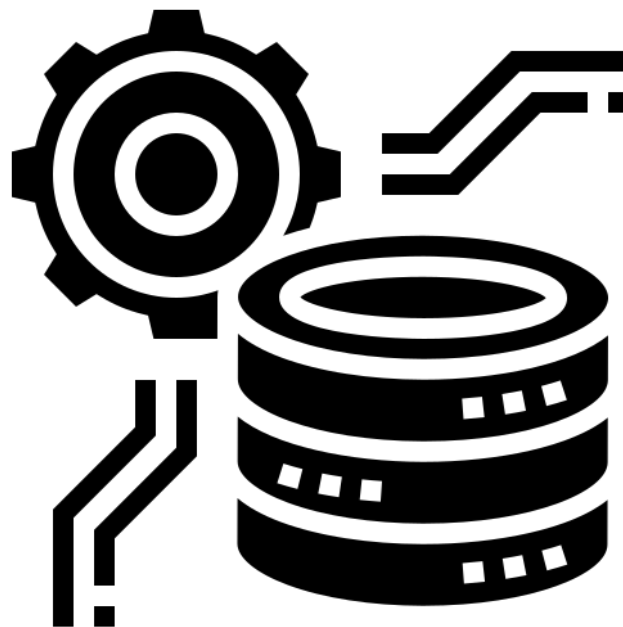


Meu Primeiro Livro de SAS®



E. Corrêa
Versão 1.0.0 - 08/07/2019

Detalhes de Copyright



Este trabalho está licenciado com uma Licença [Creative Commons - Atribuição-NãoComercial 4.0 Internacional](https://creativecommons.org/licenses/by-nc/4.0/). Isto significa que você pode alterar, transformar ou criar outra obra com base nesta, contanto que atribua o crédito ao autor original e não utilize a obra derivada para fins comerciais.

Versões

08/07/2019 – primeira versão completa do livro (v 1.0.0)

Número ISBN: 978-65-900095-1-7

Título: Meu primeiro livro de SAS

Autor: E. Corrêa

Palavras-chave: sas, banco de dados, sql, ciência de dados

Tipo de Suporte: Publicação digitalizada

Formato Ebook: PDF

Desenho da capa: https://www.flaticon.com/free-icon/data-processing_1878375

edubd 2019

SAS® e todos os outros nomes de produtos e serviços da SAS Institute Inc. são marcas registradas da SAS Institute Inc. nos Estados Unidos e demais países. ® indica registro nos Estados Unidos.

Outros nomes de marcas e produtos são marcas registradas de suas respectivas companhias.

Sumário

<i>Apresentação.....</i>	<i>7</i>
1. Muito Prazer, Sistema SAS®	9
1.1 A Arquitetura SAS.....	9
1.1.1 Base SAS®.....	9
1.1.2 SAS/ACCESS®	10
1.1.3 Outros Módulos.....	11
1.1.4 Comentários Finais.....	11
1.2 Enterprise Guide®	12
2. Introdução à Linguagem SAS.....	14
2.1 Criação de um Projeto no EG.....	14
2.2 “Olá SAS” – o Primeiro Programa SAS.....	16
2.3 Data Sets SAS.....	18
2.4 DATA e PROC: os Dois Passos de um Programa SAS.....	20
2.5 Libraries SAS.....	23
2.6 A Library WORK.....	24
2.7 LOG do SAS.....	24
3. Trabalhando com Bases de Dados no Formato Texto.....	27
3.1 INFILE e INPUT: Os Dois Comandos para Leitura de Arquivos.....	27
3.2 O Laço Implícito do Passo DATA (Built-in Loop).....	28
3.3 Leitura de Arquivos CSV.....	29
3.4 Outras Opções do Comando INFILE.....	31
3.5 A Variável Automática _INFILE_.....	34
3.6 Utilizando o Ponteiro de Colunas (@).....	36
3.7 FILE e PUT: Os Comandos para Gravação de Arquivos Texto.....	38
3.8 FILENAME.....	39
3.9 Leitura de Múltiplas Observações por Linha (@@).....	42
3.10 Trabalhando com Dados Numéricos Especiais (Nonstandard).....	44
3.11 Trabalhando com Valores do Tipo Data (DATE).....	45
3.12 Trabalhando com Valores do Tipo Hora (TIME).....	47
4. Transformação de Dados no Passo DATA.....	48
4.1 Criando Variáveis dentro do Passo DATA.....	48
4.2 As Instruções de Desvio IF-THEN-ELSE.....	49
4.3 Comparação de Comparação e Lógicos.....	51
4.4 Comparações Envolvendo o Valor Missing.....	53

4.5 Operadores Aritméticos e de String.....	54
4.6 Operação de Data Subsetting.....	54
4.7 Conhecendo as Data Set Options.....	56
4.8 Comando RETAIN.....	58
4.9 Comando LENGTH.....	60
4.11 Desvio Utilizando a Estrutura SELECT-WHEN.....	62
4.12 Comando OUTPUT.....	65
4.13 Funções do Base SAS.....	65
4.14 Trabalhando com Variáveis do Tipo DATETIME.....	70
4.15 Funções e Operações para a Manipulação de Datas.....	71
4.16 A Estrutura de Repetição DO-WHILE.....	73
4.17 A Estrutura de Repetição DO.....	75
4.18 A Estrutura de Repetição DO-UNTIL.....	76
4.19 Program Data Vector (PDV).....	76
5. Introdução às PROCs do Base SAS.....	81
5.1 O que é uma PROC?.....	81
5.2 PROC SORT.....	82
5.3 BY Variables, BY Values e BY Groups.....	84
5.4 PROC PRINT.....	85
5.5 PROC FREQ.....	90
5.6 Combinando PROC FREQ e PROC FORMAT para Tratar Variáveis Contínuas.....	95
5.7 PROC MEANS.....	96
5.8 PROC TABULATE.....	99
5.9 PROC CORR.....	103
5.10 PROC REPORT.....	104
5.11 Introdução à Tecnologia ODS – Output Delivery System.....	108
5.12 PROC TRANSPOSE.....	109
5.12 PROC APPEND.....	111
5.14 PROC COMPARE.....	114
5.15 PROC CONTENTS.....	117
6. Programação Avançada.....	119
6.1 SET.....	119
6.2 Comandos KEEP, DROP, RENAME e WHERE.....	121
6.3 Variáveis Temporárias FIRST e LAST.....	122

6.4 MERGE - Introdução.....	124
6.5 MERGE – Exemplo Básico.....	124
6.6 MERGE – Como Funciona?.....	127
6.7 MERGE – Opção IN.....	131
6.8 MERGE – Problemas mais Comuns.....	133
6.9 MERGE sem BY.....	135
6.10 MODIFY - Introdução.....	136
6.11 MODIFY com o Comando BY.....	137
6.12 Arrays.....	140
6.13 Macro Variáveis.....	142
6.14 Macro Variáveis Dinâmicas – CALL SYMPUT.....	145
6.15 Macros.....	146
6.16 Macro – Como Funciona?.....	148
6.17 Sistema SAS e os SGBD Relacionais.....	149
6.18 PROC SQL - Introdução.....	151
6.19 PROC SQL – Junção de Tabelas.....	154
6.20 PROC SQL – CREATE TABLE ... AS SELECT.....	159
6.21 MERGE x SQL JOIN.....	159
6.22 PROC SQL – Produzindo Resultados Agregados.....	160
6.23 Dicas Finais.....	164
<i>REFERÊNCIAS BIBLIOGRÁFICAS.....</i>	<i>165</i>
<i>Sobre o Autor.....</i>	<i>170</i>

Apresentação

O sistema SAS[®] pode ser definido como um **ambiente de software** completo para a execução de processos de *business intelligence* e *big data analytics* nas empresas. Trata-se de um sistema extremamente versátil, eficiente e poderoso, composto por diferentes módulos especialmente direcionados para realizar o **acesso, manipulação, integração e análise** de grandes bases de dados.

O software foi criado por um grupo de pesquisadores da *North Carolina State University*, no final dos anos 60 com o objetivo de servir como ferramenta para a análise estatística de dados provenientes de uma pesquisa agrícola. O nome SAS era usado nesta época como acrônimo para *Statistical Analysis Software*.

Durante a primeira metade dos anos 70 o programa tornou-se extremamente popular em ambiente acadêmico e também começou a ser utilizado dentro de muitas empresas. Isto motivou os seus criadores a fundar, no ano de 1976, o *SAS Institute*¹, companhia que passou a ser responsável pelo desenvolvimento e comercialização de novas versões do produto (que atualmente se encontra no *Release 9.4*). Desde então, a empresa consolidou-se como uma das mais importantes do mundo no setor de desenvolvimento de software, sendo responsável pela criação de aplicativos voltados não apenas para análise estatística tradicional, mas também para diversas outras tarefas, tais como integração de dados, *data warehousing*, segurança de informações, inteligência artificial e *machine learning*, entre outras.

Este livro apresenta os principais conceitos e técnicas para o desenvolvimento de programas de alto desempenho na **linguagem SAS** – a poderosa linguagem de programação utilizada no sistema SAS. É importante deixar claro que o livro não tem como foco o ensino da Estatística com o SAS, mas tem por objetivo principal **ensinar o leitor a programar na linguagem SAS**. Mais claramente, o objetivo do livro é tornar leitor um verdadeiro programador SAS, capacitando-o a desenvolver qualquer tipo de sistema no ambiente SAS, incluindo programas que sejam capazes de analisar grandes bases de dados através de técnicas estatísticas. Desta forma, o livro destina-se a diferentes tipos de profissionais envolvidos com big data e análise de dados, tais como estatísticos, cientistas da computação, engenheiros, matemáticos e bioinformatas, entre outros. O livro está dividido em seis capítulos, descritos a seguir:

- Capítulo 1, Muito Prazer, Sistema SAS. Fornece uma introdução à arquitetura do sistema SAS.
- Capítulo 2, Introdução à Linguagem SAS. Apresenta os conceitos básicos sobre programação em SAS: como criar e executar programas; propriedades dos data sets SAS; estrutura básica dos programas SAS (passos DATA e PROC); libraries; logs de execução de programas.
- Capítulo 3, Trabalhando com Bases de Dados no Formato Texto. Antes que os dados possam ser analisados por técnicas estatísticas, eles precisam ser carregados no SAS. Este capítulo apresenta as diferentes formas de importar bases de dados no formato texto (CSV, arquivo sequencial, etc.) para data sets SAS e também para

¹Alguns anos depois a companhia retirou a palavra “Institute” de seu nome e passou a se chamar apenas SAS[®].

realizar a operação inversa, ou seja, gravar o conteúdo de data sets para arquivos texto.

