デフォルトではシーケンシャルアクセス方式を最適化するように設定されていることに注意してください。直接(ランダム)アクセスのパフォーマンスを向上させるには、BUFSIZE=の値を変更する必要があります。

  動作環境のデフォルト値を使用する場合でも、値を指定する場合でも、エンジンは、ページがどれだけ埋まっているか空であるかに関係なく、常に完全なページを書き込みます。

  データの合計量が少なくなることがわかっている場合は、BUFSIZE=オプションを使用して小さなページサイズを設定すると、データセットの合計サイズが小さく保たれ、ページ上の無駄なスペースの量が最小限に抑えられます。対照的に、データセット内に多くのオブザベーションが含まれることがわかっている場合は、必要なオーバーヘッドができる限り少なくなるように BUFSIZE=を最適化す

る必要があります。各ページには追加のオーバーヘッドが必要になることに注意してください。

シーケンシャルアクセスされる大規模なデータセットには、データセットの読み取りに必要なシステム呼び出しの回数が減るため、ページサイズを大きくするとメリットが得られます。オブザベーションはページにまたがることができないため、通常、ページには未使用のスペースが存在することに注意してください。

“データセットサイズの計算” (683 ページ)では、データセットのサイズを見積もる方法について説明します。

詳細については、“BUFSIZE= システムオプション” (*SAS システムオプション: リファレンス*)および動作環境の SAS ドキュメントを参照してください。

CATCACHE=
SAS はこのオプションを使用して、一度に開いておく SAS カタログの数を決定します。この値を増やすと、より多くのメモリが使用される可能性がありますが、他のアプリケーションが比較的すぐに必要とするカタログをアプリケーションが使用する場合には、これが保証される可能性があります。(最初のアプリケーションによって閉じられたカタログはキャッシュされ、後続のアプリケーションからより効率的にアクセスできるようになります。)

詳細については、“CATCACHE= システムオプション” (*SAS システムオプション: リファレンス*)および動作環境の SAS ドキュメントを参照してください。

COMPRESS=
I/O 処理を削減できるもう 1 つの手法は、COMPRESS=データセットオプションを使用してデータを圧縮データセットとして保存することです。ただし、この方法でデータを保存すると、SAS がオブザベーションを利用できるようになるため、オブザベーションを解凍するためにより多くの CPU 時間が必要になります。ただし、CPU 使用率ではなく I/O が問題である場合は、データを圧縮するとアプリケーションの I/O パフォーマンスが向上する可能性があります。

詳細については、“COMPRESS= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

DATAPAGESIZE=
SAS 9.4 以降、I/O パフォーマンスを向上させるために、最適なバッファーページサイズが増加しました。ページサイズの増加により、データセットまたはユーティリティファイルのサイズが 増加する可能性があります。現在の最適化プロセスが SAS セッションにとって理想的ではない場合は、DATAPAGESIZE=COMPAT93 を使用して、SAS 9.4 より前に使用されていた最適化プロセスを使用できます。

詳細については、“DATAPAGESIZE= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

STRIPESIZE=
データが RAID (Redundant Array of Independent Disks)デバイスに保存されている場合、STRIPESIZE=システムオプションを使用して、ディレクトリの I/O バッファーサイズを RAID ストライプのサイズに設定できます。ディレクトリ内に作成される SAS データセットまたはユーティリティファイルのページサイズは、RAID ストライプサイズと一致します。このオプションを使用すると、個々のディスクのパフォーマンスが向上します。

詳細については、“STRIPESIZE= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

UBUFNO=
UBUFNO=システムオプションは、SAS がデータセットを処理するために使用するユーティリティバッファーの数を設定します。

詳細については、“UBUFNO= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

UBUFSIZE=
UBUFSIZE=オプションは、SAS がデータセットの処理に使用するユーティリティファイルのページサイズを設定します。UBUFSIZE=オプションと BUFSIZE=オプションの値が同じ場合、ディスクアクセス回数を改善できます。

詳細については、“UBUFSIZE= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

VBUFSIZE=
VBUFSIZE=オプションは、ビューバッファーのサイズを設定します。ビューのパフォーマンスは、より多くの生成されたオブザベーションを保持できる十分な大きさのビューバッファーのサイズを設定することで向上させることができます。詳細については、“VBUFSIZE= システムオプション” (*SAS システムオプション: リファレンス*)および“SAS DATA ステップビューの VBUFSIZE=および OBSBUF=の設定” (678 ページ)を参照してください。

SAS DATA ステップビューの VBUFSIZE=および OBSBUF=の設定

SAS DATA ステップビューを使用する場合、OBSBUF=データセットオプションまたは VBUSIZE=システムオプションを指定すると、タスクの切り替えが減り、処理効率が向上します。VBUFSIZE=システムオプションを使用すると、バイト数に基づいてビューバッファーのサイズを指定できます。デフォルトのバッファーサイズは 65536 です。OBSBUF=データセットオプションは、指定されたオブザベーション数に基づいてビューバッファーサイズを設定します。どちらの場合も、より多くの生成されたオブザベーションを保持できるようにビューバッファーを設定すると、タスクの切り替えが減り、実行時間が短縮されます。

VBUFSIZE=システムオプションの詳細については、“VBUFSIZE= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。OBSBUF=データセットオプションの詳細については、“OBSBUF= データセットオプション” (*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。

SASFILE ステートメントの使用

SASFILE グローバルステートメントは、SAS データセットを開き、データセット全体をメモリ内に保持するのに十分なバッファーを割り当てます。読み取られたデータはメモリ内に保持され、2 番目の SASFILE ステートメントがファイルを閉じてバッファーを解放するか、プログラムが終了して自動的にファイルが閉じてバッファーが解放されるまで、後続の DATA および PROC ステップで使用できます。

SASFILE ステートメントを使用すると、次のようにパフォーマンスが向上します。

- SAS データセットを処理するための複数の開く操作と閉じる操作(バッファーへのメモリの割り当てと解放を含む)を 1 回の開く操作と閉じる操作に削減する
- データをメモリ内に保持することで I/O 処理を削減する

SAS プログラムが SAS データセットを複数回読み取るステップで構成されており、ファイル全体を実メモリに保持できる十分な量のメモリがある場合、プログラムは SASFILE ステートメントを使用するとメリットが得られます。また、SASFILE は、SAS/SHARE サーバーなどの SAS サーバーを起動するプログラムの一部として特に便利です。

SASFILE グローバルステートメントの詳細については、*SAS DATA ステップステートメント: リファレンス*を参照してください。

DATASETS プロシジャを使用した属性の変更

PROC DATASETS が実行する必要があるタスクがこのタスクだけである限り、DATASETS プロシジャを使用して変数属性を変更する方が、DATA ステップを使用するより効率的です。DATASETS プロシジャは、データセットのデータディスクリプター情報のみを処理します。DATA ステップはデータセット全体を処理します。詳細については、"DATASETS プロシジャ" (*Base SAS プロシジャガイド*)を参照してください。

変数を文字として保存する

SAS は、DATA ステップで処理される数値ごとに 8 バイトのストレージを使用し、文字ごとに 1 バイトを使用します。数値を含む変数に対して計算を実行しない場合は、変数を文字変数として定義することでストレージを節約できます。各変数に必要なストレージの量を減らすと、I/O 操作の回数が減ります。

メモリ使用量を最適化する手法

システムオプション

メモリが重要なリソースである場合は、いくつかの手法を使用してメモリの増加への依存を減らすことができます。ただし、それらのほとんどは I/O 処理または CPU 使用率も増加させます。

MEMSIZE=システムオプションを使用すると、SAS で使用できるメモリの量を増やし、処理時間を短縮できます。メモリを増やすと、ページング、つまりメモリへのデータのページの読み取りに費やされる時間が減少するため、処理時間が短縮されます。

SORTSIZE=および SUMSIZE=システムオプションを使用すると、並べ替えと集計のプロシジャに使用できるメモリの量を制限できます。

“プログラムコンパイルの最適化変更による CPU 時間の短縮” (682 ページ)で説明したように、メモリと他のリソースの間でトレードオフを行うこともできます。I/O サブシステムを最も効果的に使用するには、より多くのより大きなバッファーを使用する必要があります。ただし、これらのバッファーは、SAS セッションの他のメモリ要求とスペースを共有します。

PROC MEANS での BY ステートメントの使用

PROC MEANS で CLASS ステートメントと分類変数を使用する場合、メモリ要件が大量になる可能性があります。SAS は、各分類変数の一意の値のコピーをメモリに保持します。PROC MEANS ですべての分類変数を集計するにはメモリが不足している場合は、CLASS ステートメントと BY ステートメントを一緒に使用して分類ごとにデータを分析することでメモリを節約できます。

詳細については、“BY ステートメントと CLASS ステートメントの比較” (*Base SAS プロシジャガイド*)を参照してください。

関連項目

MEMSIZE システムオプション

CPU パフォーマンスを最適化する手法

より多くのメモリを使用するか I/O を減らすことで CPU 時間を短縮する

単一のコードストリームを実行すると、そのコードが実行されるたびにほぼ同じ量の CPU 時間がかかります。このようなインスタンスでの CPU パフォーマンスの最適化は、通常、トレードオフになります。たとえば、より多くのメモリを使用することで CPU 時間を短縮できます。これにより、1 回の操作でより多くの情報を読み取って保存できるようになります。ただし、他のプロセスが使用できるメモリは少なくなります。

また、CPU は I/O 操作の実行に必要なすべての処理を実行するため、I/O 操作の回数を減らすオプションや手法も CPU 使用率にプラスの影響を与える可能性があります。

計算集約型の DATA ステップ用にコンパイルされたプログラムを保存する

CPU パフォーマンスを向上させるもう 1 つの手法は、SAS ステートメントではなくコンパイルされたプログラムとして繰り返し実行される DATA ステップを保存することです。これは、I/O 集約型ではない大規模な DATA ステップジョブに特に当てはまります。詳細については、*“Stored, Compiled DATA Step Programs” (SAS V9 LIBNAME Engine: Reference)*を参照してください。

SAS 実行可能ファイルの検索時間の短縮

PATH=システムオプションは、SAS 実行可能ファイルを含むディレクトリ(一部の動作環境ではライブラリ)のリストを指定します。デフォルトの構成ファイルは、これらのディレクトリの特定の順序を指定します。PATH=オプションのディレクトリ指定を再配置して、最も頻繁にアクセスされるディレクトリが最初にリストされるようにすることができます。アクセス頻度が最も低いディレクトリを最後に配置します。

変数長の指定

SAS がプログラムデータベクトルを処理するときは、通常、個々の変数ごとではなく、1 つの大きな操作でデータを移動します。データが適切に配置されている場合(8 バイト境界内)、データの移動はわずか 2 クロックサイクル(1 回のロードとそれに続く 1 回の保存)で実行できます。SAS は、より複雑な手段で、最悪の場合でも一度に 1 バイトずつ、未配置のデータを移動します。8 バイト変数の場合、これは少なくとも 8 倍遅くなります。

多くの高性能 RISC (縮小命令セットコンピューター)プロセッサは、未配置のデータの移動により、パフォーマンスに非常に大きなペナルティを課します。可能な場合は、数値データを全幅(8 バイト)のままにしてください。SAS は算術演算の場合、短い数値データを拡張する必要があることに注意してください。一方、短い数値データはメモリと I/O の両方を節約できます。動作環境や状況に応じてどの方法が最も有利かを判断する必要があります。

注: 配置は、特定の変数のみを選択してデータセットを処理する場合、または WHERE 処理を使用する場合に特に重要になります。

並列処理の使用

SAS System 9 は、並列処理に関連する新しい SAS 機能をサポートします。並列処理とは、複数の CPU で同時に処理を行うことを意味します。このテクノロジは SMP コンピューターを活用し、スレッド I/O とスレッドアプリケーション処理という 2 種類の SAS プロセスのパフォーマンスを向上させます。

プログラムコンパイルの最適化変更による CPU 時間の短縮

SAS がプログラムをコンパイルするとき、コードは冗長な命令、欠損値のチェック、配列添字の反復計算を削除するように最適化されます。このコードは命令のパターンを検出し、より効率的なシーケンスに置き換えます。また、SAS レジスタに関連する最適化も実行します。ほとんどの場合、コード生成の最適化を実行することをお勧めします。大規模な DATA ステッププログラムがある場合、コード生成の最適化を実行すると、コンパイル時間と全体の実行時間が大幅に増加する可能性があります。

CGOPTIMIZE=システムオプションを使用すると、コード生成の最適化を軽減またはオフにすることができます。次の CGOPTIMIZE=システムオプション値を使用して、SAS に実行させるコード生成の最適化を設定します。

0 コードのコンパイル中に最適化は実行されません。

1 ステージ 1 の最適化を実行するように指定します。ステージ 1 の最適化では、冗長な命令、欠損値のチェック、配列添字の反復計算が削除されます。命令のパターンを検出し、より効率的なシーケンスに置き換えます。

2 ステージ 2 の最適化を実行するように指定します。ステージ 2 では、SAS レジスタに関連する最適化を実行します。大規模な DATA ステッププログラムでステージ 2 の最適化を実行すると、コンパイル時間が大幅に増加する可能性があります。

3 ステージ 1 と 2 を組み合わせた完全な最適化を実行するように指定します。これがデフォルト値です。

詳細については、“CGOPTIMIZE= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

データセットサイズの計算

最適化手法をすでに適用しているにもかかわらず、依然として処理時間が長かったり、過剰なメモリ使用量が発生したりする場合は、データセットのサイズが非常に大きい可能性があります。その場合、これ以上の改善は不可能となる可能性があります。

データセットと同じ変数を含むダミーデータセットを作成することで、データセットのサイズを推定できます。CONTENTS プロシジャを実行すると、各オブザベーションのサイズが表示されます。このサイズにデータセット内のオブザベーションの数を乗じて、処理する必要がある合計バイト数を求めます。処理統計量を小さなデータセットと比較して、大きなデータセットのパフォーマンスがそのサイズに比例しているかどうかを判断できます。そうでない場合、さらに最適化できる可能性があります。

注: この手法を使用してデータセットのサイズを計算すると、推定値のみが得られます。変数名の保存などの内部要件により、実際のデータセットのサイズが若干異なる場合があります。

26

並列処理の使用

概要

SAS では、SAS®9 から*スレッド*テクノロジを導入し、部分的に複数の*スレッド*で実行できるように強化されたいくつかの Base SAS プロシジャを導入しました。SAS は、独自の内部サブシステムによって提供されるスレッド処理機能を活用する製品とコンポーネントの開発と強化を継続してきました。

スレッド処理は、複数の CPU コアを備えたローカルデスクトップコンピューター上でも、ハイパフォーマンスプラットフォームサーバー上でも、さまざまなプラットフォームで利用できます。

ハイパフォーマンスサーバーには、大規模な対称型マルチプロセッサ(SMP)と超並列プロセッサ(MPP)が含まれており、通常は分散クラスターとして構成されます。これらのプラットフォーム上で実行される多くの SAS コンポーネントは、スレッドテクノロジを利用しています。

SAS 9.4M5 以降、SAS Viya プラットフォームのライセンスを取得すると、マルチスレッドのインメモリ処理をサポートする分散サーバー環境である SAS Cloud Analytic Services (CAS)にアクセスできるようになります。詳細については、*SAS Cloud Analytic Services: ユーザーガイド*を参照してください。

Base SAS 9.4 の以前のリリースは、SAS DS2 プログラミング言語または SAS Federated SQL 言語で記述されたプログラムをサポートしています。これらの言語でもスレッド処理を利用します。

他の多くの SAS 製品もスレッドテクノロジを使用しています。たとえば、SAS High-Performance Analytics プロシジャ、SAS Stored Process、および SAS Embedded Process は、ハイパフォーマンス分散コンピューティング環境で実行されるコードを実行または生成します。

SAS のスレッドテクノロジとは

スレッドテクノロジは、動作環境内で複数の実行パスを提供します。各実行パスはスレッドと呼ばれ、各スレッドはプログラムタスクまたはデータ転送を処理できます。その結果、複数のプログラムタスクとデータ I/O 操作が同時に並行して実行されます。スレッドにはコンテキスト(レジスタセットやプログラムカウンターなど)、実行するコードのセグメント、およびプロセスで使用するある程度のメモリが必要です。スレッド動作環境には複数の CPU が搭載されている場合がありますが、CPU ごとに*コア*は 1 つだけです。他のよりハイパフォーマンスな構成には、CPU ごとに複数のコア、さらにはコアごとに複数のスレッドを備えた複数の CPU が含まれる場合があります。各 CPU が一度に 1 つのスレッドのみを実行する状況では、スレッド間をすばやく切り替える CPU の機能により、ほぼ同時の実行が実現します。

SAS ソフトウェアでのスレッド実行には、これら 2 つの一般的な手法の一方または両方が含まれます。

- *スレッド I/O* とは、データ(多くの場合非常に大量)がスレッドでアプリケーションに配信されるため、アプリケーションはデータを待機するのではなく、継続的に処理することになります。Base SAS および SAS/STAT では、いくつかのプロシジャがスレッド読み取りを利用します。Base SAS には、アプリケーションへのスレッド入力を最適化するためにパーティション分割されたデータセットから読み取る SPD Engine も含まれています。SAS High-Performance Analytics プロシジャでは、非常に迅速なデータ配信が必要です。大量のデータをアプリケーションに配信し(これもクラスター上で処理される)、そのデータをデータストレージアプライアンスに並行して書き込むには、コンピューティングクラスター全体に分散されたデータからのスレッド読み取りが必要です。
- *スレッドアプリケーション処理*とは、アプリケーション自体が複数の CPU マシン上で特定のタスクを並列実行するように構造化されていることを意味します。スレッドアプリケーション処理では、複数のスレッドがデータのより小さなセグメントで実行されるため、アプリケーションは大量のデータをより迅速に処理できます。アプリケーションは、ローカルの 4 ウェイデスクトップであっても、サーバークラスのマシンであっても 、複数の CPU を搭載したマシンを活用するように設計できます。SAS High-Performance Analytics Server は、データとアプリケーションのコピーの両方をアプライアンスノード全体に分散するアプライ

アンス上で実行され、データがアプリケーション処理と同じ場所に配置されます。

SAS 9.4 および SAS Analytics 12.1 を使用すると、顧客は、スレッド処理を使用して、増え続けるデータ量や計算集約型のアルゴリズムやモデルをサポートするさまざまな製品やコンポーネントにアクセスできます。Base SAS および Foundation SAS スレッドテクノロジは、これらすべてをサポートします。

SAS でのスレッド処理の制御方法

多くの SAS コンポーネントは、スレッドテクノロジを自動的に利用します。SAS 内のメカニズムは特定の環境変数を検出し、アプリケーションの予想されるパフォーマンスに応じてスレッド処理を使用したり使用しなかったりすることができます。一部の環境変数とシステムオプションは、管理者が構成できます。データセットオプション(使用可能な場合)を指定して、スレッドでの I/O またはアプリケーション処理に影響を与えることもできます。多くのコンポーネントがスレッドを自動的に使用している可能性があるため、スレッド処理を制御する特定のオプションがオフになっている場合でも、スレッドの使用が継続される可能性があります。

システムオプション THREADS、NOTHREADS、および CPUCOUNT は、スレッド処理が自動ではない SAS 全体のスレッド処理に影響します。

一部の製品には、SAS/ACCESS の DBSLICE=データセットオプション"DBSLICE= Data Set Option" (*SAS/ACCESS for Relational Databases: Reference*)など、スレッド処理を制御するための追加オプションがあります。システムオプション THREADS はすべての製品のデフォルトであるため、スレッド処理の使用が可能な場合はどこでもスレッド処理を実行でき、パフォーマンスが向上します。NOTHREADS は、Base SAS または SAS クライアント、および対称型マルチプロセッサ環境で実行される製品のスレッド処理を無効にします。プロシジャステートメントのオプションは、必要に応じてシステムオプションをオーバーライドするために提供されています。

SAS/STAT、SAS/OR、SAS/ETS、SAS Enterprise Miner、SAS High-Performance Analytics Server プロシジャなどの SAS 製品の特定のプロシジャは、SAS クライアントの SMP モード、または*分散コンピューティング*環境での超並列処理モードのいずれかで実行できます。SMP モードでは、NOTHREADS が受け入れられます。

THREADS が設定されている場合、CPUCOUNT はデフォルトでクライアントでの処理に使用できるスレッドの数になります。この機能は調整可能です。MPP モードでは、スレッドは常に想定されます。NOTHREADS は無視され、スレッド処理は常に有効になります(クライアント SAS セッションまたは SAS Enterprise Miner から実行しない限り)。ただし、MPP モードでは、NOTHREADS は効果がありません。これらの製品のほとんどでは、スレッド制御と実行モードは PERFORMANCE ステートメントで指定されます。その製品またはコンポーネントで使用されているスレッドテクノロジについては、特定の SAS 製品ドキュメントとともに *SAS システムオプション: リファレンス*を参照してください。

Base SAS のスレッド処理

Base SAS では一部のスレッド処理が自動的に行われます。さらに、THREADS システムオプションは、スレッド読み取りまたはスレッドアプリケーション処理をサポートするすべての Base SAS コンポーネントのデフォルトです。

基本言語
: SAS は、スレッドテクノロジを使用して SAS データファイルにインデックスを構築します。インデックスを使用すると、SAS 言語の WHERE 処理、BY グループ処理、SET および MODIFY ステートメント、ならびに DO ループでの ARRAY 処理のパフォーマンスを高速化できます。Base SAS の並べ替えアルゴリズムはインデックスの構築に使用され、デフォルトでスレッド対応になっています。この機能は、NOTHREADS システムオプションを指定することで無効にできます。

スレッド対応の Base SAS プロシジャ
: 特定の Base SAS プロシジャには、スレッド処理を利用できるアルゴリズムがあります。これらのプロシジャはスレッド対応になっており、プロシジャアルゴリズムの一部を分割して、アルゴリズムの一部をスレッドで実行できるようにします。たとえば、SORT プロシジャはスレッド対応になっているため、使用可能なスレッドで並べ替えが行われ、各スレッドがデータの一部を並べ替えます。次に、このプロシジャは、複数のスレッドから並べ替えられた順序でデータセットを迅速に生成します。これらのプロシジャは、スレッド内のデータを読み取ることもできます。

  プロシジャで使用できるスレッドと CPU の数は、システムまたはプロシジャのオプション CPUCOUNT および THREADS|NOTHREADS によって指定されます。NOTHREADS は、スレッド処理をサポートする SAS アプリケーションの実行にスレッド処理を使用しないことを指定します。デフォルトは THREADS です。NOTHREADS が有効な場合、CPUCOUNT は無視されます。Base SAS スレッド対応プロシジャは次のとおりです。

  - MEANS *SAS Output Delivery System: ユーザーガイド*
  - REPORT
  - SORT
  - SUMMARY
  - TABULATE
  - SQL

  詳細については、“Base SAS プロシジャのスレッド処理” (*Base SAS プロシジャガイド*)を参照してください。スレッド対応 SQL プロシジャの詳細については、*SAS SQL プロシジャユーザーガイド*を参照してください。

  SAS システムオプションの詳細については、*SAS システムオプション: リファレンス*を参照してください。

  SAS/STAT ソフトウェアの一部のプロシジャもスレッド対応であり、そのほとんどは SMP または MPP モードで実行できます。SMP モードでは、NOTHREADS および CPUCOUNT が受け入れられます。MPP モードでは、PERFORMANCE ス

テートメントはスレッド処理を制御するオプションを提供します。次は、スレッド対応の SAS/STAT プロシジャです。

- ADAPTIVEREG
- FMM
- GLM
- GLMSELECT
- LOESS
- MIXED
- QUANTLIFE
- QUANTREG
- QUANTSELECT
- ROBUSTREG

各プロシジャの詳細については、*SAS/STAT プロシジャガイド*を参照してください。

SAS Scalable Performance Data Engine
: SAS Scalable Performance Data Engine は Base SAS に含まれており、SMP ハードウェア機能を活用するように設計されています。SAS Scalable Performance Data Engine は、スレッドでのデータの読み取り用に最適化されたパーティション化されたデータセットを使用します。パーティションサイズは、SAS Scalable Performance Data Engine の PARTSIZE オプションで構成できます。THREADNUM および SPDEMAXTHREADS は、最適なスレッド読み取りのためにスレッド処理を制御します。Base SAS NOTHREADS および CPUCOUNT システムオプションは、SPD Engine のスレッド読み取りには影響しません。これらは、SPD Engine データセット上で実行される SAS スレッド対応プロシジャに対して有効なままです。SPD Engine インデックスも、NOTHREADS が設定されている場合は、NOTHREADS に関係なく、スレッド内で並列に自動的に作成されます。SPDEINDEXSORTSIZE=を使用すると、スレッドインデックスの作成を最適化できます。SPD Engine については、*SAS Scalable Performance Data Engine: リファレンス*で説明します。

SAS FedSQL 言語
: SAS FedSQL は、ANSI SQL:1999 規格に基づいた SAS 独自の SQL 実装です。ANSI SQL データ型およびその他の ANSI 準拠機能のサポートを提供します。SAS FedSQL の中核的な強みは、異種データベース環境全体で *フェデレーションクエリ*を実行し、単一の結果セットを返す機能です。FedSQL クエリは、大規模な操作を解決するために、マルチスレッドアルゴリズムを使用して自動的に最適化されます。さらに、FedSQL は SAS セッションの外部で実行できます。たとえば、SAS Federation Server および SAS Scalable Performance Data Server 環境で実行できます。NOTHREADS および CPUCOUNT オプションは、FedSQL の処理には影響しません。

  FedSQL プログラムを実行のためにサブミットする FedSQL プロシジャが含まれています。完全な情報については、*SAS FedSQL 言語リファレンス*を参照してください。

SAS DS2 プログラミング言語
: DS2 は、高度なデータ操作およびデータモデリングアプリケーションに適した SAS 独自のプログラミング言語です。DS2 は Base SAS に含まれており、SAS

DATA ステップと交差しますが、追加のデータ型、ANSI SQL 型、プログラミング構造要素、ユーザー定義のメソッド、パッケージもサポートします。DS2 SET ステートメントは埋め込み FedSQL 構文を受け入れ、ランタイム生成クエリは DS2 とサポートされているデータベース間で対話的にデータを交換できます。これにより、2 つの言語の力を効果的に組み合わせて、入力テーブルの SQL 前処理が可能になります。

DS2 プログラムは、並列実行用にコーディングされたプログラムで THREAD ステートメントを使用することでスレッド対応になります。NOTHREADS および CPUCOUNT オプションは効果がありません。DATA ステッププログラムが DS2 に変換することでメリットが得られるかどうかの詳細については、SAS 9.4 DS2 言語リファレンスを参照してください。

スレッド対応の DS2 プログラムを SAS Embedded Process にサブミットして実行する DS2 プロシジャも含まれています。DS2 プロシジャのハイパフォーマンスバージョンである PROC HPDS2 は、DS2 言語ステートメントを、別途ライセンスされた High-Performance Analytics Server にサブミットして処理します。特定の SAS プロシジャのこのバージョンおよびその他のハイパフォーマンスバージョンに関するドキュメントについては、*SAS High-Performance Analytics Server 使用ガイド*を参照してください。

DS2 は、SAS セッションの外部で実行できます。次に例を示します。

- SAS Federation Server
- SAS Scalable Performance Data Server
- SAS In-Database Scoring Accelerator、SAS Embedded Process 環境、SAS High-Performance Analytics Server 分散環境などの MPP コンピューティング環境

SAS ログ
: SAS ログ機能は、NOTHREADS および CPUCOUNT オプションを無視します。すべての受信ログイベントをスレッドで処理します。現在のスレッドまたはタスクに関連付けられているクライアント ID がログにレポートされます。ログ機能は多くの SAS 製品およびコンポーネントをサポートしていますが、Base SAS に含まれています。*SAS プログラムおよびジョブの SAS Viya ログ機能*を参照してください。

SAS Code Analyzer
: SAS Code Analyzer (SCAPROC プロシジャ)は、記録されたコメントである SAS ジョブに関するメタデータを生成するときに、既存の SAS プログラムを実行します(通常どおりプログラムを実行します)。PROC SCAPROC は、ジョブステップに関する情報、ファイルの依存関係などの I/O 情報、および実行中の SAS ジョブからのマクロシンボルの使用情報を取得します。出力は、コメントとそのコメントに記述されている依存関係を含む SAS プログラムです。アプリケーションはこのテキストを読み取り、SAS メタデータを作成したり、これらの依存関係に基づいてプロセスフローを決定したりできます。たとえば、SAS Data Integration Studio の開発者は、SAS Code Analyzer によって出力された情報を使用して、従来の SAS ジョブをリバースエンジニアリングできます。SAS Grid Manager と併用することもできます。保存されたジョブが*グリッド*上で実行されると、SAS Grid Manager は識別されたサブタスクをグリッドノードに自動的に割り当てます。詳細については、*Base SAS プロシジャガイド*の SCAPROC プロシジャのドキュメントを参照してください。

SAS/ACCESS エンジン

SAS/ACCESS エンジンは、60 を超えるリレーショナルおよび非リレーショナルデータベース、PC ファイル、データウェアハウスアプライアンス、分散ファイルシステムへの読み取り、書き込み、および更新アクセスを提供する *LIBNAME エンジン*です。これらのエンジンは Base SAS の一部ではありませんが、Base SAS に依存します。これらは個別にライセンスを取得するか、SAS BI Server や SAS Activity-Based Management などの多くの製品バンドルに含まれています。多くのバンドルでは、お客様は使用可能な多数の SAS/ACCESS エンジンから 2 つを選択できます。

SAS/ACCESS エンジンを使用すると、SAS プログラムは、SAS データセットであるかのように DBMS に接続できます。これにより、バルクロードサポート、一時テーブルサポート、明示的パススルーによるネイティブ SQL サポートなど、パフォーマンス関連の DBMS 機能と利点が活用されます。DBMS が並列サーバーの場合、エンジンは複数のスレッドを使用して DBMS サーバーに接続し、DBMS データに並列にアクセスします。SAS プログラムがこれらの SAS/ACCESS エンジンを使用してスレッド対応の SAS プロシジャを実行している場合、パフォーマンスがさらに向上する可能性があります。

SAS/ACCESS では、スレッド読み取りにより、結果セットが複数のスレッドにパーティション分割されます。Base SAS プロシジャのスレッド処理とは異なり、SAS/ACCESS のスレッド読み取りはマシン上のプロセッサの数に依存しません。かわりに、結果セットは SAS と DBMS 間の複数の接続で取得されます。SAS により、DBMS は SQL ステートメントに WHERE 句を追加することによって結果セットをパーティション分割します。これが発生すると、単一の SQL ステートメントが複数の SQL ステートメント(各スレッドに 1 つずつ)になります。DBMS もスレッドごとに 1 つのパーティションを読み取ります。

SAS/ACCESS エンジンによって提供される*スケーラビリティ*の程度は、DBMS 自体に実装された並列化の効率によって決まります。ただし、SAS/ACCESS エンジンには、SAS/ACCESS エンジン内のスレッド実装の調整を可能にする LIBNAME ステートメントで使用できるオプションがあります。SAS/ACCESS でスレッド読み取りを制御するオプションは、DBSLICE、DBSLICEPARM、THREADS|NOTHREADS、および BY、OBS、または KEY オプションが PROC または DATA ステップで使用されるかどうかです。詳細については、リレーショナルデータベース用 SAS/ACCESS のドキュメントを参照してください。

SAS Scalable Performance Data Server

SAS Scalable Performance Data Server は、包括的なセキュリティインフラストラクチャ、バックアップおよび復元ユーティリティ、ならびに管理および調整オプションを備えたマルチユーザー並列処理データサーバーです。SPD Server は、ネイティブ SQL、FedSQL、および DS2 言語をサポートします。SPD Server に固有のオプ

ションは、スレッド処理を制御します。NOTHREADS および CPUCOUNT オプションは効果がありません。詳細については、*SAS Scalable Performance Data Server: 管理者ガイド*および *SAS Scalable Performance Data Server: ユーザーガイド*を参照してください。

SAS Intelligence Platform

SAS Intelligence Platform は、エンタープライズインテリジェンスを作成、管理、配信するためのインフラストラクチャです。このインフラストラクチャは、金融サービス、ライフサイエンス、ヘルスケア、小売、製造などの業界向けの SAS ソリューションをサポートします。SAS Intelligence Platform は Base SAS の一部ではなく、Base SAS と次のコンポーネントを含む SAS Foundation に依存しています。

- メタデータを定義するための SAS 管理コンソール
- SAS Business Intelligence Server および SAS Data Integration Server テクノロジ
- SAS Integration Technologies は、次を提供します。
  - SAS サーバーおよび SAS Stored Process Servers などのサポートサービス
  - アプリケーションメッセージング
  - SAS BI Web Services
  - パブリッシングフレームワーク
  - SAS Foundation Services

*SAS Intelligence Platform の概要*では、個々のコンポーネントについて説明し、関連する幅広い SAS Intelligence Platform ドキュメントを参照します。

Intelligence Platform の SAS サーバーは次のとおりです。

- SAS Workspace Server
- SAS Stored Process Server
- SAS Pooled Workspace Server
- SAS OLAP Server
- SAS Metadata Server

各サーバーは、アクティブなスレッドのプールで開始されます。これらのスレッドはサーバーによって制御され、サーバープロセス(受信要求の処理など)によって使用されます。NOTHREADS および CPUCOUNT オプションが指定されている場合、これらのオプションを受け入れる SAS プロシジャを含むサブミットされたコードの実行中を除いて、それらは無視されます。

SAS Metadata Server の場合、スレッドの使用量は、オブジェクトサーバーパラメーター(THREADSMAX および THREADSMIN)およびメタデータサーバー構成オプション MACACTIVETHREADS のデフォルト設定によって制御されます。管理者はパフォーマンスを微調整するためにこれらの設定をオーバーライドできます。詳細と例については、*SAS Intelligence Platform: システム管理者ガイド*を参照してくださ

い。THREADMAX および THREADMIN オブジェクトサーバーパラメーターは、SAS Metadata Server 以外のサーバーではほとんど使用されません。

インテリジェントプラットフォーム中間層(Web アプリケーションのインフラストラクチャ)では、受信要求はスレッドで処理されます。これらのスレッドはジョブ実行サービスを使用して定義されます。スレッドは、NOTHREADS または CPUCOUNT オプションの制約を受けません。ジョブキュースレッド数とジョブ実行スレッド数の両方を指定できます。*SAS Intelligence Platform: 中間層管理ガイド* を参照してください。

SAS MP CONNECT は、インテリジェンスプラットフォームにバンドルされている SAS/CONNECT ソフトウェアの一部です。複数の SAS セッション間の接続を確立し、各セッションが非同期でタスクを並列実行できるようにすることで、並列処理をサポートします。同じローカルコンピューター上のプロセスへの接続を確立することにより、アプリケーションはネットワークリソースを使用して並列処理し、すべての結果をクライアント SAS セッションに調整できます。多くの SAS プロセスは SMP コンピューター上の複数のプロセッサを使用しますが、ネットワーク上の複数のリモートのシングルまたはマルチプロセッサコンピュータ上で実行することもできます。スレッドは常に使用可能であると想定されます。

一部の SAS High-Performance Analytics 製品は、MPP 環境として構成されている場合、SAS Intelligence Platform 上で実行できます。たとえば、SAS Grid Manager (次のセクションで説明します)は、SMP 構成で実行される SAS アプリケーションのワークロード管理を処理します。また、並列実行用にコーディングされ、SAS High-Performance Analytics Server のノード全体に分散されたアプリケーションを管理することもできます。

SAS Visual Analytics は、MPP 環境として構成されている場合、SAS Intelligence Platform 上で実行されるハイパフォーマンス分析アプリケーションの Web ベースのスイートです。SAS Visual Analytics のこの実行環境については、*SAS Intelligence Platform: 中間層管理ガイド*に記載されています。(SAS Visual Analytics については、次のセクションで詳しく説明します。“SAS High-Performance Analytics 製品ポートフォリオ” (693 ページ)を参照してください。)

SAS High-Performance Analytics 製品ポートフォリオ

SAS High-Performance Analytics 製品は、スレッドを最大限に活用するように設計されています。これらの製品は Base SAS または SAS Foundation の一部ではありません。ただし、Base SAS 機能に依存し、拡張して、大量のデータの迅速な分析をサポートする機能を提供します。

一部の SAS High-Performance Analytics 製品は、他の SAS アプリケーションやソリューションと連携して動作したり、直接統合したりできます。たとえば、SAS Intelligence Platform 上で実行される SAS Enterprise Guide からのワークロードを分散するように SAS Grid Manager を構成できます。SAS Grid Manager を使用すると、EMC Greenplum や Teradata などの DBMS アプライアンス上で実行される SAS High-Performance Analytic Server 上の SAS ジョブのワークロードを管理

できます。また、SAS Visual Analytics を使用すると、SAS Grid 環境で実行される SAS Enterprise Miner によって消費されるデータを探索できます。

SAS High-Performance Analytics テクノロジには次のものが含まれます。

- SAS Grid Manager
- SAS In-Database
- SAS In-Memory Analytics テクノロジには、次のものが含まれます。
  - SAS High-Performance Analytics Server
  - SAS Visual Analytics
  - SAS High-Performance Risk Management
  - その他の SAS ハイパフォーマンス製品およびソリューション

SAS Grid Manager

SAS Grid Manager は、構成されたサーバーのグリッド上のスレッドで実行されるさまざまな SAS プロセスおよびサービス全体にわたって、スケーラブルなワークロード管理と優先順位付け、高可用性処理、および最適化されたパフォーマンスを提供します。個別のサーバーのリソースを一元的に管理される SAS 環境に調整することにより、SAS Grid Manager は SAS ジョブの処理を高速化できます。

SAS プログラムは、プログラムフロー全体の中で複数の独立したステップで実行できるグリッド対応になっており、スレッドで実行されます。スレッド実行は通常、DATA ステップまたはプロシジャの境界で発生します。プログラムはスレッドで実行するように特別に作成されているため、THREADS オプションが設定されており、CPUCOUNT オプションが 1 より大きいことが想定されます。SAS Grid Manager を使用すると、個々の SAS ジョブのサブタスクをグリッドのさまざまな部分で並列実行し、リソースのプールを共有できます。スレッドサブタスクは、コード内から実行されるスレッド対応プロシジャのいずれかによって 実行されるスレッド処理からさらにメリットを得ることができます。SAS Code Analyzer を使用して並列処理用に分析されたプログラムは、SAS Grid Manager を使用してグリッド上で実行できます。これにより、識別されたサブタスクがグリッドノードに自動的に割り当てられます。