- Capítulo 4, Transformação de Dados no Passo DATA. Descreve as principais técnicas que podem ser utilizadas para transformar bases de dados.
- Capítulo 5, Introdução às PROCS do Base SAS. Apresenta as principais procedures para produção de relatórios, análise de dados e utilitárias disponibilizadas pelo módulo Base SAS (o módulo central do sistema SAS).
- Capítulo 6, Programação Avançada. Aborda tópicos avançados de programação: combinação e modificação de data sets; conexão do SAS com bancos de dados; incorporação de comandos SQL ao código SAS; utilização de arrays e macros em programas.

Grande parte do texto apresentado neste livro refere-se a código SAS (pequenos programas exemplo) e os resultados produzidos pelo código (normalmente na forma de data sets, relatórios ou tabulações). As seguintes convenções tipográficas foram adotadas:

- fonte com largura constante
 - Usada nas listagens dos programas SAS e na apresentação do conteúdo de bases de dados estruturadas em arquivos texto.
- *Fonte itálica*
 - Usada para representar os nomes dos data sets SAS.
- “palavra entre aspas duplas”
 - Usada para representar nomes de variáveis e nomes de arquivos texto.
- FONTE MAIÚSCULA
 - Usada para representar os nomes das palavras reservadas da linguagem SAS (comandos, funções e procedures). Esta convenção é adotada tanto nas listagens dos programas como nos textos que explicam o funcionamento dos mesmos.

1. Muito Prazer, Sistema SAS®

Este capítulo introduz a arquitetura SAS através de uma breve apresentação dos módulos mais conhecidos e utilizados deste sistema.

1.1 A Arquitetura SAS

O sistema SAS é composto por diversos módulos (subsistemas) que podem ser licenciados e integrados de maneira separada pelas organizações, conforme as necessidades destas. Ao conjunto de módulos licenciados, atribui-se o nome de “ambiente SAS” da empresa. As subseções a seguir realizam uma breve introdução ao Base SAS (módulo principal do sistema) e a outros módulos importantes do sistema SAS.

1.1.1 Base SAS®

Considerado o “coração” do sistema, o módulo Base SAS tem por finalidade oferecer funcionalidades para o acesso e transformação de dados, a realização de consultas livres sobre bases de dados e a produção de relatórios em diversos formatos (PDF, HTML, XML, RTF, XLS, entre outros). Além disso, o módulo também oferece suporte para a análise de dados, através de ferramentas que possibilitam a obtenção de estatísticas básicas sobre conjuntos de dados – desde o cálculo de média, variância, etc. até a geração de tabelas de frequências e a análise de correlação entre variáveis.

O Base SAS oferece todas essas funcionalidades por meio de uma linguagem de programação denominada **linguagem SAS**. Trata-se de uma linguagem dotada de recursos extremamente poderosos, cujas principais características são relacionadas abaixo:

- **Possui sintaxe “econômica”:** normalmente um programa escrito em SAS contém um número de linhas de código significativamente menor do que um programa escrito em outra linguagem para resolver a mesma tarefa (como R, C++, PL/SQL, Python ou Java). Como consequência, a linguagem SAS torna-se bem mais fácil de ser aprendida do que a maioria das outras linguagens.
- **Possui centenas de PROCs pré-definidas:** com essas PROCs, o desenvolvedor consegue transformar e analisar conjuntos de dados e produzir relatórios dos mais diversos tipos escrevendo poucas linhas de código.
- **Suporta o uso da linguagem SQL:** esta característica facilita a troca de dados entre a maioria dos repositórios de dados e o sistema SAS, já que SQL é a linguagem nativa dos bancos de dados.
- **Suporta a definição de rotinas reutilizáveis:** são as chamadas “macros” do SAS.
- **Possibilita a criação de programas “duráveis”:** um código escrito em SAS não precisa sofrer manutenção significativa mesmo quando a versão do sistema SAS é atualizada (ex: mudança da versão do sistema de 8.0 para 9.1). Por isto, é comum encontrar programas SAS que foram escritos há 10 ou mais anos em operação nas empresas, algo que raramente acontece com programas construídos em outras linguagens.

O módulo Base SAS pode ser estendido através do licenciamento de outros módulos. A incorporação de um módulo faz com que novos comandos e funções sejam acrescentados de maneira automática à linguagem SAS. As subseções a seguir apresentam alguns dos módulos mais utilizados, com ênfase no SAS/ACCESS®, uma vez que este módulo é de grande importância em projetos que envolvem a análise de Big Data.

1.1.2 SAS/ACCESS®

O módulo Base SAS é extremamente flexível para a leitura de arquivos tradicionais. Ele possibilita o acesso a diversos formatos de arquivos texto (CSV, XML, arquivos ASCII separados por coluna, etc.) e também consegue trabalhar com arquivos binários, hexadecimais, além de arquivos no antigo padrão EBCDIC. No entanto, o Base SAS não trabalha diretamente com dados nativos de alguns tipos de softwares importantes muito comumente encontrados nas empresas:

- Para começar, o Base SAS não é capaz de acessar informações armazenadas em **sistemas gerenciadores de bancos de dados** (SGBDs). Para os que não conhecem o termo, SGBD pode ser definido como um software especial e muito sofisticado que permite a construção, manipulação e compartilhamento de bancos de dados entre vários usuários e aplicações. Alguns exemplos de SGBDs conhecidos são Oracle, Microsoft SQL Server, DB2, MySQL e PostgreSQL.
- O Base SAS também não é capaz realizar a leitura e manipulação de arquivos muito populares em ambiente Windows, como **planilhas Excel** (XLS ou XLSX) e bases de dados do **Microsoft Access** (extensões MDB, DBF, etc.).
- Por fim, mas não menos importante: o Base SAS não está preparado para interagir diretamente com os modernos sistemas de arquivos distribuídos da era Big Data, como o **Hadoop**.

Para que o acesso às fontes de dados relacionadas acima seja possível, é preciso que a empresa integre uma ou mais interfaces SAS/ACCESS ao seu ambiente SAS. Existem mais de vinte diferentes interfaces, cada qual para um tipo de tecnologia de armazenamento de dados específica. Melhor explicando: existe uma interface específica para o SGBD Oracle, uma específica para o SGBD Microsoft SQL Server, uma específica para o sistema Hadoop, etc. Há também a interface *SAS/ACCESS® to PC File Formats*, utilizada para permitir a manipulação de arquivos MDB, XLSX e outros formatos associados a aplicativos de PCs. É importante mencionar que cada interface é licenciada de forma separada, ou seja, a empresa precisará adquirir todas as interfaces que considerar necessárias, uma por uma, de acordo com o seu ambiente de informática.

Para que o conceito fique mais claro, a Figura 1 apresenta um exemplo de ambiente SAS, estruturado em um servidor Windows de uma empresa hipotética. Conforme mostra a figura, o ambiente exemplo possui, além do módulo Base SAS, mais dois outros módulos incorporados, que correspondem a duas diferentes interfaces SAS/ACCESS. Observe que neste ambiente, os programas desenvolvidos na linguagem SAS poderão utilizar as funcionalidades oferecidas pelo módulo Base SAS caso precisem acessar arquivos texto convencionais (como CSV e XML). Porém, se o programa tiver que acessar dados armazenados no SGBD Oracle, precisará fazer uso de funcionalidades que estão disponíveis apenas no módulo *SAS/ACCESS® Interface to Oracle*. Ou seja: apenas com o licenciamento deste módulo, torna-se possível realizar o acesso (nativo e de alto desempenho) ao Oracle. De maneira análoga, a Figura 1 indica que um programa SAS que tenha que ler e manipular planilhas Excel, precisará utilizar as funcionalidades disponibilizadas pelo módulo *SAS/ACCESS Interface to PC File Formats*.

Após essa breve discussão, é possível concluir que o licenciamento de interfaces SAS/ACCESS representa algo praticamente obrigatório para a maioria das organizações, uma vez que, nos dias atuais, qualquer empresa de pequeno, médio ou grande porte utiliza algum SGBD para armazenar os dados de seu negócio.

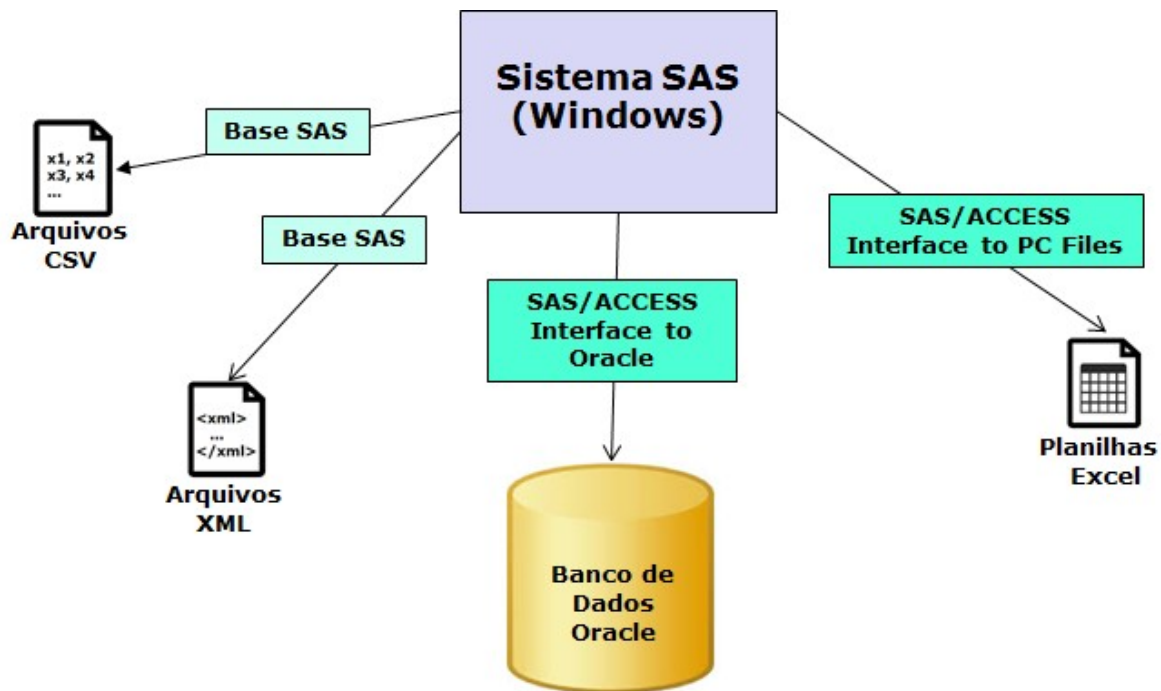


Figura 1. Exemplo de sistema SAS com duas interfaces SAS/ACCESS incorporadas

1.1.3 Outros Módulos

A seguir são relacionados outros módulos importantes do SAS:

- **SAS/Connect®**: módulo que torna possível a comunicação entre sessões SAS que estejam executando em diferentes plataformas.
- **SAS/ETS®**: pacote de funções para econometria e análise de séries temporais
- **SAS/GIS™**: incorpora um sistema de informações geográficas interativo ao SAS.
- **SAS/GRAPH®**: módulo que possibilita a produção de diferentes tipos de gráficos.
- **SAS/IML®**: módulo que oferece funções para a manipulação de matrizes.
- **SAS® Integration Technologies**: módulo que possibilita com que programas desenvolvidos em diferentes linguagens possam “conversar” com o ambiente SAS. Com este módulo é possível, por exemplo, fazer com que programas C# ou Java possam acessar data sets SAS ou até mesmo executar comandos da linguagem SAS.
- **SAS/QC®**: disponibiliza ferramentas de apoio ao controle de qualidade de processos de análise estatística.
- **SAS/STAT®**: pacote que incorpora mais de 80 PROCs para análise estatística à linguagem SAS: análise multivariada, análise bayesiana, determinação de agrupamentos (*clustering*), análise de dados categóricos, regressão, entre outras. É um dos módulos mais populares do SAS, tendo em vista que as raízes do software se encontram exatamente na análise estatística.

1.1.4 Comentários Finais

Esta seção apresentou uma breve descrição à arquitetura SAS. De acordo com o que foi discutido, a conclusão é que o ambiente SAS pode ser composto por módulos diferentes em diferentes empresas, dependendo da área de atuação de cada empresa. Uma universidade que deseje utilizar o SAS como ferramenta de apoio em um curso de Estatística poderia, por exemplo, licenciar os módulos Base SAS, SAS/STAT, SAS/IML,

SAS/GRAPH e SAS/ETS, mas não licenciar nenhum dos módulos SAS/ACCESS. Já uma empresa que deseje utilizar o SAS com o objetivo de construir programas que acessem SGBDs, precisaria do Base SAS e também de ao menos uma interface SAS/ACCESS. Mas esta empresa poderia não incorporar módulos como SAS/ETS ou SAS/IML ao seu ambiente SAS.

Na realidade, a empresa SAS oferece pacotes de soluções prontas, que trazem de forma “embutida” os módulos mais adequados para cada diferente tipo de organização (assim, o cliente não precisa conhecer qualquer detalhe sobre a arquitetura modular do sistema SAS). Alguns exemplos: indústrias farmacêuticas, instituições financeiras, órgãos governamentais, hotéis, entre outras. Existe até mesmo uma solução SAS específica para cassinos (isto é: uma solução já montada, contendo todos os módulos relevantes para os administradores e analistas que trabalham em cassinos).

Por fim, é importante comentar que o SAS é um software muito robusto e sofisticado e, por consequência, de custo elevado. Por esta razão, costuma ser encontrado apenas em grandes empresas e universidades. De acordo com o web site da SAS, nada menos do que 94 dentre as 100 maiores empresas do mundo², utilizaram software produzido pela SAS ao longo do ano de 2016.

Se a sua empresa ou universidade não possui o sistema, isto não significa que você jamais conseguirá utilizar o SAS. Felizmente, existe uma opção chamada SAS University Edition®, que corresponde a uma versão gratuita do sistema. Ela é dotada de todas as funcionalidades essenciais e destina-se a todos que desejam aprender a programação SAS³ em casa.

IMPORTANTE!

Este não é um livro de Estatística e sim um livro sobre programação na linguagem SAS. Por este motivo, possui como foco específico o pacote Base SAS, que representa o núcleo do sistema. É preciso ter em mente que para criar programas direcionados para a análise estatística na linguagem SAS, é previamente necessário ter um bom conhecimento sobre o Base SAS.

O módulo Base SAS é, por si só, extremamente sofisticado e poderoso. A partir dos exemplos apresentados ao longo deste livro, será demonstrado que é possível implementar processos inteiros de extração, carga, integração e análise de dados, utilizando apenas funcionalidades incluídas neste módulo.

1.2 Enterprise Guide®

Ambiente integrado de desenvolvimento (*Integrated Development Environment – IDE*) é um jargão utilizado na área de desenvolvimento de software para rotular aplicativos que tem o propósito de servir como ambiente para a criação, depuração e execução de programas em uma determinada linguagem. Apenas para apresentar exemplos bem conhecidos, podemos citar as IDEs Eclipse e NetBeans, ambas utilizadas para a criação de programas na linguagem Java.

²Relatório de maiores empresas da revista Fortune, ano de 2016.

³Disponível em: https://www.sas.com/en_us/software/university-edition.html

No caso da linguagem SAS, a IDE mais utilizada chama-se Enterprise Guide® (EG). Trata-se de um aplicativo cliente (software que roda em uma estação de trabalho) que funciona comunicando-se com um sistema SAS que esteja instalado na própria estação de trabalho (chamado de SAS local) ou em um servidor remoto (isto é: outro computador da rede, neste caso chamado de servidor SAS). O Enterprise Guide está disponível apenas para o sistema operacional Windows.

Curiosamente, o EG foi criado no ano de 1999 para atender às necessidades de usuários que não eram programadores, mas que desejavam utilizar o SAS através de uma interface *point and click* – ou seja, explorar e analisar bases de dados através de sequências de cliques do mouse sobre diferentes formulários e menus interativos. Porém, diferentemente do que ocorria com outras IDEs SAS da época, o EG surgiu com uma série de recursos capazes de aumentar a eficiência do processo de programação. Alguns exemplos: *autocomplete* (possibilita com que um comando seja automaticamente completado durante a digitação), *inline help* (exibe um texto de ajuda sobre uma palavra reservada quando o mouse é colocado sobre a mesma) e formatação automática de programas (realiza o alinhamento automático de comandos que estejam colocados entre o início e o fim de um bloco de instruções).

Outra vantagem importante do EG é que esta IDE baseia-se no utilíssimo conceito de **projeto SAS**, em vez de atuar apenas como um mero editor de programas. Um projeto do EG permite com que diversos programas e processos possam ser reunidos e rodados em uma sequência especificada, viabilizando a execução de tarefas complexas.

Devido a todas essas características, o EG acabou caindo no gosto dos programadores, tornando-se a principal IDE para desenvolvimento SAS. Por esta razão, o leitor deste livro aprenderá a criar projetos e executar os seus programas no EG. Apesar disso, é importante deixar claro que todos os programas que serão apresentados também podem ser normalmente executados no PC SAS® (conhecido por muitas pessoas como SAS Desktop ou “SAS de micro”), um ambiente de desenvolvimento que ainda é muito popular entre os amantes do SAS.

Agora é hora de você se preparar, pois a nossa jornada rumo ao domínio da programação SAS está oficialmente iniciada!

2. Introdução à Linguagem SAS

Este capítulo apresenta os conceitos básicos sobre programação em SAS. As Seções 2.1 e 2.2 fornecem a receita para criar e executar programas no EG, através da elaboração de um pequeno projeto exemplo. A Seção 2.3 introduz a estrutura de dados “data set”, que corresponde ao principal objeto manipulado em programas SAS. Em seguida, a Seção 2.4 descreve a estrutura básica dos programas SAS, apresentando os passos DATA e PROC. Encerrando o capítulo, as últimas seções cobrem outros dois importantes conceitos associados ao universo SAS: as libraries (Seções 2.5 e 2.6) e os logs de execução (Seção 2.7).

2.1 Criação de um Projeto no EG

Conforme descrito na Seção 1.2, qualquer tipo de tarefa realizada no EG precisa ser organizada em um projeto. Um projeto pode conter uma coleção de programas, logs, data sets e relatórios, entre outros objetos do SAS (todos esses objetos serão apresentados ao longo do livro, alguns deles ainda neste capítulo). Desta forma, para que seja possível criar um novo programa SAS, é preciso antes de tudo criar um novo projeto ou abrir um projeto existente no EG.

Nesta seção, são descritos os passos necessários para a criação de seu primeiro projeto. Basta seguir o roteiro abaixo:

1. **Iniciar o EG:** caso você esteja utilizando o EG no ambiente cliente-servidor (ou seja, o seu EG acessa um servidor remoto SAS), provavelmente o seu login e senha de rede serão requeridos antes que o EG seja iniciado. No entanto, caso o seu sistema SAS seja local (instalado na própria estação de trabalho), nenhuma tela de login será apresentada.
 - Após o EG iniciar, uma tela similar à mostrada na Figura 2 será aberta. Trata-se da tela principal do EG.