SAS Enterprise Guide、SAS Enterprise Miner、SAS Data Integration Studio、SAS Web Report Studio、SAS Marketing Automation、SAS Marketing Optimization などの多くの SAS 製品は、SAS Grid Manager と統合されています。アプリケーションまたは SAS Metadata 内のオプションにより、統合が可能になります。SAS Data Integration Studio および SAS Enterprise Miner には、並列化の機会を認識し、グリッド全体で並列実行するために SAS Grid Manager にサブミットする適切なコードを生成できるコード生成エンジンがあります。

SAS Intelligence Platform などの他の SAS コンポーネントは、SAS Grid Manager を使用して、ネットワーク上の複数のコンピューターにサーバーのワークロードを分散することにより、処理に最適な SAS サーバーを決定します。SAS Grid Manager は、ジョブを複数のサーバー間で並列実行される個別のプロセスに分割します。SAS

Grid Manager については、*SAS のグリッドコンピューティング*に記載されています。

SAS In-Database テクノロジ

SAS In-Database テクノロジは、SAS アプリケーションによってコマンドと関数をデータベース内で実行するように送信することにより、コマンドと関数の実行を高速化します。このプロセスでは、データの移動と変換が回避されますが、ホストデータベースのスレッド機能も利用されます。

SAS In-Database は、Teradata、Greenplum、Aster Data、Netezza、DB2 などのさまざまなスレッド DBMS およびハードウェアまたはソフトウェアアプライアンスアーキテクチャ上で実行されます。スレッド処理はデータベースによって 制御されるため、NOTHREADS および CPUCOUNT オプションは効果がありません。

In-Database プロセスはデータベース内の複数の並列タスクに分割され、それぞれが処理対象のデータ全体の小さなサブセットを処理します。より小さなサブセットの結果が得られると、それらは統合されて SAS アプリケーションに返されます。通常、データが SAS サーバーに転送される場合よりも実行時間は大幅に短くなります。

In-Database テクノロジは、既存のデータベース機能と標準の SAS 機能を使用して、その SAS 機能をデータベースに拡張します。たとえば、SORT や SUMMARY などの Base SAS プロシジャは、プロシジャのコードに簡単なオプションを含めることでデータベース環境内で実行できます。これらのプロシジャは、データベース環境内で実行するネイティブデータベース関数に"変換"されます。SAS 機能をデータベースに拡張する例として、SAS Enterprise Miner によって生成されたスコアリングコードを関数としてエクスポートできます。エクスポートされたこれらの関数は、SAS プロセスによって実行したり、標準 SQL ステートメントを使用してサードパーティまたはデータベースアプリケーションから呼び出したりすることができます。

SAS In-Database テクノロジについては、*SAS In-Database 製品: 管理者ガイド*および *SAS In-Database 製品: ユーザーガイド*で説明されています。。SAS In-Database コンポーネントには、サポートされているほとんどのデータベースのスコアリングおよび分析アクセラレータが含まれています。

SAS In-Memory Analytics テクノロジ

SAS In-Memory Analytics テクノロジは、Teradata や EMC Greenplum などの DBMS アプライアンスや、 *Hadoop 分散ファイルシステム*(HDFS)を使用する汎用ハードウェアで使用できる、多数のスレッドと高レベルのメモリを利用します。Teradata および EMC Greenplum アプライアンスは、特定の超並列処理(MPP)技術を使用してコンピューティングクラスターとして 組み立てられます。SAS In-Memory Analytics テクノロジを使用すると、処理されるすべてのデータがクラスター全体に分散され、分析プロシジャが開始される前にメモリにロードされます。こ

れは、データが必要に応じてブロック単位でロードされる従来の処理とは明らかに異なります。さらに、SAS High-Performance Analytics プロシジャは数百のスレッドで実行されるように設計されており、各スレッドは処理されるデータ全体の小さなサブセットを担当します。大規模なデータセットの分析が高速化されると、分析モデルがさらに改良されます。

SAS High-Performance Analytics ファミリ製品のいくつかのメンバーは、SAS High-Performance Analytics Server、SAS Visual Analytics、SAS High-Performance Risk など、SAS In-Memory Analytics テクノロジに基づいています。一連の SAS In-Memory Analytics コンポーネントの詳細については、In-Memory Analytics の Web サイト(Products & Solutions/In-Memory Analytics)を参照してください。

SAS High-Performance Analytics Server
: SAS High-Performance Analytics Server は、スレッドで実行するように設計されており、計算集約型の計算による予測モデル開発に焦点を当てたハイパフォーマンス分析プロシジャを提供します。これらのプロシジャは、SAS/STAT、SAS/QC、および SAS/ETS のライブラリから抽出されます。このプロシジャは、EMC Greenplum、Teradata アプライアンス、および Hadoop クラスターによって提供される MPP コンピューティング環境で実行されます。ハイパフォーマンス MPP 構成には、通常、最小 1.5TB のメモリと、コアあたり複数のスレッドを備えた 192 コア以上が搭載されています。

  SAS High-Performance Analytics プロシジャは、Base SAS セッションが実行されている要求元の SAS クライアント上で呼び出されます。これには、従来の Display Manager System、SAS Enterprise Guide、または SAS Enterprise Miner の **SAS High-Performance Data Mining** タブを使用できます。MPP 環境では、SAS クライアントは SAS High-Performance Analytics Server ノードと通信し、そこでシン SAS 環境が要求された SAS プロシジャまたは DS2 コードのコピーを実行します。完了すると、分析結果が SAS クライアント上の要求元アプリケーションに返されます。

  SAS High-Performance Analytics プロシジャの PERFORMANCE ステートメントを使用すると、スレッド処理と処理モード(SMP (クライアントモード)または MPP (分散モード))を制御するパラメーターを指定できます。SMP モード(クライアントマシン上の SAS セッション)では、CPUCOUNT のデフォルトはクライアントマシン上の CPU の数です。CPUCOUNT および NOTHREADS オプションは、SAS システムオプションをオーバーライドできます。この環境では、プロシジャが MPP モードで実行される場合、CPUCOUNT オプションのデフォルトでは、スレッド数はアプライアンスノード上の CPU の数によって決まります。PERFORMANCE ステートメントでのみ使用できる NOTHREADS システムオプションは、スレッドの数を調整します。

  SAS High-Performance Analytics Server とプロシジャについては、*SAS High-Performance Server 管理ガイド*に記載されています。SAS High-Performance Analytics Server は、SAS LASR Analytic Server を利用して、大量のデータと複雑な計算に最適化されたスケーラビリティと信頼性の高い分析インフラストラクチャを提供します。

SAS Visual Analytics
: SAS Visual Analytics は、SAS High-Performance Analytics Server 上で実行され、MPP 環境として構成されている場合は SAS Intelligence Platform 上で実行されます。SAS Visual Analytics は、ビジネスユーザーやビジネスアナリストが、ビッグデータを探索して価値を引き出すために特別に設計されたビジュアル化を使用して、さまざまなタスクを実行できる機能を提供します。これにより、記述統計量を超えて、より特殊な分析を非常にスケーラブルな方法で使用できるよ

うになります。これにより、将来に対するより正確な洞察が得られ、意思決定が容易になります。たとえば、ユーザーはデータを視覚的に探索し、わずか数秒で数十億行のデータの相関分析を実行し、結果を視覚的に表示できます。これにより、これまで明らかではなかったデータのパターン、トレンド、関係を迅速に特定することができます。

SAS Visual Analytics は、SAS High-Performance Analytics Server および SAS In-Database と組み合わせて、ハイパフォーマンスモデルのライフサイクルを提供できます。たとえば、SAS Visual Analytics を使用してデータを探索し、どの情報を使用するかを決定します。SAS High-Performance Analytics Server を使用してモデルを構築し、SAS In-Database を使用してモデルをスコアリング用の適切なデータベースにプッシュします。

SAS Visual Analytics には、Teradata または EMC Greenplum アプライアンス、または MPP クラスターとして構成された Hadoop HDFS などのブレードハードウェアの専用かつ特殊な構成が必要です。この環境は常にスレッド処理されています。SAS オプション CPUCOUNT および NOTHREADS は効果がありません。かわりに、PERFORMANCE ステートメントの NTHREADS オプションを使用して、スレッドの使用量を調整する方法を提供します。製品情報については、*SAS Visual Analytics: ユーザーガイド*を参照してください。

SAS Visual Analytics は、SAS LASR Analytic Server を利用して、大量のデータと複雑な計算に最適化されたスケーラビリティの高い分析インフラストラクチャを提供します。

SAS LASR Analytic Server
: SAS LASR Analytic Server は、データへの同時アクセスのための安全な環境を提供する分析プラットフォームです。データは、SAS High-Performance Analytics Server のコンピューティングノード全体のメモリにロードされます。SAS LASR Analytic Server は、SAS High-Performance Analytics Server のルートノード上で実行され、アプライアンス全体のワーカーノードがデータを並列に非常に高速でメモリに読み込みます。データが*同じ場所にあるデータプロバイダー*からのものでない場合、データは DBMS アプライアンスまたは Hadoop クラスターから読み取られ、SAS High-Performance Analytics Server のルートノードに転送されます。その後、*ワーカーノード*のメモリにロードされます。SAS LASR Analytic Server は、CPUCOUNT または NOTHREADS の影響を受けません。かわりに、PERFORMANCE ステートメントの NTHREADS オプションによってスレッドの使用量が調整されます。詳細については、*SAS LASR Analytic Server: 管理ガイド*を参照してください。

その他の SAS In-Memory Analytics 製品については、Products & Solutions/In-Memory Analytics を参照してください。

SAS High-Performance Analytics 製品の統合

SAS High-Performance Analytics ポートフォリオは、多くの SAS ソリューションと製品をサポートしています。一部の SAS 製品は、*並列処理*を活用するために、SAS

Grid Manager および(ある程度)他のハイパフォーマンス分析製品と統合されています。

SAS Enterprise Guide:
SAS Enterprise Guide は、SAS クライアントとして実行され、SAS プログラムを実行するためのフロントエンドを SAS サーバーに提供します。ユーザーは、Base SAS の並列処理機能(スレッド対応プロシジャや Stored Process など)を利用するプログラムを作成します。SAS Enterprise Guide は、SAS Grid Manager がワークロードバランシング、リソース割り当て、ジョブの優先順位付けを提供する環境を管理しているかどうかを検出できます。SAS Enterprise Guide では、異なるグリッドノード上でプロセスフローブランチを並列実行できます。これにより、同じサーバー上でタスクを並列実行できるようになり、SAS Grid Manager 環境でプロジェクトまたは個人レベルでタスクを実行できます。詳細については、*SAS Enterprise Guide* を参照してください。

SAS Data Integration Studio
SAS Data Integration Studio は SAS Data Integration Server 上で実行され、SAS Grid Manager がインストールされている場合はグリッドコンピューティングを自動的に利用します。SAS Data Integration Studio 3.4 は、SAS Grid Manager 管理グリッド上で実行できる SAS アプリケーションを自動的に生成するように強化されました。ユーザーは、プログラミングの知識や基盤となるグリッドインフラストラクチャの知識がなくても、グリッド対応の SAS アプリケーションを作成できます。

これらの SAS アプリケーションは、実行時に SAS Grid Manager 環境の存在を検出し、それに応じて実行を分散します。これらのグリッド対応アプリケーションは、SAS® Stored Process として保存でき、その後、SAS Web Report Studio、SAS Information Map Studio、SAS Add-In for Microsoft Office などの SAS Intelligence Platform コンポーネントによって実行できます。(これらのアプリケーションには、SAS Foundation に依存する SAS BI サーバーまたは SAS Enterprise BI サーバーが必要です)。

SAS Data Integration Studio は、SAS Intelligence Platform で構成されたインメモリ製品である SAS High-Performance Analytics Server および SAS Visual Analytics Server を MPP 環境として実行でき、常にスレッド処理されます。SAS Data Integration Studio は、SAS LASR Analytic Server または HDFS に High-Performance Analytics 変換を提供します。NOTHREADS および CPUCOUNT オプションは効果がありません。詳細については、*SAS Data Integration Studio: ユーザーガイド*を参照してください。SAS Data Integration Server は、SAS Intelligence Platform の一部として管理されます。

SAS Risk Dimensions
SAS Risk Dimensions の反復ワークフローは、SAS Data Integration Studio のワークフローと似ています。どちらも、データの異なるサブセットに対して同じ分析を実行します。このワークフローは、SAS Grid Manager を利用してグリッド全体に処理を分散するのに最適です。詳細については、*SAS Risk Dimensions ユーザーガイド*を参照してください。

SAS Enterprise Miner
SAS Enterprise Miner は、SAS Grid Manager グリッド上で実行できる SAS アプリケーションを自動的に生成できます。これらの SAS アプリケーションは、実行時に SAS Grid Manager 環境の存在を検出し、それに応じて実行を分散します。アプリケーションは、SAS® Stored Process として保存し、その後 SAS Web Report Studio などの SAS Business Intelligence コンポーネントによって実行することもできます。DMINE、DMREG、および DMDB プロシジャは、デフォ

ルトで THREADS を使用してスレッド対応になっています。NOTHREADS はマルチスレッド計算を無効にします。

SAS Enterprise Miner には、データビン化、補完、スコアリング、変換などのタスクのための、ハイパフォーマンス統計およびデータマイニングプロシジャのキーセットが含まれています。これらのプロシジャは、高度にスレッド化された SAS High-Performance Analytics Server で実行されます。MPP モードでは、SAS Enterprise Miner はデータ、メモリ、および計算をサーバーノード全体に分散して、予測モデルを構築します。SAS High-Performance Analytics Server がインストールされており、特定の Hadoop HPDM コードが有効になっている場合、SAS Enterprise Miner で HPDM タブが使用可能になり、サーバーノードでの実行が許可されます。Enterprise Miner のスコアリングコードは、SAS In-Database テクノロジを使用してデータベース内で実行できます。詳細については、*SAS Enterprise Miner: 管理と構成*を参照してください。

SAS Viya プラットフォーム

SAS 9.4M5 では、さまざまなハイパフォーマンス製品と SAS Cloud Analytic Services へのアクセスを提供するソフトウェアである SAS Viya プラットフォームのライセンスを取得できます。詳細については、*SAS Viya プラットフォームプログラミング: 入門ガイド*を参照してください。

5 部

出力の作成

27

出力

SAS 出力の定義

SAS 出力は、SAS プログラムの実行結果です。ほとんどの SAS プロシジャと一部の DATA ステッププログラムは出力を生成します。SAS プログラムは、次のタイプの出力の一部またはすべてを 生成できます。

出力先
Output Delivery System (ODS)が特定のタイプの出力を生成するために使用する、SAS プログラム内の指定。簡単に言うと、ODS が出力をどこにどのようにルーティングするかということです。たとえば、ODS は出力を HTML としてブラウザーにルーティングしたり、ファイルや端末にルーティングしたり、単純なリストレポートとして表示したりできます。詳細については、"Overview of How ODS Works" (*SAS Output Delivery System: Procedures Guide*)を参照してください。出力先は次によって異なります。

- 動作環境
- SAS の実行モード
- SAS のバージョン

プログラム結果
SAS プロシジャおよび SAS DATA ステップアプリケーションからのプログラム結果が含まれます。これらの結果は、ファイルに送信したり、レポートとして印刷したりできます。SAS には、出力をカスタマイズするために使用できるさまざまなオプション、出力形式、ステートメント、コマンドがあります。Output Delivery System を使用すると、出力の保存方法を制御する出力先、出力の構造を制御するテーブル定義、出力のスタイル要素を制御するスタイルテンプレートを指定できます。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。

SAS プログラムの実行から取得できる出力のタイプの例をいくつか示します。

- SAS データセット
- Web 表示用の HTML ファイル
- 単純なリストレポート
- Microsoft Word での表示に適した RTF 出力
- モバイルデバイスでの表示に適した SVG 出力
- 複数の出力先を一度に指定するための ODS ドキュメント
- PostScript や PDF などの高解像度プリンター用にフォーマットされた出力
- さまざまなマークアップ言語(HTML に加えて)でフォーマットされた出力

SAS ログ出力
SAS ログ
SAS セッションの説明が含まれており、実行されたソースコードの行がリストされています。

“すべてのモードでのログへの書き込み” (714 ページ)で説明されている SAS ステートメントを使用して、特定の情報(変数値やテキスト文字列など)を SAS ログに書き込むことができます。

条件付きプロシジャ JulieM

このログは、DATASETS プロシジャや OPTIONS プロシジャなど、ユーティリティ関数を実行する一部の SAS プロシジャでも使用されます。*Base SAS プロシジャガイド*を参照してください。

SAS ログはプログラム処理のジャーナルを提供するため、必須のデバッグツールです。ただし、SAS プログラムのデバッグにログを有効にするには、特定のシステムオプションが有効である必要があります。“ログのカスタマイズ” (715 ページ)では、使用できるいくつかの SAS システムオプションについて説明します。

SAS コンソールログ
通常の SAS ログがアクティブでないときに、情報、警告、およびエラーメッセージを記録するために作成されます。SAS ログがアクティブな場合、SAS コンソールログは、致命的なシステム初期化エラーまたは遅延終了メッセージにのみ使用されます。

SAS ログ機能の出力
SAS ログ機能を使用して作成したログメッセージが含まれています。SAS ログ機能はオプションです。ログ機能メッセージは、SAS プログラム内で作成することも、SAS サーバーメッセージのログを記録するために SAS によって作成することもできます。ログ機能のログメッセージは、認証、管理、パフォーマンスなどのメッセージカテゴリ、または SAS プログラムのカスタマイズされたメッセージカテゴリに基づいています。SAS プログラムでは、ログ機能関数、自動呼び出しマクロ、または DATA ステップコンポーネントオブジェクトを使用して、ログ機能環境を作成します。

アペンダーは、マクロプログラムまたは DATA ステップ中に定義されます。ロガーは、SAS セッション中に定義されます。

詳細については、*SAS プログラムおよびジョブの SAS Viya ログ機能*を参照してください。

デフォルトの出力先

デフォルトの出力先は、SAS のバージョンと SAS の実行モードによって異なります。

次の表は、SAS バージョンに基づいた各操作方法のデフォルトの出力先を示しています。

表 27.1 デフォルトの出力先の比較

SAS の実行モード	ビューアー	ODS 出力先
SAS Studio	**結果**タブまたはブラウザーウィンドウ	HTML5
Enterprise Guide	プロジェクトペイン、プロセスフロー、またはブラウザーウィンドウ	HTML5
バッチモード	動作環境により異なる	

新しいデフォルトと ODS 出力先の詳細については、*SAS Output Delivery System: ユーザーガイド*を参照してください。

出力先の変更

SAS では、ログ、プロシジャ、および DATA ステップ出力の送信先を制御するさまざまな方法があります。

次のリストでは、出力のルーティングに使用される一般的に使用される ODS ステートメントとその他の SAS 言語要素の一部について説明します。

表 27.2 出力先を変更する方法

使用方法	出力結果
PRINTTO プロシジャ	DATA ステップ、ログ、またはプロシジャ出力をシステムのデフォルトの出力先から選択した出力先にルーティングします。PRINTTO プロシジャは、非 ODS 出力先を定義します。
FILENAME ステートメント	ファイル参照名を外部ファイルまたは出力デバイスに関連付け、ファイルおよびデバイスの属性を指定できるようにします。
ODS OUTPUT ステートメント	出力オブジェクトから SAS データセットを生成し、OUTPUT 出力先の選択リストと除外リストを管理します。
ODS DOCUMENT ステートメント	さまざまな方法でデータの再構築、移動、再生を可能にする ODS ドキュメントを生成します。また、分析を再実行したり、データベース

使用方法	出力結果
	クエリを繰り返したりすることなく、複数の出力先を指定することもできます。
ODS HTML5 ステートメント	HTML 出力先を開いたり、管理したり、閉じたりします。これにより、埋め込みスタイルシートを含む HTML 5 出力が生成されます。
ODS MARKUP ステートメント	MARKUP 出力先を開いたり、管理したり、閉じたりします。これにより、多くのさまざまなマークアップ言語の 1 つを使用してフォーマットされた SAS 出力が生成されます。
ODS WORD ステートメント	Microsoft Word の ODS 出力先を開いたり、管理したり、閉じたりします。これにより、Microsoft Office 2010 以降のバージョンと互換性のある Microsoft Word 出力が生成されます。
SAS システムオプション	SAS プログラム全体のログと出力の出力先を再定義します。これらのシステムオプションは、SAS を起動するときに指定します。出力のルーティングに使用されるシステムオプションは、ALTLOG=、ALTPRINT=、LOG=、PRINT= オプションです。

グローバル ODS ステートメントの概念的な情報については、次のリソースを参照してください。

- *"出力先カテゴリテーブル" (SAS Output Delivery System: ユーザーガイド)*。

出力のカスタマイズ

出力を説明的にする

SAS では、出力をカスタマイズできるステートメントとシステムオプションが多数用意されています。有益なタイトル、フットノート、およびラベルを追加して出力をカスタマイズし、ページ上での情報のレイアウト方法を制御できます。

次のリストでは、使用できるステートメントと SAS システムオプションの一部について説明します。

表 27.3　出力をカスタマイズする方法

SAS 言語要素	機能
CENTER | NOCENTER システムオプション	出力を中央に配置するかどうかを制御します。デフォルトでは、SAS はタイトルとプロシジャ出力をページおよびパーソナルコンピューターのディスプレイの中央に配置します。
DATE | NODATE システムオプション	日付と時間の値の出力を制御します。このオプションを有効にすると、SAS は出力の各ページの上部に SAS ジョブが開始された日付と時間を出力します。
FOOTNOTE ステートメント	各出力ページの下部にフットノートを出力します。この目的で **FOOTNOTES** ウィンドウを使用することもできます。 null のフットノートステートメント(footnote;)を使用して、FOOTNOTE ステートメントを抑制することもできます。
FORMCHAR=システムオプション	CALENDAR、FREQ、REPORT、TABULATE などの一部のプロシジャのデフォルトの出力フォーマット文字を指定します。このシステムオプションは、従来の LISTING 出力にのみ影響します。
LABEL ステートメント	説明ラベルを変数に関連付けます。ほとんどのプロシジャ出力では、SAS は変数名ではなくラベルを書き込みます。 LABEL ステートメントは、特定の SAS プロシジャで使用される場合、説明ラベルも提供します。特定のプロシジャでの LABEL ステートメントの使用については、*Base SAS プロシジャガイド*を参照してください。
LINESIZE=および PAGESIZE= システムオプション	印刷出力のデフォルトの 1 ページあたりの行数(ページサイズ)と 1 行あたりの文字数(行サイズ)を変更します。デフォルトは、SAS の実行方法と特定の SAS システムオプションの設定によって異なります。新しいページと行のサイズは、OPTIONS ステートメントまたは **OPTIONS** ウィンドウで指定します。FILE ステートメントで DATA ステップ出力の行サイズとページサイズを指定することもできます。 LINESIZE=および PAGESIZE=システムオプションに使用する値は、一部の SAS プロシジャによって生成される出力の外観に大きな影響を与える可能性があります。

SAS 言語要素	機能
NUMBER | NONUMBER および PAGENO=システムオプション	ページ番号を制御します。NUMBER システムオプションは、印刷出力の各ページの最初のタイトル行にページ番号を印刷するかどうかを制御します。PAGENO=システムオプションを使用して、SAS によって生成される出力の次のページの開始ページ番号を指定することもできます。
TITLE ステートメント	各出力ページの上部にタイトルを出力します。 TITLE ステートメントまたは **TITLES** ウィンドウを使用して、デフォルトのタイトルを置き換えたり、SAS プログラムに他の説明的なタイトルを指定したりできます。null のタイトルステートメント(title;)を使用して、TITLE ステートメントを抑制することもできます。

ODS を使用した出力のスタイルと構造のカスタマイズ

ODS は、出力先を制御できるだけではありません。また、出力の構造とスタイルをカスタマイズすることもできます。ODS はテーブルとスタイルテンプレート(定義)を使用してプロシジャと DATA ステップの結果を表示するため、カスタマイズした*テーブル*と*スタイルテンプレート*を作成することでこれらの結果を制御できます。定義を最初から作成しない場合は、既存のスタイル定義とテーブル定義を変更することもできます。

Output Delivery System 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。

出力内の値の再フォーマット

特定の SAS ステートメント、プロシジャ、およびオプションを使用すると、指定された出力形式を使用して値を出力できます。

FORMAT プロシジャを使用すると、独自の出力形式と入力形式を設計できるため、値の表示がさらに柔軟になります。FORMAT プロシジャの詳細については *"FORMAT プロシジャ" (Base SAS プロシジャガイド)*、他のすべての SAS システムオプションについては *SAS システムオプション: リファレンス*を参照してください。

欠損値の出力

SAS は、SAS リスト内の通常欠損数値を単一のピリオドとして表し、欠損文字値を空白スペースとして 表します。数値変数に特殊欠損値を指定すると、SAS は文字またはアンダースコアを 書き込みます。文字変数の場合、SAS は変数の長さに等しい一連の空白を書き込みます。

MISSING=システムオプションを使用すると、通常欠損数値のピリオドのかわりに出力する文字を指定できます。詳細については、“MISSING= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

SAS ログ

ログの構造

SAS ログは、SAS セッションまたは SAS プログラムで実行したすべての記録です。SAS システムオプションの設定、SAS の実行方法、および指定したプログラムステートメントに応じて、ログには次の種類の情報が含まれる場合があります。

- プログラムステートメント
- プログラムによって 作成されたデータセットの名前
- プログラムの実行中に発生した注記、警告、またはエラーメッセージ
- 各データセットに含まれる変数とオブザベーションの数
- 各ステップに必要な処理時間

例のコード 27.1　サンプル SAS ログ

```
NOTE: Copyright (c) 2016 by SAS Institute Inc., Cary, NC, USA.  1
NOTE: SAS (r) Proprietary Software  2     Licensed to SAS Institute Inc., Site 1.  3
NOTE: This session is executing on the W32_7PRO  platform.  4
NOTE: SAS initialization used:
      real time           4.19 seconds
      cpu time            0.85 seconds
1     options pagesize=24
2     linesize=64 pageno=1 nodate;  5
3     data logsample;  6
5       infile
5 !  '\\myserver\my-directory-path\sampledata.dat';  7
6       input LastName $ ID $ Gender $ Birth : date7. score1
6 !  score2 score3 score4 score5 score6 score7 score8;
7       format Birth mmddyy8.;
8     run;
NOTE: The infile
      '\\myserver\my-directory-path\sampledata.dat' is:  8
      Filename=\\myserver\my-directory-path\sampledata.dat,
      RECFM=V,LRECL=256,File Size (bytes)=296,
      Last Modified=08Jun2009:15:42:26,
      Create Time=08Jun2009:15:42:26
NOTE: 5 records were read from the infile  9
      '\\myserver\my-directory-path\sampledata.dat'.
      The minimum record length was 58.
      The maximum record length was 59.
NOTE: The data set WORK.LOGSAMPLE has 5 observations and 12
      variables.  10
NOTE: DATA statement used (Total process time):
      real time           0.21 seconds  11
      cpu time            0.03 seconds
9
10    proc sort data=logsample;  12
11      by LastName;
12    run;
NOTE: There were 5 observations read from the data set
      WORK.LOGSAMPLE.
NOTE: The data set WORK.LOGSAMPLE has 5 observations and 12
      variables.  13
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
13
14    proc print data=logsample;  14
15      by LastName;
16    run;
NOTE: There were 5 observations read from the data set
      WORK.LOGSAMPLE.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds
```

次のリストは、前述の SAS ログ内の丸で囲まれた数字に対応します。

1　著作権情報

2　このプログラムの実行に使用された SAS リリース

3　プログラムが実行されたコンピューターの名前とサイト番号

4　プログラムを実行するために使用されるプラットフォーム

注: 著作権情報、ライセンスおよびサイト情報、ならびにデータセット内のオブザベーションと変数の数は、NOTES | NONOTES システムオプションを使用して抑制できます。

5 OPTIONS ステートメントは、ページサイズ 24、行サイズ 64 を設定し、SAS 出力をページ 1 に設定し、出力内の日付を抑制します。

6 プログラムを構成する SAS ステートメント(SAS システムオプション SOURCE が有効な場合)

7 長いステートメントが次の行に続きました。継続行の前には感嘆符(!)が付いており、行番号は変わらないことに注意してください。

8 入力ファイル情報: 生データとその取得場所に関する注記または警告メッセージ(SAS システムオプション NOTES が有効な場合)

9 入力ファイルから読み取られたレコードの数とレコード長(SAS システムオプション NOTES が有効な場合)

10 プログラムが作成した SAS データセットと、作成された各データセットのオブザベーションと変数の数を含む注記(SAS システムオプション NOTES が有効な場合)

11 STIMER オプションまたは FULLSTIMER オプションが設定されている場合にレポートされるパフォーマンス統計量

12 データセットを並べ替えるプロシジャ

13 並べ替えられた SAS データセットに関する注記

14 データセットを出力するプロシジャ

バッチ、ライン、またはオブジェクトサーバーモードの SAS ログ

SAS の起動時に LOGCONFIGLOC=システムオプションが指定されていない場合は、LOG=システムオプションまたは LOGPARM=システムオプションを使用して SAS ログを構成できます。これらのオプションは、バッチモード、ラインモード、またはオブジェクトサーバーモードで指定できます。LOGCONFIGLOC=システムオプションが指定されている場合、ログは SAS ログ機能によって実行され、LOGPARM=オプションは無視されます。LOG=オプションは、%S{App.Log}変換文字がログ構成ファイルで指定されている場合にのみ適用されます。これらのログシステムオプションについては、*"LOGPARM= システムオプション" (SAS システムオプション: リファレンス)*を参照してください。SAS ログ機能については、*SAS プログラムおよびジョブの SAS Viya ログ機能*を参照してください。

次のセクションでは、LOGPARM=システムオプションを使用して構成できるログオプションと、ログ機能が開始されていない場合にこれらのオプションの SAS ログに名前を付ける方法について説明します。LOG=システムオプションは、SAS ログに名前を付けます。LOGPARM=システムオプションを使用すると、SAS ログに追加または置換したり、SAS ログに書き込むタイミングを決定したり、特定の条件下で新しい SAS ログを開始したりできます。

SAS ログの追加または置換

LOG=システムオプションで SAS ログの出力先を指定すると、SAS ログがすでに存在するかどうかが SAS によって確認されます。ログが存在する場合は、LOGPARM=システムオプションの OPEN=オプション使用して、SAS ログにコンテンツを書き込む方法を指定できます。

次の SAS コマンドでは、1 日以上経過した既存の SAS ログを置き換えるために、LOG=と LOGPARM=の両方のシステムオプションが指定されています。

```
sas -sysin "my-batch-program"  -log "my-file-path/SASlogs/mylog"
   -logparm open=replaceold
```

LOGPARM=システムオプションの ROLLOVER=オプションが特定のサイズ *n* に設定されている場合、OPEN=オプションは無視されます。

SAS ログに書き込むタイミングの指定

コンテンツは、ログコンテンツの生成時に SAS ログに書き込むことも、バッファーリングしてバッファーがいっぱいになったときに書き込むこともできます。デフォルトでは、ログバッファーがいっぱいになると、SAS はログに書き込みます。ログコンテンツをバッファーリングすることにより、SAS は一度に 1 行ずつ書き込むのではなく、定期的にログファイルに書き込むことでより効率的に実行します。

SAS ログコンテンツをいつ書き込むかを構成するには、LOGPARM=システムオプションの WRITE=オプションを使用します。ログコンテンツが生成時に書き込まれるようにするには LOGPARM="WRITE=IMMEDIATE"を設定し、バッファーがいっぱいになったときにログコンテンツを書き込むようにするには LOGPARM="WRITE=BUFFERED"を設定します。

SAS ログのロールオーバー

SAS ログは、長時間実行されているサーバーやバッチジョブの場合、非常に大きくなる可能性があります。LOGPARM=システムオプションと LOG システムオプションを一緒に使用すると、SAS ログを新しい SAS ログにロールオーバーするように指定できます。SAS がログをロールオーバーすると、ログが閉じて新しいログが開きます。

LOGPARM=システムオプションは、ログファイルをいつ開くか閉じるかを制御し、LOG=システムオプションは SAS ログファイルに名前を付けます。SAS セッションの開始時、ログが特定のサイズに達したとき、またはまったくないときに、ログを自動的にロールオーバーできます。SAS ログ名でフォーマットディレクティブを使用すると、各 SAS ログに一意識別子を付けることができます。SAS ログをロールオーバーする方法の詳細については、次の例を参照してください。

- "例: ディレクティブを使用して SAS ログに名前を付ける" (720 ページ)

- “例: ディレクティブの変更時に SAS ログを自動的にロールオーバーする” (721 ページ)
- “例: SAS セッションごとの SAS ログのロールオーバー” (722 ページ)
- “例: ログサイズごとの SAS ログのロールオーバー” (723 ページ)

ログをまったくロールオーバーしないようにするには、SAS の起動時に LOGPARM=“ROLLOVER=NONE”オプションを指定します。ディレクティブは 解決されず、ロールオーバーは発生しません。たとえば、LOG=“March#b.log”の場合、ディレクティブ#b は解決されず、ログ名は March#b.log になります。

すべてのモードでのログへの書き込み

すべてのモードで、次のステートメントを使用して、ログに追加情報を書き込むように SAS に指示できます。

表 27.4　SAS ログに情報を書き込むその他のステートメント

ステートメント	説明	例
/NESTING オプションを使用した DATA ステートメント	各 DO-END および SELECT-END ネストレベルの開始と終了について、SAS ログに注記を出力することを指定します。このオプションを使用すると、一致しない DO-END ステートメントと SELECT-END ステートメントをデバッグでき、ネストレベルが明らかでない大規模なプログラムで特に役立ちます。	“DATA ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
ERROR ステートメント	_ERROR_を 1 に設定します。SAS ログに書き込まれるメッセージはオプションです。	“ERROR ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
LIST ステートメント	処理中のオブザベーションの入力データレコードを SAS ログに書き込みます。	“LIST ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
PUT ステートメント	SAS ログ、SAS 出力ウィンドウ、または最新の FILE ステートメントで指定された外部の場所に行を書き込みます。	“PUT ステートメント” (*SAS DATA ステップステートメント: リファレンス*)
%PUT ステートメント	テキストまたはマクロ変数情報を SAS ログに書き込みます。	“%PUT マクロステートメント” (*SAS マクロ言語: リファレンス*)

PUTLOG ステートメント	メッセージを SAS ログに書き込みます。	“PUTLOG ステートメント” (*SAS DATA ステップステートメント: リファレンス*)

PUT、PUTLOG、LIST、DATA、ERROR ステートメントを条件付き処理と組み合わせて使用し、選択した情報をログに書き込むことで DATA ステップをデバッグします。

ログのカスタマイズ

ログのコンテンツを抑制する

大規模な SAS 運用プログラムや、変更せずに定期的に実行するアプリケーションがある場合、ログの一部を抑制することが必要な場合があります。SAS システムオプションを使用すると、SAS ステートメントとシステムメッセージを抑制したり、エラーメッセージの数を制限したりできます。すべての SAS システムオプションは、セッション中、またはオプションを変更するまで有効です。プログラムがエラーなく正常に実行されるまで、ログメッセージを抑制しないでください。

次のリストでは、ログのコンテンツを変更するために使用できるいくつかの SAS システムオプションについて説明します。これらのオプションの一部の使用例については、“例: SAS ログへの出力を抑制する”を参照してください。

表 27.5 ログをカスタマイズする方法

SAS システムオプション	説明
CPUID | NOCPUID	CPU ID 番号を SAS ログに書き込むかどうかを指定します。
ECHOAUTO | NOECHOAUTO	入力ファイル内の autoexec コードをログに書き込むかどうかを指定します。
ERRORS=	SAS が完全なエラーメッセージを発行するオブザベーションの最大数を指定します。
HOSTINFOLONG	SAS の起動時に追加の動作環境情報を SAS ログに書き込みます。
LOGPARM "OPEN=APPEND | REPLACE | REPLACEHOLD"	ログファイルがすでに存在し、SAS がそのログを開いている場合、LOGPARM オプションは、既存のログに追加するか、既存のログを置き換えるかを指定します。REPLACEHOLD オプションは、1 日以上経過したログを置き換えることを指定します。
MLOGIC	マクロ実行トレース情報を SAS ログに書き込みます。

MLOGICNEST	SAS ログの MLOGIC 出力にマクロのネスト情報を表示するかどうかを指定します。
MPRINT | NOMPRINT	マクロ実行によって生成された SAS ステートメントを SAS ログに書き込むかどうかを指定します。
MPRINTNEST	SAS ログの MLOGIC 出力にマクロのネスト情報を表示するかどうかを指定します。
MSGLEVEL=	SAS ログに書き込まれるメッセージの詳細レベルを指定します。
NEWS=	サイトで維持されるニュース情報を SAS ログに書き込むかどうかを指定します。
NOTE | NONOTES	注記(NOTE で始まるメッセージ)を SAS ログに書き込むかどうかを指定します。NONOTES は、エラーまたは警告メッセージを抑制しません。
OVP | NOOVP	エラーメッセージを太字にするオーバープリントを有効にするかどうかを指定します。
PAGEBREAKINITIAL	SAS ログとリストファイルを新しいページで開始するかどうかを指定します。
PRINTMSGLIST | NOPRINTMSGLIST	メッセージの拡張リストを SAS ログに書き込むかどうかを指定します。
SOURCE | NOSOURCE	SAS がソースステートメントを SAS ログに書き込むかどうかを指定します。
SOURCE2 | NOSOURCE2	SAS が%INCLUDE ステートメントに含まれるファイルからの 2 次ソースステートメントを SAS ログに書き込むかどうかを指定します。
SYMBOLGEN | NOSYMBOLGEN	マクロ変数参照の解決結果を SAS ログに書き込むかどうかを指定します。

これらおよび他の SAS システムオプションの使用方法の詳細については、*SAS システムオプション: リファレンス*を参照してください。

ログの外観のカスタマイズ

次の SAS ステートメントと SAS システムオプションを使用すると、ログをカスタマイズできます。ログをカスタマイズすると、レポートの作成や永久的な記録の作成にログを使用するときに役立ちます。