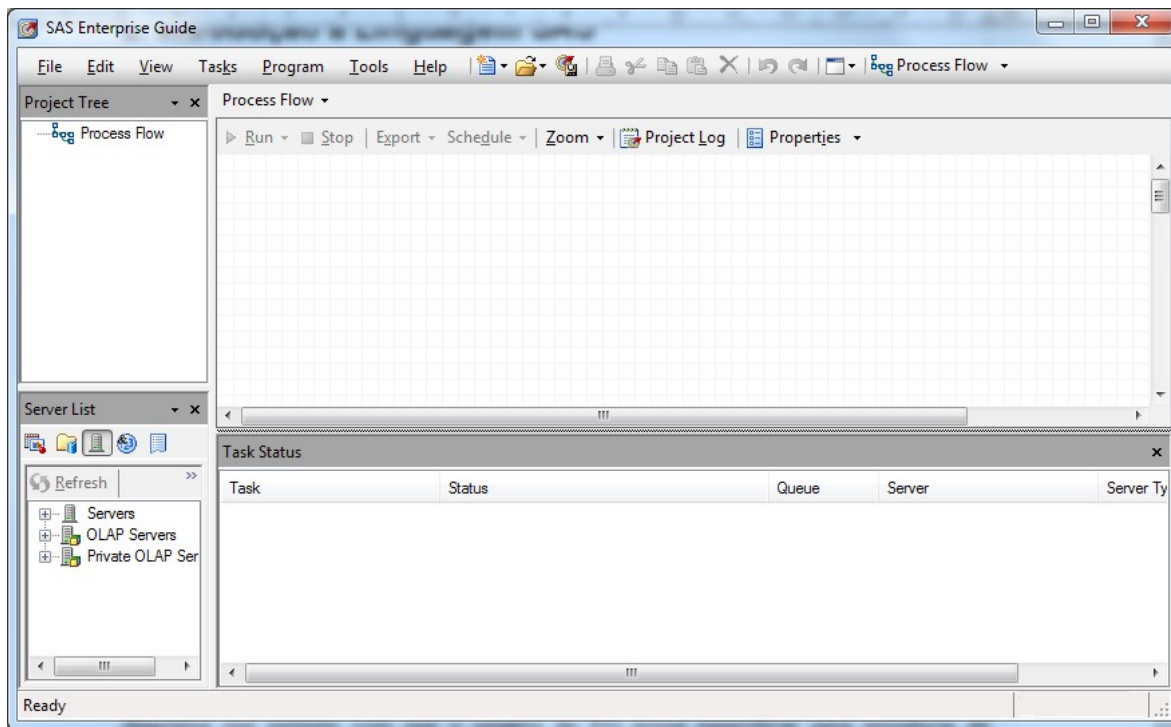


Figura 2. Tela principal do EG

- Observe a janela “**Project Tree**” (chamada de “Project Explorer” em versões antigas do EG), localizada no canto superior esquerdo. Ela é responsável por apresentar o nome do projeto e todos os seus componentes. Por enquanto, o seu projeto possui apenas um objeto, do tipo “**Process Flow**”, que foi criado automaticamente. O “Process Flow” é um diagrama que permite com que o usuário do EG possa especificar a sequência de programas ou ações a serem executadas para resolver um dado problema.
 - Também no canto esquerdo da tela, mas na parte inferior, localiza-se outra janela importante, a “**Server List**”. Nela são apresentadas informações a respeito dos servidores SAS da empresa. Quando o sistema SAS está instalado em um servidor remoto, o nome deste servidor aparecerá na relação apresentada abaixo do ícone “Servers”. No entanto, se o sistema SAS estiver instalado na mesma máquina do EG, o nome “Local” é que constará da relação apresentada em “Servers”.
2. **Salvar o projeto:** o projeto ainda está em branco, pois nenhum programa foi criado. Entretanto, isso não impede que o mesmo seja salvo. Há duas formas de salvar o projeto: com o uso do menu **File > Save Project** ou digitando **CTRL + S**. Basta escolher uma pasta e dar um nome para o projeto, como, por exemplo “prjLivroSAS”. Ele será gravado com a extensão ***.egp** (Enterprise Guide Project).

2.2 “Olá SAS” – o Primeiro Programa SAS

Execute as etapas descritas a seguir para criar e executar o seu primeiro programa SAS:

1. **Acrescentar um programa ao projeto EG:** utilize o menu **File > New > Program** ou clique no ícone **New** e selecionar a opção **Program**.
 - Após executar esta ação, um objeto do tipo “**Program**” será adicionado automaticamente ao “**Process Flow**”. Além disso, o editor de código será aberto, para que você possa digitar o código do programa (Figura 3).
 - Embora não seja obrigatório, é possível renomear o programa que você inseriu. Para tal, clique com o botão direito sobre o ícone “**Program**”, selecione a opção **Rename** e digite um novo nome: “**Exemplo1**” (Figura 4).

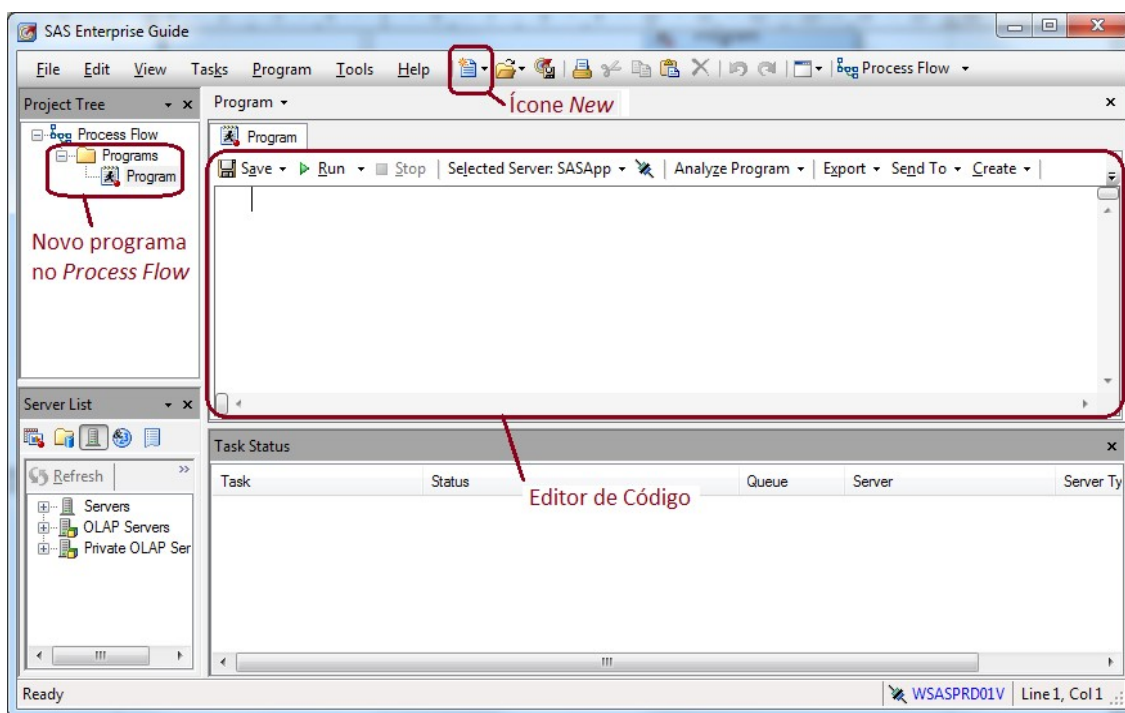


Figura 3. Novo programa adicionado ao projeto

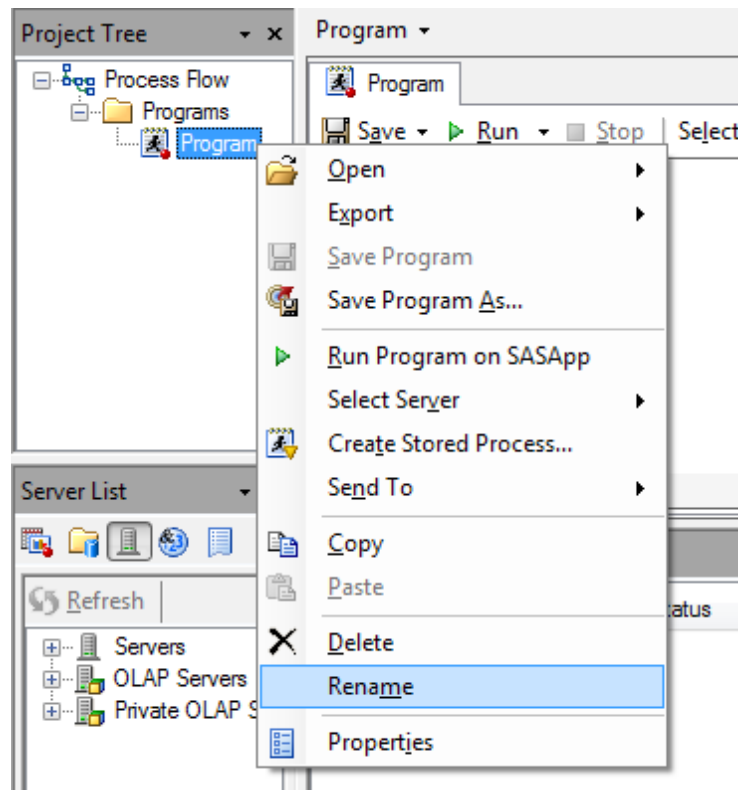


Figura 4. Renomeando o programa recém-inserido

2. **Digitar o programa:** digite na janela do editor, o pequeno programa listado abaixo. Este programa cria um **data set** (tabela) chamado *Ola*, que contém apenas uma **variável** (coluna), chamada “mensagem”, e uma única **observação** (linha) com o valor ‘Hello SAS!’.

```
/* Programa 1: Hello SAS */
* Cria o data set "Ola", contendo 1 linha e 1 observação;
DATA Ola;
    mensagem = 'Hello SAS!';
RUN;
```

3. **Analisar o programa:** o programa possui cinco linhas de código.
 - As duas primeiras linhas contêm comentários (ou seja, não constituem instruções, mas apenas textos com comentários do programador). Na linguagem SAS, os comentários podem ser definidos entre os símbolos “/*” e “*/” (como na primeira linha) ou entre os símbolos “ * ” e “ ; ” (como na segunda linha).
 - As instruções propriamente ditas correspondem às três últimas linhas do programa. Por enquanto não será explicado o que cada linha faz. Por ora, basta que você observe que toda instrução SAS é encerrada com um ponto-e-vírgula (“ ;”).

4. **Executar o programa:** existem algumas maneiras distintas:
- Selecione a opção de menu **Program > Run**; ou
 - Clique com o botão direito do mouse sobre o nome do programa no *Process Flow* e depois selecione a opção “Run”; ou
 - Simplesmente pressione F8 ou F3;

Após o término da execução do programa, duas abas serão automaticamente acrescentadas juntas ao editor de código: “**Log**” e “**Output Data**” (Figura 5). A aba “Log” apresenta um resumo contendo informações sobre a execução do programa, tais como tempo de execução, status (execução normal ou com erro), etc. Maiores detalhes são apresentados na Seção 2.7. Já a aba “Output Data”, contém o data set *Ola*, gerado exatamente através da execução do Programa 1. Clique nesta aba para examinar o conteúdo do data set. Veja que ele possui apenas uma linha e uma coluna. Esta única célula, mostra a mensagem ‘Hello SAS!’.

Pronto! O seu primeiro programa SAS foi criado e executado no EG. Não esqueça de salvar o projeto para que ele não seja perdido quando o EG for encerrado. De agora em diante, basta utilizar os mesmos passos para acrescentar os outros programas ao projeto recém-criado.

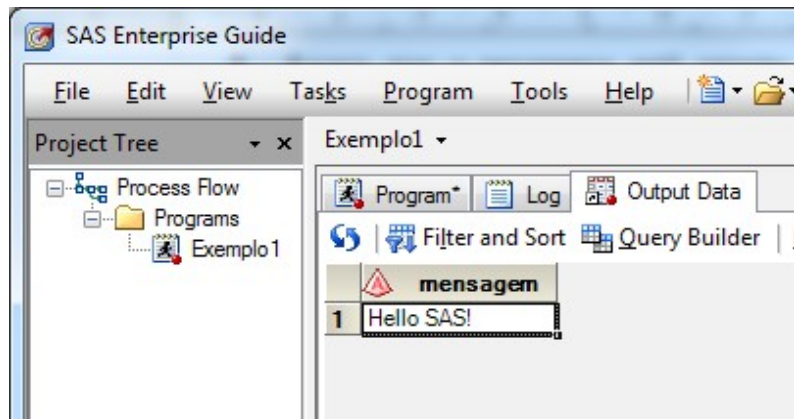


Figura 5. As abas Log e Output Data e o data set “Hello”

2.3 Data Sets SAS

Data set é o objeto utilizado pelo SAS para o armazenamento de dados. É similar a um data frame da linguagem R ou a uma tabela de um banco de dados. A Figura 6 mostra um data set que contém dados de filmes produzidos em diferentes anos e países. O data set contém seis variáveis: “título” (título do filme), “ano” (ano de produção), “país” (país de produção), “duracao” (duração em minutos) e “avaliacao” (nota obtida através da avaliação do público).






	 titulo	 ano	 pais	 duracao	 avaliacao
1	Monty Python em Busca do Cálice Sagrado	1975	GBR	91	8.3
2	Noel: Poeta da Vila	2006	BRA	99	8.1
3	As Pontes de Madison	1995	USA	135	7.3
4	Meu Melhor Amigo	2006	FRA	94	6.8
5	La La Land	2016	USA	128	8.4
6	Intocáveis	2011	FRA	112	8.6
7	O Filho da Noiva	2001	ARG	.	9.1
8	Vidas Secas	1963	BRA	103	.
9	Hair	1979	USA	121	7.6
10	Edukators	2004	DEU	127	7.5

Figura 6. Data set de filmes

IMPORTANTE!

O data set é a estrutura de dados mais importante do SAS. Para que seja possível realizar a análise estatística, produzir relatórios ou manipular dados com o uso dos comandos, funções e procedimentos disponibilizados por qualquer módulo SAS, primeiro é preciso gerar um data set com os dados que serão trabalhados.

A seguir são relacionadas algumas características inerentes às variáveis de data sets:

2.3.1 Tipos de Dados

As variáveis SAS podem ser de apenas dois tipos: numérico e caractere. As variáveis numéricas podem ser positivas ou negativas, inteiras ou reais com qualquer número de casas decimais e são armazenadas internamente utilizando a representação em ponto flutuante. Já as variáveis do tipo caractere, também chamadas de tipo *string*, podem conter letras, dígitos e qualquer outro tipo de caractere especial. No data set da Figura 6 existem três variáveis numéricas (“ano”, “duracao” e “avaliacao”) e duas variáveis do tipo caractere (“titulo” e “pais”). Note que o EG usa ícones e cores diferentes para apresentar variáveis numéricas (o ícone é um triângulo vermelho) e caractere (o círculo azul é adotado como ícone).

2.3.2 Tamanho

Tanto as variáveis numéricas como as do tipo caractere podem ocupar de 2 a 8 bytes, sendo 8 bytes o tamanho default. O número máximo de caracteres suportados por uma variável do tipo caractere é 32.767. Já uma variável numérica consegue armazenar um número inteiro com valor máximo de 9.007.199.254.740.992.

2.3.3 Nomes

Um nome de variável pode ter até 32 caracteres. O primeiro caractere deve ser uma letra ou underscore (_). Os caracteres seguintes podem ser letras, números ou underscores. É recomendável que você não crie variáveis que começam e terminam com underscore, pois este padrão é adotado pelas variáveis automáticas (como será mostrado a partir do Capítulo 3).

2.3.4 Valores Missing

Um valor nulo em uma variável é chamado de *missing* no vocabulário SAS. Em um data set SAS, as variáveis do tipo caractere representam um valor missing com um espaço

em branco, enquanto as variáveis numéricas representam o missing com um ponto ("."). Veja que na Figura 5, a variável “duracao” do filme “Intocáveis” possui valor missing. Da mesma forma, a variável “avaliacao” do filme “Vidas Secas” também “está missing” (utilizando o jargão dos programadores).

2.4 DATA e PROC: os Dois Passos de um Programa SAS

A linguagem SAS possui uma sua sintaxe econômica que favorece a criação de programas menores e mais simples do que o de outras linguagens. Curiosamente, esta linguagem não segue exatamente os paradigmas de programação mais comuns, como a programação estruturada ou a programação orientada a objetos. Na realidade, trata-se de uma linguagem **orientada a passos**.

Por que orientada a passos? Isto ocorre porque todo e qualquer programa SAS é sempre construído a partir de **apenas dois tipos de passos**: DATA e PROC. Melhor explicando: na linguagem SAS, o conjunto de instruções que irá compor um programa deve ser sempre colocado dentro de um ou do outro passo.

O esquema apresentado na Figura 7 ilustra o fluxo de um programa SAS típico. Veja que o programa começa com um passo DATA, que é responsável pela criação de um data set SAS. O data set em questão pode ser construído a partir da importação de dados oriundos de fontes de dados distintas como, por exemplo, arquivos texto ou tabelas de SGBDs relacionais. A seguir, o programa segue a sua execução com um passo PROC, que normalmente executará algum tipo de tarefa de análise sobre os dados do data set criado no passo DATA. Em função desta análise, será produzido um relatório, gráfico, modelo ou tabela de resultados.

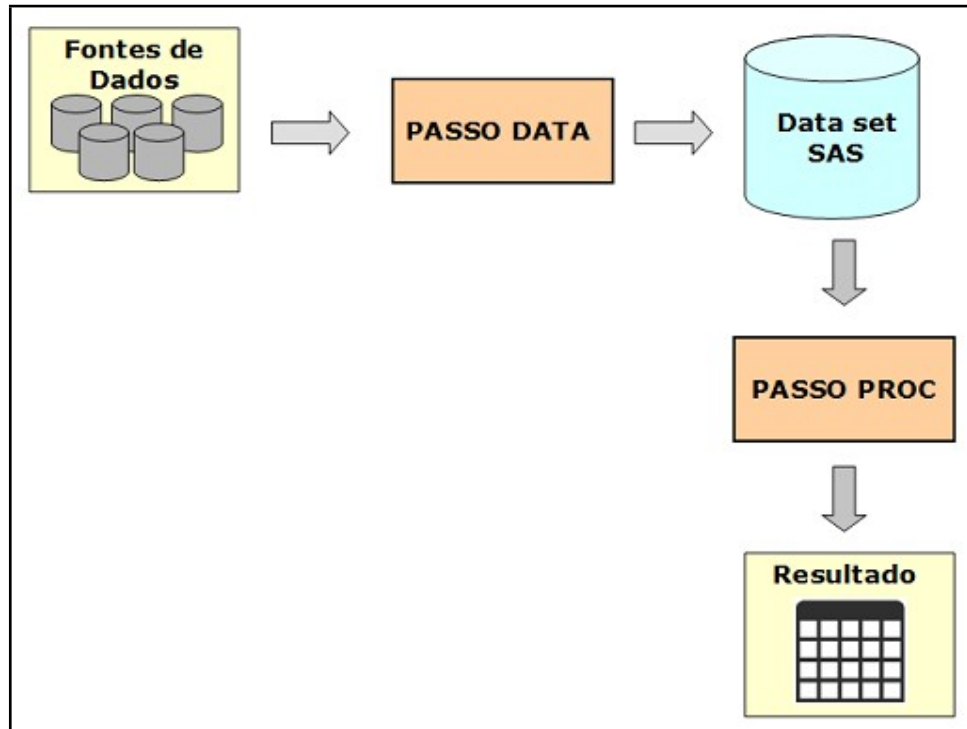


Figura 7. Fluxo de um típico programa SAS

Para que o conceito fique mais claro, digite e execute o Programa 2. Após executar, examine o texto restante dessa seção, onde é feita uma explicação passo-a-passo sobre cada linha do programa.

```
* Programa 2: Introduz os passos DATA e PROC;

* Passo Data - cria o data set "Funcionarios";
DATA Funcionarios;

    INPUT matricula $ nome $ idade sexo $;

    DATALINES;
M01 George 58 M
M02 Jane 32 M
M03 Aldous 40 F
M04 Thomas 28 M
M05 Mary 43 F
    ;
RUN;

* Passo PROC - gera um relatório a partir de "Funcionarios";
PROC PRINT DATA=Funcionarios;
RUN;
```