表 27.6 ログをカスタマイズするための SAS ステートメントとシステムオプション

DETAILS システムオプション	ファイルが SAS ライブラリにリストされるときに追加情報を含めるかどうかを指定します。
DTRESET システムオプション	SAS ログおよびリストファイルの日付と時間を更新するかどうかを指定します。
FILE ステートメント	PUT ステートメントの結果を外部ファイルに書き込むことができます。FILE ステートメントで LINESIZE=および PAGESIZE=オプションを使用して、レポートをカスタマイズできます。
LINESIZE=システムオプション	DATA ステップおよびプロシジャで使用される SAS ログおよび SAS 出力の行サイズ(プリンターの行幅)を指定します。
MISSING=システムオプション	欠損数値変数値に対して出力する文字を指定します。
NUMBER システムオプション	印刷出力の各ページの最初のタイトル行にページ番号を印刷するかどうかを制御します。
PAGE ステートメント	SAS ログ内の指定された行数をスキップします。
PAGESIZE=システムオプション	SAS 出力の 1 ページあたりに印刷できる行数を指定します。
SKIP ステートメント	SAS ログ内の指定された行数をスキップします。

例: 出力先の管理

例: デフォルトの HTML 出力の作成

コード例

SAS プログラムを実行する場合、デフォルトの出力先は HTML です。SAS Data and AI Studio(2026.06 より前は SAS Studio という名前)では、結果が**結果**ウィンドウに表示されます。

```
title 'Student Weight';
  proc print data=sashelp.class;
    where weight>100;
  run;
```

アウトプット 27.1　デフォルトの HTML 出力

Student Weight

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
8	Janet	F	15	62.5	112.5
14	Mary	F	15	66.5	112.0
15	Philip	M	16	72.0	150.0
16	Robert	M	12	64.8	128.0
17	Ronald	M	15	67.0	133.0
19	William	M	15	66.5	112.0

注: 以前に HTML 出力先を閉じた場合、出力はデフォルトで WORK ディレクトリに送信されます。HTML 出力先を閉じて、同じ SAS セッションで再度開くと、すべての出力が現在のディレクトリに保存されます。出力を表示するために ods html close;を指定する必要はありません。

要点

- HTML はデフォルトの出力先です。
- 出力先は、動作環境、SAS の実行モード、および SAS のバージョンによって異なります。

例: ODS ステートメントを使用して出力先を変更する

コード例

出力を LISTING 出力先に送信する場合は、SAS プログラムで ODS ステートメントを使用して出力先を変更できます。

この例では、ODS LISTING ステートメントと ODS HTML CLOSE ステートメントを指定することで、出力先を HTML から LISTING に変更しています。出力先を LISTING に変更すると、出力は **SAS 出力ウィンドウ**にリストレポートとして自動的に表示されます。

```
ods html close;
options nodate;

title 'Students';
    proc print data=sashelp.class;
     where weight>100;
    run;
ods html;
```

アウトプット 27.2　ウィンドウ環境でのリスト出力

Students

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
8	Janet	F	15	62.5	112.5
14	Mary	F	15	66.5	112.0
15	Philip	M	16	72.0	150.0
16	Robert	M	12	64.8	128.0
17	Ronald	M	15	67.0	133.0
19	William	M	15	66.5	112.0

要点

- より永続的なソリューションが必要な場合は、SAS を実行するたびに出力がデフォルトで LISTING 出力先に送信されるように設定を変更できます。

関連項目

- “ODS 出力先について” (*SAS Output Delivery System: ユーザーガイド*)

例: SAS ログのロールオーバー

例: ディレクティブを 使用して SAS ログに名前を付ける

コード例

次の例は、LOG=システムオプションでディレクティブを 使用して、SAS ログ名に年、月、日を含める方法を示しています。

```
-log "my-file-path/saslog/#Y#b#dsas.log
```

SAS ログが 2019 年 2 月 2 日に作成された場合、ログの名前は 2019Feb02sas.log になります。

要点

- ディレクティブを 使用すると、SAS ログ名に日、時間、システムノード名、一意識別子などの情報を追加できます。LOG システムオプションでログ名を指定する場合、SAS ログの名前に 1 つ以上のディレクティブを 含めることができます。
- ディレクティブは、SAS ログに一意の名前を付けるために使用される処理命令です。
- ディレクティブは、LOGPARM=システムオプションの ROLLOVER=オプションの値が AUTO または SESSION に設定されている場合にのみ解決されます。
- ログ名にディレクティブが指定されており、ROLLOVER オプションの値が NONE または特定のサイズ *n* である場合、#b や#Y などのディレクティブ文字はログ名の一部になります。
- LOG システムオプションに関する前述の例を使用すると、LOGPARM=システムオプションで ROLLOVER=NONE が指定されている場合、SAS ログの名前は #Y#b#dsas.log になります。

関連項目

- ディレクティブの完全なリストについては、“LOGPARM= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

例: ディレクティブの変更時に SAS ログを自動的にロールオーバーする

コード例

SAS ログ名に 1 つ以上のディレクティブが含まれており、LOGPARM=システムオプションの ROLLOVER=オプションが AUTO に設定されている場合、SAS はログを閉じて、ディレクティブ値が変更されたときに新しいログを開きます。新しい SAS ログ名には、新しいディレクティブ値が含まれます。

次の表は、次の SAS コマンドを使用して、毎月 2 日の午前 6 時 15 分に SAS が開始されたときに作成されるログ名の一部を示しています。

```
sas -objectserver -log "london#n#d#%H.log"
-logparm
"rollover=auto"
```

要点

- ディレクティブ#n は、システムノード名をログ名に挿入します。#d は、ログ名に日付を追加します。#H はログ名に時間を追加します。この例のノード名は Thames です。
- この SAS セッションのログは、時間が変わるときと日が変わるときにロールオーバーされます。

表 27.7 ロールオーバーログのログ名

ロールオーバー時間	ログ名
SAS 初期化	londonThames0206.log
最初のロールオーバー	londonThames0207.log
その日の最後のログ	londonThames0223.log

ロールオーバー時間	ログ名
真夜中過ぎの最初のログ	londonThames0300.log

関連項目

- ディレクティブの完全なリストについては、“LOGPARM= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

例: SAS セッションごとの SAS ログのロールオーバー

コード例

SAS セッションの開始時にログをロールオーバーするには、SAS の起動時に LOGPARM=“ROLLOVER=SESSION”オプションを指定します。次の例では、SAS セッションを開始したユーザー名を含むログファイル名を作成します。

```
sas -log "%l.log"
-logparm "rollover=session"
```

要点

- SAS は、LOG=システムオプションで指定されたシステム固有のディレクティブを解決し、解決された値を使用して新しいログファイルに名前を付けます。
- SAS セッション中にロールオーバーは発生せず、ログファイルは SAS セッションの終了時に閉じられます。

関連項目

- “LOGPARM= システムオプション” (*SAS システムオプション: リファレンス*)

例: ログサイズごとの SAS ログのロールオーバー

コード例

ログが特定のサイズに達したときにログをロールオーバーするには、SAS の起動時に LOGPARM=“ROLLOVER=*n*”オプションを指定します。

```
sas -log "test%H%M.log"
-logparm "rollover=80"
```

要点

- *n* はログの最大サイズ(バイト単位)であり、10K (10,240)バイトより小さくすることはできません。
- ログが指定されたサイズに達すると、SAS はログを閉じ、ファイル名にテキスト“old”を追加します(例: londonold.log)。SAS は、ログ名に LOG=オプションの値を使用して新しいログを開き、LOGPARM システムオプションの OPEN=オプションステートメントを無視します。
- これは、SAS が既存のログファイルを決して上書きしないようにするのに役立ちます。ログサイズに基づいてロールオーバーするログでは、ログ名のディレクティブは無視されます。
- サーバー間でログファイル名を一意にするために、SAS はログファイル名に基づいてロックファイルを作成します。ロックファイル名は *logname*.lck です。ここで、*logname* は LOG=オプションの値です。
- サーバーログ用のロックファイルが存在し、別のサーバーが同じログ名を指定している場合、2 番目のサーバーのログファイル名とロックファイル名に番号が追加されます。番号は 2 から始まり、同じログファイル名に対する後続の要求に対して 1 ずつ増加します。たとえば、ログファイル london.log にロックが存在する場合、2 番目のサーバーログは london2.log となり、ロックファイルは london2.lck になります。

関連項目

- “LOGPARM= システムオプション” (*SAS システムオプション: リファレンス*)

例: SAS ログへの出力を抑制する

例: NOSOURCE システムオプションを使用してソースプログラムの SAS ログへの出力を抑制する

コード例

この例では、最初の DATA ステップでソースコードを SAS ログに出力します。この場合のソースコードにはパスワードが含まれており、DATA ステップの実行時に SAS ログに表示されます。

```
data mytable;
  password="ABC123";
  x = 1;
run;
```

例のコード 27.2　SAS ログのソースコードにパスワードがどのように出力されるかを示す SAS ログ出力

```
73      data mytable;
74      password="ABC123";
75      x = 1;
76      run;
```

NOSOURCE システムオプションを使用すると、ソースプログラムのログへの出力を抑制できます。

```
options nosource;
data mytable;
   password="ABC123";
run;
```

例のコード 27.3　ソースコードのログへの出力がどのように抑制されているかを示す SAS ログ出力

```
114  options nosource;

NOTE: The data set WORK.MYTABLE has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time          0.02 seconds
      cpu time           0.01 seconds
```

要点

- SOURCE | NOSOURCE システムオプションは、SAS がソースステートメントを SAS ログに書き込むかどうかを指定します。
- NOSOURCE システムオプションは、SAS ソースステートメントを SAS ログに書き込みません。

関連項目

- “SOURCE システムオプション” (*SAS システムオプション: リファレンス*)

例: NOLIST オプションを使用して SAS ログへの変数データの出力を抑制する

コード例

この例では、NOSOURCE オプションにより、ソースプログラムの SAS ログへの出力が抑制されます。ただし、2 番目の DATA ステップでエラーが生成されるため、パスワードデータが SAS ログのログ情報に出力されます。

```
options nosource;
data mytable;
   password="123ABC";
run;
data mytable2;
   set mytable;
   password=input(password,datetime.);
run;
```

例のコード 27.4　SAS ログの NOTE にパスワードがどのように出力されるかを示すログ出力

```
NOTE: The data set WORK.MYTABLE has 1 observations and 1 variables.
  NOTE: DATA statement used (Total process time):
      real time          0.02 seconds
      cpu time           0.03 seconds

  NOTE: Numeric values have been converted to character values at the places given by:
      (Line):(Column).
      123:14
  NOTE: Invalid argument to function INPUT at line 123 column 14.
  password=. _ERROR_=1 _N_=1
  NOTE: Mathematical operations could not be performed at the following places. The results of the
      operations have been set to missing values.
      Each place is given by: (Number of times) at (Line):(Column).
      1 at 123:14
  NOTE: There were 1 observations read from the data set WORK.MYTABLE.
  NOTE: The data set WORK.MYTABLE2 has 1 observations and 1 variables.
  NOTE: DATA statement used (Total process time):
      real time          0.03 seconds
      cpu time           0.00 seconds
```

DATA ステートメントに NOLIST オプションを指定すると、エラー発生時の変数データの出力を抑制できます。

```
data mytable2 / NOLIST;
  set mytable;
  password=input(password,datetime.);
run;
```

例のコード 27.5　エラー発生時に NOLIST オプションが SAS ログへの変数データの出力を抑制する方法を示すログ出力

```
NOTE: Numeric values have been converted to character values at the places given by:
      (Line):(Column).
      127:14
  NOTE: Invalid argument to function INPUT at line 127 column 14.
  NOTE: NOLIST option on the DATA statement suppressed output of variable listing.
  NOTE: Mathematical operations could not be performed at the following places. The results of the
      operations have been set to missing values.
      Each place is given by: (Number of times) at (Line):(Column).
      1 at 127:14
  NOTE: There were 1 observations read from the data set WORK.MYTABLE.
  NOTE: The data set WORK.MYTABLE2 has 1 observations and 1 variables.
  NOTE: DATA statement used (Total process time):
      real time          0.02 seconds
      cpu time           0.00 seconds
```

要点

- NOLIST DATA ステートメントのオプションは、_ERROR_の値が 1 の場合、SAS ログへのすべての変数の出力を抑制します。
- NOLIST は、DATA ステートメントの最後のオプションである必要があります。

関連項目

- “DATA ステートメント” (*SAS DATA ステップステートメント: リファレンス*)

例: NONOTES システムオプションを使用して SAS ログへのすべての注記の出力を抑制する

コード例

これらの例では、NONOTES システムオプションを使用して、SAS ログへの注記の出力を抑制します。

```
data example;
  password="ABC123";
run;
```

例のコード 27.6　SAS ログに注記がどのように出力されるかを示すログ出力

```
17   data example;
18     password="ABC123";
19   run;

NOTE: The data set WORK.EXAMPLE has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time          0.02 seconds
      cpu time           0.01 seconds
```

NONOTES システムオプションを使用すると、注記の出力を抑制できます。

```
options NONOTES;
data example;
  password="ABC123";
run;
```

例のコード 27.7　SAS ログで注記がどのように抑制されるかを示すログ出力

```
20   options NONOTES;
21   data example;
22     password="ABC123";
23   run;
```

要点

- NOTES システムオプションは、注記を SAS ログに書き込むかどうかを指定します。

- NONOTES は、SAS が SAS ログに注記を書き込まないことを指定します。

関連項目

- “NOTES システムオプション” (*SAS システムオプション: リファレンス*)

28

印刷

ユニバーサル印刷

ユニバーサル印刷の定義

ユニバーサル印刷は、さまざまなドキュメントおよびグラフィック出力形式を作成できるシステムです。たとえば、ユニバーサル印刷を使用して、HTML5 ファイルや PDF および RTF ドキュメントを作成できます。

ユニバーサル印刷を使用してプリンターを定義し、印刷出力を制御するオプションを設定します。さまざまなドキュメントおよびグラフィック出力タイプの作成に加えて、出力をプリンターに送信できます。

SAS は、すべての印刷をユニバーサル印刷サービスを通じてルーティングします。すべてのユニバーサル印刷機能はシステムオプションによって制御されるため、バッチモードでも多くの印刷機能を制御できます。

SAS Viya プラットフォームユニバーサル印刷.

6 部

ファイルの管理

29

ファイルの保護

パスワード

パスワードの種類

パスワードは、SAS 内の SAS データセットへのアクセスを制限しますが、SAS データセットが動作環境のシステムレベルで表示されるのを防ぐことはできません。パスワードは、外部プログラムによる SAS データセットの読み取りを防ぐことはできません。

SAS Compute Server ではパスワードを使用できますが、CAS では使用できません。

パスワードに関するいくつかの重要な概念を次に示します。

- パスワードは有効な SAS 名である必要があります。“SAS 名” (36 ページ)を参照してください。
- パスワード保護の 3 つのレベルは、読み取り、書き込み、および変更です。PW=パスワードは、3 つのレベルすべてに同じパスワードを割り当てます。
- パスワードが割り当てられると、ログ内に大文字の X として表示されます。1 つ以上のパスワードレベルをデータセットに割り当てる場合は、SAS ログから適切に消去されるように、各パスワードオプションを個別の行あるいは DATA または PROC ステートメントの一部にコーディングする必要があります。
- 既存のデータセットにパスワードを割り当てることも、データセットの作成時にパスワードを割り当てることもできます。
- SAS ビューに対するパスワードの動作は、SAS データセットに対する場合とは異なります。“ビュー付きのパスワード”を参照してください。
- SAS は、元の保護されたファイルに関連付けられた他のファイルにもパスワード保護を拡張します。これには、世代データセット、インデックス、監査証跡、コピーが含まれます。

> 重要　割り当てたパスワードは記録しておいてください。パスワードを忘れた場合、またはパスワードがわからない場合、SAS テクニカルサポートからパスワードを取得することはできません。

次の表に、データセットオプションとそれぞれの保護レベルを示します。

表 29.1 パスワードのデータセットオプション

データセットオプション	保護レベル	注
READ=	ファイルの読み取りから保護します。	READ=データセットオプションは、SAS 内の SAS データセットへのアクセスを制限します。 SAS がファイルを開く方法により、読み取りのみが保護されている SAS データセットを更新するには、読み取りパスワードを指定する必要があります。 変更または書き込みパスワードを知らなくても、変更保護または書き込み保護された SAS データセットを読み取ることができます。
WRITE=	オブザベーションの追加、変更、削除から保護します。	書き込み保護では、読み取りアクセスにパスワードは必要ありません。
ALTER=	ファイル全体の削除または置換、変数属性の変更、インデックスの作成または削除から保護します。	変更保護では、読み取りアクセスにパスワードは必要ありません。
PW=	各保護レベルに同じパスワードを割り当てます。	PW=データセットオプションを使用する場合、アクセス権を持つユーザーは、完全なアクセスのために 1 つのパスワードを覚えておくだけで済みます。
エンコードされたパスワード	プレーンテキストでパスワードを指定しなくても、SAS プログラムを作成できます。 エンコードされたパスワードは、悪意のないカジュアルなパスワードの閲覧を防ぐことを目的としています。	PWENCODE プロシジャを使用します。
消去されたパスワード	パスワードが SAS ログに表示されないようにします。	パスワードが SAS ログに表示されないようにするには、各パスワードをコード内の別の行に置く必要があります。

関連項目

- “例: パスワードの割り当てと使用” (741 ページ)
- “例: パスワードの消去またはエンコード” (751 ページ)

ビュー付きのパスワード

SAS ビューおよびストアドプログラムの保護レベルは、他のタイプの SAS ファイルの保護レベルと同様です。ただし、SAS ビューの場合、パスワードは基になるデータだけでなく、ビューの定義(またはソースステートメント)にも影響します。

SAS ビューには、読み取り、書き込み、および変更の 3 つの保護レベルを指定できます。これらのデータセットオプションは、ビューのディスクリプター情報だけでなく、基になるデータにも影響します。特に断りのない限り、"ビュー"という用語は任意のタイプの SAS ビューを指し、"基になるデータ"という用語は SAS ビューによってアクセスされるデータを指します。

ビューの読み取り、書き込み、および変更のパスワードは階層構造になっています。パスワードで保護されたビューを記述するには、そのパスワードを指定する必要があります。ビューが複数のパスワードを使用して作成された場合、ビューの DESCRIBE には最も制限の厳しいパスワードを使用する必要があります。

DATA および PROC のほとんどのステップでは、パスワードで保護されたビューの使用方法は、パスワードで保護された他のタイプの SAS ファイルの使用方法と一致します。

注: 基になるデータセットになんらかの種類の保護がすでに設定されている場合、SAS ビューに保護を設定すると、予期しない結果が発生する可能性があります。

表 29.2　ビュータイプ

ビューのタイプ	説明	注
DATA ステップビュー	パスワードで保護された SAS データセットを使用して DATA ステップビューを作成する場合は、ビュー定義でパスワードを指定します。このようにして、ビューを使用するときに、パスワードを再指定せずに基になるデータにアクセスできます。	“例: パスワードで保護されたデータセットから DATA ステップビューを作成する”を参照してください。
PROC SQL ビュー	通常、パスワードで保護された SAS データセットから PROC SQL ビューを作成する場合は、データセッ	“例: パスワードで保護されたデータセットからパスワードで保護された SQL ビ

ビューのタイプ	説明	注
	トオプションを使用して、CREATE VIEW ステートメントの FROM 句にパスワードを指定します。こうすることで、後でビューを使用するときにパスワードを再指定することなく、基になるデータにアクセスできます。	ューを作成する”を参照してください。 パスワードを指定せずに、パスワードで保護された SAS データセットから PROC SQL ビューを作成できます。 SAS をバッチモードまたは非対話モードで実行している場合は、エラーメッセージが表示されます。

関連項目

“例: パスワードの割り当てと使用” (741 ページ)

エンコードされたパスワード

エンコードされたパスワードは CAS では認識されませんが、SAS Compute Server では使用できます。

PWENCODE プロシジャは、エンコーディングを使用してパスワードを偽装します。エンコードでは、ある文字セットがなんらかの形式のテーブルルックアップを通じて別の文字セットに変換されます。エンコードされたパスワードは、悪意のないカジュアルなパスワードの閲覧を防ぐことを目的としています。ただし、データセキュリティのニーズすべてをエンコードされたパスワードに 依存すべきではありません。決意と知識のある攻撃者は、エンコードされたパスワードをデコードすることができます。

エンコードされたパスワードが使用される場合、構文パーサーはパスワードをデコードしてファイルにアクセスします。エンコードされたパスワードがプレーンテキストで SAS ログに書き込まれることはありません。SAS は 8 文字を超えるパスワードを受け入れません。エンコードされたパスワードがデコードされて 8 文字を超える場合、SAS はそれを間違ったパスワードとして読み取り、エラーメッセージを SAS ログに送信します。

関連項目

“例: PWENCODE=プロシジャを使用してパスワードをエンコードする” (753 ページ)

暗号化

SAS データセットの暗号化の種類

SAS は、データセットを暗号化するために 2 種類のアルゴリズムを提供します。

- SAS Proprietary 暗号化は、ENCRYPT=YES データセットオプションを使用して実装されます。
- AES (Advanced Encryption Standard)暗号化は、ENCRYPT=AES または ENCRYPT=AES2 を使用して実装されます。

暗号化に関する重要な概念をいくつか示します。

- ENCRYPT=データセットオプションは CAS では認識されませんが、SAS Compute Server では使用できます。
- 暗号化は、暗号化されたデータをディスクに書き込むことで、SAS の外部で SAS データのセキュリティを提供します。データはディスクから読み取られるときに SAS によって復号化されますが、オペレーティングシステムレベルまたは外部プログラムによって読み取られるときには復号化されません。
- 暗号化はファイルアクセスには影響しません。ただし、SAS はファイルアクセスを制御するすべてのホストセキュリティメカニズムを 尊重します。
- 変数レベルでのデータの暗号化については、“例: 変数値の暗号化” (161 ページ)を参照してください。
- SAS は、元の保護されたファイルに関連付けられた他のファイルにも暗号化を拡張します。これには、世代データセット、インデックス、監査証跡、コピーが含まれます。
- 暗号化された SAS データセットでは PROC CPORT を使用できません。
- SAS ビューにはデータが含まれていないため、暗号化できません。
- 暗号化には、圧縮とほぼ同じ量の CPU リソースが必要です。
- SAS Proprietary または AES 暗号化のために追加のインストールは必要ありません。

SAS Proprietary 暗号化

データセットの SAS Proprietary 暗号化には次のルールが適用されます。

- SAS データセットの作成時に ENCRYPT=データセットオプション を指定して暗号化を割り当てます。

- SAS Proprietary 暗号化を使用してデータセットを暗号化する場合も、パスワードを割り当てる必要があります。少なくとも、ENCRYPT=YES を指定するのと同時に、READ=または PW=データセットオプションを指定する必要があります。
- この暗号化手法ではパスワードが使用されるため、データセットを再作成しない限り、暗号化されたデータセットのパスワードを変更することはできません。

関連項目

- “例: データセットの暗号化” (749 ページ)
- “セキュリティ” (*SAS Cloud Analytic Servcies: 基本*)

AES 暗号化

データセットを作成するときに、ENCRYPT=AES または ENCRYPT=AES2 を指定します。AES は、最大 64 文字の長さのキー値を使用して強力な暗号化を生成します。AES および AES2 は、SAS Proprietary 暗号化よりも強力な暗号化を提供します。AES2 は、AES よりも強力なキー生成アルゴリズムを提供します。

SAS Proprietary 暗号化のようにデータセットに保存されるパスワードのかわりに、AES および AES2 は、データセットに保存されないキー値を使用します。キー値は、データセットの作成時に ENCRYPTKEY=データセットオプションを使用して作成されます。AES 暗号化データセットの ENCRYPTKEY=キー値を変更するには、データセットを再作成する必要があります。

データセットの AES および AES2 暗号化には、次のルールが適用されます。

- AES 暗号化データセットを作成またはアクセスする場合は、ENCRYPTKEY=データセットオプションを使用する必要があります。オプションで、ALTER=または PW=データセットオプションを指定して、SAS でデータセットが削除または置換されないように保護することもできます。
- AES 暗号化データセットをコピーするには、出力エンジンが AES 暗号化をサポートしている必要があります。それ以外の場合、データセットはコピーされません。
- 2 つ以上のデータセットが参照的に関連しており、それらのいずれかが AES 暗号化されている場合は、すべてのデータセットが AES 暗号化されている必要があります。すべてのファイルの暗号化キーは同じである必要があります。
- データセットに AES 暗号化がある場合、関連するすべてのインデックスも AES 暗号化されます。

ENCRYPTKEY=データセットオプションは、AES 暗号化ファイルを削除または置換から保護しません。AES 暗号化データセットは、ENCRYPTKEY=値を指定せずに、次のいずれかのシナリオを使用して削除できます。

- PROC DATASETS の KILL オプション
- PROC DATASETS の DELETE ステートメント
- PROC DELETE

- PROC SQL の DROP ステートメント

暗号化キーにより、ファイルの内容へのアクセスのみが防止されます。SAS による不正な削除または置換からファイルを保護するには、ファイルに ALTER=または PW=パスワードも含まれている必要があります。

関連項目

- “例: データセットの暗号化” (749 ページ)
- “セキュリティ” (*SAS Cloud Analytic Servcies: 基本*)

ログからパスワードと暗号化キーの値を消去する

パスワードと暗号化キーを消去するための重要な概念は次のとおりです。

- **SAS ログを確認する**: SAS ログを確認して、パスワード値または暗号化キー値が消去されていることを確認する必要があります。これは、READ=、WRITE=、ALTER=、PW=、ENCRYPTKEY=データセットオプションに適用されます。
- **マクロ内のパスワード**: マクロ内でパスワードが割り当てられている場合、マクロの実行時にパスワードが SAS ログ内で消去されません。SAS ログでパスワードが公開されるのを防ぐために、SAS ログをファイルにリダイレクトできます。詳細については、“PRINTTO プロシジャ” (*Base SAS プロシジャガイド*)を参照してください。
- **パスワードの長さ**: 場合によっては、表示されるパスワードの長さが消去された 8 文字に固定されます。それ以外の場合は、消去された文字の数がパスワードの長さになります。OPTIONS プロシジャからの出力は一例です。

関連項目

“例: パスワードの消去またはエンコード” (751 ページ)

例: パスワードの割り当てと使用

例: DATA ステップでのパスワードの割り当て

コード例

このコードは、書き込みパスワードと変更パスワードを使用したデータセットの削除または変更を防ぎます。

注意
割り当てたパスワードは記録しておいてください。 パスワードを忘れた場合、またはパスワードがわからない場合、SAS テクニカルサポートからパスワードを取得することはできません。

```
libname mydata '/home/userid/mydata/';
data mydata.students(write=yellow alter=red);
  input name $ sex $ age;
  datalines;
Amy f 25
Sue f 23
Padma f 21
Peter m 25
Rick m 22
;
run;
```

パスワードは DATA ステートメント内にあるため、SAS ログでは消去されます。

例のコード 29.1　SAS ログ出力

```
83   data mydata.students(write=XXXXXX alter=XXX);
84   input name $ sex $ age;
85   datalines;
NOTE: The data set MYDATA.STUDENTS has 5 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds

86   ;
87   run;
```

要点

- ログを確認して、パスワードが消去されていることを確認します。
- 変更パスワードは、ファイル全体の削除または置換、変数属性の変更、インデックスの作成または削除から保護します。変更パスワードはデータの読み取りを保護しません。
- 書き込みパスワードは、オブザベーションの追加、変更、削除から保護します。書き込みパスワードはデータの読み取りを保護しません。

関連項目

- “ALTER= データセットオプション” (*SAS データセットオプション: リファレンス*)
- “WRITE= データセットオプション” (*SAS データセットオプション: リファレンス*)

例: PW=データセットオプションを使用して完全な保護を割り当てる

コード例

この例では、DATA ステップでデータセットを作成し、PW=データセットオプションを使用してパスワードを割り当てます。次に、DATASETS プロシジャは、ALTER=データセットオプションを使用してデータセットを削除します。PW=オプションは、データセットの削除にも適しています。

```
data mydata.college(pw=red);
  input name $ 1-10 location $ 12-25;
  datalines;
Vanderbilt Nashville
Rice      Houston
Duke      Durham
Tulane    New Orleans
;

proc datasets library=mydata;
  delete college(alter=red);
run;
quit;
```

例のコード 29.2　SAS ログ出力

```
12   data mydata.college(pw=XXX);
13     input name $ 1-10 location $ 12-25;
14     datalines;

NOTE: The data set MYDATA.COLLEGE has 4 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds


19   ;
20
21   proc datasets library=mydata;
NOTE: Writing HTML Body file: sashtml.htm
22     delete college(alter=XXX);
23   run;

NOTE: Deleting MYDATA.COLLEGE (memtype=DATA).
24   quit;
```

要点

表 29.3　要点

- データセットが削除または置換されないように保護するには、PW=または ALTER=データセットオプションを割り当てます。
- PW=オプションは、ALTER=オプションよりも強力な保護を提供します。PW=オプションは、ALTER=、READ=、および WRITE=オプションを割り当てる場合と同じ保護を提供します。
- PW=オプションを使用してデータセットを作成すると、読み取り、書き込み、および変更に同じパスワードが割り当てられます。PW=オプションを割り当てた後、そのパスワードを ALTER=、READ=、または WRITE=オプションとともに使用できます。PW=データセットオプションを使用する場合、ユーザーは完全なアクセスのために 1 つのパスワードを覚えておくだけで済みます。

関連項目

- “ALTER= データセットオプション” (*SAS データセットオプション: リファレンス*)
- “PW= データセットオプション” (*SAS データセットオプション: リファレンス*)

例: 既存のデータセットにパスワードを割り当てる

コード例

この例では、DATA ステップによって、seminar という名前のデータセットが作成されます。次に、PROC DATASETS によってパスワードが割り当てられます。

```
data mydata.seminar;
  set sashelp.class;
run;
proc datasets library=mydata;
  modify seminar(pw=orange);
run;
quit;
```

例のコード 29.3　SAS ログ出力

```
25   data mydata.seminar;
26     set sashelp.class;
27   run;

NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set MYDATA.SEMINAR has 19 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time          0.02 seconds
      cpu time           0.00 seconds

28   proc datasets library=mydata;
29     modify seminar(pw=XXXXXX);
30   run;

NOTE: MODIFY was successful for MYDATA.SEMINAR.DATA.
31   quit;
```

要点

- PROC DATASETS の MODIFY ステートメントを使用して、既存のデータセットにパスワードを割り当てることができます。

関連項目

- “ALTER= データセットオプション” (*SAS データセットオプション: リファレンス*)

- “MODIFY ステートメント” (*Base SAS プロシジャガイド*)

例: プロシジャでのパスワードの割り当て

コード例

この例では、書き込みパスワードと変更パスワードを含む、tutorial という名前のデータセットを作成します。

```
proc sort data=mydata.seminar(pw=orange)
  out=mydata.tutorial(write=yellow alter=red);
  by age;
run;
```

例のコード 29.4　SAS ログ出力

```
36   proc sort data=mydata.seminar(pw=XXXXXX)
37     out=mydata.tutorial(write=XXXXXX alter=XXX);
38     by age;
39   run;

NOTE: There were 19 observations read from the data set MYDATA.SEMINAR.
NOTE: The data set MYDATA.TUTORIAL has 19 observations and 5 variables.
```

要点

- ほとんどの SAS プロシジャを使用するときにパスワードを割り当てることができます。

関連項目

- “SORT プロシジャ” (*Base SAS プロシジャガイド*)

例: パスワードで保護されたデータセットから DATA ステップビューを作成する

コード例

次のステートメントは、PW=パスワードを持つデータセットから project という名前の DATA ステップビューを作成します。DROP=オプションは、ビューから変数を削除します。ビューはパスワードで保護されていないため、PRINT プロシジャコードにはパスワードは必要ありません。

```
libname mydata '/home/userid/mydata/';
data mydata.project / view=mydata.project;
  set mydata.seminar(pw=orange drop=height);
run;
proc print data=mydata.project;
run;
```

要点

- 基になる SAS データセットを変更せずに、SAS ビューから変数を削除できます。
- パスワードで保護された SAS データセットの SAS ビューを作成すると、パスワードがビューに保存されます。ビューにパスワードを割り当てない限り、ビューはパスワードで保護されません。

関連項目

- “SAS Views” (*SAS V9 LIBNAME Engine: Reference*)
- “DROP= データセットオプション” (*SAS データセットオプション: リファレンス*)

例: パスワードで保護されたデータセットからパスワードで保護された SQL ビューを作成する

コード例

次のステートメントは、PW=パスワードを持つデータセットから test という名前の PROC SQL ビューを作成します。PROC PRINT はビューのパスワードを指定します。

```
libname mydata '/home/userid/mydata/';
proc sql;
  create view mydata.test(pw=blue) as
    select * from mydata.seminar(pw=orange);
quit;
proc print data=mydata.test(pw=blue);
run;
```

要点

- パスワードで保護された SAS データセットから SAS ビューを作成すると、パスワードがビューに保存されます。ビューにパスワードを割り当てない限り、ビューはパスワードで保護されません。

関連項目

- “SAS Views” (*SAS V9 LIBNAME Engine: Reference*)

例: パスワードで保護されたビューで DESCRIBE ステートメントを使用する

コード例

複数の種類のパスワードを持つビューの DESCRIBE には、最も制限の厳しいパスワードを指定します。次のプログラムでは、最初の DATA ステップで、読み取りパスワードと変更パスワードを持つデータビューを作成します。変更パスワードは読み取りパスワードより制限が厳しいため、2 番目の DATA ステップには ALTER=データセットオプションを指定する DESCRIBE ステートメントがあります。

```
data exam / view=exam(read=rose alter=violet);
  set sashelp.class;
run;

data view=exam(alter=violet);
  describe;
run;
```

例のコード 29.5　SAS ログ出力

```
10   data exam / view=exam(read=XXXX alter=XXXXXX);
11     set sashelp.class;
12   run;

NOTE: DATA STEP view saved on file WORK.EXAM.
NOTE: A stored DATA STEP view cannot run under a different operating system.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds

13
14   data view=exam(alter=XXXXXX);
15     describe;
16   run;

NOTE: DATA step view WORK.EXAM is defined as:

data exam / view=exam(read=XXXX alter=XXXXXX);
  set sashelp.class
```

要点

- SAS ビューまたはストアドコンパイル DATA ステッププログラムには、複数の種類のパスワードを設定できます。DESCRIBE ステートメントを使用するには、最も制限の厳しいパスワードを指定する必要があります。

たとえば、読み取り保護と書き込み保護の両方が設定されているビューの場合は、その書き込みパスワードを指定します。読み取り保護と変更保護の両方が設定されているビューの場合は、その変更パスワードを指定します。変更は読み取りよりも制限が厳しくなります。

関連項目

- "SAS Views" (*SAS V9 LIBNAME Engine: Reference*)
- "DESCRIBE ステートメント" (*SAS DATA ステップステートメント: リファレンス*)

例: データセットの暗号化

例: SAS Proprietary 暗号化を使用したデータセットの作成

コード例

次の例では、salary という名前の SAS データセットを作成します。ENCRYPT=YES データセットオプションは、SAS Proprietary アルゴリズムを使用してファイルを暗号化します。READ=オプションは、読み取りアクセス用のパスワード green を割り当てます。

読み取りパスワードは、データセットの削除または置換を保護しません。このコードを再度実行して salary を上書きすることができます。SAS でデータセットが削除または上書きされないように保護するには、ALTER=または PW=データセットオプションを使用します。

```
data salary(encrypt=yes read=green);
  input name $ yrsal bonuspct;
  datalines;
Muriel   34567  3.2
Bjorn    74644  2.5
Freda    38755  4.1
Benny    29855  3.5
Agnetha  70998  4.1
;
run;
```

このデータセットを出力するには、パスワードを指定します。

```
proc print data=salary(read=green);
run;
```

要点

- SAS Proprietary アルゴリズムを使用してデータセットを暗号化するには、ENCRYPT=YES を指定します。この暗号化では、データセットに保存されているパスワードが使用されます。
- 少なくとも、ENCRYPT=YES を指定するのと同時に、READ=データセットオプションまたは PW=データセットオプションを指定する必要があります。暗号化法ではパスワードが使用されるため、データセットを再作成しない限り、暗号化されたデータセットのパスワードを変更することはできません。
- SAS でデータセットが削除または上書きされないように保護するには、ALTER=または PW=データセットオプションを使用します。

関連項目

- *“ENCRYPT= データセットオプション” (SAS データセットオプション: リファレンス)*
- *“セキュリティ” (SAS Cloud Analytic Servcies: 基本)*

例: AES 暗号化を使用したデータセットの作成

コード例

次の例では、salary という名前のデータセットを作成します。ENCRYPT=AES2 データセットオプションは、AES (Advanced Encryption Standard)アルゴリズムを使用してファイルを暗号化します。ENCRYPTKEY=オプションは、暗号化キー green を割り当てます。

暗号化は、データセットの削除または置換を保護しません。このコードを再度実行して salary を上書きすることができます。SAS でデータセットが削除または上書きされないように保護するには、ALTER=または PW=データセットオプションを追加します。

```
data salary(encrypt=aes2 encryptkey=green);
  input name $ yrsal bonuspct;
  datalines;
Muriel   34567  3.2
Bjorn    74644  2.5
```

```
Freda    38755  4.1
Benny    29855  3.5
Agnetha  70998  4.1
;
```

このデータセットを出力するには、ENCRYPTKEY=値を指定します。

```
proc print data=salary(encryptkey=green);
run;
```

要点

- AES 暗号化データセットを作成およびアクセスする場合は、ENCRYPTKEY=データセットオプションが必要です。
- 暗号化キーにより、ファイルの内容へのアクセスが防止されます。SAS による不正な削除または置換からファイルを保護するには、ALTER=または PW=パスワードを割り当てます。

例: パスワードの消去またはエンコード

例: 消去時のエラーを回避する

コード例

次の例では、パスワード値と暗号化キーの値は SAS ログで消去されません。

- DATA ステートメントでは、ライブラリ参照名またはデータセット名にマクロ変数を使用しないでください。

```
%let ds=dataset;

data &ds(alter=secret);
  x=1
;
run;
```

次のように SAS ログに書き込まれます。

```
111  %let ds=dataset;
112  data &ds(alter=secret);
113    x=1;
114  run;
```