O programa exemplo possui um passo DATA (linhas 4-15) e um passo PROC (linhas 18-19). A primeira instrução do passo DATA é:

```
DATA Funcionarios;
```

Todo passo DATA começa com a palavra reservada DATA seguida por um nome, que representa o nome do data set que será produzido ao final da execução do passo (neste exemplo, *Funcionarios*). Todo passo DATA produz um ou mais data sets ao final de sua execução. Normalmente os dados usados para montar o data set são obtidos através de fontes externas (arquivos ou tabelas de bancos de dados). No entanto, no Programa 2, o data set *Funcionarios* foi populado com dados internos (dados especificados no próprio código fonte), como o uso de dois comandos: INPUT e DATALINES.

O comando INPUT é um dos mais importantes do SAS e será discutido em detalhes no Capítulo 3. No Programa 2, ele foi utilizado com o propósito de especificar as variáveis do data set *Funcionarios*. Estas variáveis são “matricula” (tipo caractere), “nome” (tipo caractere), “idade” (tipo numérico) e “sexo” (tipo caractere). O símbolo “\$” ao lado de “matricula”, “nome” e “sexo” indica ao SAS que estas variáveis são do tipo caractere.

O comando DATALINES é utilizado para o lançamento das observações do data set. Todas as linhas que forem encontradas após a palavra DATALINES serão consideradas como novas observações até que o SAS encontre um ponto-e-vírgula (linha 14). Os valores de cada variável devem ser separados com espaço em branco. O comando DATALINES não é muito utilizado em aplicações práticas, pois, conforme mencionado anteriormente, normalmente os dados de um data set são montados a partir de fontes externas. No entanto, ele costuma ser muito utilizado para fins didáticos (em exemplos, livros e manuais do SAS). Duas observações importantes: (i) a linguagem SAS possui um outro comando que possui a mesma funcionalidade DATALINES, chamado CARDS

(talvez você esbarre com ele em algum código antigo); (ii) a linguagem SAS **não** é *case-sensitive*, ou seja, os comandos e palavras reservadas (como INPUT, DATA, DATALINES, RUN, etc.) podem ser escritos em letra maiúscula ou minúscula. No entanto, por convenção, esses comandos serão sempre especificados em maiúsculo nas listagens apresentadas neste livro.

O comando RUN marca o fim do passo DATA. Na realidade, um passo DATA ou PROC é encerrado de três formas distintas: (i) quando o SAS encontra um novo passo (marcado pelas palavras-chave DATA ou PROC); (ii) quando o fim do programa é alcançado; ou (iii) explicitamente, quando o comando RUN é encontrado (forma utilizada no Programa 2). Apesar de não ser obrigatório, o comando RUN é muito utilizado porque melhora a legibilidade do código e do log gerado.

Após o final do passo DATA, inicia-se o passo PROC, que é composto por apenas uma instrução:

```
PROC PRINT DATA=Funcionarios;
```

Um passo PROC do SAS começa com a palavra PROC seguida do nome da *procedure* a ser executada (como PRINT, FREQ, MEANS, CORR, entre outras). A procedure PRINT é uma das mais utilizadas do SAS e sua função é produzir um relatório com o conteúdo de um data set. Maiores detalhes sobre a PROC PRINT serão apresentados no Capítulo 5. O formato do relatório gerado pela PROC PRINT no EG é configurável. Para definir o formato de sua preferência, acesse a opção de menu **Tools > Options ... > Results General**. No item **Results Format**, você poderá escolher entre as seguintes opções: SAS Report (saída padrão do SAS), HTML, PDF, RTF e Text Output (texto comum).

2.4.1 Características dos Passos DATA e PROC

O Programa 2, apresentado nesta seção, possui apenas um passo DATA e um passo PROC. No entanto, um programa SAS pode ser composto por inúmeros passos DATA e PROC, que podem ser intercalados da forma que o programador desejar. Além disso, é possível criar programas que possuam apenas passos DATA (algo até certo ponto comum) ou apenas passos PROC (situação bem menos comum). Existem comandos do SAS que devem ser utilizados apenas dentro dos passos DATA e aqueles usados apenas dentro de passos PROC. Um erro cometido por muitos iniciantes é o de tentar utilizar um comando no passo errado. Para que você não cometa esse erro, procure lembrar das diferenças básicas entre os dois passos:

- **Passo DATA**
 - Começa com a palavra DATA.
 - É usado para leitura e modificação de dados, oriundos de fontes diversas.
 - Cria um data set.
- **Passo PROC**
 - Começa com a palavra PROC.
 - É normalmente usado para invocar alguma procedure para análise de dados do SAS.
 - Produz algum tipo de relatório, gráfico, modelo ou tabela de resultados.

IMPORTANTE!

Na prática, nem sempre as diferenças entre os passos DATA e PROC são tão claras e rígidas. Conforme será apresentado ao longo desse livro, existem situações especiais onde é possível criar um data set dentro de um passo PROC ou gerar relatórios dentro de um passo DATA.

Conforme mencionado no início da seção, a linguagem SAS é uma linguagem orientada a passos e não uma linguagem procedural ou orientada a objetos. Nas linguagens convencionais as informações são passadas entre as rotinas através do uso de parâmetros e estruturas em memória. No SAS, a comunicação entre os passos é feita, tipicamente, com o uso de data sets. Inicialmente, a estrutura de um programa SAS pode parecer estranha, se comparada com a de outras linguagens. No entanto, lembre-se de que o SAS não é uma linguagem de propósito geral, como Python, C# ou Java, que são utilizadas para a criação de sistemas. Na realidade a linguagem SAS tem um propósito específico: criar programas para a integração, modificação e análise de grandes bases de dados.

2.5 Libraries SAS

Uma *library* SAS é um **repositório de data sets SAS**. No ambiente Windows, uma library SAS corresponde simplesmente a uma pasta qualquer da máquina local do usuário ou de algum compartilhamento de rede. O comando LIBNAME indica ao SAS a localização da library. Para que o conceito fique mais claro, veja o seguinte exemplo que define uma library chamada “my_lib” associada à pasta “c:\bases” do computador local.

```
* Programa 3a: cria uma library para uma pasta da máquina local;  
libname my_lib 'c:\bases';
```

Antes de explicar o Programa 3a, é preciso apresentar um conceito muito importante: em ambientes onde o EG acessa um servidor SAS remoto, você **não** conseguirá utilizar uma library que aponte para uma pasta da sua estação de trabalho (como foi feito no exemplo acima). Isto ocorre porque quando se usa o SAS em rede, para que você possa acessar qualquer pasta ou arquivo, é normalmente necessário que estes estejam armazenados em algum compartilhamento de rede que possa ser enxergado pelo seu login de rede (direito de escrita, se você precisar apenas ler arquivos ou direito de gravação caso precise ler e criar/alterar). O Programa 3b apresenta um exemplo de library associada ao compartilhamento de rede chamado “\\X001\bases”. Converse com o administrador do sistema SAS de sua empresa para obter maiores detalhes.

```
* Programa 3b: cria uma library para um compartilhamento da rede;  
libname my_lib '\\X001\bases';
```

Supondo que você colocou um nome de pasta ou de compartilhamento que pode ser acessado pela sua conta, execute o programa exemplo. Após a execução, acesse a janela de

servidores (“Servers”). Expanda o servidor e a pasta “**Libraries**”. Observe que a library “my_lib” será apresentada. Outras libraries poderão estar sendo mostradas, como por exemplo MAPS, SASHELP, SASUSER e WORK. Todas são criadas automaticamente pelo SAS sempre que uma sessão é iniciada. A mais importante delas é a library WORK, a ser apresentada na próxima seção. Algumas dessas libraries poderão conter um ou mais data sets SAS. Para visualizar o conteúdo de um data set clique duas vezes sobre o nome do mesmo.

Como comentário final, observe que o comando libname é um comando especial de configuração. Por esta razão, ele deve ser especificado fora dos passos DATA e PROC.

2.6 A Library WORK

WORK é uma library especial criada **automaticamente** pelo SAS sempre que uma **sessão é iniciada**. Cada sessão SAS possui a sua própria library WORK associada.

Todo data set produzido por um passo DATA é criado dentro da WORK, caso nenhum outro local seja especificado no código de um programa SAS. A Figura 8 ilustra esse fato: veja que os data sets *Funcionarios* (criado na Seção 2.4) e *Ola* (criado na Seção 2.2) estão armazenados exatamente dentro da WORK (Obs.: o outro data set chamado *_PRODSAVAIL* foi criado automaticamente pelo SAS. Ele contém a lista de módulos licenciados no sistema em uso).