- 特定のプロシジャでデータセットのパスワードが正しくないと、SAS ログにパスワードが記録されます。たとえば、次のコードでは 2 つのパスワードが使用されており、1 つは間違って入力されています。

```
proc append base=here(pw=scret1) data=more(read=secret2);
run;
```

次の出力が SAS ログに書き込まれます。

```
160
161  proc append base=here(PW=XXXXX) data=more(READ=secret2);
ERROR: Invalid or missing WRITE password on member WORK.HERE.DATA.
162  run;
```

- コードによってエラーメッセージが表示された場合、パスワードは消去されません。たとえば、次のコードでは、ライブラリ参照名のスペルが間違っているため、SAS はエラーメッセージを発行しますが、パスワードは消去されません。

```
libname mylib '/home/userid/data/';
  data mylub.abc(
  read=secret
  );
  x=1;
  run;
```

次の出力が SAS ログに書き込まれます。

```
636  libname mylib '/home/userid/data';
NOTE: Libref MYLIB was successfully assigned as follows:
     Engine:       V9
     Physical Name: home/userid/data
637     data mylub.abc(
638     read=secret
639     );
ERROR: Libref MYLUB is not assigned.
640     x=1;
641     run;
```

- パスワード値が報告される場合、その長さは 8 に固定されます。ただし、パスワード値がサブミットされたステートメントからエコーされると、その長さは保持されます。この例は、パスワードの長さを示しています。

```
options pdfpassword=(open=a owner=dog);
proc options option=pdfpassword;
run;
```

次のように SAS ログに書き込まれます。最初の行で、パスワードの長さを確認できます。

```
166  options pdfpassword=(open=X owner=XXX);
167  proc options option=pdfpassword;
168  run;

   SAS (r) Proprietary Software

 PDFPASSWORD=(OPEN = XXXXXXXX OWNER = XXXXXXXX )
 Specifies the password to use to open a PDF document and the password
 used by a PDF document owner.
```

要点

- DATA ステートメントでは、ライブラリ参照名またはデータセット名にマクロ変数を使用しないでください。
- 入力ミスにより、SAS ログのエラーメッセージにパスワードが表示される場合があります。
- サブミットされたステートメントからパスワード値がエコーされると、消去された値の長さは保持されます。

関連項目

- *SAS 暗号化*
- *“セキュリティ” (SAS Cloud Analytic Servcies: 基本)*

例: PWENCODE=プロシジャを使用してパスワードをエンコードする

コード例

この例では、PWENCODE プロシジャを使用してパスワードをエンコードします。

```
proc pwencode in='my-password' method=sas005;
run;
```

SAS ログでは、パスワードの各文字が X に置き換えられることに注意してください。

例のコード 29.6　SAS ログ出力

```
19   proc pwencode in=XXXXXXXXXXXXX method=sas005;
20   run;

{SAS005}1CF7A13AE8BFCD5DB41E4E76B1047B480C649D0A0F61DF40
```

要点

- PWENCODE=プロシジャを使用すると、SAS ログ内で IN=パスワードが X 文字で消去されます。
- エンコードされたパスワードを保存する 1 つの方法は、OUT=引数にファイル参照名を指定することです。もう 1 つの方法は、エンコードされたパスワードをログから手動でコピーし、マクロ変数として指定することです。

関連項目

- “PWENCODE プロシジャ” (*Base SAS プロシジャガイド*)

30

SAS ファイルの修復

SAS データセットと SAS カタログの破損の修復

概要

SAS は、特定のシステムエラーに起因する SAS データセットおよび SAS カタログの構造的破損を修復できます。また、オペレーティングシステムコマンドを使用して誤って削除または破損した単一インデックスと一貫性制約を再構築することもできます。これらの修復は、DLDMGACTION=システムオプション、DLDMGACTION=データセットオプション、または PROC DATASETS REPAIR ステートメントを使用して実行できます。SAS が修復できる破損とその方法の詳細については、“SAS で修復できる破損について理解する” (757 ページ)を参照してください。

SAS では次の修復はできません。

- 切り捨てられた SAS データセット

- 監査ファイルが破損しているか削除されているデータセット
- オペレーティングシステムツールを使用してコピーされた一貫性制約を含むデータセット
- 破損した記憶装置上にあるデータセットとカタログ
- SAS ビュー
- ストアドコンパイル DATA ステッププログラム
- 複合インデックス
- SORT プロシジャで FORCE オプションを使用したときに削除されたインデックス。FORCE オプションは、元のファイルを上書きします。

このような状況では、次の表で推奨されているアクションを実行してください。

表 30.1　ファイルまたは破損のタイプ別の推奨アクション

SAS ファイルタイプまたは破損タイプ	対策
DATA ステップビューと PROC SQL ビュー	ファイルを再作成します。
ストアドコンパイル DATA ステッププログラム	
オペレーティングシステムツールを使用してコピーされた一貫性制約を含むデータセット	“Operations That Affect Integrity Constraints” (*SAS V9 LIBNAME Engine: Reference*)の手順に従います。
切り捨てられた SAS データセット、または監査ファイルが破損または削除された SAS データセット	バックアップデバイスから回復します。[1]
破損した記憶装置上にある SAS データセットとカタログ	
DLDMGACTION=システムオプションやデータセットオプション、または PROC DATASETS REPAIR ステートメントを使用して修復できないカタログ	
複合インデックス	インデックスのないデータセットを修復し、PROC DATASETS の“INDEX CREATE ステートメント” (*Base SAS プロシジャガイド*)を使用してインデックスを再作成します。または、バックアップデバイスからデータセットを回復します。
SORT プロシジャで FORCE オプションを使用したときに削除されたインデックス	

1 監査ファイルが破損した場合、または誤って削除された場合は、データセット、そのインデックス、および監査ファイルを回復します。

SAS で修復できる破損について理解する

SAS は、次のいずれかのイベントが発生したときに、SAS データセット(インデックスと一貫性制約を含む)および SAS カタログを回復および修復できます。

- データセットまたはカタログの更新中にシステム障害が発生する。
- データセット(インデックスファイルを含む)またはカタログが保存されているディスクが、ファイルへの書き込みが完了する前にいっぱいになる。
- データセット、インデックスファイル、またはカタログへの書き込み中に入出力エラーが発生する。
- オペレーティングシステムのコマンドを使用してインデックスが誤って削除されてしまう。
- オペレーティングシステムコマンドを 使用してデータセットをコピーまたは名前変更したときに、データセットが破損したが、関連するインデックスは破損していない。

SAS は、データセットをコピーして再作成することにより、データセットの構造を修復します。データセットのインデックスと一貫性制約は、データセットヘッダーのメタデータから再構築されます。SAS はデータセット内のデータを復元しようとしますが、データが書き込まれる前にディスクがいっぱいになった場合、システム障害が発生する前に行われた最後のいくつかの更新がデータセットに含まれていない可能性があります。この問題が発生した場合は、バックアップデバイスからデータセットを回復することをお勧めします。

破損したデータセットは、BASE エンジンがデータセットの破損を検出した開始時間に修復されます。データセットが修復された後でのみ、データとそのインデックスをコピーできます。

SAS は、破損したカタログを次のように修復します。SAS はカタログをチェックして、どのエントリが破損しているかを確認します。エントリの読み取り中にエラーが発生した場合、エントリはコピーされます。コピー処理中にエラーが発生した場合、エントリは自動的に削除されます。重大な破損がある場合は、カタログ全体が新しいカタログにコピーされます。

修復ツールについて理解する

DLDMGACTION=システムオプションを使用すると、データセットまたはカタログへの破損が検出された場合に、プロアクティブな 修復応答をプログラムできます。DLDMGACTION=システムオプションを設定して、SAS セッション内のデータセットおよびカタログの破損ケースの大部分に適用するアクションを実行します。たとえば、対話型セッションの場合、DLDMGACTION=システムオプションを PROMPT に設定すると、その場で修復オプションのメニューが表示されます。または、システムオプションを FAIL に設定し、DLDMGACTION=データセットオプションと PROC DATASETS REPAIR ステートメントを使用してすべての 修復を実行することもできます。

DLDMGACTION=NOINDEX および REPAIR オプションは、実際の問題を確認し、それを修正しようとしている場合にのみ使用してください。DLDMGACTION=REPAIR は、データセットとそのすべてのインデックスを 可能な限り修復します。DLDMGACTION=REPAIR がシステムオプションとして設定されている場合、破損したデータセットとカタログは、次回データセットまたはカタログにアクセスしたときに自動的に修復されます。これにより、データセットで失われたオブザベーションを調べることができなくなる可能性があります。DLDMGACTION=REPAIR も、通常のデータセットと世代データセットを区別しません。世代データセットの修復は難しい場合があります。自動修復は適切ではありません。世代データセットを修復する方法については、“Generation Data Sets” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。

DLDMGACTION=データセットオプションと PROC DATASETS REPAIR ステートメントの違いは次のとおりです。

- PROC DATASETS REPAIR ステートメントは、1 つまたは複数のデータセットまたはカタログを一度に修復できます。MEMTYPE=オプションを使用すると、修復を DATA または CATALOG タイプのファイルのみに制限できます。DLDMGACTION=データセットオプションは、単一のデータセットまたはカタログに対して動作します。
- DLDMGACTION=データセットオプションを使用すると、NOINDEX オプションを指定することで、インデックスおよび一貫性制約の有無にかかわらず、データセットを修復できます。PROC DATASETS REPAIR ステートメントは常にすべてのインデックスを修復します。NOINDEX オプションは、インデックスと一貫性制約のないデータセットを自動的に修復し、インデックスファイルを削除し、無効になったインデックスと一貫性制約を反映するようにデータセットを更新し、INPUT モードで開かれるデータセットを制限します。無効化されたインデックスと一貫性制約を修正または削除するために *Base SAS プロシジャガイド*の "REBUILD ステートメント"を実行するよう指示する警告が SAS ログに書き込まれます。

SAS アプリケーションの中には、デフォルトの DLDMGACTION=システムオプション設定が出荷されているものもあれば、そうでないものもあります。DLDMGACTION=システムオプション設定を調べて、適切な値に設定されていることを確認することをお勧めします。

SAS データセットごとに、データセットが修復された回数と最後の修復日を示す破損ログが維持されます。PROC CONTENTS で修復ログを表示できます。“例: データセットが修復された回数を確認する” (766 ページ)を参照してください。

カタログには破損ログはありません。

SAS で SAS データセットとカタログを修復するための推奨タスク

表 30.2　SAS でファイルを修復するためのタスクとツール

タスク	コード	説明	例
SAS セッションの DLDMGACTION=システムオプション設定を決定する	proc options option=dldmgaction value; run;	DLDMGACTION=システムオプションの設定を SAS ログに出力します。	“例: SAS セッションのデフォルトの修復アクションを決定する” (760 ページ)
破損したファイルを検出したときの SAS の応答方法を変更する(オプション)	options dldmgaction=*value*;	SAS セッションの修復応答を変更または構成できます。	
破損したファイルを自分で積極的に修復する	(dldmgaction=*value*)	特定の SAS データセットまたはカタログの修復アクションを指定できます。	“例: DLDMGACTION=データセットオプションを使用してデフォルトの修復アクションをオーバーライドする” (764 ページ)
	proc datasets; repair *filename*; run;	破損した SAS データセットまたはカタログを修復しようとします。	“例: REPAIR ステートメントを使用して SAS データセットを修復する” (762 ページ)
	proc datasets; rebuild *filename*; run;	DLDMGACTION=NOINDEX オプションによって無効にされたインデックスおよび一貫性制約を復元するか削除するかを指定します。	“例: REBUILD ステートメントを使用して破損した SAS データセットを再構築する” (765 ページ)
データセットの修復履歴を表示する	proc contents data=*filename*; run;	SAS データセットに関する情報をリストします。出力のエンジン/ホスト関連情報セクションの"データセットの修復数"と"最終修復"エントリを確認します。	“例: データセットが修復された回数を確認する” (766 ページ)

例 :ファイルの修復

例: SAS セッションのデフォルトの修復アクションを決定する

コード例

DLDMGACTION=システムオプションは、SAS セッションのデフォルトの修復アクションを制御します。DLDMGACTION=システムオプションの設定は、PROC OPTIONS を使用して決定できます。

```
proc options option=dldmgaction value;
run;
```

SAS は、次のような情報をログに返します。

例のコード 30.1　OPTIONS プロシジャからのログ出力

```
Option Value Information For SAS Option DLDMGACTION
   Value: REPAIR
   Scope: Default
   How option value set: Shipped Default
```

要点

- 管理者が構成ファイルで別の値を構成していない限り、ほとんどの SAS セッションは値 DLDMGACTION=REPAIR を返します。他の DLDMGACTION=値については、*“DLDMGACTION= システムオプション” (SAS システムオプション: リファレンス)* を参照してください。
- DLDMGACTION=システムオプションの設定を変更することで、デフォルトの修復応答を変更できます。または、DLDMGACTION=データセットオプションを使用して、特定のデータセットおよびカタログのシステムオプション設定をオーバーライドすることもできます
- DLDMGACTION=REPAIR では修復できないアイテムが多数あります。修復できるものとできないものについては、“SAS で修復できる破損について理解する” (757 ページ)を参照してください。DLDMGACTION=システムオプションは、現在のバックアップのかわりにはなりません。

関連項目

- “OPTIONS プロシジャ”
- “DLDMGACTION= システムオプション” (*SAS システムオプション: リファレンス*)
- “DLDMGACTION= データセットオプション” (*SAS データセットオプション: リファレンス*)
- “例: REPAIR ステートメントを使用して SAS データセットを修復する” (762 ページ).
- “例: DLDMGACTION=データセットオプションを使用してデフォルトの修復アクションをオーバーライドする” (764 ページ).

例: DLDMGACTION=システムオプションの変更

コード例

次のコードは、DLDMGACTION=システムオプションを PROMPT に設定し、PROC OPTIONS で設定を確認します。

```
options dldmgaction=prompt;

proc options option=dldmgaction value;
run;
```

アウトプット 30.1　PROC OPTIONS からのログ出力

```
73         options dldmgaction=prompt;
74
75         proc options option=dldmgaction value;
76          run;

    SAS (r) Proprietary Software Release 9.4  TS1M6

Option Value Information For SAS Option DLDMGACTION
    Value: PROMPT
    Scope: Default
    How option value set: OPTIONS Statement
```

要点

- OPTIONS ステートメントは、現在の SAS セッション内のすべての対話の修復設定を変更します。システムオプションは、SASV9_OPTIONS 環境変数でも指定できます。
- PROMPT 設定は、特定の SAS データセットまたはカタログに対してユーザーが修復を決定できるダイアログボックスを表示するように SAS に指示します。
- 特定の SAS データセットまたはカタログのデフォルト設定をオーバーライドするには、DLDMGACTION=データセットオプションを使用します。

関連項目

- "OPTIONS ステートメント" (*SAS グローバルステートメント: リファレンス*)
- "DLDMGACTION= システムオプション" (*SAS システムオプション: リファレンス*)

例: REPAIR ステートメントを使用して SAS データセットを修復する

コード例

次のコードは、暗号化された mylib.myfile という名前の破損 SAS データセットを修復することを指定します。PROC DATASETS REPAIR ステートメントは、データセットを修復するために使用されます。

```
libname mylib "C:\dldmgactiontest";

proc datasets lib=mylib;
  repair myfile /encryptkey=secret;
run;
quit;
```

プロセスが完了すると、次のメッセージがログに書き込まれます。

例のコード 30.2 PROC DATASETS REPAIR ステートメントからのログメッセージ

```
NOTE: Repairing MYLIB.MYFILE (memtype=DATA).
NOTE: File MYLIB.MYFILE.DATA is damaged.
NOTE: Data set MYLIB.MYFILE contained no structural errors, however some
changes during last update may not have been written to disk.
NOTE: Indexes recreated:
 1 Simple indexes
```

要点

- PROC DATASETS REPAIR ステートメントの機能は、DLDMGACTION=REPAIR ステートメントの機能と似ています。このステートメントは、破損した SAS データセットとカタログを使用可能な状態に復元しようとします。データセットの単一インデックスと一貫性制約を修復します。
- REPAIR ステートメントを使用すると、DLDMGACTION=システムオプションをサポートしない SAS プログラミング言語で使用される SAS データセットを修復できます。
- DLDMGACTION=REPAIR と同様、REPAIR ステートメントは現在のバックアップのかわりにはなりません。データセットに大きな破損がある場合、REPAIR ステートメントでは修正されません。システムバックアップからデータセットを回復する必要がある場合があります。
- 破損したカタログを修復するには、カタログが作成された環境と同じホスト動作環境で実行されるバージョンの SAS を使用する必要があります。CEDA によるカタログへのアクセスはありません。

関連項目

- “REPAIR ステートメント” (*Base SAS プロシジャガイド*).

例: DLDMGACTION=データセットオプションを使用してデフォルトの修復アクションをオーバーライドする

コード例

次の要求は、破損したデータセット mylib.myfile に対する要求に対して DLDMGACTION=データセットオプションを設定します。この例では、SAS セッションのデフォルトのアクションは DLDMGACTION=REPAIR です。データセットオプションは、DLDMGACTION=データセットオプションを使用して、データセットに対する PROC PRINT 要求の NOINDEX オプションを指定します。

```
proc print data=mylib.myfile(dldmgaction=noindex);
run;
```

SAS は、次のような情報をログに返します。

例のコード 30.3　DLDMGACTION=NOINDEX オプションを使用して修復されたデータセットに対して返されるログメッセージ

```
100  proc print data=mylib.myfile(dldmgaction=noindex);
NOTE: File MYLIB.MYFILE.DATA is damaged.
NOTE: Data set MYLIB.MYFILE contained no structural errors, however some changes during last
      update may not have been written to disk.
WARNING: SAS data file MYLIB.MYFILE.DATA was damaged and has been partially repaired.  To
         complete the repair, execute the DATASETS procedure REBUILD statement.
101  run;
```

要点

- DLDMGACTION=データセットオプションは、既存のデータセットに指定する必要があります。このオプションは、SET ステートメントを除き、データセットを作成するステートメントでは無効です。
- NOINDEX オプションは、インデックスと一貫性制約を使用せずに、破損したデータセットを修復することを指定します。この修復では、インデックスファイルも削除され、無効になったインデックスと一貫性制約を反映するようにデータセットが更新されます。データセットが破損していない場合、データセットオプションは効果がありません。
- このデータセットは破損しているため、PROC DATASETS REBUILD ステートメントを実行して、無効になっているインデックスと一貫性制約を修正または削除するように指示する警告が SAS ログに書き込まれます。変更を加えるまで、データセットは入力モードでのみ開くことができます。

- DLDMGACTION=データセットオプションを SET ステートメントで使用すると、新しいデータセットからインデックスが単純に省略されます。

関連項目

- “DLDMGACTION= データセットオプション” (*SAS データセットオプション: リファレンス*)
- “例: REBUILD ステートメントを使用して破損した SAS データセットを再構築する” (765 ページ)

例: REBUILD ステートメントを使用して破損した SAS データセットを再構築する

コード例

次のコードは、DLDMGACTION=NOINDEX データセットオプションによって無効にされたデータセットのインデックスと一貫性制約を再構築します。この要求により、“例: DLDMGACTION=データセットオプションを使用してデフォルトの修復アクションをオーバーライドする” (764 ページ)。で削除されたインデックスと一貫性制約が再構築されます。

```
proc datasets library=mylib;
rebuild myfile;
run;
quit;
```

SAS は、次のような情報をログに返します。

例のコード 30.4　PROC DATASETS REBUILD ステートメントによって書き込まれるログメッセージ

```
NOTE: Rebuilding MYLIB.MYFILE (memtype=DATA).
NOTE: Indexes recreated:
 2 Simple indexes
```

要点

- PROC DATASETS REBUILD ステートメントを使用すると、DLDMGACTION=NOINDEX オプションで無効にされたインデックスおよび一貫性制約を復元するか削除するかを指定できます。

- REBUILD ステートメントは、デフォルトで無効なインデックスと一貫性制約を復元します(ここに示されています)。データセットからインデックスと一貫性制約を削除するには、NOINDEX オプションを指定します。

関連項目

- “PROC DATASETS REBUILD ステートメント”
- “例: データセットが修復された回数を確認する” (766 ページ).

例: データセットが修復された回数を確認する

コード例

データセットが修復された回数を確認するには、CONTENTS プロシジャを使用します。この例は、データセット mylib.myfile が修復された回数を示しています。

```
proc contents data=mylib.myfile;
run;
```

修復の回数は、CONTENTS プロシジャ出力のエンジン/ホスト関連情報セクションに報告されます。

アウトプット 30.2　CONTENTS プロシジャ出力のエンジン/ホスト関連情報セクション

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	2
First Data Page	1
Max Obs per Page	2715
Obs in First Data Page	200
Index File Page Size	4096
Number of Index File Pages	3
Number of Data Set Repairs	2
Last Repair	13:44 Tuesday, November 13, 2018
ExtendObsCounter	YES
Filename	C:\dldmgactiontest\myfile.sas7bdat
Release Created	9.0401M6
Host Created	X64_10PRO
Owner Name	Computer_Name/User_ID
File Size	192KB
File Size (bytes)	196608

要点

- CONTENTS プロシジャには、修復に関する情報を報告する 2 つのフィールドがあります。
  - データセットの修復数
  - 最終修復

関連項目

"CONTENTS プロシジャ"

31

SAS データセットの圧縮

SAS での圧縮

SAS データセットの圧縮は、各オブザベーションを表すために必要なバイト数を削減するプロセスです。圧縮データセットでは、各オブザベーションは可変長のレコードですが、非圧縮データセットでは、各オブザベーションは固定長のレコードです。

圧縮には次のような利点があります。

- ファイルのストレージ要件の削減
- 処理中にデータの読み取りまたは書き込みに必要な I/O 操作が減少する

圧縮には次のような欠点があります。

- 各オブザベーションを圧縮解除し、更新する場合はストレージに書き込むときに再度圧縮するオーバーヘッドのため、圧縮ファイルを読み取るための CPU リソース要件が増加する
- 結果のファイルサイズが減少するのではなく増加する可能性がある状況

スペースの節約が主な関心事の場合は、テストすることをお勧めします。圧縮の詳細については、次の表にリストされているドキュメントを参照してください。

表 31.1　SAS エンジンでの圧縮

エンジン	ドキュメント	使用上の注意
V9 エンジン	“Compression” (*SAS V9 LIBNAME Engine: Reference*)	COMPRESS=NO | CHAR | BINARY をサポートします。
SPD Engine	“Updates to a Compressed SPD Engine Data Set” (*SAS*	COMPRESS=NO | CHAR | BINARY をサポートします。

エンジン	ドキュメント	使用上の注意
	Scalable Performance Data Engine: Reference)	SPD Engine の追加の言語要素により、圧縮をより詳細に制御できるようになります。
CAS エンジン	*“データ圧縮” (SAS Cloud Analytic Services: ユーザーガイド*)	COMPRESS=YES | NO をサポートします。

32

SAS ファイルの移動

動作環境間でのファイルの移動

SAS ファイルをある動作環境から別の動作環境に移動する手順は、動作環境、移動する SAS ファイルのメンバータイプとバージョン、およびファイルの移動に使用できる方法によって異なります。

この主題の詳細については、*SAS ファイルの移動とアクセス*を参照してください。

33

クロス環境データアクセス

クロス環境データアクセス(CEDA)の定義

このドキュメントでは、CEDA 処理の制限、利点、および動作について説明します。ここでは、いくつかの有用な概念を示します。

クロス環境データアクセス(CEDA)
ディレクトリベースの動作環境(UNIX や Windows など)で作成された SAS ファイルを、互換性のない環境または互換性のないセッションエンコーディングで処理できるようにします。CEDA を使用すると、処理は自動的かつ透過的に行われます。移送ファイルを作成したり、ファイルを変換する SAS プロシジャを使用したり、SAS プログラムを変更したりする必要はありません。CEDA は Base SAS 機能です。

データ表現
: データが特定の動作環境で保存される形式です。動作環境が異なれば、データの保存に使用する標準や規則も異なります。(“互換性のあるデータ表現” (777 ページ)を参照してください。)

  - 浮動小数点数は、IEEE 浮動小数点形式または IBM 浮動小数点形式で表現できます。
  - データの配置は、動作環境のデータ型要件に応じて、1 バイト、4 バイト、または 8 バイト境界で行うことができます。
  - データ型の長さは、文字データ型の場合は 8 ビット以上、整数データ型の場合は 16 ビット、32 ビット、または 64 ビット、単精度浮動小数点データ型の場合は 32 ビット、倍精度浮動小数点データ型の場合は 64 ビットです。
  - バイトの順序はビッグエンディアンまたはリトルエンディアンになります 。

エンコーディング
: コンピューターが使用できる数値(コードポイントと呼ばれる)にマップされた文字(文字、表語文字、数字、句読点、記号、制御文字など)のセットです。コードポイントは、エンコーディング方式を適用することによって文字セット内の文字に割り当てられます。エンコーディングの例としては、Wlatin1 や UTF-8 などがあります。(“トランスコーディングのための CEDA 処理が必要ないエンコーディングの組み合わせ” (*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。)

互換性のない
: 現在実行中のセッションとは異なるデータ表現またはエンコーディングを持つファイルについて説明します。CEDA を使用すると、さまざまな種類の互換性のないファイルへのアクセスが可能になります。

CEDA の利点

CEDA には次の利点があります。

- ファイルのデータ表現やエンコーディングを知らなくても、サポートされている SAS ファイルを透過的に処理できます。
- 移送ファイルは作成されません。CEDA では、ソース表現から移送ファイル、ターゲット表現への複数の変換ではなく、現在のセッションのデータ表現への単一の変換が必要です。
- CEDA を使用すると、ファイルを処理するために複数の手順を実行する必要がなくなります。
- CEDA では、SAS/CONNECT で必要なサインオンは必要ありません。

CEDA による SAS ファイル処理

CEDA を呼び出す条件

CEDA は次の状況で使用されます。

- SAS ファイルの文字値のエンコーディングが、現在実行中のセッションのエンコーディングと互換性がない場合。たとえば、ファイルを Latin1 から UTF-8 に移動すると、非互換性が発生する可能性があります。“互換性のあるエンコーディング” (778 ページ)を参照してください。
- SAS ファイルのデータ表現が、現在実行中のセッションのデータ表現と互換性がない場合。たとえば、ファイルを Windows から Linux に移動すると、非互換性が発生する可能性があります。“互換性のあるデータ表現” (777 ページ)を参照してください。

CEDA は、UNIX や Windows などのディレクトリベースの動作環境で作成された SAS ファイルをサポートします。CEDA は、これらの SAS エンジンに対して次の SAS ファイル処理を提供します。

BASE
: デフォルトの Base SAS エンジンと、SAS Viya プラットフォーム上の V9 エンジンのエイリアスが出荷されました。CEDA トピックでは V9 エンジンと呼ばれます。

SASESOCK
: SAS/CONNECT ソフトウェア用の TCP/IP ポートエンジン。

SPDE
: SAS Scalable Performance Data Engine (一部の例外を除く)。詳細については、“Accessing SPD Engine Files on Another Host” (*SAS Scalable Performance Data Engine: Reference*)を参照してください。

表 33.1 CEDA が提供する SAS ファイル処理

SAS ファイルタイプ	エンジン	サポートされている処理
SAS データセット	V9、SASESOCK、SPDE	入出力 [1]
PROC SQL ビュー (DATA ステップビューはサポートされていません)	V9	入力

SAS ファイルタイプ	エンジン	サポートされている処理
MDDB ファイル [2]	V9	入力

1 既存の SAS データセットを置き換える出力処理の場合、動作の違いがあります。詳細については、“出力処理がエンコーディングとデータ表現に与える影響” (778 ページ)を参照してください。
2 CEDA は、SAS 8 以降の MDDB ファイルをサポートします。

CEDA のログメッセージ

CEDA 処理が発生すると、SAS は 1 つ以上のメッセージをログに書き込みます。メッセージについては、“例: CEDA” (780 ページ)で説明します。CEDA メッセージの種類は次のとおりです。

- NOTE は情報提供を目的としたものであり、エラーを示すものではありません。
- ERROR は、操作がサポートされていないことを示している場合があります。たとえば、CEDA は更新モードをサポートしていません。データセットを置換せずに変更すると、更新モードになります。
- ERROR は、データセットが出力モードで開いている場合のデータ損失を示している可能性があります。データセットを作成または置換するときは、出力モードになります。
- WARNING は、データセットが入力モードで開いている場合のデータ損失を示します。入力モードの例は、データセットの出力です。
- INFO メッセージは、追加情報がある場合はそれを提供します。これらのメッセージを表示するには、システムオプション MSGLEVEL=I を指定する必要があります。

CEDA の制限事項

CEDA には次の制限があります。

- SAS カタログはサポートされていません。カタログエントリには、出力形式、入力形式、ストアドコンパイルマクロ、SAS コード、データ、およびさまざまな SAS プロシジャに固有のその他のエントリタイプが含まれる場合があります。
- 更新処理はサポートされていません。
- 一貫性制約は読み取りまたは更新できません。
- 監査証跡ファイルは更新できませんが、読み取ることはできます。
- インデックスはサポートされていません。したがって、インデックスを使用した WHERE 最適化はサポートされていません。
- 拡張属性は更新できませんが、読み取ることはできます。

- サポートされていないその他のファイルには、DATA ステップビュー、ストアドコンパイル DATA ステッププログラム、アイテムストア、DMDB ファイル、FDB ファイル、または SAS 7 より前に作成された SAS ファイルが含まれます。
- SAS は、データの読み取り時にデータ表現間の変換、またはエンコーディング間のトランスコードを行います。複数のプロシジャを使用する場合、SAS はデータを複数回変換またはトランスコードする必要があるため、システムのパフォーマンスに影響を与える可能性があります。
- データセットが破損している場合、CEDA はファイルを処理して修復することができません。CEDA は、破損したデータセットを修復するために必要な更新処理をサポートしていません。ファイルを修復するには、ファイルを作成した環境、または CEDA 処理を呼び出さない互換性のある環境に戻す必要があります。破損したデータセットを修復する方法については、*Base SAS プロシジャガイド*の DATASETS プロシジャの REPAIR ステートメントを参照してください。
- エンコーディングに互換性がない場合、トランスコーディングにより文字データが失われる可能性があります。"トランスコーディングの留意点" (*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。
- 動作環境間でデータを移動すると、数値変数の精度が失われることがあります。数値変数が短い長さで定義されている場合は、変数の長さを増やしてみることができます。フルサイズの数値変数は、CEDA によって精度が失われる可能性が低くなります。詳細については、"数値精度" (100 ページ)を参照してください。
- 数値変数の最小長は、動作環境に応じて 2 バイトまたは 3 バイトです。最小 3 バイトをサポートする動作環境(Windows や UNIX など)では、CEDA は 2 バイトの長さで作成された数値変数(z/OS など)を処理できません。この制限が発生した場合は、CEDA のかわりに XPORT エンジン、または CPORT および CIMPORT プロシジャを使用してください。

注: ファイルが以前のバージョンの SAS で作成されているためにこれらの制限が発生する場合は、MIGRATE プロシジャの使用を検討してください。このプロシジャについては、*Base SAS プロシジャガイド*に記載されています。PROC MIGRATE は、一貫性制約、インデックス、監査証跡などの多くの機能を保持します。

互換性のあるデータ表現

SAS Viya プラットフォームは Linux for x64 上で実行され、LINUX_X86_64 の SAS データ表現を備えています。次のデータ表現には互換性があり、CEDA を呼び出しません。

- Linux for x64 (LINUX_X86_64)
- Power Architecture 上の Linux (LINUX_POWER_64)
- Solaris for x64 (SOLARIS_X86_64)

他のほとんどのデータ表現は CEDA を呼び出します。データ表現値のリストについては、"OUTREP= データセットオプション" (*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。

データ表現またはエンコーディングに互換性がないかどうかを確認するには、SAS ログで CEDA メッセージを確認します。“例: CEDA に関するログメッセージを理解する” (780 ページ)を参照してください。

互換性のあるエンコーディング

SAS データセットを作成すると、文字エンコーディングがデータセットのメタデータに保存されます。互換性のあるエンコーディングでは、トランスコーディングに CEDA 処理は必要ありません。“トランスコーディングのための CEDA 処理が必要ないエンコーディングの組み合わせ” (*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。

SAS Viya プラットフォームでは、Compute Server のデフォルトのセッションエンコーディングは UTF-8 です。セッションエンコーディングは変更できます。SAS Data and AI Studio(2026.06 より前は SAS Studio という名前)では、このプリファレンスは**テキストエンコーディング**と呼ばれます。

ただし、エンコーディングに互換性がある場合でも、互換性のないデータ表現によって CEDA 処理が呼び出される可能性があります。さらに、スマート引用符などの一部の Microsoft Word 文字は切り捨てエラーを引き起こす可能性があります。文字には、UTF-8 エンコーディングで 2 バイト以上が必要です。

データに 7 ビット ASCII 文字(米国英語)のみが含まれている場合、データは UTF-8 を含む他の ASCII セッションと互換性があります。他の ASCII エンコーディングでは、さまざまな各国文字の上位ビットが使用されます。

出力処理がエンコーディングとデータ表現に与える影響

既存の SAS データセットを置き換える出力処理の場合、次の属性に関して動作が異なります。

エンコーディング

- V9 エンジンは、ソースライブラリからのファイルのエンコーディングを 使用します。つまり、エンコーディングが複製されます。
- V9 エンジンの場合、デフォルトで PROC COPY はソースライブラリからのファイルのエンコーディングを使用します。かわりに、現在のセッションのエンコーディングを使用する場合は、NOCLONE オプションを指定します。別のエンコーディングを使用する場合は、NOCLONE オプションと ENCODING=オプションを指定します。
- SAS/CONNECT で PROC COPY を使用する場合、デフォルトの動作では、(サーバーセッションではなく)クライアントセッションのエンコーディングが使用されます。
- SPD Engine は現在のセッションエンコーディングを使用します。PROC COPY の CLONE オプションはサポートされていません。

データ表現
- V9 エンジンは、PROC COPY を除き、現在のセッションのデータ表現を使用します。
- V9 エンジンの場合、デフォルトで PROC COPY はソースライブラリからのファイルのデータ表現を使用します。かわりに、現在のセッションのデータ表現を使用する場合は、NOCLONE オプションを指定します。別のデータ表現を使用する場合は、NOCLONE オプションと OUTREP=オプションを指定します。
- SAS/CONNECT で PROC COPY を使用する場合、デフォルトの動作では、(サーバーセッションではなく)クライアントセッションのデータ表現が使用されます。
- SPD Engine は、現在のセッションのデータ表現を使用します。PROC COPY の CLONE オプションはサポートされていません。

CEDA の使用に代わる方法

制限があるため、CEDA を使用できない場合があります。動作環境間でファイルを移動するには、次の方法を使用できます。

MIGRATE プロシジャ
通常、PROC MIGRATE は、SAS ライブラリのメンバーを現在の SAS リリースに移行する最良の方法です。PROC MIGRATE は、ほとんどのユーザーがデータ移行で必要とするデータ属性を保持する 1 ステップのコピープロシジャです。

場合によっては、SAS/CONNECT サーバーが必要になります。“Examples: Migrate SAS Libraries” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。

CPORT および CIMPORT プロシジャ
ソース環境では、PROC CPORT はデータセットまたはカタログを移送形式に書き込みます。ターゲット環境では、PROC CIMPORT が移送ファイルをターゲット環境の形式に変換します。切り捨てが発生した場合は、変数長を拡張する必要があります。CVP エンジンを PROC CPORT で使用することも、EXTENDVAR=オプションを PROC CIMPORT で使用することもできます。

SAS/CONNECT ソフトウェアのデータ転送サービス
データ転送サービスは、データのコピーを転送し、ある環境の表現から別の環境の表現へのデータの必要な変換や、SAS リリース間の必要な変換を実行するバルクデータ転送メカニズムです。SIGNON コマンドを使用して 2 つのセッション間の接続を確立し、PROC UPLOAD または PROC DOWNLOAD を実行してデータを移動する必要があります。

SAS/CONNECT ソフトウェアのリモートライブラリサービス
リモートライブラリサービスでは、LIBNAME ステートメントを使用してリモートデータに透過的にアクセスできます。

DATA ステップまたは PROC COPY を使用した XPORT エンジン
ソース環境では、XPORT エンジンと DATA ステップまたは PROC COPY を使用した LIBNAME ステートメントにより、SAS データセットから移送ファイルが作

成されます。ターゲット環境では、同じ方法で移送ファイルがターゲット環境の形式に変換されます。XPORT エンジンは、8 バイトを超えるファイル名や変数名などの SAS 7 以降の機能をサポートしていないことに注意してください。

XPORT エンジンはデータ表現を変換しますが、トランスコーディングは実行しません。XPORT エンジンは、シングルバイトエンコーディング間でのみデータセットを移送する場合に使用します。XPORT エンジンは、文字変数内のバイトがシングルバイトであることを想定しており、ASCII システムの場合はそれらをそのまま移送ファイルに出力し、EBCDIC システムの場合は ASCII に変換します。UTF-8 データを含む文字変数は、移送に書き込む前に(KCVT 関数を使用して)シングルバイトエンコーディングにトランスコードする必要があります。

例: CEDA

例: CEDA に関するログメッセージを理解する

コード例

この例を実行する場合は、まず、Linux for x64 以外の SAS®9 環境で simple という名前のデータセットを作成します。次のコードは、Windows のパス名を示しています。LIBNAME ステートメントでは独自のパス名を置き換えます。

```
libname test v9 'c:\examples';
data test.simple;
  x=1;
run;
```

SAS Viya プラットフォームで、simple データセットを作成し、次のコードをサブミットします。LIBNAME ステートメントでは独自のパス名を置き換えます。

```
libname myfiles v9 'library-path';
proc print data=myfiles.simple;        /* 1 */
run;

proc sql;
  insert into myfiles.simple           /* 2 */
    values (2);
quit;
```

1 データセットと処理環境のデータ表現が異なるため、PRINT プロシジャは CEDA 注記を生成します。この注記は情報提供を目的としたものであり、エラーを示すものではありません。PROC PRINT は、simple データセットを出力します。

2 SQL プロシジャは、myfiles.simple データセットに行を挿入しようとします。このコードによりエラーメッセージが生成されます。CEDA は、INSERT INTO ステートメントなどの更新処理をサポートしていません。

例のコード 33.1 CEDA 情報注記を示すログ出力

```
NOTE: Data file MYFILES.SIMPLE.DATA is in a format that is native
      to another host, or the file encoding does not match the session
      encoding. Cross Environment Data Access will be used, which might
      require additional CPU resources and might reduce performance.
```

例のコード 33.2 CEDA 更新エラーを示すログ出力

```
ERROR: File MYFILES.SIMPLE cannot be updated because its encoding
       does not match the session encoding or the file is in a format
       native to another host, such as WINDOWS_64.
```

要点

- SAS Viya プラットフォームでは、SAS Compute Server セッションのデータ表現は LINUX_X86_64 です。
- CEDA 処理は透過的かつ自動ですが、ほとんどのユーザーは CEDA 処理がいつ行われるかを知りたいと考えています。CEDA にはいくつかの制限があります。たとえば、更新処理はサポートされていません。
- CEDA 処理が発生すると、SAS はログに注記を書き込みます。この注記は情報提供を目的としたものであり、エラーを示すものではありません。
- 処理が CEDA によってサポートされていない場合、SAS はエラーメッセージをログに書き込みます。
- トランスコーディングにより文字データが失われる可能性がある場合、SAS は警告またはエラーメッセージをログに書き込みます。“例: 切り捨てに関するログメッセージを理解する” (782 ページ)および“例: 使用できない文字に関するログメッセージを理解する” (784 ページ)を参照してください。

関連項目

- “CEDA の制限事項” (776 ページ)
- “トランスコーディングの留意点” (*SAS 各国語サポート(NLS): リファレンスガイド*)
- “Compatibility and Migration” (*SAS V9 LIBNAME Engine: Reference*)

例: 切り捨てに関するログメッセージを理解する

コード例

この例では、SAS のダブルバイト文字セット(DBCS)セッションで mytrunctest データセットを作成します。セッションエンコーディングは shift-jis です。a 変数の長さは 22 バイトです。

```
libname myfiles v9 'c:\examples';
data myfiles.mytrunctest;
  a='サンプルテキスト文字列';
run;
proc print data=myfiles.mytrunctest;
run;
```

ユーザーは、UTF-8 エンコーディングを使用するセッションで mytrunctest データセットにアクセスします。このセッションでは、PROC PRINT 出力は切り捨てられます。追加情報メッセージが使用可能な場合は、システムオプション msglevel=i を指定して表示します。

```
options msglevel=i;
libname myfiles 'library-path';
proc print data=myfiles.mytrunctest;
run;
```

SAS ログの切り捨て警告は次のとおりです。システムオプション msglevel=i は、変数を識別する追加メッセージを生成します。

例のコード 33.3　ログ出力

```
WARNING: Some character data in the data set "MYFILES.MYTRUNCTEST"
      was lost during transcoding. Truncation occurred during
      transcoding to the new encoding. To avoid the transcoding error,
      please refer to "Troubleshooting Truncation and Data loss
      Issue during Transcoding" in SAS National Language Support
      (NLS): reference guide.

NOTE: There were 1 observations read from the data set MYFILES.MYTRUNCTEST.
INFO: The length of the variable "a" in the data set "MYFILES.MYTRUNCTEST" is
      insufficient. You might need to increase the length of the variable.
```

アウトプット 33.1　切り捨てられたデータを示す PROC PRINT 出力

Obs	a
1	サンプルテキス

切り捨てが発生するのは、a 変数では SHIFT-JIS エンコーディングよりも UTF-8 エンコーディングの方がさらに多くのバイトを必要とするためです。切り捨てを防ぐには、CVP エンジンを使用して変数長を拡張します。

```
libname cvp myfiles 'library-path';
proc print data=myfiles.mytrunctest;
```

```
run;
```

アウトプット 33.2　完全なデータ値を示す PROC PRINT 出力

Obs	a
1	サンプルテキスト文字列

要点

- SAS Viya プラットフォームでは、SAS Compute Server セッションのデフォルトのエンコーディングは UTF-8 です。SAS 管理者はこの値を変更できます。*SAS Viya: プログラミングランタイムサーバー*の"SAS ロケールと SAS エンコーディングのデフォルト値の変更"を参照してください。
- SAS が CEDA 注記をログに書き込む場合、この注記は情報提供です。CEDA 注記はエラーを示していません。
- SAS が切り捨てに関するエラーまたは警告をログに書き込む場合は、新しいエンコーディングで文字データの損失が発生している可能性があります。

  文字を表現するためにより多くのバイトを使用するエンコーディングでファイルを処理する場合、列の長さがより大きな文字サイズに対応できないと切り捨てが発生する可能性があります。たとえば、文字は wlatin1 エンコーディングでは 1 バイトとして表現されますが、UTF-8 では 2 バイトとして表現される場合があります。切り捨てにより、出力に"文字化け"文字や不正な文字が表示される可能性もあります。

  スマート引用符などの一部の Microsoft Word 文字には、UTF-8 で複数バイトが必要です。これらの文字をシングルバイトエンコーディングからトランスコードする場合、データバッファーが十分に大きくないと切り捨てが発生する可能性があります。
- 追加情報メッセージが使用可能な場合は、システムオプション msglevel=i を指定して表示できます。SAS のほとんどの操作では、トランスコーディングエラーが発生したときに追加のメッセージが生成されることがあります。
- 切り捨てを防ぐには、CVP エンジンを使用して変数長を拡張します。LIBNAME ステートメントで CVP エンジンを指定すると、データを変更せずに変数長を拡張できます。CVP エンジンを PROC COPY または PROC MIGRATE とともに使用して、変数長を永久拡張することもできます。
- 切り捨ては変数ラベルでも発生する可能性があります。CVP エンジンは変数ラベルの長さを拡張できません。必要に応じて、変数ラベルの内容を永久変更できます。たとえば、DATASETS プロシジャを MODIFY ステートメントおよび LABEL ステートメントとともに使用できます。

関連項目

- “CEDA の制限事項” (776 ページ)

- “トランスコーディングの留意点” (*SAS 各国語サポート(NLS): リファレンスガイド*)
- “例: CVP エンジンと V9 エンジンを使用して切り捨てを回避する” (281 ページ)
- “Example: Avoid Truncation When Migrating a SAS Library by Using the CVP Engine” (*SAS V9 LIBNAME Engine: Reference*)

例: 使用できない文字に関するログメッセージを理解する

コード例

この例では、SAS のダブルバイト文字セット(DBCS)セッションで special データセットを作成します。セッションエンコーディングは ms-950 Traditional Chinese (PCMS) です。var1 変数に十分な長さを設定すると、値を保存するためにより多くのバイトを必要とする他のセッションエンコーディングで使用された場合に、データが切り捨てられなくなります。

```
libname myfiles v9 'c:\examples';
data myfiles.special;
  length var1 $32;
  var1='示例文本';
run;
```

ユーザーは、セッションエンコーディングが latin1 である環境から special データセットにアクセスします。このセッションエンコーディングでは、中国語の文字は使用できません。追加情報メッセージが使用可能な場合は、システムオプション msglevel=i を指定して表示します。

```
options msglevel=i;
libname myfiles 'library-path';
proc print data=myfiles.special;
run;
```

SAS ログの警告は次のとおりです。システムオプション msglevel=i は、変数、行、およびエンコーディングを識別する追加メッセージを生成します。

例のコード 33.4　ログ出力

```
WARNING: Some character data was lost during transcoding in the data set
      "MYFILES.SPECIAL". It contains one or more characters that
      are not available in the new encoding. To avoid the transcoding error,
      please refer to "Troubleshooting Truncation and Data loss Issue during
      Transcoding" in SAS National Language Support (NLS): reference guide

NOTE: There were 1 observations read from the data set MYFILES.SPECIAL.
INFO: One or more characters in the variable "var1" at row number 1 in
      the encoding "ms-950" are not available in the encoding "latin1".
```

正しくない出力は次のとおりです。一部の Web ブラウザーは、使用できない文字を表示しようとします。一部のブラウザーでは空白が表示されます。

アウトプット 33.3　PROC PRINT の出力

Obs	var1
1	□□

このエラーは、ms-950 Traditional Chinese (PCMS)データセットから latin1 セッションエンコーディングで新しいデータセットを作成しても修正できません。エラーが発生し、不完全なデータセットが生成されてしまいます。

これらのエラーを回避するために、SAS では、データセットが作成されたセッションエンコーディングでデータセットにアクセスすることをお勧めします。もう 1 つの解決策は、セッションエンコーディングとして UTF-8 を一貫して使用することです。UTF-8 エンコーディングにはほとんどの言語の文字が含まれるため、文字データが正しく保持される可能性が高くなります。

要点

- SAS が CEDA 注記をログに書き込む場合、この注記は情報提供です。CEDA 注記はエラーを示していません。

  ただし、エンコーディングに互換性がない場合、トランスコーディングにより文字データが失われる可能性があります。たとえば、LATIN1 などのシングルバイトエンコーディングの文字は、LATIN2 などの別のシングルバイトエンコーディングでは使用できない場合があります。したがって、異なるエンコーディングでデータセットを処理するときは、常に出力を確認してください。
- SAS が 1 つ以上の文字が使用できないことを示すエラーまたは警告をログに書き込む場合、新しいエンコーディングで文字データの損失が発生している可能性があります。
- 追加情報メッセージが使用可能な場合は、システムオプション msglevel=i を指定して表示できます。SAS のほとんどの操作では、トランスコーディングエラーが発生したときに追加のメッセージが生成されることがあります。
- "使用できません"というエラーが発生したときにデータが失われないようにするために、SAS では、データセットが作成されたセッションエンコーディングでデータセットにアクセスすることをお勧めします。
- SAS Viya プラットフォームでは、SAS Compute Server セッションのデフォルトのエンコーディングは UTF-8 です。SAS 管理者はこの値を変更できます。*SAS Viya: プログラミングランタイムサーバー*の"SAS ロケールと SAS エンコーディングのデフォルト値の変更"を参照してください。
- ベストプラクティスは、一貫して UTF-8 セッションエンコーディングを使用することです。UTF-8 エンコーディングにはほとんどの言語の文字が含まれるため、文字データが正しく保持される可能性が高くなります。UTF-8 では多くの文字に複数バイトが使用されるため、列の長さが切り捨てを防ぐのに十分な長さであることを確認する必要があります。一般的な問題の詳細については、"WLATIN1 から UTF-8 へのデータの移行" (*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。
- "使用できません"というエラーは、変数ラベルでも発生する可能性があります。前述の推奨事項に加えて、必要に応じて変数ラベルの内容を永久変更できます。たと

えば、DATASETS プロシジャを MODIFY ステートメントおよび LABEL ステートメントとともに使用できます。

- あまり一般的ではありませんが、文字列の操作中に文字列内の文字が切り捨てられると、"使用できません"というエラーが発生することがあります。たとえば、SUBSTR 関数はバイト位置に基づいてデータを分割するため、ダブルバイト文字でエラーが発生する可能性があります。このエラーを回避するには、かわりに KSUBSTR 関数を使用します。エラーを防ぐことができない場合にエラーを修正するには、KPROPDATA 関数を使用して壊れた文字を特定し、それを置換文字に置き換えます。
- KPROPDATA 関数は文字の置換に役立ちます。たとえば、ある文字をサポートしないエンコーディングを使用する場合、KPROPDATA は、元の文字列内の文字を、結果の文字列内の Unicode エスケープ形式に変換できます。結果のデータにはさらに多くのスペースが必要ですが、データは保存されます。

関連項目

- “CEDA の制限事項” (776 ページ)
- “トランスコーディングの留意点” (*SAS 各国語サポート(NLS): リファレンスガイド*)
- “KSUBSTR 関数” (*SAS 各国語サポート(NLS): リファレンスガイド*)
- “KPROPDATA 関数” (*SAS 各国語サポート(NLS): リファレンスガイド*)

例: CEDA を回避するエンコーディングの指定

コード例

この例では、ユーザーは Linux for x64 上で SAS ®9 を実行しています。セッションエンコーディングは latin1 Western (ISO)です。次の DATA ステップでは、ENCODING=データセットオプションを使用して、utf8 エンコーディングを持つデータセットを作成します。

```
libname mylnx v9 '/mydata';
data mylnx.mytest (encoding=utf8);
  a='sample data string';
run;
proc contents data=mylnx.mytest;
run;
```

データセットの作成時に、SAS ログに次のメッセージが表示されます。このメッセージは、データセットを作成するために CEDA が呼び出されたことを示します。ただし、mytest データセットが UTF-8 セッションエンコーディングでアクセスされる場合、CEDA は呼び出されません。CEDA は、UTF-8 以外のセッションエンコーディ

ングで呼び出されます。また、動作環境に互換性がない場合には CEDA が呼び出されます。

```
NOTE: Data file MYLNX.MYTEST.DATA is in a format that is native
      to another host, or the file encoding does not match the session
      encoding. Cross Environment Data Access will be used, which might
      require additional CPU resources and might reduce performance.
```

CONTENTS プロシジャが SAS®9 で実行されると、CEDA メッセージがログに再度書き込まれます。セッションエンコーディングが UTF-8 である環境でデータセットにアクセスする場合、CEDA メッセージは表示されません。

次は PROC CONTENTS からの出力で、エンコーディングとして utf-8 Unicode (UTF-8) を示しています。

アウトプット 33.4　PROC CONTENTS 出力の一部

Data Set Name	MYLNX.MYTEST	**Observations**	1
Member Type	DATA	**Variables**	1
Engine	V9	**Indexes**	0
Created	08/27/2019 15:31:24	**Observation Length**	18
Last Modified	08/27/2019 15:31:24	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

要点

- ファイルのデータ表現またはエンコーディングが現在のセッションのデータ表現またはエンコーディングと異なる場合、SAS は自動的に CEDA 処理を呼び出します。
- CEDA 処理によりパフォーマンスが低下する可能性があります。したがって、別のセッションエンコーディングでアクセスされるデータセットを作成する場合は、データセットの作成時にそのエンコーディングを指定することが必要になる場合があります。
- ENCODING=データセットオプションまたは OUTENCODING= LIBNAME ステートメントオプションを使用して、ターゲット環境のエンコーディングを指定します。CEDA は、データセットの作成時に現在のセッションで呼び出されます。CEDA は、データセットと同じエンコーディングを持つターゲットセッションの後半では回避されます。

関連項目

- “ENCODING= データセットオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)
- “INENCODING=, OUTENCODING= LIBNAME Statement Options” (*SAS V9 LIBNAME Engine: Reference*)
- “各国語サポート関連のエンコーディング” (*SAS 各国語サポート(NLS): リファレンスガイド*)

例: データ表現と UTF-8 エンコーディングの指定

コード例

この例は、OUTREP=データセットオプションまたは LIBNAME ステートメントオプションを使用するときに、UTF-8 などのデフォルト以外のエンコーディングを指定する正しい方法を示しています。OUTREP=オプションを指定すると、SAS はセッションエンコーディングを無視し、デフォルトのエンコーディングを割り当てます。デフォルトのエンコーディングを使用ない場合は、エンコーディングオプションを指定する必要があります。

この例では、ユーザーは SAS®9 for Windows を実行しており、Linux for x64 上のターゲットセッション用のデータセットを作成したいと考えています。ユーザーのエンコーディングとターゲットのエンコーディングは両方とも SAS 構成ファイルで UTF-8 に設定されています。ただし、OUTREP=オプションのみを指定した場合、ユーザーの UTF-8 セッションエンコーディングは無視されます。ユーザーは、次に示すように、OUTREP=オプションと ENCODING=オプションの両方を指定する必要があります。

次の DATA ステップでは、ターゲット環境のデータ表現とエンコーディングの両方を指定します。

```
libname myfiles 'c:\examples';
data myfiles.test (outrep=linux_x86_64 encoding=utf8);
  x=1;
run;
proc contents data=myfiles.test;
run;
```

データセットの作成時と CONTENTS プロシジャの実行時に、SAS ログに CEDA メッセージが表示されます。

```
NOTE: Data file MYFILES.TEST.DATA is in a format that is native
      to another host, or the file encoding does not match the session
      encoding. Cross Environment Data Access will be used, which might
      require additional CPU resources and might reduce performance.
```

PROC CONTENTS 出力は、データ表現とエンコーディングが正しく割り当てられていることを示します。これで、ユーザーは CEDA を呼び出すことなく、データセットをターゲット環境に移送できるようになります。

アウトプット 33.5　PROC CONTENTS 出力の一部

Data Set Name	MYFILES.TEST	Observations	1
Member Type	DATA	Variables	1
Engine	V9	Indexes	0
Created	09/03/2019 14:51:14	Observation Length	8
Last Modified	09/03/2019 14:51:14	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

要点

- SAS には、次の 2 つの値に基づくデフォルトのセッションエンコーディングがあります。
  - 現在のセッションの動作環境
  - 現在のセッションのロケール
- 多くのサイトでは、ENCODING システムオプションを使用して、デフォルト以外のセッションエンコーディングを指定します。ENCODING システムオプションは、構成時または呼び出し時に有効です。
- ベストプラクティスとして、OUTREP=オプションを指定して別のデータ表現でデータセットを作成するときは、エンコーディングオプションも指定します。ENCODING=データセットオプションまたは OUTENCODING= LIBNAME ステートメントオプションを使用します。

  エンコーディングオプションを指定しない場合、現在のセッションエンコーディングは無視され、かわりにデフォルトのエンコードが割り当てられます。このデフォルトのエンコーディングは、次の 2 つの値に基づいています。
  - OUTREP=オプションで表される動作環境
  - 現在のセッションのロケール
- デフォルトの動作は、COPY プロシジャまたは DATASETS プロシジャの COPY ステートメントの OVERRIDE=(OUTREP=*data-rep-value*)構文でも発生します。デフォ

ルト以外のエンコーディング値が必要な場合は、OVERRIDE=(OUTREP= *data-rep-value* ENCODING=*encoding-value*)を指定します。

関連項目

- " POSIX ロケール値および Linux エンコーディング値に基づく DATESTYLE=および PAPERSIZE=システムオプションのデフォルト値" (*SAS 各国語サポート(NLS): リファレンスガイド*)
- "ENCODING システムオプション" (*SAS 各国語サポート(NLS): リファレンスガイド*)
- "ENCODING= データセットオプション" (*SAS 各国語サポート(NLS): リファレンスガイド*)
- "INENCODING=, OUTENCODING= LIBNAME Statement Options" (*SAS V9 LIBNAME Engine: Reference*)
- "Compatibility and Migration" (*SAS V9 LIBNAME Engine: Reference*)

例: SAS Compute Server セッションまたはデータセットのエンコーディングとデータ表現の確認

コード例

次のステートメントは、セッションエンコーディングとロケールを返します。

```
%put %sysfunc(getoption(encoding));
%put %sysfunc(getoption(locale));
```

SAS ログに書き込まれる情報の例を次に示します。セッションエンコーディングは UTF-8、ロケールは EN_US です。

```
82   %put %sysfunc(getoption(encoding));
UTF-8
83   %put %sysfunc(getoption(locale));
EN_US
```

OPTIONS プロシジャは、セッションエンコーディングとロケールを確認するもう 1 つの方法です。

```
proc options option=(encoding locale) define value;
run;
```

PROC OPTIONS は、オプション設定に関する詳細情報を返します。次の出力例では、関連する行を強調するために多くの行が省略されています。

```
Option Value Information For SAS Option ENCODING
   Value: UTF-8
.
.
.
Option Value Information For SAS Option LOCALE
   Value: EN_US
```

現在実行中のセッションのデータ表現を確認するには、一時データセットを作成し、CONTENTS プロシジャまたは DATASETS プロシジャの CONTENTS ステートメントをサブミットします。

```
data sessiontest;
  x=1;
run;
proc contents data=sessiontest;
run;
```

sessiontest データセットは、一時的な Work ライブラリに作成されます。次は PROC CONTENTS 出力の一部です。データセットの作成時に OUTREP=オプションが指定されていなかったため、sessiontest にはセッションのデータ表現が含まれます。出力にはセッションエンコーディングも表示されます。

アウトプット 33.6　セッションエンコーディングとデータ表現を学習するための PROC CONTENTS 出力の一部

Data Set Name	WORK.SESSIONTEST	**Observations**	1
Member Type	DATA	**Variables**	1
Engine	V9	**Indexes**	0
Created	04/07/2020 13:44:43	**Observation Length**	8
Last Modified	04/07/2020 13:44:43	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64, LINUX_POWER_64		
Encoding	utf-8 Unicode (UTF-8)		

要点

- 異なる環境間でデータを共有する場合、現在実行中のセッションのエンコーディングとデータ表現を確認する必要がある場合があります。ロケールを知っておくことも役に立ちます。
- PROC OPTIONS または GETOPTION 関数を使用して、セッションエンコーディングのクエリを実行します。

- データセットのデータ表現とエンコードを確認するには、PROC CONTENTS (または PROC DATASETS の CONTENTS ステートメント)を使用します。
- OUTREP=オプションはシステムオプションとして使用できないため、PROC OPTIONS または GETOPTION 関数を使用してセッションのデータ表現値のクエリを実行することはできません。かわりに、OUTREP=または ENCODING=オプションを使用せずに一時データセットを作成し、一時データセットにセッションのデータ表現とエンコーディングが含まれるようにします。PROC CONTENTS (または DATASETS プロシジャの CONTENTS ステートメント)をサブミットします。PROC SQL を使用する例については、サンプル 55054 を参照してください。
- 一部の動作環境では、PROC CONTENTS (または PROC DATASETS の CONTENTS ステートメント)の出力に、いくつかのデータ表現がリストされます。この場合、動作環境には互換性があり、CEDA 処理は呼び出されません。ただし、一部の互換環境ではカタログに互換性がない場合があります。“互換性のあるデータ表現” (777 ページ)を参照してください。
- PROC CONTENTS (または PROC DATASETS の CONTENTS ステートメント)によって返される情報は、動作環境とエンジンによって異なります。
- 環境間でデータセットを共有する場合、役立つ可能性があるもう 1 つの属性は、PROC CONTENTS 出力の**エンジン/ホスト関連情報**部分でレポートされる、**作成したホスト**です。

関連項目

- “GETOPTION 関数” (*SAS 関数と CALL ルーチン: リファレンス*)
- “OPTIONS プロシジャ” (*SAS システムオプション: リファレンス*)
- “CONTENTS プロシジャ” (*Base SAS プロシジャガイド*)

34

SAS カタログの管理

SAS カタログの定義

SAS カタログは、カタログエントリと呼ばれる小さな単位でさまざまな種類の情報を保存する特別な SAS ファイルです。各エントリには、SAS に対してその目的を識別するエントリタイプがあります。単一の SAS カタログには、複数のタイプのカタログエントリを含めることができます。例としては、出力形式、入力形式、マクロ、グラフィック出力などがあります。CATALOG プロシジャを使用して、カタログの内容をリストできます。

詳細については、“Catalogs” (*SAS V9 LIBNAME Engine: Reference*)を参照してください。

7 部

その他の SAS 機能

35

SAS レジストリ

SAS レジストリの概要

SAS レジストリとは

SAS レジストリは、SAS の構成データの中央記憶域です。たとえば、レジストリには次のものが保存されます。

- SAS が起動時に割り当てるライブラリとファイルのショートカット
- 使用するために定義されているプリンター
- さまざまな SAS 製品の構成データ

レジストリは階層構造で編成されています。キーと値が含まれ、キーには他のキーを含めることができ、値はこれらのキーの下に保存されている実際の設定です。こ

の形式は、Linux オペレーティングシステムでのディレクトリベースのファイル 構造の動作と同様の方法で動作します。SAS レジストリの調整は、プロシジャのデフォルト設定、ユーザーインターフェイスの外観、SAS サーバーの構成など、ソフトウェアの特定の部分の動作に影響を与える可能性があります。

注: ホストプリンターは SAS レジストリでは参照されません。

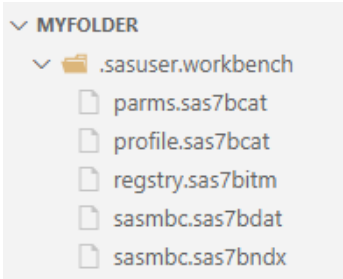
SAS レジストリの使用対象者

SAS レジストリは、システム管理者および経験豊富な SAS ユーザーが使用できるように設計されています。このセクションでは、レジストリツールの概要を示し、レジストリの一部をインポートおよびエクスポートする方法について説明します。

注意
レジストリの編集を誤ると、システムが不安定になったり、使用できなくなったりする可能性があります。

可能な限り、レジストリを直接編集するのではなく、管理ツールを使用して構成を変更してください。管理ツールを使用すると、構成を変更するときに値がレジストリに適切に保存されます。

SAS レジストリの保存場所



Sasuser ライブラリと Sashelp ライブラリのレジストリファイル

SAS レジストリは論理的には 1 つのデータストアですが、物理的には、Sasuser ライブラリと Sashelp ライブラリの両方にある 2 つの異なるファイルで構成されています。レジストリの物理ファイル名は regstry.sas7bitm です。デフォルトでは、これらのレジストリファイルは、Sashelp および Sasuser ライブラリの SAS エクスプローラービューでは非表示になります。

- Sashelp ライブラリレジストリファイルには、サイトのデフォルトが含まれています。通常、システム管理者は、サイトで使用するプリンター、起動時に割り当てられるグローバルファイルショートカットまたはライブラリ、およびサイトのその他の構成デフォルトを構成します。
- Sasuser ライブラリレジストリファイルには、ユーザーのデフォルトが含まれています。構成情報を変更する場合、設定は Sasuser ライブラリに保存されます。

サイトのデフォルトを復元する方法

SAS セッションに元のサイトのデフォルトを復元する場合は、Sasuser ライブラリから regstry.sas7bitm ファイルを削除し、SAS セッションを再起動します。

SAS レジストリを表示するには

次のコード行をサブミットして、SAS レジストリを表示できます。

```
proc registry list;
run;
```

このメソッドでは、レジストリを SAS ログに出力し、サブキーを含むすべてのレジストリエントリを含む大きなリストを生成します。サイズが大きいため、このメソッドを使用してレジストリを表示するには数分かかる場合があります。

SAS レジストリを表示する方法の詳細については、“REGISTRY プロシジャ” (*Base SAS プロシジャガイド*)の REGISTRY プロシジャを参照してください。

SAS レジストリの定義

SAS レジストリは、DOS や Linux のファイルシステムのようなディレクトリやサブディレクトリを使用するのではなく、キーとサブキーを構造の基礎として使用します。これらの用語とここで説明する他のいくつかの用語は、SAS レジストリについて説明するときに頻繁に使用されます。

キー
: SAS の特定の側面を参照するレジストリファイル内のエントリ。レジストリファイルの各エントリはキー名で構成され、次の行に 1 つ以上の値が続きます。キー名は角かっこ([と])の間に 1 行で入力します。

  キーは、値やサブキーが関連付けられていないプレースホルダーにすることも、値が関連付けられた多数のサブキーを持つこともできます。サブキーはバックスラッシュ(\)で区切られます。単一のキー名または一連のキー名の長さは 255 文字を超えることはできません(角かっことバックスラッシュを含む)。キー名にはバックスラッシュを除く任意の文字を含めることができ、大文字と小文字は区別されません。

SAS レジストリには、SAS_REGISTRY という最上位キーが 1 つだけ含まれています。SAS_REGISTRY の下のすべてのキーはサブキーです。

サブキー
別のキーの中にあるキー。サブキーはバックスラッシュ(\)で区切られます。サブキー名では大文字と小文字が区別されません。次のキーには、1 つのルートキーと 2 つのサブキーが含まれています: [SAS_REGISTRY\HKEY_USER_ROOT\CORE]

SAS_REGISTRY
ルートキーです。

HKEY_USER_ROOT
SAS_REGISTRY のサブキーです。SAS レジストリには、このレベルに別のサブキーが 1 つあります。それは HKEY_SYSTEM_ROOT です。

CORE
HKEY_USER_ROOT のサブキーで、プリンターやウィンドウ処理などの多くのデフォルト属性が含まれています。

リンク
内容がキーを参照する値。リンクは SAS 内部でのみ使用するように設計されています。これらの値は常に"link:"という単語で始まります。

値
キーまたはサブキーに関連付けられた名前と内容。値には、値の名前と値の内容(値データとも呼ばれる)の 2 つのコンポーネントがあります。

.SASXREG ファイル
実際のバイナリ SAS レジストリファイルのテキスト表現を含むファイル拡張子.SASXREG を持つテキストファイル。

SAS レジストリの管理

SAS レジストリの管理に関する主な懸念事項

注意
レジストリの編集を誤ると、システムが不安定になったり、使用できなくなったりする可能性があります。 可能な限り、レジストリを編集するのではなく、管理ツールを使用して構成を変更します。これは、構成を変更するときに値がレジストリに適切に保存されるようにするためです。

Sasuser レジストリのバックアップ

Sasuser レジストリをバックアップする理由

Sasuser[1] 部分のレジストリには個人設定が含まれています。SAS セッションに大幅なカスタマイズを行った場合は、レジストリの Sasuser 部分をバックアップすることをお勧めします。大幅なカスタマイズには次のようなものがあります。

- 新しいプリンターのインストール
- システム管理者が提供するデフォルトのプリンター設定からプリンター設定を変更
- ローカリゼーション設定の変更
- TRANTAB を使用した変換テーブルの変更

SAS がデフォルト設定にリセットされる場合

SAS は起動時にレジストリファイルを自動的にスキャンします。SAS は、次の 2 つの条件下でレジストリを元の設定に復元します。

- SAS がレジストリの破損を検出した場合、SAS はファイルを再構築します。
- regstry.sas7bitm というレジストリファイルを削除すると、SAS は Sasuser レジストリをデフォルト設定に復元します。レジストリファイルは Sasuser ライブラリにあります。

注意
Sahelp にあるレジストリファイルは削除しないでください。これにより、SAS が起動できなくなります。

レジストリをバックアップする方法

レジストリをバックアップするには 2 つの方法があり、それぞれ異なる結果が得られます。

方法 1: Sasuser レジストリファイルのコピーを regstry.sas7bitm という名前で保存します。
結果は、コピーした時点のレジストリの正確なコピーです。レジストリのそのコピーを使用して、レジストリの壊れたコピーを復元する必要がある場合、コピー日以降にレジストリに加えられた変更はすべて失われます。ただし、元のデフォ

1. レジストリの Sashelp 部分には、サイトのすべてのユーザーに共通の設定が含まれています。Sahelp は書き込み保護されており、システム管理者のみが更新できます。

ルトのレジストリに戻すよりも、このバックアップファイルを作成しておいた方がよいでしょう。

方法 2: PROC REGISTRY を使用して、Sasuser レジストリの変更された部分をバックアップします。

その結果、レジストリの連結コピーが作成され、バックアップファイルから復元できます。PROC REGISTRY の EXPORT=ステートメントを使用してバックアップファイルを作成する場合、SAS は、変更されたレジストリの部分を保存します。SAS がこのバックアップファイルをレジストリに復元すると、バックアップファイルは次の方法で現在のレジストリと連結されます。

- バックアップ日以降に Sasuser レジストリに追加されたまったく新しいキー、サブキー、または値は、新しいレジストリに保持されます。
- SAS の最初のインストール後に変更され、バックアップ後に再度変更された既存のキー、サブキー、または値は上書きされ、復元後にバックアップファイルの値に戻ります。
- 既存のキーまたはサブキー(または元のデフォルト値を保持する値)は、復元後にデフォルト値になります。

レジストリ障害からの回復

このセクションでは、バックアップファイルを使用してレジストリを復元する手順と、破損したレジストリファイルを修復する方法を説明します。

SAS エクスプローラーまたはオペレーティングシステムのコピーコマンドを使用して作成されたレジストリバックアップファイルをインストールするには、次の操作を行います。

1 破損したレジストリファイルの名前を別の名前に変更します。

2 バックアップファイルの名前を、レジストリファイルの名前である *regstry.sas7bitm* に変更します。

3 名前を変更したレジストリファイルを、以前のレジストリファイルがあった Sasuser の場所にコピーします。

4 SAS セッションを再起動します。

PROC REGISTRY で作成されたレジストリバックアップファイルを復元するには、次の操作を実行します。

1 プログラムエディターを開き、次のプログラムをサブミットして、前に作成したレジストリファイルをインポートします。

```
proc registry import=<registry file specification>;
 run;
```

これにより、レジストリファイルが Sasuser ライブラリにインポートされます。

2 ファイルの名前がまだ適切に指定されていない場合は、エクスプローラーを使用してレジストリファイルの名前を regstry.sas7bitm に変更します。

3 SAS を再起動します。

破損したレジストリの修復を試みるには、次の操作を行います。

1 破損したレジストリファイルの名前を"registry"以外の名前(*temp* など)に変更します。

2 SAS セッションを開始します。

3 *temp* レジストリの場所を指すライブラリを定義します。

```
libname here '.'
```

4 REGISTRY プロシジャを実行し、Sasuser レジストリを再定義します。

```
proc registry setsasuser="here.temp";
 run;
```

5 HKEY_USER_ROOT キーの下にある破損したフィールドを編集します。

6 SAS セッションを閉じ、変更したレジストリの名前を元の名前に戻します。

7 新しい SAS セッションを開いて、変更によって問題が解決されたかどうかを確認します。

SAS レジストリを使用した色の制御

色と SAS レジストリの概要

SAS レジストリには、ほとんどの Web ブラウザーに共通する色の名前の RGB 値が含まれています。これらの色は、ODS および GRAPH 出力に使用できます。RGB 値はトリプレット(赤、緑、青)であり、各成分の範囲は 00-FF (0-255)です。

色のレジストリ値は、COLORNAMES\HTML サブキーにあります。

プログラムによる色の追加

SAS コードを使用して、レジストリの COLORNAMES\HTML サブキーに追加することで、独自の新しい色の値を作成できます。

最も簡単な方法は、まず REGISTRY プロシジャが予期するレイアウトでファイルに色の値を書き込むことです。次に、REGISTRY プロシジャを使用してファイルをインポートします。この例では、スペイン語の色の名前がレジストリに追加されます。

```
filename mycolors temp;
data _null_;
  file "mycolors";
  put "[colornames\html]";
  put ' "rojo"=hex:ff,00,00';
  put ' "verde"=hex:00,ff,00';
  put ' "azul"=hex:00,00,ff';
  put ' "blanco"=hex:ff,ff,ff';
```

```
  put ' "negro"=hex:00,00,00';
  put ' "anaranjado"=hex:ff,a5,00';
run;

proc registry import="mycolornames";
run;
```

これらの色をレジストリに追加すると、SAS が提供する色名を使用する場所であればどこでも、これらの色名を使用できます。たとえば、次のコードに示すように、GOPTIONS ステートメントで色の名前を使用できます。

```
goptions cback=anaranjado;
proc gtestit;
run;
```

レジストリの構成

ユニバーサル印刷の定義

REGISTRY プロシジャでは、プリンター定義をバックアップしたり、SASXREG ファイルからプリンター定義を復元したりできます。レジストリ値のその他の直接変更は、SAS テクニカルサポートの指導の下でのみ実行してください。

SAS レジストリに関するライブラリ参照名(Libref)の問題の解決

ライブラリ参照名(libref)は SAS レジストリに保存されます。ライブラリ参照名が以前は機能していた後に失敗するという状況が発生する可能性があります。状況によっては、レジストリを編集することが問題を解決する最も早い方法です。このセクションでは、欠損または失敗したライブラリ参照名の修復に必要な内容について説明します。

SAS レジストリに保存されている永久ライブラリ参照名が起動時に失敗すると、SAS ログに次の注記が表示されます。

```
NOTE: One or more library startup assignments were not restored.
```

次のエラーは、ライブラリ割り当ての問題の一般的な原因です。

- SAS レジストリ内のライブラリ参照名割り当てに必要なフィールド値が欠損しています。

- SAS レジストリ内のライブラリ参照名割り当てに必要なフィールド値が無効です。たとえば、ライブラリ名は 8 文字に制限されており、エンジン値は実際のエンジン名と一致する必要があります。
- SAS レジストリ内のライブラリ参照名の暗号化されたパスワードデータが変更されました。

注意
SAS レジストリ内の多くのライブラリ参照名割り当てエラーを修正できます。 ライブラリ参照名や SAS レジストリに詳しくない場合は、テクニカルサポートにお尋ねください。SAS レジストリではエラーが発生しやすく、起動時にライブラリが割り当てられなくなる可能性があります。

ライブラリ参照名の割り当てエラーを修正するには、動作環境に応じて次のパスのいずれかを選択し、必要に応じてキーとキー値を変更します。

CORE\OPTIONS\LIBNAMES

または

CORE\OPTIONS\LIBNAMES\CONCATENATED

注: これらの修正は永久ライブラリ参照名に対してのみ可能です。

たとえば、永久連結ライブラリのキーの名前が正の整数以外に変更されたと判断した場合は、準拠するようにそのキーの名前を再度変更できます。

SMTP メールを送信するための IGNOREFROM SAS レジストリキーの構成

IGNOREFROM レジストリキーを使用すると、FROM= オプションで指定された電子メールエイリアスが SMTP MAIL FROM コマンドで使用されるのを防ぐことができます。このオプションが有効な場合:

- SMTP 認証では FROM= 値は使用されません。
- SAS では、引き続き FROM= 値がメールメッセージの From: アドレスとして使用されます。

このオプションは、FILENAME EMAIL ステートメントを使用して送信される電子メールにのみ適用されます。

注: SASHELP レジストリを変更するには、管理者レベルのアクセス許可が必要です。

1 現在のレジストリを registry.txt という名前のファイルにエクスポートします。

```
proc registry
  listuser
  export='registry.txt';
run;
```

2 レジストリファイルを更新します。registry.txt を開きます。次のエントリを追加します:

```
[CORE\EMAIL]
"IGNOREFROM"=dword:00000001
```

3 更新されたレジストリファイルを SASHELP にインポートします。

```
filename source 'registry.txt';
proc registry import=source usesashelp;
run;
```

4 レジストリエントリが正しく追加されたことを確認します。

```
proc registry listhelp startat='CORE\EMAIL';
run;
```