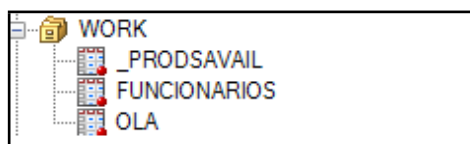


Figura 8. A library WORK

Como a WORK é uma library temporária, todo conteúdo nela armazenado também é excluído ao final de uma sessão SAS. De maneira mais clara, isto quer dizer que **os data sets da WORK também morrem no momento uma sessão SAS é finalizada!** Se você desejar persistir (ou seja, gravar permanentemente) um data set, precisará criá-lo em outra library.

ATENÇÃO!

Os data sets criados no SAS e persistidos fora da library WORK são gravados com a extensão “.sas7bdat” no ambiente Windows.

2.7 LOG do SAS

Um dos objetos mais interessantes do SAS é o seu log de execução de programas. Sempre que um programa é submetido para execução, uma aba chamada “Log” é automaticamente associada ao projeto do EG assim que a execução é terminada. Este log irá conter as linhas de código do programa e mensagens do sistema SAS relacionadas ao

status da execução. A Figura 9 apresenta alguns trechos do log associado à execução do programa que gerou e imprimiu o data set *Funcionarios*, apresentado na Seção 2.4. Três diferentes seções deste log foram marcadas com os identificadores (i), (ii) e (iii).

```

(i) The SAS System                                15:45 Tuesday, May 9, 2017
...
14      GOPTIONS ACCESSIBLE;
15      /* Programa 2: cria um data set e produz relatório a partir do mesmo */
16
17      * Passo Data - cria o data set "Funcionarios";
18      DATA Funcionarios;
19
20      INPUT matricula $ nome $ idade sexo $;
21
22      DATALINES;

(ii)
NOTE: The data set WORK.FUNCIONARIOS has 5 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds

28      ;

29      RUN;
30
31      * Passo PROC - gera relatório a partir do data set "Funcionarios";
32      PROC PRINT DATA=Funcionarios;
33      RUN;

(iii)
NOTE: There were 5 observations read from the data set WORK.FUNCIONARIOS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

...

45      QUIT; RUN;
46

```

Figura 9. Trechos do log de execução de um programa SAS

Veja que o log inicia em (i) com informações sobre a data e hora de execução do programa. A seguir, em (ii) são apresentadas informações a respeito do passo DATA que gerou o data set *Funcionarios*: número de observações, variáveis e tempo de execução. A mesma coisa é feita com relação à execução da PROC PRINT, no trecho rotulado como (iii).

O comando PUT pode ser utilizado para escrever mensagens no log. Este é um recurso útil no processo de depuração de programas. Para verificar esse recurso, digite e execute o Programa 4. Em seguida, examine o log de execução do mesmo.

```

/* Programa 4: escrevendo mensagens no LOG */
DATA Ola;
  PUT 'Essa Mensagem vai para o Log!!!';
  mensagem = 'Hello SAS!';
RUN;

```

Em resumo: qualquer erro, aviso ou informação relevante sobre a execução de um programa é colocada de forma automática pelo SAS no log. E você também pode escrever as suas mensagens nele, se assim desejar.

3. Trabalhando com Bases de Dados no Formato Texto

Este capítulo apresenta as maneiras pelas quais bases de dados estruturadas em diferentes formatos de arquivos texto podem ser acessadas e importadas para data sets SAS. O capítulo está dividido da seguinte forma. Inicialmente, na Seção 3.1, são introduzidos os comandos básicos para a leitura de arquivos (INFILE e INPUT). A seguir, a Seção 3.2 descreve a técnica de laço implícito empregada pelo passo DATA para automatizar o processo de varredura das informações de arquivos. As Seções 3.3 a 3.6 abordam as diversas opções que podem ser utilizadas no comando INPUT para permitir a importação de diferentes formatos de arquivo texto. Em seguida, a Seção 3.7 trata da gravação de arquivos (transferência do conteúdo de um data set para um arquivo em disco). A Seção 3.8 introduz comando global FILEREF, útil tanto para a leitura como para a gravação de arquivos. Encerrando o capítulo, as Seções 3.9 a 3.11 cobrem as máscaras de leitura (informats) utilizadas para o tratamento de datas e valores numéricos especiais (valores numéricos que contêm caracteres não-numéricos).

3.1 INFILE e INPUT: Os Dois Comandos para Leitura de Arquivos

Os comandos INFILE e INPUT são utilizados no passo DATA para possibilitar a leitura de arquivos. A receita básica para utilizar estes comandos é bastante simples: o arquivo a ser lido deve ser indicado na instrução INFILE, enquanto as variáveis que serão obtidas a partir do mesmo deverão ser declaradas com o uso do INPUT. A seguir será apresentado um exemplo de utilização destes comandos na leitura de um **arquivo texto sequencial separado por colunas**. O exemplo utiliza o arquivo “ARQ_COLUNAS.txt” apresentado abaixo. Considere que este arquivo armazena informações de duas variáveis: “n1”, uma variável numérica, que inicia na coluna 1 e possui tamanho fixo de 4 colunas; e “c1”, uma variável do tipo caractere, iniciando na coluna 5 e com tamanho fixo de 5 colunas.

```
1001aaaaa
1002bbbbb
1003cccccc
1004dddddd
1005eeeeee
```

Suponha que este arquivo esteja armazenado na pasta “C:\CursoSAS” (utilizaremos essa suposição para todos os exemplos de agora em diante – modifique para a pasta ou compartilhamento de rede que você estiver utilizando em sua empresa). O Programa 5 apresenta o código necessário para importar este arquivo, gerando um data set SAS com o conteúdo do mesmo. O conteúdo do data set gerado é apresentado logo após o código do programa (em geral, os demais exemplos apresentados neste livro farão uso desta mesma convenção: primeiro apresentar a listagem do programa SAS e depois o data set gerado).

```
* Programa 5: Importação de um arquivo separado por colunas;
DATA Colunas;
  INFILE 'C:\CursoSAS\ARQ_COLUNAS.txt';
  INPUT n1 1-4 c1 $ 5-9;
RUN;
```

<i>Colunas</i>	
n1	c1
1001	aaaaa
1002	bbbbb
1003	ccccc
1004	ddddd
1005	eeeee

A primeira instrução do programa é:

```
DATA Colunas;
```

Ela serve para dizer ao SAS: “crie um data set chamado *Colunas* na library WORK”. Na instrução abaixo, o comando INFILE especifica que o data set será gerado a partir de informações obtidas no arquivo “C:\CursoSAS\ARQ_COLUNAS.txt”. Esta é a forma mais simples de usar o comando: escrever INFILE e depois o caminho e nome do arquivo a ser acessado. A instrução aceita o uso de alguns parâmetros que serão apresentados ao longo deste capítulo.

Na linha 4, o comando INPUT é usado para determinar as variáveis que farão parte do data set *Colunas* e indicar de que maneira elas podem ser obtidas no arquivo texto de origem,. Existem várias maneiras de utilizar a instrução INPUT. No Programa 5, a seguinte sintaxe foi utilizada:

```
VAR1 col_ini-col_fim ... VARn col_ini-col_fim
```

Todas as variáveis são especificadas em uma única linha. Para cada variável, basta indicar um nome (que representará o nome da variável no data set) e a sua coluna inicial e final no arquivo de entrada, ligadas por um traço (ex: 5-9). Variáveis do tipo caractere precisam ser especificadas com uso do símbolo de cifrão “ \$ ” (ex: c1 \$ 5-9). O SAS assume que variáveis indicadas sem o cifrão representam variáveis do tipo numérico (ex: n1 1-4). Espaços em branco devem ser utilizados para separar as informações.

3.2 O Laço Implícito do Passo DATA (Built-in Loop)

Em linguagens de programação tradicionais, para que seja possível abrir um arquivo e depois percorrer todas as suas linhas, normalmente é necessário implementar um bloco de código conhecido como laço, utilizando comandos de repetição como WHILE, FOR e REPEAT. Entretanto, ao analisar o programa da seção anterior, você deve ter notado que não foi preciso criar nenhum tipo de laço dentro do passo DATA para percorrer as observações do arquivo texto uma por uma. Isto ocorre porque **o passo DATA funciona baseado em um laço implícito (*built-in loop*)**, ou seja, um laço que é executado de **forma automática**. Esta é uma característica bastante inusitada, porém extremamente interessante da linguagem SAS, pois ela contribui decisivamente para que o código de um programa

SAS seja mais simples e curto do que o código produzido em outras linguagens. Você deve ter observado que o Programa 5, apresentado na seção anterior, é realmente muito pequeno.

Uma regra importantíssima associada ao laço implícito é a seguinte: **todos os comandos de um passo DATA são executados para cada observação que for lida do arquivo de entrada.** Para que o conceito fique mais claro, observe o esquema apresentado na Figura 10, encontrado em diversos livros sobre programação SAS. O esquema mostra como uma observação lida de um arquivo de entrada é tratada dentro de um passo DATA. O SAS lê a primeira observação do arquivo e a processa usando o primeiro comando do passo DATA. Depois processa a mesma observação usando o segundo comando, o terceiro, até processar contra o último comando (comando imediatamente anterior ao comando RUN). Nesse momento, a observação é copiada para o data set de saída (data set gerado pelo passo DATA). Uma vez que o SAS termine de processar a primeira observação, o controle do programa é retornado automaticamente ao início do passo DATA, para que a segunda observação seja processada da mesma forma (por isso, o nome “laço implícito”). Quando a última observação é processada, o laço termina de forma automática e o controle do programa passa para o passo seguinte, que pode ser outro passo DATA ou um passo PROC.