アウトプット 35.1 ログ出力の例

```
1          proc registry listhelp startat='CORE\EMAIL'; run;

NOTE: Contents of SASHELP REGISTRY starting at subkey [CORE\EMAIL]
[  CORE\EMAIL]
  IGNOREFROM=dword:0000000000000001
NOTE: PROCEDURE REGISTRY used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

注: 8 桁 (00000001) を入力しても、値は 16 進数 (0000000000000001) として表示されます。これは予期されたものです。これは同じ値です。SAS は、表示のために値を先頭のゼロで埋め込みます。**dword:0000000000000001** の代わりに **int:1** が表示された場合、インポートで正しい種類が使用されておらず、IGNOREFROM は機能しません。エントリを削除して再作成する必要があります。

関連項目

- (807 ページ)
- “REGISTRY プロシジャ” (*Base SAS プロシジャガイド*)
- “FILENAME ステートメント: EMAIL (SMTP)アクセスメソッド” (*SAS グローバルステートメント: リファレンス*)

36

SAS で使用される業界プロトコル

SMTP 電子メール

概要

SAS では、SAS システムオプションおよび SAS ステートメントを使用して、SAS セッションから直接メールを送信できます。この機能は、Simple Mail Transfer Protocol (SMTP)を使用して、プログラムでメッセージを生成および配信します。

注: SAS Viya Mail Service は、メールを送信する SAS 言語要素とは独立しています。メールサービス(SAS Viya: 構成辞書で参照)は、構成された SMTP サーバーを介して電子メールを送信するための REST API を提供します。メールサービスをカスタマ

イズしても、FILENAME EMAIL ステートメントまたは関連するシステムオプションを使用する SAS コードの動作は変更されません。SAS プログラムからメールを直接送信するには、この章で説明するオプションおよびステートメントを使用します。

SAS セッションから SMTP メールを送信する方法の例については、“SMTP インターフェイスを使用して電子メールを送信する” (814 ページ)を参照してください。

SMTP 電子メールを制御する SAS 言語要素

SAS システムオプション

いくつかの SAS システムオプションは SMTP 電子メールを制御します。動作環境と、サイトで SMTP 電子メールインターフェイスがサポートされているかどうかに応じて、起動時または SAS 構成ファイルでこれらのオプションを指定する必要がある場合があります。

EMAILSYS システムオプションは、SAS 内から電子メールを送信するためにどの電子メールシステムを使用するかを指定します。EMAILSYS システムオプションの詳細については、ご使用の動作環境の SAS ドキュメントを参照してください。

次のシステムオプションは、SMTP 電子メールインターフェイスがサイトでサポートされている場合にのみ指定されます。

- EMAILACKWAIT=は、SAS が SMTP サーバーからの確認応答を受信するまで待機する秒数を指定します。
- EMAILAUTHPROTOCOL=は、SMTP 電子メールの認証プロトコルを指定します。
- EMAILFROM=は、FILE または FILENAME ステートメントのいずれかを使用して電子メールを送信するときに、FROM 電子メールオプションを必要とするかどうかを指定します。
- EMAILHOST=は、サイトの電子メールアクセスをサポートする SMTP サーバーを指定します。
- EMAILID=は、ログオン ID、電子メールプロファイル、電子メールアドレスのいずれかを指定して、電子メール送信者を識別します。
- EMAILPORT=は、SMTP サーバーが接続されるポートを指定します。
- EMAILPW=は、電子メールログインパスワードを指定します。
- EMAILUTCOFFSET=は、電子メールメッセージの日付:ヘッダーフィールドで使用される UTC オフセットを指定します。

SAS ステートメント

FILENAME ステートメントでは、EMAIL (SMTP)アクセス方式により、SMTP 電子メールインターフェイスを使用して SAS からプログラムで電子メールを送信できます。

FILE ステートメントで電子メールオプションを指定できます。FILE ステートメントで指定した電子メールオプションは、FILENAME ステートメントで指定した対応する電子メールオプションをオーバーライドします。

DATA ステップでは、FILE ステートメントを使用して電子メールのファイル参照名を出力先として定義した後、PUT ステートメントを使用してメッセージの本文を定義します。PUT ステートメントのディレクティブは、FILE ステートメントおよび FILENAME ステートメント内の他の電子メールオプションをオーバーライドします。

次の SAS ステートメントは、メールの送信に使用されます。

- FILENAME ステートメント、EMAIL (必須)では、SMTP(簡易メール転送プロトコル)電子メールインターフェイスを使用して SAS からプログラムで電子メールを送信できます。
- FILENAME ステートメントは、SAS ファイル参照名を外部ファイルまたは出力デバイスに関連付けたり、ファイル参照名と外部ファイルの関連付けを解除したり、外部ファイルの属性をリストしたりします。
- FILE ステートメントは、PUT ステートメントの現在の出力ファイルを指定します。
- PUT ステートメントは、SAS ログ、SAS 出力ウィンドウ、または最新の FILE ステートメントで指定された外部の場所に行を書き込みます。

SMTP 電子メールインターフェイスによるユーザーの認証方法

SMTP (簡易メール転送プロトコル)電子メールインターフェイスを使用して、SAS からプログラムで電子メールを送信できます。SMTP は、SAS が実行されるすべての動作環境で利用できます。SAS 電子メールサポートを使用して SMTP 電子メールを送信するには、SMTP をサポートするイントラネットまたはインターネット接続が必要です。

SMTP サーバーには、ログイン ID としてユーザー ID のみが必要な場合と、完全な電子メールアドレスが必要な場合があります。SAS SMTP 電子メールインターフェイスは、次の順序でユーザー ID を認証します。

1 EMAILHOST=システムオプションの USERID=オプションでユーザー ID が指定されている場合、SAS はこのユーザー ID を使用して認証します。

2 ユーザー ID が USERID=オプションで指定されていない場合、SAS は FILENAME ステートメント、EMAIL (SMTP アクセス方式)の FROM=オプションで指定されたユーザー ID を使用して認証します。

3 FILENAME ステートメント、EMAIL (SMTP アクセス方式)の FROM=オプションでユーザー ID が指定されていない場合、SAS は EMAILID システムオプションで指定されたユーザー ID を使用して認証します。

4 EMAILID=システムオプションでユーザー ID が指定されていない場合、SAS はオペレーティングシステムからユーザー ID を検索し、ユーザー ID の認証を試みます。

ユニバーサル一意識別子

ユニバーサル一意識別子とは

ユニバーサル一意識別子(UUID)は、日時情報とホストの IEEE ノードアドレスで構成される 128 ビットの識別子です。UUID は、SAS アプリケーションの行やその他のコンポーネントなどのオブジェクトを一意に識別する必要がある場合に役立ちます。たとえば、SAS がサーバーとして実行され、オブジェクトを複数のクライアントに同時に配布している場合、UUID を各オブジェクトに関連付けることができます。これにより、特定のクライアントと SAS が同じオブジェクトを参照することが保証されます。

オブジェクトスポーナーとは

オブジェクトスポーナーは、サーバー上で実行され、要求をリッスンするプログラムです。要求を受信すると、オブジェクトスポーナーは接続を受け入れ、接続が行われたポートまたはサービスに関連付けられたアクションを実行します。オブジェクトスポーナーは、要求元の SAS セッションの UUID を作成する UUID ジェネレーターデーモン(UUIDGEND)として構成できます。

UUID ジェネレーターデーモンは、次の場合には必要ありません。

- Windows 上で実行される SAS アプリケーション
- SAS バージョン 9.4M2 (以降)を実行している Linux 環境で実行される SAS アプリケーション

UUID ジェネレーターデーモンの定義

UUIDGEND の定義は、オブジェクトスポーナーを呼び出すときに指定するセットアップ構成ファイルに含まれています。この構成ファイルは、UUID 要求をリッスンするポートを識別し、また、構成ファイルは UUID ノードも識別します。

Windows 以外の動作環境に UUIDGEND をインストールする場合は、SAS テクニカルサポート(http://support.sas.com/techsup/contact/)に連絡して UUID ノードを取得してください。UUIDGEND が真に一意の UUID を保証するには、UUID ノードが UUIDGEND インストールごとに一意である必要があります。

Windows 以外の動作環境の UUIDGEND セットアップ構成ファイルの例を次に示します。

```
## Define our UUID Generator Daemon. Since this UUIDGEND is
## executing on a Linux host, we contacted SAS Technical
## Support to get the specified sasUUIDNode.
#
dn: sasSpawnercn=UUIDGEND,sascomponent=sasServer,cn=SAS,o=ABC Inc,c=US
objectClass: sasSpawner
sasSpawnercn: UUIDGEND
sasDomainName: unx.abc.com
sasMachineDNSName: medium.unx.abc.com
sasOperatorPassword: myPassword
sasOperatorPort: 6340
sasUUIDNode: 0123456789ab
sasUUIDPort: 6341
description: SAS Session UUID Generator Daemon on UNIX
```

Windows の UUIDGEND セットアップ構成ファイルの例を次に示します。

```
#
## Define our UUID Generator Daemon. Since this UUIDGEND is
## executing in a Windows operating environment, we do not need to specify
## the sasUUIDNode.
#
dn: sasSpawnercn=UUIDGEND,sascomponent=sasServer,cn=SAS,o=ABC Inc,
c=US
objectClass: sasSpawner
sasSpawnercn: UUIDGEND
sasDomainName: wnt.abc.com
sasMachineDNSName: little.wnt.abc.com
sasOperatorPassword: myPassword
sasOperatorPort: 6340
sasUUIDPort: 6341
description: SAS Session UUID Generator Daemon on XP
```

UUID ジェネレーターデーモンのインストール

セットアップ構成ファイルを作成したら、オブジェクトスポーナープログラム(objspawn)を起動し、次の構文でセットアップ構成ファイルを指定することで、UUIDGEND をインストールできます。

objspawn -configFile *filename*
configFile オプションは、-cf と省略できます。

filename は、UUIDGEND セットアップ構成ファイルへの完全修飾パスを指定します。空白が含まれるパス名は単一引用符または二重引用符で囲みます。Windows では、空白が含まれるパス名を二重引用符で囲みます。z/OS では、次のように構成ファイルを指定します。

//dsn:*myid.objspawn*.log (MVS ファイル)

//hfs:*filename*.ext (OpenEdition ファイル)

Windows では、**objspawn.exe** ファイルは ***SAS-installation-directory*\SASFoundation*SAS-version***ディレクトリにインストールされます。たとえば、一般的な Windows インストールでは、**objspawn.exe** ファイルは次のディレクトリにインストールされます。

```
C:\Program Files\SASHome\SASFoundation\9.4
```

UNIX では、objspawn ファイルは、インストールされている SAS ディレクトリの **utilities/bin** ディレクトリにインストールされます。

VMS 動作環境では、**OBJSPAWN_STARTUP.COM** ファイルはデタッチプロセスとして OBJSPAWN.COM ファイルを実行します。OBJSPAWN.COM ファイルはオブジェクトスポーナーを実行します。OBJSPAWN.COM ファイルには、オブジェクトスポーナーが開始される前にサイトで実行する必要がある次のコマンドも含まれています。

- 表示ノードを設定するコマンド
- 適切なバージョンのスポーナーを実行するコマンド
- テンプレート DCL ファイル(OBJSPAWN_TEMPLATE.COM)を指すプロセスレベルの論理名を定義するコマンド

OBJSPAWN_TEMPLATE.COM ファイルは、クライアントプロセスを実行するために必要なセットアップを実行します。オブジェクトスポーナーは、最初に論理名 SAS $OBJSPAWN_TEMPLATE が定義されているかどうかをチェックします。もしそうであれば、テンプレートファイル内のコマンドは、クライアントセッションの開始時に使用されるコマンドシーケンスの一部として実行されます。論理名を定義する必要はありません。

完全修飾ドメイン名(FQDN)

IP アドレスは簡単に変更される可能性があるため、ハードコーディングされた IP アドレスを含む SAS アプリケーションはメンテナンスの問題が発生しやすくなります。

このような問題を回避するには、IP アドレスよりも FQDN を使用することをお勧めします。TCP/IP プロトコルの一部である名前解決システムは、FQDN に関連付けられた IP アドレスを見つける役割を果たします。

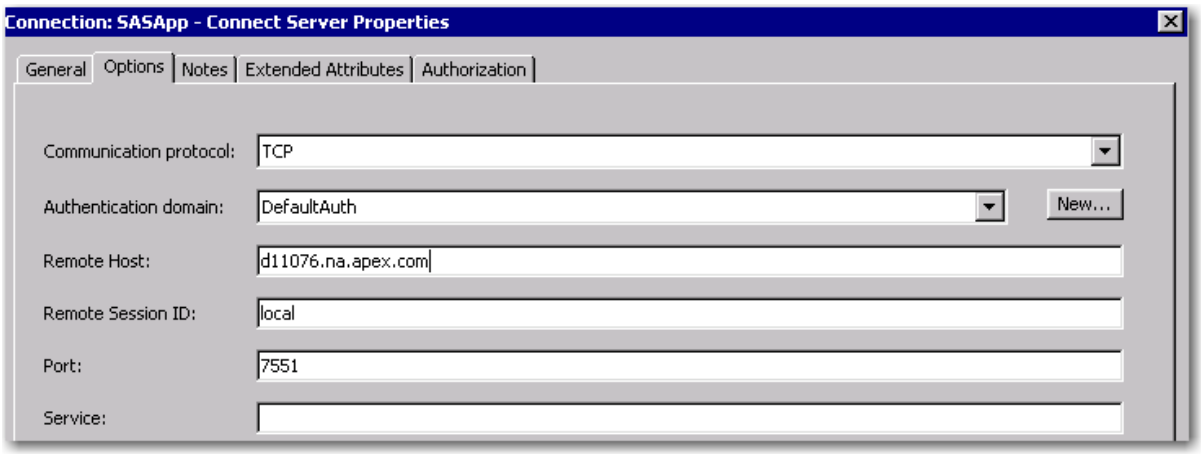
次の例では、一時停止されたリポジトリ内のクライアントアクティビティを 復元します。

```
PROC METAOPERATE
  SERVER="d6292.us.company.com"
  PORT=2222
  USERID="myuserid"
  PASSWORD="mypassword"
  PROTOCOL=BRIDGE

  ACTION=RESUME
   OPTIONS=""
  NOAUTOPAUSE;
```

IP アドレスが使用されており、コンピューターノード名に関連付けられている IP アドレスが変更されている場合、コードは不正確になります。

FQDN はコード内でそのまま維持できますが、基になる IP アドレスは予測できない結果を引き起こすことなく変更できます。TCP/IP 名前解決システムは、FQDN をそれに関連付けられた IP アドレスに自動的に解決します。

SAS GUI アプリケーションで指定される FQDN の例を次に示します。



完全な FQDN である d11076.na.apex.com は、SAS 管理コンソールの **Connect Server プロパティ**ウィンドウの**リモートホスト**フィールドで指定されます。

一部の SAS 製品では、コンピューター名の長さに制限が設けられています。

次のコードは、SAS メニュー変数に割り当てられる FQDN の例です。

```
%let sashost=hrmach1.dorg.com;
rsubmit sashost.sasport;
```

FQDN は 8 文字を超えるため、FQDN を RSUBMIT ステートメントで使用される SAS マクロ変数に割り当てる必要があります。

例: SMTP インターフェイスを使用して電子メールを送信する

SMTP インターフェイスを使用して電子メールを送信する

この例では、SMTP(Simple Mail Transfer Protocol)メールインターフェイスを介して、SAS からメールをプログラムによって送信する方法を示します。SMTP メール送信の詳細については、“SMTP 電子メールを制御する SAS 言語要素” (808 ページ)を参照してください。

SAS セッションから SMTP メールを送信する方法の例については、“”例”” (*SAS グローバルステートメント: リファレンス*)を参照してください。

```
options emailsys=smtp                  /* 1 */
      emailhost="mailhost.example.com"   /* 2 */
      emailport=25;                  /* 3 */

filename mymail email                  /* 4 */
  to=("jtk@example.com")               /* 5 */
  from="bmccoy@example.com"            /* 6 */
  subject="Crew Health";


data _null_;                           /* 7 */
  file mymail;
  put "Hello J,";
  put "Here is the health report.";
  put "This message was sent automatically from my SAS program.";
  put "The job finished successfully.";
  put " ";
  put "Regards,";
  put "B";
run;
```

1 EMAILSYS システムオプションで SMTP を指定します。SAS Viya 環境のほとんどでは、SMTP がデフォルト値です。
2 EMAILHOST=システムオプションに SMTP ホスト名を指定します。
3 EMAILPORT=システムオプションに SMTP ポートを指定します。
4 FILENAME ステートメント: EMAIL アクセス方式を指定して、電子メールのファイル参照名を割り当てます。
5 FILENAME ステートメントの TO オプションで受信者のメールアドレスを指定します。
6 FILENAME ステートメントの FROM オプションに送信者のメールアドレスを指定します。
7 電子メールの本文を書き込みます。

アウトプット 36.1 SMTP インターフェイスを使用して電子メールを送信するためのログ出力

```
NOTE: The file MYMAIL is:
      E-Mail Access Device
Message sent
      To:          "jtk@example.com"
      Cc:
      Bcc:
      Subject:     Crew Health
      Attachments:
NOTE: 7 records were written to the file MYMAIL.
      The minimum record length was 1.
      The maximum record length was 56.
NOTE: DATA statement used (Total process time):
      real time           0.16 seconds
      cpu time            0.03 seconds
```

FILENAME ステートメント: EMAIL (SMTP)アクセス方式を使用した電子メール送信の詳細については、"FILENAME ステートメント: EMAIL (SMTP)アクセスメソッド" (*SAS グローバルステートメント: リファレンス*)を参照してください。

関連項目

ステートメント

- "メールオプション" (*SAS グローバルステートメント: リファレンス*)

システムオプション:

- "EMAILAUTHPROTOCOL= システムオプション" (*SAS システムオプション: リファレンス*)
- "EMAILID= システムオプション" (*SAS システムオプション: リファレンス*)
- "EMAILPORT システムオプション" (*SAS システムオプション: リファレンス*)
- "EMAILPW= システムオプション" (*SAS システムオプション: リファレンス*)

デフォルトの SAS Data and AI Studio(2026.06 より前は SAS Studio という名前でした)

結果のコピーと関連するコードおよびログファイルを電子メールで他のユーザーに送信できます。送信できるファイルには、HTML5、RTF、および PDF 形式の結果、および結果に関連付けられたコードおよびログファイルが含まれます。コードは SAS プログラムファイルとして送信され、ログおよびプログラム要約ファイルは HTML5 ファイルとして送信されます。電子メールでファイルを送信するには、SMTP サーバーにアクセスできる必要があります。詳細については、サイト管理者に問い合わせてください。

SAS Data and AI Studio(2026.06 より前は SAS Studio という名前)環境からメールを送信する手順については、"別のユーザーへの結果の送信" (*SAS Studio: ユーザーガイド*)を参照してください。

例: CAS セッションの UUID を取得する

コード例

```
cas casauto;          /* 1 */

proc cas;
print uuid("casauto");  /* 2 */
run;
```

1 CAS ステートメントを指定して CAS セッションを開始します。

2 CAS プロシジャと UUID 関数を指定して、Casauto セッションに割り当てられた UUID を出力します。

例のコード 36.1　CAS セッションの UUID 取得のログ出力

```
NOTE: The session CASAUTO connected successfully to Cloud Analytic Services
      controller.sas-cas-server-default using port 5570.
      The UUID is ec849c3b-ba1c-da4b-87eb-e0921fdd1656.
      The user is sasdemo and the active caslib is CASUSER(sasdemo).
```

関連項目

- “UUIDGEN 関数” (*SAS 関数と CALL ルーチン: リファレンス*)

8 部

付録

付録 1

例で使用されるデータセット

例で使用されるデータセット

Animal

```
data animal;
input common $ animal $;
datalines;
a Ant
b Bird
c Cat
d Dog
e Eagle
f Frog
;
```

AnimalDupes

```
data animalDupes;
input common $ animal $;
datalines;
a Ant
a Ape
b Bird
c Cat
d Dog
e Eagle
;
```

AnimalMissing

```
data animalMissing;
input common $ animal $;
datalines;
a Ant
c Cat
d Dog
e Eagle
;
```

CarSales

```
data carSales;
input transID $1-9 make $11-19 model $21-38;
datalines;
CSH203073 Chevrolet Aveo 4dr
CCA460564 Chevrolet Aveo LS 4dr htc
DEB848135 Honda     Civic DX 2dr
CCA762350 Honda     Insight 2dr
CCA314633 Hyundai   Accent 2dr hatch
CSH118553 Hyundai   Accent GL 4dr
CCA295057 Hyundai   Accent GT 2dr htc
CSH285627 Kia       Rio 4dr auto
DEB898883 Kia       Rio 4dr manual
CCA803483 Kia       Rio Cinco
DEB661568 Mazda     MX-5 Miata LS
CSH36345  Mazda     MX-5 Miata
CCA562255 Scion     xA 4dr hatch
CCA651575 Scion     xB
DEB322009 Toyota    Echo 2dr auto
CSH607254 Toyota    Echo 2dr manual
CSH879119 Toyota    Echo 4dr
CSH230940 Toyota    MR2 Spyder
;
```

CarsSmall

```
data carsSmall;
input make $1-9  model $11-39 type $41-46  driveTrain $48-52  weight $54-57 length 59-61
makeModelDrive $63-120;
datalines;
Chevrolet Aveo 4dr                Sedan  Front 2370 167 Chevrolet Aveo 4dr - Front wheel
drive
Chevrolet Aveo LS 4dr hatch          Sedan  Front 2348 153 Chevrolet Aveo LS 4dr hatch -
Front wheel drive
Honda     Civic DX 2dr            Sedan  Front 2432 175 Honda Civic DX 2dr - Front wheel
drive
Honda     Insight 2dr (gas/electric)    Hybrid Front 1850 155 Honda Insight 2dr (gas/electric)
- Front wheel drive
Hyundai   Accent 2dr hatch          Sedan  Front 2255 167 Hyundai Accent 2dr hatch - Front
wheel drive
Hyundai   Accent GL 4dr             Sedan  Front 2290 167 Hyundai Accent GL 4dr - Front
wheel drive
Hyundai   Accent GT 2dr hatch         Sedan  Front 2339 167 Hyundai Accent GT 2dr hatch -
Front wheel drive
Kia Rio   4dr auto                Sedan  Front 2458 167 Kia Rio 4dr auto - Front wheel drive
```

```
Kia Rio   4dr manual                Sedan  Front 2403 167 Kia Rio 4dr manual - Front wheel
drive
Kia Rio   Cinco                     Wagon  Front 2447 167 Kia Rio Cinco - Front wheel drive
Mazda     MX-5 Miata LS convertible 2dr Sports Rear  2387 156 Mazda MX-5 Miata LS
convertible 2dr - Rear wheel drive
Mazda     MX-5 Miata convertible 2dr    Sports Rear  2387 156 Mazda MX-5 Miata
convertible 2dr - Rear wheel drive
Scion     xA 4dr hatch              Sedan  Front 2340 154 Scion xA 4dr hatch - Front wheel
drive
Scion     xB                        Wagon  Front 2425 155 Scion xB - Front wheel drive
Toyota    Echo 2dr auto             Sedan  Front 2085 163 Toyota Echo 2dr auto - Front wheel
drive
Toyota    Echo 2dr manual           Sedan  Front 2035 163 Toyota Echo 2dr manual - Front
wheel drive
Toyota    Echo 4dr                  Sedan  Front 2055 163 Toyota Echo 4dr - Front wheel drive
Toyota    MR2 Spyder convertible 2dr    Sports Rear  2195 153 Toyota MR2 Spyder
convertible 2dr - Rear wheel drive
;
```

Class

```
data class;
input name $ age height weight;
format weight comma8.;
format height comma8.;
datalines;
Alice   13 56.5   84.00
Barbara 13 65.3   98.00
Carol   14 62.8  102.50
Jane    12 59.8   84.50
Janet   15 62.5  112.50
Joyce   11 51.3   50.50
Judy    14 64.3   90.00
Louise  12 56.3   77.00
Mary    15 66.5  112.00
;
```

Classfit

```
data classfit;
input name $ age height weight predict;
format weight comma8.2;
format predict comma8.1;
label weight="Weight in Pounds";
datalines;
Janet   15 62.5 112.5 100.662
Mary    15 66.5 112.0 116.259
Philip  16 72.0 112.0 137.703
Ronald  15 67.0 133.0 118.208
William 15 66.5 150.0 116.259
```

```
;
```

Inventory

```
data Inventory;
input partNumber $ partName $;
datalines;
K89R seal
M4J7 sander
LK43 filter
MN21 brace
BC85 clamp
NCF3 valve
KJ66 cutter
UYN7 rod
JD03 switch
BV1E timer
;
```

InventoryAdd

```
data InventoryAdd;
input partNumber $ partName $ newStock newPrice;
format newPrice dollar12.2;
datalines;
K89R seal 6 247.50
AA11 hammer 55 32.26
BB22 wrench 21 17.35
KJ66 cutter 10 24.50
CC33 socket 7 22.19
BV1E timer 30 36.50
;
```

One

```
data one;
  input ID state $;
 datalines;
1 AZ
2 MA
3 WA
4 WI
;
```

Many

```
data many;
  input ID city $ state $;
  datalines;
1 Phoenix   Ariz
2 Boston    Mass
2 Foxboro   Mass
3 Olympia   Mass
3 Seattle   Wash
3 Spokane   Wash
4 Madison   Wis
4 Milwaukee Wis
4 Madison   Wis
4 Hurley    Wis
;
```

Master

```
data master;
input common $ animal $ plant $;
datalines;
a Ant Apple
b Bird Banana
c Cat Coconut
d Dog Dewberry
e Eagle Eggplant
f Frog Fig
;
```

Minerals

```
data minerals;
input common $ plant $ mineral $;
datalines;
a Apricot Amethyst
b Barley Beryl
c Cactus .
e . .
f Fennel .
g Grape Garnet
;
```

Plant

```
data plant;
input common $ plant $;
datalines;
a Apple
b Banana
c Coconut
d Dewberry
e Eggplant
f Fig
;
```

PlantDupes

```
data plantDupes;
input common $ plant $;
datalines;
a Apple
b Banana
c Coconut
c Celery
d Dewberry
e Eggplant
;
```

PlantG

```
data plantG;
input common $ plant $;
datalines;
a Apple
b Banana
c Coconut
d Dewberry
e Eggplant
g Fig
;
```

PlantMissing

```
data plantMissing;
```

```
input common $ plant $;
datalines;
a Apple
b Banana
c Coconut
;
```

PlantMissing2

```
data plantMissing2;
input common $ plant $;
datalines;
a Apple
b Banana
c Coconut
e Eggplant
f Fig
;
```

PlantNew

```
data plantNew;
input common $ plant $;
datalines;
a Apricot
b Barley
c Cactus
d Date
e Escarole
f Fennel
;
```

PlantNewDupes

```
data plantNewDupes;
input common $ plant $;
datalines;
a Apricot
b Barley
c Cactus
d Date
d Dill
e Escarole
f Fennel
;
```

Product_List

```
data product_list;
input Product_Id Product_Name $14-49 Supplier_ID;
datalines;
240200100101 Grandslam Staff Tour Mhl Golf Gloves 3808
210200100017 Sweatshirt Children's O-Neck 3298
240400200022 Aftm 95 Vf Long Bg-65 White 1280
230100100017 Men's Jacket Rem 50
210200300006 Fleece Cuff Pant Kid'S 1303
210200500002 Children's Mitten 772
210200700016 Strap Pants BBO 798
210201000050 Kid Children's T-Shirt 2963
210200100009 Kids Sweat Round Neck,Large Logo 3298
210201000067 Logo Coord.Children's Sweatshirt 2963
220100100019 Fit Racing Cap 1303
220100100025 Knit Hat 1303
220100300001 Fleece Jacket Compass 772
220200200036 Soft Astro Men's Running Shoes 1747
230100100015 Men's Jacket Caians 50
230100500004 Backpack Flag, 6,5x9 Cm. 316
210200500006 Rain Suit, Plain w/backpack Jacket 772
230100500006 Collapsible Water Can 316
224040020000 Bat 5-Ply 3808
220200200035 Soft Alta Plus Women's Indoor Shoes 1747
240400200066 Memhis 350,Yellow Medium, 6-pack 1280
240200100081 Extreme Distance 90 3-pack 3808
;
```

Quarter1、Quarter2、Quarter3、Quarter4

```
data quarter1;
length mileage 4;
input account mileage;
datalines;
1 932
2 563
;
data quarter2;
length mileage 8;
input account mileage;
datalines;
1 1288
2 1087
;
data quarter3;
length mileage 6;
input account mileage;
```

```
datalines;
1 2781
2 1990
;
data quarter4;
length mileage 6;
input account mileage;
datalines;
1 3278
2 2209
;
```

Sales

```
data Sales;
input partNumber $ partName $ salesPerson $;
datalines;
NCF3 valve  JN
BV1E timer  JN
LK43 filter KM
K89R seal   SJ
LK43 filter JN
M4J7 sander KM
BV1E timer  KM
;
```

Sales2019

```
data Sales2019;
input partNumber $ lastSoldDate mmddyy10.;
format lastSoldDate mmddyy10.;
datalines;
BC85 10/15/2019
BV1E 10/23/2019
KJ66 11/11/2019
LK43 09/12/2019
MN21 09/13/2019
NCF3 07/24/2019
UYN7 12/11/2019
;
```

Supplier

```
data Supplier;
input Supplier_ID Supplier_Name $6-32 Supplier_Address $34-52 Country $54-55;
```

```
datalines;
50   Scandinavian Clothing A/S   Kr. Augusts Gate 13 NO
316  Prime Sports Ltd            9 Carlisle Place    GB
755  Top Sports                  Jernbanegade 45     DK
772  AllSeasons Outdoor Clothing 553 Cliffview Dr    US
798  Sportico                    C. Barquillo 1      ES
1280 British Sports Ltd          85 Station Street   GB
1303 Eclipse Inc                 1218 Carriole Ct    US
1684 Magnifico Sports            Rua Costa Pinto 2   PT
1747 Pro Sportswear Inc          2434 Edgebrook Dr   US
3298 A Team Sports               2687 Julie Ann Ct   US
3808 Carolina Sports             3860 Grand Ave      US
;
```

Table1

```
data Table1;
  set sashelp.class(where=(age=14));
run;
```

Table2

```
data Table2;
  set sashelp.classfit(where=(age=14));
  drop uppermean lower upper;
run;
proc sort data=Table2; by name; run;
```

Year1

```
data Year1;
input date;
datalines;
2009
2010
2011
2012
;
```

Year2

```
data Year2;
input date;
```

```
datalines;
2010
2011
2012
2013
2014
;
```

付録 2

MODIFY ステートメントと KEY= オプションの使用

MODIFY ステートメントと KEY=オプションを使用したデータの更新

MODIFY ステートメントまたは“SET ステートメント” (*SAS DATA ステップステートメント: リファレンス*)のいずれかで KEY=オプションを使用して、SAS データセットを更新できます。また、“SET ステートメント” (*SAS DATA ステップステートメント: リファレンス*)、MERGE ステートメント、および UPDATE ステートメントを使用して、データを変更および結合することもできます。MODIFY ステートメントで KEY=オプションを使用すると、次のような利点があります。

- MODIFY ステートメントで使用するディスク領域は、SET、MERGE または UPDATE ステートメントよりも少なくなります。
- MODIFY ステートメントの KEY=オプションを使用すると、インデックス付けを利用できます。

SAS データセットの結合の詳細については、“変更” (526 ページ)を参照してください。

定義

この付録で使用される次の SAS 用語と定義に注意してください。

data set
SAS で処理可能な行(SAS オブザベーション)と列(SAS 変数)のテーブルとして編成されるディスクリプター情報とデータ値で構成される SAS ファイル。

IORC
KEY=オプションを指定した MODIFY ステートメントまたは SET ステートメントを使用するときに SAS によって作成される自動変数。_IORC_の値は、SAS データセットのオブザベーションに対して実行された最新の I/O 操作のステータスを示す数値のリターンコードです。リターンコードは、一致するオブザベーションの取得が成功したかどうかを示します。

表 A2.1 _IORC_のリターンコード値

コード	値
0	successful execution
-1	end-of-file error
その他すべての値	non-match

通常、_IORC_は、自動呼び出しマクロ%SYSRC と組み合わせて使用します。これにより、I/O 操作の潜在的な結果を説明するニーモニック名を指定できます。

注: バージョン 7 では、IORCMSG 関数は、_IORC_の現在値に関連付けられたフォーマット済みエラーメッセージを返します。

index
SAS データセットのインデックスを定義するときに作成されるファイル。

program data vector (PDV)
SAS が一度に 1 つのオブザベーションずつデータセットを構築するメモリの物理領域。プログラムが実行されると、ソフトウェアは入力バッファーから値を読み取るか、SAS 言語ステートメントを実行して値を作成します。値は PDV の適切な変数に割り当てられます。ここから、ソフトウェアは値を単一のオブザベーションとして SAS データセットに書き込みます。

%SYSRC
最後に実行された KEY=オプション付きの MODIFY ステートメントまたは SET ステートメントによって発生した特定の I/O エラー状態を簡単にテストできる自動呼び出しマクロ。%SYSRC は、渡されたニーモニック文字列に対応する数値を返します。ニーモニックは、_IORC_自動変数の数値に対応するリテラルです。

SAS は自動呼び出しマクロのライブラリを各サイトに提供します。自動呼び出し機能を使用すると、同じ SAS プログラム内でマクロを事前に定義していなくてもマクロを呼び出すことができます。自動呼び出し機能を使用するには、MAUTOSOURCE システムオプションを指定します。

注: DATA=オプションは、マスターデータセットを指定するために PRINT プロシジャで使用されます。DATA=を指定しない場合、PROC PRINT は、出力用に開かれた最後に作成されたデータセットを表示します。最後に作成されたデータセットは、今変更されたデータセットではない可能性があることに注意してください。

view
データ値を取得するために必要な情報のみが含まれます。データは別のファイルから取得されます。ビューには次の 3 種類があります。

DATA step view
1 つ以上の SAS データセットまたは外部ファイルから読み取ることができるコンパイル済み DATA ステッププログラム。

SAS/ACCESS view
他のソフトウェア製品によってフォーマットされたデータを定義します。SAS/ACCESS ビューが処理されるとき、アクセスするデータは元の形式のままです。

PROC SQL view
1 つ以上のデータファイルまたはビュー、あるいは SQL プロシジャパススルー機能から値を取得するコンパイル済みクエリ。

MODIFY ステートメントについて

MODIFY ステートメントは、データセットのコピーを作成せずに、既存のデータセットにあるオブザベーションの置き換え、削除、追加を行います。これは*インプレース更新*と呼ばれます。MODIFY ステートメントは、更新のために開いているデータセットのコピーを作成しないため、処理に使用されるディスク領域は、MERGE、SET、UPDATE の各ステートメントよりも少なくなります。これらのステートメントは新しいデータセットを作成します。

変更されたデータセットで KEEP、RENAME、DROP などのデータセットオプションが使用される場合、それらのオプションは処理中にのみ有効です。更新モードで開いた SAS データセットのディスクリプター部分は変更できません。

ディスクリプター部分を含む SAS データセットの各部分の詳細については、“SAS データセットの各部分” (300 ページ)を参照してください。

MODIFY ステートメントで参照されるデータセットは、DATA ステートメントでも参照される必要があります。そして、DATA ステップのロジックに基づいて、新しいオブザベーションの置換、削除、追加を実行します。たとえば、次の単純な DATA ステップでは、MODIFY ステートメントを使用して、変数 Recdate のすべてのオブザベーションに含まれる日付値を現在の日付に置き換えることで、データセット Invty.Stock を更新します。

```
data invty.stock;
```

```
  modify invty.stock; recdate=today();
run;
```

SAS が前述の DATA ステップを処理すると、次のことが起こります。

- MODIFY ステートメントは、更新処理のために SAS データセット Invty.Stock を開きます。
- 最初のオブザベーションがデータセットから読み取られ、値が PDV に書き込まれます。(変数 Recdate はデータセット Invty.Stock に存在します。)
- 変数 Recdate の値は TODAY 関数の結果で置き換えられます。
- PDV の値はデータセットの値を置き換えます。
- このプロセスは、ファイルの終わりのマーカーに到達するまで、順次アクセス方式を使用してオブザベーションごとに繰り返されます。

処理をさらに制御するために、DATA ステップの実行に MODIFY ステートメントと組み合わせて次のステートメントを含めることができます。

- OUTPUT ステートメント: 現在のオブザベーションを新しいオブザベーションとしてデータセットの最後に追加します。MODIFY ステートメントで指定したデータセットは、DATA ステートメントでも出力データセットとして指定する必要があります。
- REMOVE ステートメントデータセット。
- REPLACE ステートメント: 現在のオブザベーションを同じ物理的な場所に書き込みます。つまり、データセット内で置き換えます。DATA ステップの下部にある暗黙的な REPLACE ステートメントがデフォルトアクションになります。

MODIFY ステートメントの処理は、DATA ステップにこれらのステートメントが含まれているかどうかによって異なります。

- DATA ステップで REPLACE、REMOVE、OUTPUT ステートメントが指定されていない場合、ソフトウェアは、REPLACE ステートメントが DATA ステップの最後のステートメントであるかのように、現在のオブザベーションをデータセットの元の場所に書き込みます。つまり、MODIFY ステートメントは DATA ステップの最後に REPLACE ステートメントを生成します。これは暗黙的な REPLACE と呼ばれます。
- REPLACE、REMOVE、または OUTPUT ステートメントが含まれている場合、暗黙的な REPLACE がオーバーライドされます。

REPLACE、REMOVE、および OUTPUT ステートメントは互いに独立しています。複数のステートメントを同じオブザベーションに適用できます。OUTPUT ステートメントと REPLACE または REMOVE ステートメントの両方を同じオブザベーションに対して実行する場合、インデックス内の適切な位置を維持するために OUTPUT ステートメントが最後に実行されるようにしてください。

MODIFY ステートメントは、次の 4 つの異なるアクセス方式をサポートします。

- 順次アクセス
- BY ステートメントを使用した一致アクセス
- POINT=を使用したオブザベーション番号による直接(ランダム)アクセス
- KEY=を使用したインデックス付き値による直接アクセス

順次アクセス

順次アクセスは、MODIFY ステートメントを使用する最も単純な処理形式です。順次アクセスは、他のアクセス方式に比べて制御が低くなりますが、データセット内のすべてのオブザベーションを更新するための最も迅速な方式を提供します。順次アクセスの構文は次のとおりです。

MODIFY*master-data-set*<*(data-set-option(s))*><NOBS=*variable*><END=*variable*>

次のコードでは、SAS は Master データセット内の各オブザベーションに順次アクセスし、変数 MOD の値 u を含むオブザベーションを検索します。オブザベーションが見つかると、変数 x の値は 1 に置き換えられます。暗黙的な REPLACE を実行すると、オブザベーションが更新された値で書き換えられます。変数 x は、Master データセット内に存在する必要があります。MODIFY ステートメントはデータセットを更新モードで開きます。更新のために開いたデータセットのヘッダー情報は変更できません。

```
data master;
  modify master;
  if mod='u' then x=1;
run;
```

BY ステートメントを使用した一致アクセス

一致アクセスでは、マスターデータセット内の 1 つ以上の BY 変数の値を、トランザクションデータセットと呼ばれる 2 番目のデータセットにある同じ変数と一致させます。一致アクセスの構文は次のとおりです。

MODIFY*master-data-set*<*(data-set-option(s))*>*transaction-data-set*<*(data-set-option(s))*><NOBS=*variable*><END=*variable*><UPDATEMODE=<MISSINGCHECK>NOMISSINGCHECK>;**BY***by-variable(s)*;

BY ステートメントが必要です。指定した変数は、マスターデータセットとトランザクションデータセットの両方に存在する必要があります。MODIFY を BY ステートメントとともに使用する場合、データセットが並べ替え順である必要がないことを除き、UPDATE ステートメントに適用されるルールが MODIFY にも適用されます。

MODIFY ステートメントは次のように実行されます。

1 MODIFY ステートメントは、更新処理のために指定されたデータセットを開きます。

2 MODIFY ステートメントは、トランザクションデータセットからオブザベーションをメモリの一時保存場所に読み込みます。

3 次に、MODIFY ステートメントは WHERE ステートメントを生成し、トランザクションオブザベーションからの BY 変数(例: Partno)の値を指定して、マスターデータセットから一致する BY 変数値を含むオブザベーションを検索してフェッチします。このオブザベーションは PDV に配置されます。

4 一時ストレージにあるトランザクションオブザベーションからの値は、マスターオブザベーションの PDV にある同じ名前の変数の値をオーバーレイするために使用されます。

一致アクセスには、次の動作が適用されます。

- BY ステートメントで指定した変数と一致する場合、マスターデータセットにある同じ名前の変数はすべてトランザクションデータセットの値で自動的に更新されます。
- マスターデータセットに BY 変数の重複値が存在する場合、その BY グループの最初のオブザベーションのみが更新され、BY 値が重複するオブザベーションは無視されます。これは、SAS が BY 変数のトランザクションデータセット値を含む WHERE ステートメントを生成することにより、マスターデータセットのオブザベーションを検索するためです。WHERE ステートメントは、常に、マスターデータセットで更新するために最初に出現した BY グループを返します。
- トランザクションデータセットの BY グループに重複が存在する場合、マスターデータセットの最初の一致オブザベーションに対して 1 つずつ更新が適用されます。重複の値を増やすには、累積ステートメントを使用する必要があります。それ以外の場合は、最後の重複の値がマスターデータセットの一致オブザベーションに適用されます。
- WHERE ステートメントが生成されるため、マスターデータセットとトランザクションデータセットのどちらも BY 変数で並べ替える必要はありません。ただし、両方のデータセットを並べ替えることにより、マスターデータセットからオブザベーションを取得するための I/O を大幅に削減できます。マスターデータセットに BY 変数のインデックスが付けられている場合は、生成される WHERE ステートメントでそのインデックスを使用できます。
- UPDATEMODE=MISSINGCHECK がデフォルトの動作である場合、トランザクションデータセットからの欠損値によってマスターデータセットを更新できます。オプション UPDATEMODE=NOMISSINGCHECK が現在のリリースで使用できない場合、マスターデータセットのオブザベーションはトランザクションデータセットのオブザベーションに含まれる特殊欠損値によって更新できます。数値データの場合、欠損値の種類を区別できます。つまり、最大 27 文字(アンダースコア(_)と文字 A から Z)を指定して、さまざまな種類の欠損値を指定できます。もう 1 つの方法は、RENAME=データセットオプションを使用してトランザクションデータセットにある変数の名前を変更し、マスターデータセットの変数と名前が変更されたトランザクションデータセットの変数を同じように設定できるようにします。

次の例では、一致アクセス方式を使用して、トランザクションデータセット Work.Addinv からの情報を使用してマスターデータセット Invty.Stock を更新し、在庫を受け取った日付を更新します。

```
data invty.stock;
  modify invty.stock addinv;
  by partno; recdate=today();
  instock=instock + nwstock;
run;
```

トランザクションデータセットまたはマスターデータセットのいずれかを並べ替えたり、インデックス付けしたりする必要はありませんが、次の操作を行うことでパフォーマンスを向上させることができます。

- BY 変数として使用される変数のマスターデータセットのインデックスを作成する
- 両方のデータセットを BY 変数順に並べ替える

一致アクセスと UPDATE ステートメントの比較

一致アクセスを行う MODIFY ステートメントは、UPDATE ステートメントとほぼ同じです。マスターデータセットとトランザクションデータセットの両方が必要であり、両方とも BY ステートメントが必要です。MODIFY ステートメントと UPDATE ステートメントの比較は次のとおりです。

- MODIFY ステートメントは、MODIFY および DATA ステートメントで指定された SAS データセットを更新モードで開きます。ただし、UPDATE ステートメントは指定された SAS データセットを入力用に開き、DATA ステートメントで指定された SAS データセットは出力用に開きます。
- MODIFY ステートメントでは、SAS データセットをインプレース更新します。UPDATE ステートメントを使用すると、元の SAS データセットが更新されたバージョンのコピーに置き換えられます。
- MODIFY ステートメントを実行すると、DATA ステップの反復時に、暗黙的な REPLACE ステートメントが実行されます。UPDATE では、DATA ステップの反復時に、暗黙的な OUTPUT ステートメントが実行されます。

次の例では、暗黙的な REPLACE ステートメントと暗黙的な OUTPUT ステートメントの違いを示します。この例では、Master と Trans という 2 つのデータセットを使用します。それぞれの DATA ステップを次に示します。

```
data master;
  input ssn : $11. nickname $;
datalines;
134-56-9094 Megan
160-58-1223 Kathryn
161-60-5881 Joshua
;

data trans;
  input ssn : $11. nickname $;
datalines;
134-56-9094 Meg
142-67-9888 Bill
160-58-1223 Kate
161-60-5881 .
;
```

以前に並べ替えたデータを処理するには、UPDATE ステートメントを使用します。UPDATE ステートメントは、暗黙的な OUTPUT ステートメントを生成します。OUTPUT ステートメントは、現在のプログラムデータベクトルの値をデータセットの最後に追加します。これは、BY 変数で一致があるかどうかに関係なく発生します。

```
data master;
  update master trans
  updatemode=nomissingcheck;
```

```
  by ssn;
run;
```

ただし、次に示すように、UPDATE ステートメントを MODIFY ステートメントに変更すると、失敗します。生成された WHERE ステートメントでは、ssn 142-67-9888 に対する一致が見つからず、DATA ステップの下部にある暗黙的な REPLACE ステートメントは、取得できなかったオブザベーションを置き換えようとします。

```
data master;
  modify master trans
  updatemode=nomissingcheck;
  by ssn;
run;
```

SAS ログは、Trans データセットの 2 番目のオブザベーションに対する一致が Master データセットに存在しないため、自動変数_IORC_の値 **1230013** を反映します。エラーが発生したため、自動変数_ERROR_の値は **1** に設定されます。

```
ERROR: The TRANSACTION data set observation does not exist on the MASTER data set.
ERROR: No matching observation was found in MASTER data set.
ssn=142-67-9888 nickname=Bill FIRST.ssn=1 LAST.ssn=1 _ERROR_=1 _IORC_=1230013 _N_=2
NOTE: The SAS System stopped processing this step because of errors. NOTE: There were 1 observations
read from the dataset WORK.MASTER.
NOTE: The data set WORK.MASTER has been updated. There were 1 observations rewritten,    0 observations
added and 0 observations deleted.
NOTE: There were 3 observations read from the dataset WORK.TRANS.
```

不一致レコードに対する暗黙的な REPLACE ステートメントによって引き起こされるエラーを防ぐには、条件付き IF ステートメントを使用して、一致が見つかった場合にのみデータが置換されるようにします。自動変数_IORC_は、I/O 操作が成功すると値 0 を保持します。

```
data master;
  modify master trans
  updatemode=nomissingcheck;
  by ssn;
  if _iorc_=0 then replace;
run;
```

不一致レコードが SAS ログに表示されないようにするには、自動変数_ERROR_をゼロにリセットします。(エラーチェックについてはこの記事で後述します。)

```
data master;
  modify master trans
  updatemode=nomissingcheck;
  by ssn;
  if _iorc_=0 then replace;
  else _error_=0;
run;
```

POINT=を使用したオブザベーション番号による直接(ランダム)アクセス

オブザベーション番号による直接(ランダム)アクセスでは、POINT=を使用して更新対象のオブザベーションを識別する必要があります。POINT=には、別のデータソース(マスターデータセットではない)からの変数、または DATA ステップで定義した変数を指定します。この変数の値は、マスターデータセットの中で変更するオブザベーションの番号になります。MODIFY は、POINT=変数の値を使用して、データセット内の変更対象のオブザベーションを取得します。オブザベーション番号による直接(ランダム)アクセスの構文は次のとおりです。

MODIFY *master-data-set* <(*data-set-options*)> <NOBS=*variable*> POINT=*variable*;

オブザベーション番号による直接アクセスを説明するために、Newp という名前のデータセットがあると仮定します。Newp には、ツール会社のマスターデータセット Invty.Stock にある各ツールのオブザベーション番号を含む変数 Tool_Obs と、各ツールの新価格を含む変数 Newprice が含まれています。

次の例では、データセット Newp の情報を使用して、データセット Invty.Stock を更新します。Newp は SET ステートメントで指定され、Tool_Obs 変数と Newprice 変数に提供される値が読み取られます。変数 Tool_Obs は、POINT=オプションの値として指定されます。変数 Newprice は、Invty.Stock 内の PRICE の既存の値を置き換えるために、割り当てステートメントで指定されます。SET ステートメントを実行すると、Tool_Obs の値が Newp から読み取られ、PDV に配置され、その後、MODIFY ステートメントで使用されて Invty.Stock からオブザベーションが直接取得されます。Tool_Obs 内のオブザベーション番号は、オブザベーションを直接取得できるようにするためのキーとして使用されます。

```
data invty.stock;
  set newp;
  modify invty.stock point=tool_obs;
  price=newprice;
  recdate=today():
run;
```

KEY=を使用したインデックス値による直接アクセス

インデックス値による直接アクセスでは、KEY=を使用して既存のインデックスを指定する必要があります。次に、ソフトウェアはインデックス値に基づいてインデックスを介してデータセットのオブザベーションを取得します。インデックス値による直接アクセスの構文は次のとおりです。

MODIFY *master-data-set* <(*data-set-options*)> KEY=*index* </ UNIQUE> <NOBS=*variable*> <END=*variable*>;

インデックス値による直接アクセスにより、別の SAS データセットなどのセカンダリデータソースからルックアップ値を提供できます。SET ステートメントを使用してオブザベーションが読み取られ、ルックアップ値が提供され、その値がキーとして使用され、マスターデータセットを検索してオブザベーションが特定されます。検索は、そのデータセット用に作成されたインデックスによって実行されます。KEY=オプションを使用してインデックス名を指定します。オブザベーションが見つかったら、暗黙的な REPLACE の前に、変数に新しい値を割り当て、オブザベーションに対して他の処理を実行できます。オブザベーションが見つからない場合、OUTPUT などの適切なアクションが実行されます。ルックアップ値を提供する SAS データセット(SET ステートメントで指定)には、KEY=オプションで指定された名前と同じ名前が含まれている必要があります。

次の例では、KEY=オプションを使用して、データセット Work.Addinv の変数 Partno の値とデータセット Invty.Stock の変数 Partno のインデックス付き値を一致させることにより、取得するオブザベーションを識別するためのインデックスを指定します。

```
data invty.stock;
  set addinv;
  modify invty.stock key=partno;
  if _iorc_=0 then do;
    instock=instock+nwstock;
    recdate=today();
    replace;
  end;
  else _error_=0;
run;
```

BY ステートメントを使用した一致アクセス方式とインデックス値を使用した直接アクセス方式の比較

一致アクセス方式は、BY ステートメントを使用してトランザクションデータセットのオブザベーションをマスターデータセットのオブザベーションと一致させる MODIFY ステートメントの形式です。一致アクセス方式を示す例については、*“トランザクションデータセットを使用したオブザベーションの変更” (SAS DATA ステップステートメント: リファレンス)*を参照してください。

直接アクセス方式は、MODIFY ステートメントの KEY=オプションを使用して、変更するデータセットからインデックス付き変数に名前を付ける MODIFY ステートメントの形式です。インデックス付き値による直接アクセス方式を示す例については、*“インデックスにより検索されるオブザベーションの変更” (SAS DATA ステップステートメント: リファレンス)*を参照してください。

これら 2 つの方式(BY ステートメントを使用)と、インデックスを使用した直接アクセス方式(KEY=オプションを使用)を比較すると、次のようになります。

表 A2.2 BY ステートメントを使用した一致アクセス方式とインデックス値を使用した直接アクセス方式の比較

BY ステートメントを使用した MODIFY	KEY=オプションを使用した MODIFY
マスターデータセット内の一致オブザベーションは、生成された WHERE ステートメントによって取得されます。WHERE ステートメントでは、マスターデータセットに作成されたインデックスを使用できます。	マスターデータセット内の一致オブザベーションは、KEY=オプションで指定されたインデックスを通じて取得されます。
マスターデータセットに重複する BY 値が存在する場合、最初に出現した値のみが更新されます。	マスターデータセットに重複する KEY=値が存在する場合、DO UNTIL ループを使用して重複値を更新できます。DO UNTIL ループを記述すると、不一致条件が発生するまでマスターデータセットで KEY=オプションを強制的に実行できます。“マスターデータセット内の重複キー値での MODIFY の使用” (850 ページ)を参照してください。
トランザクションデータセットに重複する BY 値が存在する場合、累積ステートメントを記述しない限り、その値はマスターデータセットの同じオブザベーションに次々に適用されます。それ以外の場合は、最後の重複が適用されます。	トランザクション内の連続する重複はすべて、UNIQUE オプションを使用して適用されます。UNIQUE オプションが指定されていない場合は、最初の連続する重複のみが適用されます。 連続していない重複はすべてマスターデータセット内のオブザベーションに次々に適用されるため、最後の重複のみが適用されます。“トランザクションデータセット内の重複キー値での MODIFY の使用” (848 ページ)を参照してください。
マスターデータセットの一致オブザベーションで、同じ名前の変数の自動更新を実行します。	マスターデータセットの一致オブザベーションで、同じ名前の変数の自動更新を実行しません。“KEY=を使用して同じ名前の変数の自動更新を実行” (844 ページ)を参照してください。

更新処理のモニタリング

_IORC_の値をテストする最良の方法は、%SYSRC 自動呼び出しマクロを使用することです。これにより、I/O 操作の潜在的な結果を説明するニーモニック名を指定できます。%SYSRC によって提供されるニーモニックコードは、SAS 自動呼び出しマクロライブラリの一部です。各ニーモニックコードは、一致オブザベーションの取得が成功したかどうかを判断するための特定の条件を表します。%SYSRC で使用可能な最も一般的なニーモニックのリストを次に示します。

表 A2.3　%SYSRC 自動呼び出しマクロの一般的なニーモニック

アクセス方式	ニーモニック	説明
KEY=オプションを使用した MODIFY	_DSENOM	マスターデータセットにオブザベーションが含まれていないことを示します。
BY ステートメントを使用した MODIFY	_DSENMR	トランザクションデータセットのオブザベーションがマスターデータセットに存在しないことを示します。
BY ステートメントを使用した MODIFY	_DSEMTR	指定された BY 値を持つ複数のトランザクションデータセットオブザベーションがマスターデータセットに存在しないことを示します。
KEY=または BY ステートメントを使用した MODIFY	_SOK	オブザベーションが見つかったことを示します。

_IORC_のニーモニックの完全なリストと、それに対応する現在の数値および説明は、自動呼び出しマクロライブラリの SYSRC メンバーに含まれています。次のコードをサブミットすると、SAS ログの SYSRC メンバーのコンテンツを表示できます。

```
options source2;
%include sasautos(sysrc);
run;
```

次の例では、%SYSRC を使用してオブザベーションの一致が成功したかどうかをテストします。この例では、ニーモニック_SOK を使用しています。これは、I/O 操作が成功したことを示します。

```
data master;
modify master trans updatemode=nomissingcheck;
by ssn;
if _iorc_=%sysrc(_sok) then replace;
else _error_ =0;
run;
```

注: バージョン 7 以降、IORCMSG 関数は、_IORC_の現在値に関連付けられたフォーマット済みエラーメッセージを返します。

KEY=を使用して同じ名前の変数の自動更新を実行

BY ステートメントを使用する場合とは異なり、KEY=オプションでは、同じ名前の変数の自動更新は行われません。ただし、更新する変数が多数ある場合は、この機能があると便利です。この機能は、SET ステートメントと POINT=オプションを

MODIFY ステートメントと組み合わせて使用することで実現できます。次のステートメントの順序を守ってください。

```
data master;
  set trans;
  modify master key=ssn;
  i+1;
  if _iorc_=%sysrc(_sok) then do;
    13
    set trans point=i;
    replace;
  end;
  else _error_=0;
run;
```

SET ステートメントは、Trans 変数の値を PDV にロードします。次に、MODIFY ステートメントが実行されます。フェッチが成功すると、PDV 内のすべての同じ名前の変数の値が Master データセットの値でオーバーレイされます。この時点で REPLACE または OUTPUT ステートメントが実行されると、Master は、同じ名前の変数の PDV 内の Master 値で更新されます。

ただし、前述のコードでは 2 番目の SET ステートメントを発行し、POINT=オプション (カウンター i+1)を使用して Trans データセット内の同じオブザベーションを読み取ります。これにより、Trans データセットの同じ名前の変数を使用して PDV が効果的に更新されます。前述のように、POINT=オプションでは、Master データセットで変更するオブザベーションの番号を値とする別のデータソースの変数を指定します。

更新する変数が少数の場合は、明示的な割り当てステートメントとともに、KEY=オプションの通常の処理に依存できます。KEY=を使用する場合、Trans データセットには、Master データセットのインデックスを作成するために使用されるキー変数と同じ名前の変数が必要です。Trans データセットは SET ステートメントで読み取られ、キー変数の値が PDV にロードされます。PDV にロードされた値は、KEY=オプションで指定された Master データセットのインデックス値に対して使用され、一致オブザベーションがフェッチされます。

Trans データセットと Master データセット間で、同じ名前の変数の値の自動更新は行われません。そのため、特定の変数の値を変更する場合、Master データセット変数の明示的な割り当てが行われます。

ただし、同じ名前の変数値の明示的な割り当ては、割り当てる変数が Master データセットと Trans データセットの両方で同じ名前を持つため、問題が発生します。RENAME=データセットオプションを使用すると、値を割り当てる前に 1 つのデータセット内の変数の名前を変更できます。

RENAME=データセットオプションの使用方法を次の例に示します。両方のデータセットには、キー変数を除いて同じ名前の変数が含まれています。明示的な割り当てステートメントを使用して、Master データセットの変数 NICKNAME を Trans データセットの変数 NICKNAME の値で更新します。明示的な割り当てを行うには、RENAME=データセットオプションを使用して、Trans データセット内の変数 NICKNAME の名前を TNICKNAM に変更します。Master データセットの NICKNAME 変数は、Trans データセットの TNICKNAM 変数と等しく設定されます。

表 A2.4　同じ名前の変数を含む入力データセット

data master(index=(ssn));	data trans;

```
  input ssn : $11. nickname $;
datalines;
161-60-5881 Joshua
160-58-1223 Kathryn
134-56-9094 Megan
;
```

```
  input ssn : $11. nickname $;
datalines;
161-60-5881 Josh
160-58-1223 Kate
134-56-9094 Meg
142-67-9888 Bill
;
```

```
data master;
  set trans(rename=(nickname=tnicknam));
  modify master key=ssn;
  if _iorc_=%sysrc(_sok) then do;
    nickname=tnicknam;
    replace;
  end;
  else _error_ = 0;
run;
```

KEY=オプションについて

MODIFY ステートメントと SET ステートメントの両方で KEY=オプションがサポートされます。このオプションを使用すると、アプリケーションでインデックスを指定し、インデックス付き値に基づいてデータセット内の特定のオブザベーションを取得できます。MODIFY ステートメントで KEY=を使用する方法については、この付録のインデックス付き値による直接アクセス方式で前述しました。次のトピックでは、SET ステートメントで KEY=を使用する方法について説明します。以降のトピックでは、キー変数に重複値が存在する場合など、MODIFY ステートメントと SET ステートメントの両方の KEY=に関する問題について説明します。

SET ステートメントでの KEY=の使用

このトピックでは一般的な SET ステートメントではなく、SET ステートメントでの KEY=オプションの使用について説明します。SET ステートメントの詳細については、*"SET ステートメント" (SAS DATA ステップステートメント: リファレンス)*を参照してください。

SET ステートメントの構文は次のとおりです。

SET *SAS-data-set(s)* <(*data-set-options*)><NOBS=*variable*> <END=*variable*> <POINT=*variable* | KEY=*index*> </ UNIQUE> ;

KEY=は、1 つ以上の変数に対して作成されたインデックスを通じて、データセット内のオブザベーションへの非順次アクセスを提供します。インデックスは単一または複合のいずれかになります。ただし、KEY=を POINT=オプションと一緒に使用することはできません。

MODIFY ステートメントと同様に、_IORC_自動変数を%SYSRC 自動呼び出しマクロと組み合わせて使用すると、より多くのエラー処理情報が提供されます。KEY=を指定した SET ステートメントを使用すると、_IORC_が作成され、データセット内のオブザベーションに対して実行された最新の I/O 操作のステータスを示すリターンコ

ードに設定されます。マスターデータセットに KEY=値が見つからない場合、_IORC_は%SYSRC 自動呼び出しマクロのニーモニック_DSENOM に対応する数値を返し、自動変数_ERROR_は 1 に設定されます。

注: KEY=オプションを使用して複数の SET ステートメントを発行する場合、各データセットの自動変数_IORC_の個別のエラーチェックを実行する必要があります。つまり、正確な出力データセットを生成するには、KEY=オプションを使用して各 SET ステートメントの後に続く_IORC_変数をテストします。

KEY=を指定した SET ステートメントを使用すると、ルックアップ値の概念がサポートされます。ルックアップ値は、別の SAS データセット、外部ファイル、ビュー、FRAME エントリのいずれかによって提供できます。バージョン 6 では、ルックアップデータセットはネイティブな SAS データセットである必要があります。バージョン 7 では、SAS/ACCESS ビューと LIBNAME ステートメントの DBMS エンジンがサポートされています。

```
data combine;
  set lookup;
  set primary key=partno;
  select(_iorc_);
  when (%sysrc(_sok)) do;
    output;
  end;
  when (%sysrc(_dsenom)) do;
    _error_ = 0;
  end;
  otherwise;
  end;
run;
```

ルックアップデータソースからのルックアップ値は、プライマリデータセットまたはマスターデータセットのオブザベーションを見つけるためのキーとして使用されます。このプライマリデータセットは、インデックス(単一または複合)を付け、KEY=オプションで指定する必要があります。ルックアップ値は、プライマリデータセット内のキー変数と同じ名前の変数を通じて提供する必要があります。たとえば、ルックアップデータソースが SAS データセットの場合、プライマリデータセットのインデックスに定義されたものと同じ名前の変数が含まれている必要があります。プライマリデータセットからオブザベーションが正常にフェッチされると、そのオブザベーションに対して OUTPUT などのアクションを実行できます。

注: DROP または KEEP が指定されていない場合、KEY=オプションを使用すると、ルックアップデータセットとプライマリデータセットのすべての変数が結合されます。つまり、出力データセットには、ルックアップデータセットの変数だけでなく、プライマリデータセットのすべての変数も含まれます。KEY=および MODIFY ステートメントでは、これは発生しません。

キー変数の重複値の処理

KEY=オプションを使用してインデックスを作成する場合、複数のオブザベーションに、入力データセットまたはインデックス値を提供するデータセット内のキー変数に対して同じ値が含まれる場合があります。

トランザクションデータセット内の重複キー値での MODIFY の使用

トランザクションデータセット内のキー変数に重複値があると、マスターデータセットに適用されたときに結果に影響する可能性があります。さらに、重複する値が連続しているか連続していないかも結果に影響します。

たとえば、Trans データセット内のキー変数 ssn の重複値の順序を除いて同一である次の 2 つのプログラムを考えてみます。プログラム 1 では、Trans データセットに 160-58-1223 という重複キー値がありますが、それらは連続していません。プログラム 2 にも重複値がありますが、それらは連続しています。

表 A2.5　重複 KEY=値の順序が出力に与える影響

プログラム 1

```
data master(index=(ssn));
  input ssn : $11. nickname $;
datalines;
161-60-5881 Joshua
160-58-1223 Kathryn
134-56-9094 Megan
;

data trans;
  input ssn : $11. tnicknam $;
datalines;
161-60-5881 Josh
160-58-1223 Kathy
134-56-9094 Meg
142-67-9888 Bill
160-58-1223 Kate
;

data master;
  set trans;
  modify master key=ssn;
  if _iorc_=%sysrc(_sok) then do;
    nickname=tnicknam;
    replace;
  end;
  else_error_=0;
run;
```

プログラム 2

```
data master(index=(ssn));
  input ssn : $11. nickname $;
datalines;
161-60-5881 Joshua
160-58-1223 Kathryn
134-56-9094 Megan
;

data trans;
  input ssn : $11. tnicknam $;
datalines;
161-60-5881 Josh
160-58-1223 Kathy
160-58-1223 Kate
134-56-9094 Meg
142-67-9888 Bill
;

data master;
  set trans;
  modify master key=ssn;
  if _iorc_=%sysrc(_sok) then do;
    nickname=tnicknam;
    replace;
  end;
  else_error_=0;
run;
```

次の PRINT プロシジャは、前述のプログラムの結果を示します。

注: DATA=オプションは、Master データセットを指定するために PROC PRINT で使用されます。DATA=が指定されていない場合、PROC PRINT は、出力のために開かれた最後に作成されたデータセットを表示します。これは、更新のために最後に開いたものではない場合があります。

```
proc print data=master;
run;
```

- プログラム 1 では、結果の Master データセットの NICKNAME の値は Kate になります。これは、対応する社会保障番号を含む Trans データセットの最後のオブザベーションの値です。

```
OBS   SSN          NICKNAME
1     161-60-5881   Josh
2 160-58-1223 Kate
3     134-56-9094   Meg
```

- プログラム 2 では、Trans データセットの ssn の TNICKNAM の連続値は Kate です。その値は Master データセットを更新しません。Kate の連続レコードでは実際には不一致が発生し、対応する値を持つ Trans の最初のオブザベーションのみが適用されます。

```
OBS   SSN          NICKNAME
1     161-60-5881    Josh
2 160-58-1223 Kathy
3     134-56-9094    Meg
```

Trans 内の重複キー値の順序は、Master データセットの結果に影響します。インデックスを検索する場合、SAS は、Trans データセット内のキー変数の値が変更された場合にのみ、インデックスの先頭から開始します。

Trans データセット内の重複キー値が*連続していない*場合、社会保障番号 **160-58-1223** の両方の検索でインデックスの先頭から検索が開始されます。したがって、Master データセット内で唯一の一致した値が 2 回とも検索され更新されるため、最後の重複オブザベーションの値が適用されます。

Trans データセット内の*連続する*重複キー変数値の場合、最初の値 **160-58-1223** が Trans データセットによって提供されると、キー変数の値が変更されます。Master データセットのインデックス検索が先頭から開始され、オブザベーションが見つかります。Master の NICKNAME 変数が Kathy に更新されます。Trans データセットによって連続値 **160-58-1223** が提供される場合、値は変更されません。したがって、検索は Master インデックスの先頭から開始されるのではなく、インデックス構造内の現在の位置から開始されます。オブザベーションは見つからず、後続の重複オブザベーションに対して更新は実行されません。最初の重複の値のみが適用されます。

Trans 内の重複が連続しているかどうかに関係なく同じ結果を保証するには、MODIFY ステートメントで UNIQUE オプションを指定して、ソフトウェアがインデックスの先頭から検索を開始するように強制することができます。UNIQUE オプションは、キー変数値が反復ごとに変わるかどうかに関係なく、インデックスの先頭から検索するように指定します。

次のコードでは、UNIQUE オプションを使用して各プログラムで同じ結果を生成しています。読みやすくするために、IF ステートメントは SELECT ステートメントとしてコーディングされています。

表 A2.6　入力マスターデータセットとトランザクションデータセット

```
data master(index=(ssn));
  input ssn : $11. Nickname $;
datalines;
161-60-5881 Joshua
160-58-1223 Kathryn
134-56-9094 Megan
;
```

```
data trans;
  input ssn : $11. tnicknam $;
datalines;
161-60-5881 Josh
160-58-1223 Kathy
160-58-1223 Kate
134-56-9094 Meg
142-67-9888 Bill
;
```

```
data master;
  set trans;
  modify master key=ssn/unique;
  select (_iorc_);
  when (%sysrc(_sok)) do;
    nickname=tnicknam;
    replace master;
  end;
  when (%sysrc(_dsenom)) do;
    _error_=0;
  end;
  otherwise;
  end;

proc print data=master;
run;
```

```
OBS   SSN          NICKNAME
1     161-60-5881  Josh
2 160-58-1223 Kate
3     134-56-9094  Meg
```

マスターデータセット内の重複キー値での MODIFY の使用

マスターデータセット内のキー変数に重複値があると、マスターデータセット内のすべての 重複をトランザクションデータセット内の対応する一意の値で更新する場合に、結果に影響する可能性があります。

たとえば、ssn 値が **161-60-5881** の Trans データセットのオブザベーションにより、Master データセットの ssn 変数の最初の対応するオブザベーションのみが Josh の値に更新されます。PROC PRINT の結果では、強調表示されたレコードが更新されていることに注意してください。

表 A2.7　入力データセット Master と Trans

data master(index=(ssn));	data trans;
input ssn : $11. nickname $;	input ssn : $11. nickname $;
datalines;	datalines;
161-60-5881 Joshua	161-60-5881 Josh
161-60-5881 Joshua	160-58-1223 Kathy
160-58-1223 Kathryn	134-56-9094 Meg
134-56-9094 Megan	142-67-9888 Bill
;	;

```
data master;
  set trans(rename=(nickname=tnicknam));
  modify master key=ssn;
  select (_iorc_);
  when (%sysrc(_sok)) do;
    nickname=tnicknam;
    replace master;
  end;
  when (%sysrc(_dsenom)) do;
    _error_=0;
  end;
  otherwise;
  end;

proc print data=master;
run;
```

```
OBS  SSN          NICKNAME
1 161-60-5881 Josh
2    161-60-5881  Joshua
3    160-58-1223  Kathy
4    134-56-9094  Meg
```

Master データセット内の複数のオブザベーションの変数 Nickname を、Trans データセット内の対応する一意のオブザベーションで更新するには、DO UNTIL ループを使用して、不一致条件が検出されるまで Master データセットのインデックスファイルの継続的な検索を強制します。PRINT プロシジャからの強調表示されたオブザベーションは、ssn 値 **161-60-5881** の対応するすべてのレコードが Master データセットで更新されたことを示していることに注意してください。

表 A2.8　入力データセット Master と Trans

data master(index=(ssn));	data trans;
input ssn : $11. nickname $;	input ssn : $11. tnicknam $;
datalines;	datalines;
161-60-5881 Joshua	161-60-5881 Josh
161-60-5881 Joshua	160-58-1223 Kathy
160-58-1223 Kathryn	134-56-9094 Meg
134-56-9094 Megan	142-67-9888 Bill
;	;

```
data master;
```

```
  set trans;
  do until (_iorc_=%sysrc(_dsenom));
    modify master key=ssn;
    select (_iorc_);
      when (%sysrc(_sok)) do;
        nickname=tnicknam;
        replace master;
      end;
      when (%sysrc(_dsenom)) do;
        _error_=0;
      end;
      otherwise;
  end;
end;

proc print data=master;
run;
```

```
OBS  SSN          NICKNAME
1 161-60-5881 Josh 2 161-60-5881 Josh
3    160-58-1223  Kathy
4    134-56-9094  Meg
```

マスターデータセットとトランザクションデータセット内の重複キー値での MODIFY の使用

キー変数の重複値がマスターデータセットとトランザクションデータセットの両方に存在する場合、対応する各マスターオブザベーションを対応する各トランザクションオブザベーションで更新するには、追加の BY 処理が必要になります。

トランザクションデータセットによって ssn の連続する重複値が提供される場合、UNIQUE オプションによって強制的にインデックスの先頭から検索が開始されることがわかっています。ただし、この状況では、UNIQUE オプションを使用すると、Master データセット内の同じオブザベーションが検索され、更新されます。したがって、Master 内の重複オブザベーションは更新されません。

UNIQUE オプションを使用せずにキー変数を変更して、インデックスの先頭から検索を強制的に開始する必要があり、これは同じ ssn の各オブザベーション間で実行する必要があります。Trans データセット内の ssn の連続する値間でキー変数を変更する必要があるため、BY 処理を使用して、Trans 内の同じ ssn のオブザベーションがさらに提供されるかどうかを判断します。したがって、Trans データセットは BY 処理用に並べ替える必要があります。インデックス内で対応するすべての一致の検索を続行するには、DO UNTIL ステートメントを使用して、Master で不一致状況が発生するまで処理します。

次の例では、キー変数値を、Master データセットにない値に一時的に変更する方法を示します。

表 A2.9 入力データセット Master と Trans

```
data master(index=(ssn));
  input ssn : $11. nickname $;
datalines;
161-60-5881 Joshua
161-60-5881 Joshua
160-58-1223 Kathryn
160-58-1223 Kathryn
134-56-9094 Megan
;
```

```
data trans;
  input ssn : $11. tnicknam $;
datalines;
161-60-5881 Josh
160-58-1223 Kathy
160-58-1223 Kate
134-56-9094 Meg
142-67-9888 Bill
;
```

```
proc sort data=trans;
  by ssn;

data master;
  set trans;
  by ssn;
  dummy=0;
  do until (_iorc_=%sysrc(_dsenom));
    if dummy then ssn='999-99-9999';
    modify master key=ssn;
    select (_iorc_);
      when (%sysrc(_sok)) do;
        nickname=tnicknam;
        replace master;
      end;
      when (%sysrc(_dsenom)) do;
        _error_=0;
        if not last.ssn and not dummy then do;
          dummy=1;
          _iorc_=0;
        end;
      end;
      otherwise;
    end;
  ;

proc print data=master;
run;
```

```
OBS  SSN          NICKNAME
1    161-60-5881  Josh
2    161-60-5881  Josh
3 160-58-1223 Kate 4 160-58-1223 Kate5   134-56-9094    Meg
```

ルックアップデータセット内の重複キー値での SET の使用

MODIFY での KEY=と同様に、KEY=オプションを指定した SET ステートメントを実行すると、フェッチ操作間でルックアップデータセット内の KEY=変数の値が変わった場合にのみ、強制的にインデックスの先頭から検索が開始されます。キー変数の重複値がルックアップデータセットに連続して存在する場合、実行間で値は変わりません。したがって、強制的に先頭から検索を開始するには、UNIQUE オプションを使用します。連続していない重複値の場合、検索はインデックス構造の先頭から開始します。

プライマリデータセット内の重複キー値での SET の使用

プライマリデータセット内のキー変数の重複値は、前述の MODIFY の場合と同じ効果を SET にもたらします。KEY=オプションを指定したステートメントが同じキー値を使用して強制的に再度実行されない限り、ソフトウェアはインデックスで最初に出現したキー変数を検索して返します。MODIFY と同様に、DO UNTIL ステートメントを使用して、このプライマリインデックス構造の検索を強制的に継続させることができます。PRINT プロシジャからの太字のオブザベーションは、Partno A066 のすべての対応するオブザベーションが出力データセットに存在することを示していることに注意してください。

表 A2.10　入力データセット Lookup と Primary

```
data lookup;
  input partno $ quantity;
datalines;
A066 8
A220 2
A812 10
;
```

```
data primary(index=(partno));
  input partno $ desc $;
datalines;
A021 motor
A033 bolt
A043 nut
A055 radiator
A066 hose
A066 hose2
A066 hose3
A078 knob
A220 switch
A223 bridge
;
```

```
data combine;
  set lookup;
  do until(_iorc_=%sysrc(_dsenom));
    set primary key=partno;
    select(_iorc_);
      when (%sysrc(_sok)) do;
        output;
```

```
      end;
      when (%sysrc(_dsenom)) do;
        _error_=0;
      end;
      otherwise;
    end;
  end;

proc print data=combine;
run;
```

```
OBS  PARTNO QUANTITY DESC
1 A066 8 hose 2 A066 8 hose2 3 A066 8 hose3
4    A220   2        switch
```

プライマリデータセットとルックアップデータセット内の重複キー値での SET の使用

プライマリデータセットとルックアップデータセットの両方にキー変数の重複値が存在する場合にデータセットを結合するには、MODIFY ステートメントで説明したのと同じ概念を使用します。"マスターデータセットとトランザクションデータセット内の重複キー値での MODIFY の使用"を参照してください。

注: SET ステートメントを使用するのではなく、SQL プロシジャを使用することを検討してください。これにより、次の PROC SQL の例に示すように、簡略化されたコードで同じ結果が得られます。両方の手法については、"SAS データセットの結合と変更: 例"の例 4.6 を参照してください。

表 A2.11　入力データセット Lookup と Primary

```
data lookup;
  input partno $ district;
datalines;
A066 1
A066 2
A220 2
A812 10
;
```

```
data primary(index=(partno));
  input partno $ desc $;
datalines;
A021 motor
A033 bolt
A043 nut
A055 radiator
A066 hose
A066 hose2
A066 hose3
A078 knob
A220 switch
223 bridge
;
```

```
proc sql;
  create table shiplst as
  select a.*, b.desc
  from lookup as a, primary as b
```

```
  where a.partno=b.partno;
quit;

proc print data=shiplst;
run;
```

```
OBS PARTNO DISTRICT DESC
1 A066 1 hose 2 A066 2 hose 3 A066 1 hose2 4 A066 2 hose2 5 A066 1 hose3 6 A066 2
hose3
7   A220    2       switch
```

不一致オブザベーションを含むデータセットの結合

SET ステートメントが実行されるたびに、PDV への入力用に開かれた現在のデータセットから 1 つのオブザベーションが読み取られます。SET ステートメントは、特に指定しない限り、入力データセットからすべての変数とすべてのオブザベーションを読み取ります。複数の SET ステートメントを使用して、指定されたデータセットの 1 対 1 読み取り(1 対 1 の一致とも呼ばれる)を実行できます。新しいデータセットには、すべての入力データセットのすべての変数が含まれます。新しいデータセット内のオブザベーションの数は、最小の元のデータセット内のオブザベーションの数になります。データセットに共通変数が含まれている場合、最後のデータセットから読み込まれた値によって、前のデータセットから読み込まれた値が置き換えられます。

複数の SET ステートメントで SAS データセットを結合する場合、SAS は、各 DATA ステップの反復で既存の変数を欠損値に再初期化しません。出力データセットの非欠損値は、発生した最新の一致の PDV で保持されている値です。

次の例では、Partno の値が A812 であるルックアップデータセットのオブザベーションは、プライマリデータセットでは見つかりません。PDV にロードされた DESC 変数は現在、Partno 値 A220 の最後に成功した一致からの値スイッチを保持します。一致が発生しないため、この DESC 変数値は PDV で Partno A812 の新しい値で上書きされません。

表 A2.12　入力データセット Lookup と Primary

```
data lookup;
  input partno $ quantity;
datalines;
A066 8
A220 2
A812 10
;
```

```
data primary(index=(partno));
  input partno $ desc $;
datalines;
A021 motor
A033 bolt
A043 nut
A055 radiator
A066 hose
A078 knob
A220 switch
A223 bridge
;
```

```
data combine;
  set lookup;
```

```
  set primary key=partno;
  select(_iorc_);
    when (%sysrc(_sok)) do;
      output;
    end;
    when (%sysrc(_dsenom)) do;
      _error_=0;
      *desc = ' ';
      output;
    end;
    otherwise;
  end;

proc print data=combine;
run;
```

```
OBS PARTNO QUANTITY DESC
1   A066    8       hose
2   A220    2       switch
3 A812 10 switch
```

プライマリデータセットに存在しない DESC の欠損値を出力データセットに書き込むには、明示的な割り当てステートメントを実行する必要があります。割り当てステートメントで、DESC 値を欠損に等しく設定します。これにより、PDV に保持されている値が、欠損している目的の値で上書きされます。前述のコード例で、目的の出力を得るには、DESC=コードのコメントを解除します。

出力データセットへの変数の追加

SET ステートメントを使用して出力のためにデータセットを開くと、割り当てステートメントを通じて新しい変数を出力データセットに追加できます。これが発生すると、DATA ステップの各反復で新しい変数が欠損にリセットされます。次の例は、プライマリデータセットからのオブザベーションで不一致が発生した場合に、欠損値を割り当てずに STATUS 変数が自動的に欠損に設定されることを示しています。

表 A2.13　入力データセット Lookup と Primary

```
data lookup;
  input partno $ quantity;
datalines;
A066 8
A220 2
A812 10
;
```

```
data primary(index=(partno));
  input partno $ desc $;
datalines;
A021 motor
A033 bolt
A043 nut
A055 radiator
A066 hose
A078 knob
A220 switch
A223 bridge
;
```

```
data combine;
```

```
  set lookup;
  set primary key=partno;
  select(_iorc_);
    when (%sysrc(_sok)) do;
      status="available";
      output;
    end;
    when (%sysrc(_dsenom)) do;
      desc=' ';
      _error_=0;
      output;
    end;
    otherwise;
  end;

proc print data=combine;
run;
```

```
OBS PARTNO QUANTITY DESC   STATUS
1   A066     8       hose   available
2   A220     2      switch  available
3 A812 10
```

バージョン 7 とバージョン 8 の考慮事項

KEY=を指定した DBKEY= SAS/ACCESS データセットオプションの使用

SAS/ACCESS ソフトウェアは、DBMS 内のデータにアクセスするときのパフォーマンスを向上させるために、いくつかのデータセットオプションを提供します。DBKEY=データセットオプションを使用すると、インデックスとして使用する変数名を指定できます。ソフトウェアは、WHERE 式を構築して DBMS に渡すことにより、指定された変数名をインデックスとして使用します。

SAS データファイルにインデックスを付けることでパフォーマンスが向上するのと同様に、DBKEY=を使用するとパフォーマンスを向上させることができます。たとえば、パフォーマンスを向上させるには、SET ステートメントの KEY=オプションとともに DATA ステップで DBKEY=オプションを使用できます。KEY=オプションの値としてキーワード DBKEY を指定する必要があることに注意してください。

次の DATA ステップでは、データファイル Keyvalues と DBMS テーブル Mytable を結合して、新しいデータファイルを作成します。ソフトウェアは、DBKEY=データセットオプションとともに変数 Deptno を使用して、WHERE 式を DBMS に渡します。DBMS オプティマイザーが DBMS テーブル上の既存のインデックスを使用して WHERE 式を最適化することを選択した場合、パフォーマンス上の利点が得られる可能性があります。

```
data sasuser.new;
  set sasuser.keyvalues;
  set dblib.mytable (dbkey=deptno) key=dbkey;
```

```
run;
```

付録 3

カラムバイナリデータストレージの説明

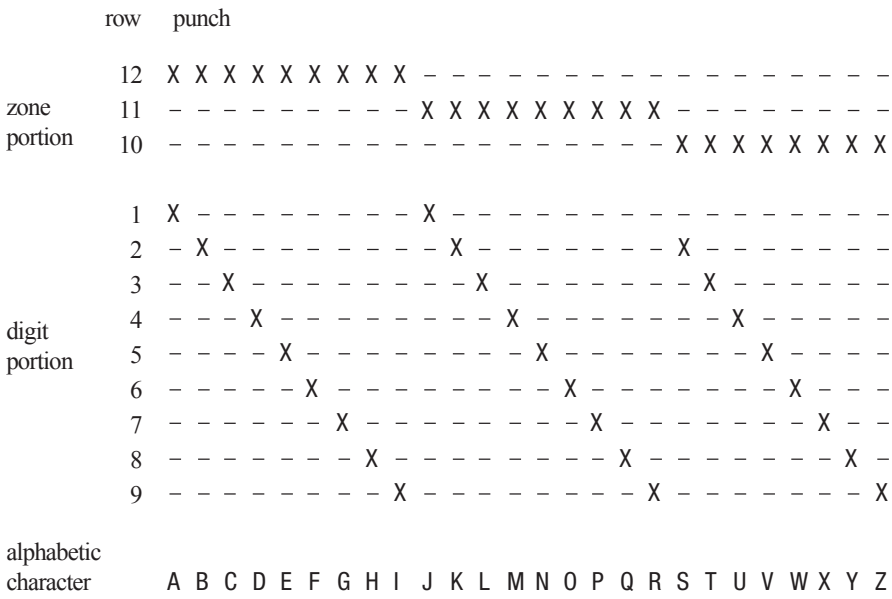
カラムバイナリデータストレージの説明

カラム内の行の配置と番号付けは、文字と数字をエンコーディングするホレリスシステムに由来します。これは、値のペアを使用して文字または数字を表すのが基本です。ホレリスシステムでは、カードのカラムごとに最大 2 つのパンチ(ゾーン部分に 1 つ、数字部分に 1 つ)が存在しました。これらのパンチは値のペアに対応し、値の各ペアは特定のアルファベット文字または記号と数字に対応します。

パンチカードのゾーン部分(最初の 3 行)では、ペアのゾーン構成要素の値は 12、11、0 (または 10)になるか、パンチされない場合があります。カードの数字部分(4 行目から 12 行目まで)では、ペアの数字構成要素は 1 から 9 までの値になるか、パンチされない場合があります。

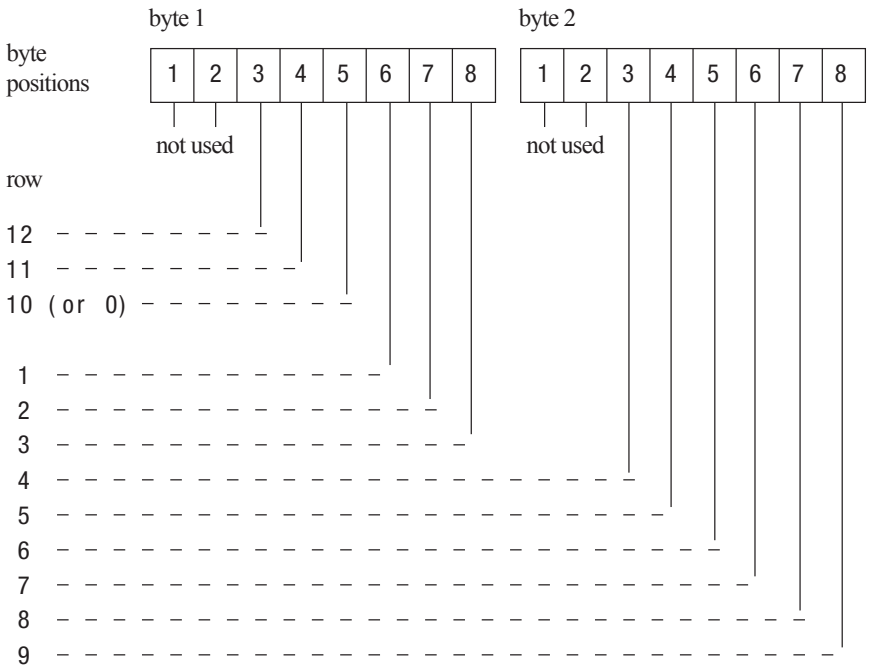
次の図は、アルファベットの文字に対応するマルチパンチの組み合わせを示しています。

図 A3.1　パンチカードのカラムと行



SAS では、カラムバイナリデータ("仮想"パンチカード)の各カラムを 2 バイトで保存します。各カラムには 12 位置のみが含まれ、2 バイトには 16 位置が含まれるため、バイト内の追加位置は各バイトの先頭に配置されます。次の図は、"仮想"パンチカードデータの行と、SAS がその保存に使用する 2 バイト内の位置との対応を示しています。SAS では、パンチされた位置をバイナリ 1 ビットとして保存し、パンチされていない位置をバイナリ 0 ビットとして保存します。

図 A3.2　"仮想"パンチカード上のカラムバイナリ表現



付録 4

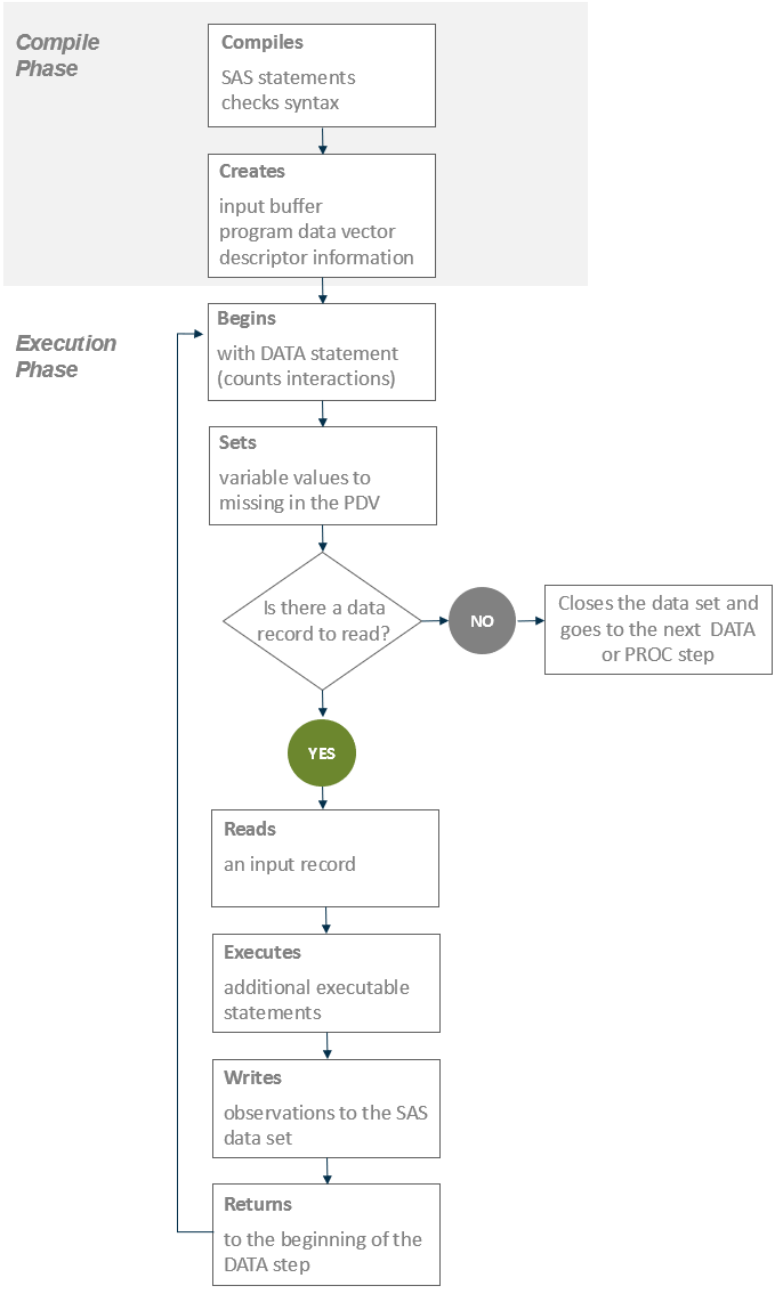
DATA ステップの仕組みを理解する

DATA ステップのデータの処理方法

アクションのフロー

DATA ステップを実行のためにサブミットする場合、DATA ステップは最初にコンパイルされ、次に実行されます。次の図は、典型的な SAS DATA ステップのアクションフローを示しています。

図 A4.1　DATA ステップのアクションフロー



コンパイルフェーズ

DATA ステップを実行のためにサブミットすると、SAS は SAS ステートメントの構文をチェックします。つまり、ステートメントを自動的にマシンコードに変換します。このフェーズでは、SAS は新しい変数ごとに型と長さを識別し、後続の変数参照ごとに変数型の変換が必要かどうかを決定します。コンパイルフェーズでは、SAS は次の 3 つのアイテムを作成します。

入力バッファー
SAS が INPUT ステートメントを実行するときに、SAS が生データの各レコードを読み込むメモリの論理領域です。このバッファーは、DATA ステップが生データを読み取る場合にのみ作成されます。DATA ステップが SAS データセットを読み取るとき、SAS はデータをプログラムデータベクトルに直接読み込みます。

プログラムデータベクトル(PDV)
SAS が一度に 1 つのオブザベーションずつデータセットを構築するメモリの論理領域です。プログラムが実行されると、SAS は入力バッファーからデータ値を読み取るか、SAS 言語ステートメントを実行してデータ値を作成します。データ値は、プログラムデータベクトル内の適切な変数に割り当てられます。ここから、SAS は値を単一のオブザベーションとして SAS データセットに書き込みます。

データセット変数と計算変数とともに、PDV には 2 つの自動変数_N_と_ERROR_が含まれています。_N_変数は、DATA ステップの反復が開始される回数をカウントします。_ERROR_変数は、実行中にデータによって引き起こされたエラーの発生を示します。_ERROR_の値は、0 (エラーが存在しないことを示す)または 1 (1 つ以上のエラーが発生したことを示す)です。SAS はこれらの変数を出力データセットに書き込みません。

ディスクリプター情報
SAS が各 SAS データセットについて作成および維持する、データセット属性および変数属性などの情報です。たとえば、データセットの名前、そのメンバータイプ、データセットが作成された日時、変数の数、名前、データ型(文字または数値)が含まれます。ディスクリプター情報には、拡張属性に関する情報も含まれます(データセットで定義されている場合)。拡張属性ディスクリプター情報には、属性の名前、変数の名前、および属性の値が含まれます。

実行フェーズ

デフォルトでは、単純な DATA ステップは、作成されるオブザベーションごとに 1 回反復されます。単純な DATA ステップの実行フェーズのアクションフローは次のとおりです。

1 DATA ステップは、DATA ステートメントから始まります。DATA ステートメントを実行するたびに、DATA ステップの新しい反復が開始され、自動変数_N_の値が 1 ずつ増加します。

2 SAS は、新しく作成されたプログラム変数をプログラムデータベクトル(PDV)で欠損に設定します。

3 SAS は、生データファイルからデータレコードを入力バッファーに読み込むか、SAS データセットからのオブザベーションをプログラムデータベクトルに直接読み込みます。INPUT、MERGE、SET、MODIFY、UPDATE の各ステートメントを使用して、レコードを読み取ることができます。

4 SAS は、現在のレコードに対して後続のプログラミングステートメントを実行します。

5 ステートメントの最後では、出力、戻り、リセットが自動的に実行されます。SAS がオブザベーションを SAS データセットに書き込むと、システムは自動的に

DATA ステップの先頭に戻り、INPUT ステートメントと割り当てステートメントで作成された変数の値はプログラムデータベクトルで欠損にリセットされます。SET、MERGE、MODIFY、UPDATE の各ステートメントで読み取った変数は、ここでは欠損にリセットされないことに注意してください。

6 SAS は別の反復をカウントし、次のレコードまたはオブザベーションを読み取り、現在のオブザベーションに対して後続のプログラミングステートメントを実行します。

7 SAS データセットまたは生データファイルでファイルの終わりが検出されると、DATA ステップは終了します。

注: この図は、DATA ステップのデフォルト処理を示しています。データ読み取りステートメント(INPUT や SET など)またはデータ書き込みステートメント(OUTPUT など)は、プログラム内に任意の順序で配置できます。

DATA ステップの処理: 手順説明

サンプル DATA ステップ

次のステートメントは、生データを読み取り、合計を計算し、データセットを作成する DATA ステップの例を示しています。

```
data total_points (drop=TeamName);  1
  input TeamName $ ParticipantName $ Event1 Event2 Event3;  2
  TeamTotal + (Event1 + Event2 + Event3);  3
  datalines;
Knights Sue   6 8 8
Kings Jane    9 7 8
Knights John  7 7 7
Knights Lisa  8 9 9
Knights Fran  7 6 6
Knights Walter 9 8 10
;

proc print data=total_points;
run;
```

1 DROP=データセットオプションは、変数 TeamName が Total_Points という出力 SAS データセットに書き込まれるのを防ぎます。

2 INPUT ステートメントは、各変数に名前を付け、そのデータ型(文字または数値)を識別し、データレコード内の相対位置を識別することにより、データを記述します。

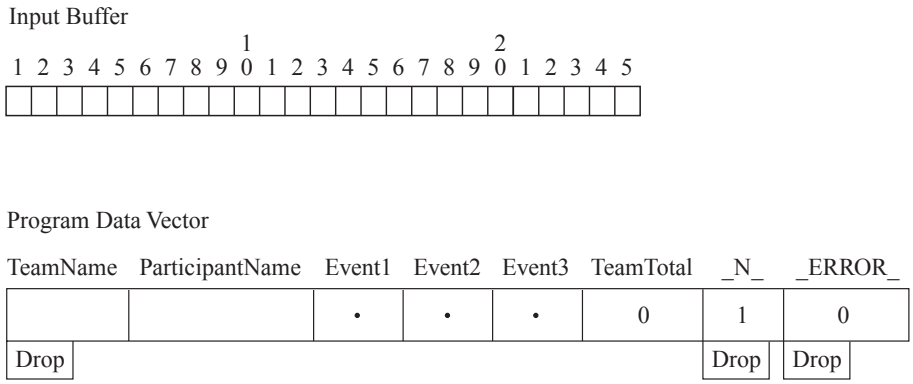
3 SUM ステートメントは、変数 TeamTotal に 3 つのイベントのスコアを累積します。

入力バッファーとプログラムバッファーの作成

DATA ステップステートメントがコンパイルされると、SAS は入力バッファーを作成するかどうかを決定します。入力ファイルに生データが含まれている場合(前述の例のように)、SAS はデータをプログラムデータベクトル(PDV)に移動する前に、データを保持するための入力バッファーを作成します。(ただし、入力ファイルが SAS データセットの場合、SAS は入力バッファーを作成しません。SAS は入力データを PDV に直接書き込みます。)

PDV には、入力データセット内のすべての変数、DATA ステップステートメントで作成された変数、および DATA ステップごとに自動生成される 2 つの変数_N_と_ERROR_が含まれます。_N_変数は、DATA ステップが反復された回数を表します。_ERROR_変数はバイナリスイッチのように動作し、DATA ステップにエラーが存在しない場合は値が 0 になり、1 つ以上のエラーが存在する場合は値が 1 になります。次の図に、DATA ステップのコンパイル後の入力バッファーとプログラムデータベクトルを示します。

図 A4.2　入力バッファーとプログラムデータベクトル



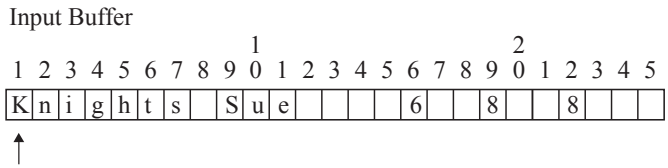
INPUT ステートメントおよび合計ステートメントで作成される変数(TeamName、ParticipantName、Event1、Event2、Event3、TeamTotal)は、初期値として欠損に設定されます。この表現では、数値変数はピリオドで初期化され、文字変数は空白で初期化されます。自動変数_N_は 1 に設定され、自動変数_ERROR_は 0 に設定されます。

変数 TeamName は、DATA ステートメントで DROP=データセットオプションが指定されているため、PDV で削除とマークされています。削除された変数は SAS データセットに書き込まれません。DATA ステップにより作成された自動変数は SAS データセットに書き込まれないため、_N_変数と_ERROR_変数は削除されます。自動変数の詳細については、4 章, “変数” (69 ページ)を参照してください。

レコードの読み取り

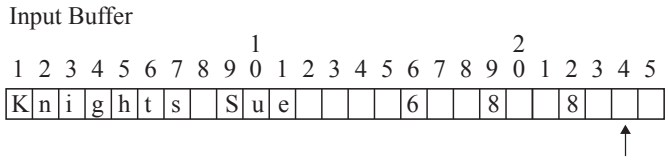
SAS は最初のデータ行を入力バッファーに読み込みます。SAS が入力バッファーからデータを読み取る際にその位置を保持するために使用する入力ポインターは、バッファーの先頭に配置され、データレコードを読み取る準備が整います。次の図は、SAS がデータを読み取る前の入力バッファー内の入力ポインターの位置を示しています。

図 A4.3　SAS がデータを読み取る前の入力バッファー内のポインターの位置



次に、INPUT ステートメントは、入力バッファーのレコードからデータ値を読み取り、それを PDV に書き込み、変数値にします。次の図は、入力バッファー内のポインターの位置と、SAS が最初のレコードを読み取った後の PDV 内の値の両方を示しています。

図 A4.4　最初のレコードの値がプログラムデータベクトルに読み込まれる



Program Data Vector

TeamName	ParticipantName	Event1	Event2	Event3	TeamTotal	_N_	_ERROR_
Knights	Sue	6	8	8	0	1	0
Drop						Drop	Drop

INPUT ステートメントが各変数の値を読み取った後、SAS は合計ステートメントを実行します。SAS は、変数 TeamTotal の値を計算し、PDV に書き込みます。次の図は、SAS がオブザベーションをデータセットに書き込む前の、すべての値を含む PDV を示しています。

図 A4.5　合計ステートメントの計算値を含むプログラムデータベクトル

Program Data Vector

TeamName	ParticipantName	Event1	Event2	Event3	TeamTotal	_N_	_ERROR_
Knights	Sue	6	8	8	22	1	0
Drop						Drop	Drop

SAS データセットへのオブザベーションの書き込み

SAS が DATA ステップの最後のステートメントを実行すると、削除するようにマークされた変数を除く PDV のすべての値が、単一のオブザベーションとしてデータセット Total_Points に書き込まれます。次の図は、Total_Points データセットの最初のオブザベーションを示しています。

図 A4.6　データセット Total_Points の最初のオブザベーション

Output SAS Data Set TOTAL_POINTS: 1st observation

ParticipantName	Event1	Event2	Event3	TeamTotal
Sue	6	8	8	22

その後、SAS は DATA ステートメントに戻り、次の反復を開始します。SAS は PDV の値を次の方法でリセットします。

- INPUT ステートメントで作成される変数の値は、欠損に設定されます。
- 合計ステートメントで作成された値は自動的に保持されます。
- 自動変数_N_の値は 1 増加し、_ERROR_の値は 0 にリセットされます。

次の図は、PDV の現在の値を示しています。

図 A4.7　プログラムデータベクトルの現在の値

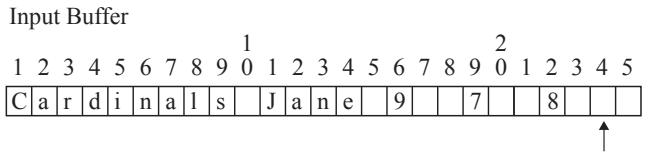
Program Data Vector

TeamName	ParticipantName	Event1	Event2	Event3	TeamTotal	_N_	_ERROR_
		.	.	.	22	2	0
Drop						Drop	Drop

次のレコードの読み取り

SAS は次のレコードを入力バッファーに読み込みます。INPUT ステートメントは入力バッファーからデータ値を読み取り、PDV に書き込みます。合計ステートメントは、Event1、Event2、および Event3 の値を TeamTotal に加算します。変数_N_の値 2 は、SAS が DATA ステップの 2 回目の反復を開始していることを示します。次の図は、入力バッファー、2 番目のレコードの PDV、および最初の 2 つのオブザベーションを含む SAS データセットを示しています。

図 A4.8　入力バッファー、プログラムデータベクトル、および最初の 2 つのオブザベーション



Program Data Vector

TeamName	ParticipantName	Event1	Event2	Event3	TeamTotal	_N_	_ERROR_
Cardinals	Jane	9	7	8	46	2	0
Drop						Drop	Drop

Output SAS Data Set TOTAL_POINTS: 1st and 2nd observations

ParticipantName	Event1	Event2	Event3	TeamTotal
Sue	6	8	8	22
Jane	9	7	8	46

SAS がレコードの読み取りを続行すると、より多くの参加者のスコアが変数に加算されるにつれて TeamTotal の値が大きくなります。_N_は、DATA ステップの各反復の開始時に増加します。このプロセスは、SAS が入力ファイルの終わりに到達するまで続行されます。

DATA ステップの実行終了時

DATA ステップは、最後の入力レコードを処理した後、実行を停止します。PROC PRINT を使用して、Total_Points データセットに出力できます。

```
data total_points (drop=TeamName);
  input TeamName $ ParticipantName $ Event1 Event2 Event3;
  TeamTotal + (Event1 + Event2 + Event3);
  datalines;
Knights Sue   6 8 8
Cardinals Jane 9 7 8
Knights John  7 7 7
Cardinals Lisa 8 9 9
Cardinals Fran 7 6 6
Knights Walter 9 8 10
;
proc print data=total_points;
  title 'Total Team Scores';
run;
```

アウトプット A4.1　DATA ステップ手順説明の出力

Total Team Scores

Obs	ParticipantName	Event1	Event2	Event3	TeamTotal
1	Sue	6	8	8	22
2	Jane	9	7	8	46
3	John	7	7	7	67
4	Lisa	8	9	9	93
5	Fran	7	6	6	112
6	Walter	9	8	10	139

DATA ステップ実行の詳細

デフォルトの実行順序

次の表は、DATA ステップ内のステートメントのデフォルトの実行順序の概要を示しています。DATA ステートメントは、ステップを開始し、通常、ステップで作成する 1 つ以上の SAS データセットを識別します。(出力データセットを作成しない場合は、データセット名としてキーワード_NULL_を使用できます)。オプションのプログラミングステートメントはデータを処理します。SAS は、オブザベーション処理の最後にデフォルトのアクションを実行します。

表 A4.1　DATA ステップのステートメントのデフォルト実行

DATA ステップの構造	アクション
DATA ステートメント	ステップを開始する 反復をカウントする
データ読み取りステートメント:[1]	
INPUT	生データソースからの入力データレコードでの値の配置を記述する
SET	1 つ以上の SAS データセットからオブザベーションを読み取る

DATA ステップの構造	アクション
MERGE	2 つ以上の SAS データセットからのオブザベーションを単一のオブザベーションに結合する
MODIFY	既存の SAS データセットにあるオブザベーションを置換、削除、または追加する
UPDATE	トランザクションを適用してマスターファイルを更新する
オプションの SAS プログラミングステートメント。例:	現在のオブザベーションのデータをさらに処理する
FirstQuarter=Jan+Feb+Mar; if RetailPrice < 500;	現在のオブザベーションの FirstQuarter の値を計算する 現在のオブザベーションの変数 RetailPrice の値によるサブセット化
オブザベーション処理の最後のデフォルトアクション	
DATA ステップの最後: 自動書き込み、自動戻り DATA ステップの先頭: 自動リセット	SAS データセットにオブザベーションを書き込む DATA ステートメントに戻る プログラムデータベクトルの値を欠損にリセットする

1 この表は、DATA ステップのデフォルト処理を示しています。DATA ステップのステートメントの順序は変更できます。データ読み取りステートメントをコーディングする前に、定数の作成や再初期化などのオプションのプログラミングステートメントをコーディングできます。

注: 関数を使用してデータを読み取って処理することもできます。ステートメントと関数のデータの処理方法については、*"関数を使用したファイルの操作" (SAS 関数と CALL ルーチン: リファレンス)*を参照してください。SAS 関数に関する具体的な情報については、*"カテゴリ別の SAS 関数と CALL ルーチン" (SAS 関数と CALL ルーチン: リファレンス)*の SAS ファイル I/O および外部ファイルカテゴリを参照してください。

ステートメントを使用した実行順序の変更

デフォルトの実行順序を変更して、プログラムの実行方法を制御できます。SAS 言語ステートメントを使用すると、DATA ステップでこれを実行するための柔軟性が大

幅に向上します。次のリストは、DATA ステッププログラムの実行フローを制御するための最も一般的な方法を示しています。

表 A4.2　実行順序を変更する一般的な方法

タスク	可能な方法
レコードの読み取り	データセットのマージ、変更、結合 複数のレコードを読み取って単一のオブザベーションを作成する 処理対象のレコードをランダムに選択する 複数の外部ファイルから読み取る ステートメントまたはデータセットオプションを使用してレコードから選択したフィールドを読み取る
データの処理	条件付きロジックを使用する 変数値を保持する
オブザベーションの書き込み	SAS データセットまたは外部ファイルに書き込む 出力をデータセットに書き込むタイミングを制御する 複数ファイルに書き込む

詳細については、*SAS DATA ステップステートメント: リファレンス*の個々のステートメントを参照してください。

関数を使用した実行順序の変更

関数を使用してデータを読み取って処理することもできます。ステートメントと関数のデータ処理の違いについては、"関数を使用したファイルの操作" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

特定のオブザベーションのフローの変更

ステートメント、ステートメントオプション、およびデータセットオプションを使用して、SAS が特定のオブザベーションを処理する方法を変更できます。次の表に、SAS 言語要素とその処理への影響を示します。

表 A4.3　プログラミングフローを変更する言語要素

SAS 言語要素	機能
サブセット化 IF ステートメント	条件が偽の場合に現在の反復を停止し、現在のオブザベーションをデータセットに書き込まずに、DATA ステップの先頭に制御を戻します。
IF-THEN/ELSE ステートメント	現在の条件に一致するオブザベーションに対して、SAS ステートメントを実行し、次のステートメントから続行します。
DO ループ	DATA ステップの一部が複数回実行されるようになります。
LINK ステートメントと RETURN ステートメント	制御フローを変更し、指定したラベルに続くステートメントを実行して、LINK ステートメントに続く次のステートメントにプログラム制御を戻します。
FILE ステートメントの HEADER=オプション	PUT ステートメントによって新しい出力ページが開始されるたびに、制御フローが変更されます。HEADER=オプションで指定されたラベルに続くステートメントは、RETURN ステートメントが検出されるまで実行され、その時点で制御は HEADER=オプションがアクティブ化されたポイントに戻ります。
GO TO ステートメント	GO TO ステートメントで指定したラベルに分岐して、実行フローを変更します。SAS は後続のステートメントを実行し、制御を DATA ステップの先頭に戻します。
INFILE ステートメントの EOF=オプション	入力ファイルの終わりに達したときに実行フローを変更します。EOF=オプションで指定されたラベルに続くステートメントがその時点で実行されます。
IF-THEN コンストラクトの_N_自動変数	DATA ステップの一部が特定の反復でのみ実行されるようになります。
SELECT ステートメント	SAS ステートメントのグループのいずれかが条件付きで実行されます。
IF-THEN コンストラクトの OUTPUT ステートメント	条件に基づいて、DATA ステップの終了前にオブザベーションを出力します。DATA ステップの下部で自動出力が行われないようにします。

SAS 言語要素	機能
IF-THEN コンストラクトの DELETE ステートメント	条件に基づいてオブザベーションを削除し、DATA ステップの先頭に戻ります。
IF-THEN コンストラクトの ABORT ステートメント	DATA ステップの実行を停止し、次の DATA ステップまたは PROC ステップから実行を再開するように SAS に指示します。ABORT ステートメントで指定されたオプションと操作方法に応じて、SAS プログラムの実行を完全に停止することもできます。
WHERE ステートメントまたは WHERE=データセットオプション	SAS が 1 つ以上の指定された基準に基づいて特定のオブザベーションを読み取るようにします。

ステップ境界

ステップ境界は SAS ステートメントが有効になるタイミングを決定するため、ステップ境界を理解することは SAS プログラミングにおいて重要な概念です。SAS では、SAS がデフォルトまたはステップの境界を越えた場合にのみプログラムステートメントを実行します。次の DATA ステップについて考えてみます。

```
data _null_; 1
  set allscores(drop=score5-score7);
  title 'Student Test Scores'; 2

data employees; 3
  set employee_list;
run;
```

1 DATA ステートメントは DATA ステップを開始し、ステップ境界となります。

2 TITLE ステートメントは最初の DATA ステップの境界の前に記述されているため、両方の DATA ステップで有効です。(TITLE ステートメントはグローバルステートメントです。)

3 DATA ステートメントは、最初の DATA ステップのデフォルトの境界です。

この例の TITLE ステートメントは、最初の DATA ステップと 2 番目の DATA ステップで有効です。これは、最初の DATA ステップの境界の前に TITLE ステートメントが記述されているためです。この例では、デフォルトのステップ境界 data employees;を使用しています。

次の例は、RUN ステートメントの後に挿入された OPTIONS ステートメントを示しています。

```
data scores; 1
  set allscores(drop=score5-score7);
run; 2
```

```
options firstobs=5 obs=55;  3

data test;
  set alltests;
run;
```

1 DATA ステートメントはステップ境界です。

2 RUN ステートメントは、最初の DATA ステップの境界です。

3 OPTIONS ステートメントは、2 番目の DATA ステップにのみ影響します。

OPTIONS のステートメントは、入力データセットから最初に読み取られるオブザベーションが 5 番目で、最後に読み取られるオブザベーションが 55 番目であることを指定します。OPTIONS ステートメントの直前に RUN ステートメントを挿入すると、SAS が OPTIONS ステートメントを検出する前に、最初の DATA ステップが境界(run;)に到達します。したがって、OPTIONS ステートメントの設定は、2 番目の DATA ステップに対してのみ有効になります。

DATA ステップ内のステートメントの後に RUN ステートメントを続けるのが、ステップの実行を開始するための最も簡単な方法ですが、RUN ステートメントは必ずしも必要なわけではありません。SAS は、SAS ステップに対していくつかのステップ境界を認識します。

- 別の DATA ステートメント
- PROC ステートメント
- RUN ステートメント

注: 対話モードで実行される SAS プログラムの場合、サブミットする最後のステップのステップ境界を示すために RUN ステートメントが必要です。

- データ行の後のセミコロン(DATALINES または CARDS ステートメントを使用)または 4 つのセミコロン(DATALINES4 または CARDS4 ステートメントを使用)
- ENDSAS ステートメント
- 非対話モードまたはバッチモードでは、SAS プログラミングステートメントを含むプログラムファイルの終わり
- QUIT ステートメント(一部のプロシジャの場合)

対話型処理中に DATA ステップをサブミットすると、SAS がステップ境界を検出するまで実行は開始されません。このため、すべてのステートメントを入力するまでステップが実行されないようにしながら、ステートメントを記述するときにサブミットできます。

DATA ステップの実行が停止する原因

DATA ステップは、入力ソースの種類と数によって異なる状況で実行を停止します。

表 A4.4　DATA ステップの実行が停止する原因

データ読み取り	データソース	SAS ステートメント	DATA ステップ停止
データなし			1 回のみ反復後
すべてのデータ			STOP または ABORT 実行時 データが尽きたとき
生データ	インストリームデータ行	INPUT ステートメント	最後のデータ行の読み取り後
	1 つの外部ファイル	INPUT および INFILE ステートメント	ファイルの終わりに達したとき
	複数の外部ファイル	INPUT および INFILE ステートメント	いずれかのファイルで初めてファイルの終わりに達したとき
順次オブザベーション	1 つの SAS データセット	SET および MODIFY ステートメント	最後のオブザベーションの読み取り後
	複数の SAS データセット	1 つの SET、MERGE、MODIFY、UPDATE ステートメント	すべての入力データセットが使い果たされたとき
	複数の SAS データセット	複数の SET、MERGE、MODIFY、UPDATE ステートメント	いずれかのデータ読み取りステートメントによってファイルの終わりに達したとき

POINT=オプションを使用する SET ステートメントで SAS データセットからオブザベーションを読み取る DATA ステップでは、入力 SAS データセットの終わりを検出する方法がありません。(この方法は、直接アクセスまたはランダムアクセスと呼ばれます。)このような DATA ステップでは、通常、STOP ステートメントが必要です。

DATA ステップは、STOP または ABORT ステートメントを実行したときにも停止します。OBS=などの一部のシステムオプションやデータセットオプションでは、DATA ステップが通常よりも早く停止することがあります。

VARINITCHK=システムオプションが ERROR に設定されている場合、変数が初期化されていないと、DATA ステップは処理を停止し、SAS ログにエラーを書き込みます。詳細については、*“VARINITCHK= システムオプション” (SAS システムオプション: リファレンス)*を参照してください。