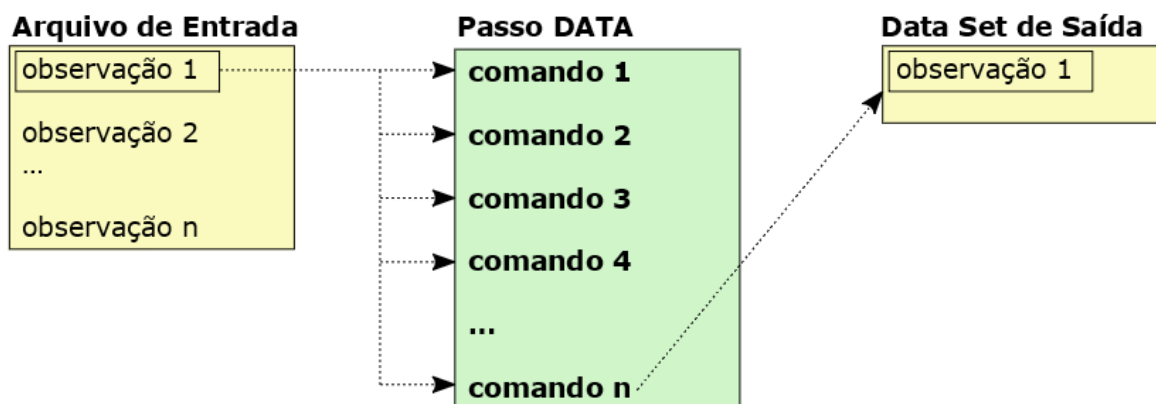


Figura 10. O laço implícito do passo DATA

3.3 Leitura de Arquivos CSV

Esta seção apresenta a forma básica para a leitura de arquivos CSV (*comma-separated values* - valores separados por vírgula) e de outros tipos de arquivos baseados em caracteres delimitadores. O arquivo “CEPS.csv” é um arquivo deste tipo. Ele armazena informações sobre faixas de CEPs utilizadas em estados da região Sudeste. Neste arquivo, o primeiro valor corresponde ao CEP inicial, o segundo valor representa o CEP final, e o último armazena o nome do estado. Observe que a primeira linha do arquivo contém o cabeçalho, ou seja, a descrição das variáveis.

```
cep_ini, cep_fim, nome_uf
20000000, 28999999, Rio de Janeiro
29000000, 29999999, Espírito Santo
30000000, 39999999, Minas Gerais
01000000, 19999999, São Paulo
```

O Programa 6a apresenta a forma básica para realizar leitura deste arquivo CSV. Veja que, no programa, três parâmetros são acrescentados no final do comando INFILE:

- **DLM=','**: indica o caractere utilizado como delimitador. Neste caso, é a vírgula.
- **DSD**: é colocado no INFILE para que o SAS possa tratar valores ausentes (ex: duas vírgulas consecutivas em um arquivo CSV).
- **FIRSTOBS=2**: informa que a primeira observação (primeira linha de dados) é a linha 2. Este parâmetro é necessário sempre que o arquivo CSV contiver uma primeira linha de cabeçalho, como é o caso do arquivo “CEPS.csv”.

```
/* Programa 6a: importação de um arquivo CSV (forma básica) */
DATA CEPS1;
  INFILE 'C:\CursoSAS\CEPS.CSV' DLM=',' DSD FIRSTOBS=2;
  INPUT cep_inicial $ cep_final $ uf $;
RUN;
```

CEPS1

cep_inicial	cep_final	uf
20000000	28999999	Rio de J
29000000	29999999	Espírito
30000000	39999999	Minas Ge
01000000	19999999	São Paul

Apesar de o Programa 6a rodar, ele não faz um bom trabalho, uma vez que o data set gerado fica com um problema na variável “uf”: apenas as 8 primeiras letras de cada nome foram gravadas em cada observação. Não se assuste com isso, pois a solução é simples e o problema ocorre somente porque o SAS assume por padrão o tamanho máximo de 8 posições ao trabalhar com variáveis do tipo caractere. Para que a situação possa ser resolvida, basta utilizar o comando LENGTH para declarar explicitamente o tamanho da variável, como mostra o Programa 6b.

```
/* Programa 6b: corrige o problema dos nomes truncados */
DATA CEPS2;
  LENGTH uf $ 15;
  INFILE 'C:\CursoSAS\CEPS.CSV' DLM=',' DSD FIRSTOBS=2;
  INPUT cep_inicial $ cep_final $ uf $;
RUN;
```

CEPS2

uf	cep_inicial	cep_final
Rio de Janeiro	20000000	28999999
Espírito Santo	29000000	29999999
Minas Gerais	30000000	39999999
São Paulo	01000000	19999999

Problema corrigido, a variável “uf” agora está com o conteúdo correto. No entanto, observe que a mesma passou a ser a **primeira** do data set. Isto ocorre porque no Programa 6b, a declaração da variável “uf” (linha 3) é feita antes da declaração das variáveis “cep_inicial” e “cep_final” (ambas são declaradas, ou seja, “mencionadas” pela primeira vez, no INPUT, linha 5). Se for necessário posicionar “uf” como a última variável do data set, para que o mesmo padrão do arquivo CSV seja respeitado, a solução é declarar “cep_inicial” e “cep_final” antes de “uf”.

```
/* Programa 6c: corrige o problema da ordem das variáveis */
DATA CEP3;
  LENGTH cep_inicial $ 8;
  LENGTH cep_final $ 8;
  LENGTH uf $ 15;
  INFILE 'C:\CursoSAS\CEPS.CSV' DLM=',' DSD FIRSTOBS=2;
  INPUT cep_inicial $ cep_final $ uf $;
RUN;
```

CEPS3

cep_inicial	cep_final	uf
20000000	28999999	Rio de Janeiro
29000000	29999999	Espírito Santo
30000000	39999999	Minas Gerais
01000000	19999999	São Paulo

3.4 Outras Opções do Comando INFILE

A seguir são apresentadas outras opções disponíveis para o comando INFILE:

3.4.1 Leitura de Arquivos Delimitados por TAB

Neste caso, usa-se a opção DLM = ‘09’x (especificação do valor hexadecimal do código ASCII para o caractere TAB):

```
INFILE 'C:\CursoSAS\ARQ_TAB.txt' DLM='09'x DSD;
```

3.4.2 Leitura de Arquivos Delimitados por Espaço

Para este tipo de arquivo, não é preciso especificar nenhum parâmetro no INFILE

```
INFILE 'C:\CursoSAS\ARQ_ESP.txt';
```

3.4.3 Opção LRECL

Nos ambientes Linux e Windows, o SAS assume que as linhas de um arquivo têm um comprimento máximo de 256 caracteres. Caso seja preciso abrir um arquivo de entrada com comprimento maior, é preciso usar a opção LRECL do comando INFILE. O exemplo a

seguir mostra a forma de utilizar a opção para ler o arquivo “COMPRIDO.txt”, da Figura 13, que possui comprimento de 260 caracteres.

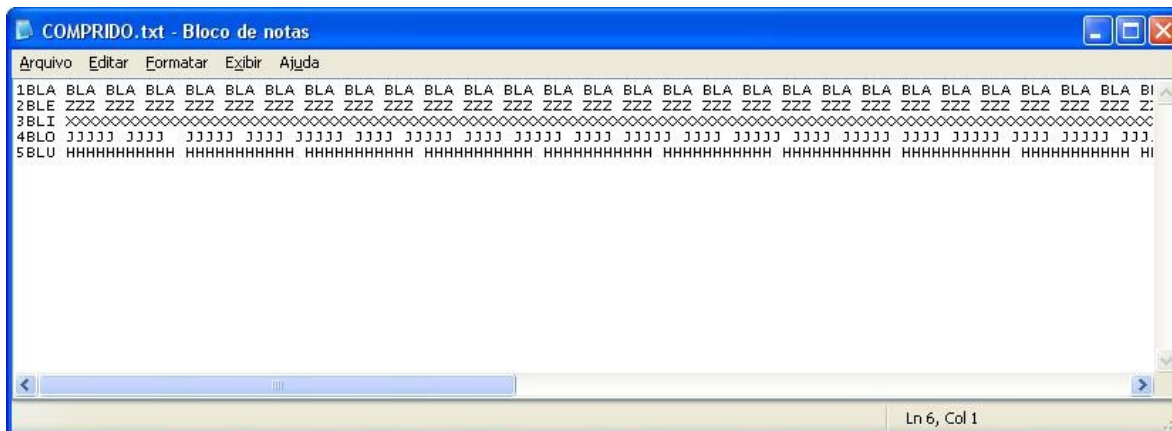


Figura 13. Arquivo separado por colunas com comprimento de 260 caracteres

```
/* Programa 7: leitura de um arquivo comprido */
DATA Comprido;
  INFILE 'C:\CursoSAS\COMPRIDO.txt' LRECL=260;
  INPUT codigo 1-1 descricao $ 2-260;
RUN;
```

IMPORTANTE!

Nos ambientes Windows e Linux, a opção LRECL precisa ser usada para garantir a correta leitura de arquivos com observações de comprimento superior a 256 caracteres. No Programa 7, a definição LRECL=260 se deu em função do fato de o arquivo de entrada possuir 260 colunas.

3.4.4 Opção OBS=*k*

Esta opção deve ser utilizada sempre que você desejar ler apenas uma parte do arquivo de entrada, mais precisamente as *k* primeiras linhas.

- **Exemplo 1:** apenas as 5 primeiras linhas de “arq.txt” são lidas.

```
o INFILE 'arq.txt' OBS=5;
```

- **Exemplo 2:** as opções FIRSTOBS e OBS são combinadas, fazendo com que o SAS inicie a leitura do arquivo a partir da terceira linha e pare após a leitura da sétima linha.

```
o INFILE 'arq.txt' FIRSTOBS=3 OBS=7;
```


3.4.6 Opção TRUNCOVER

A opção TRUNCOVER do comando INFILE é útil quando se deseja realizar a leitura de um arquivo que termina com uma variável cujo comprimento **não** é fixo. Considere, por exemplo, o arquivo “PRODUTOS.txt”, apresentado abaixo, que contém o código e o nome de diversos produtos. Observe que os nomes dos produtos possuem comprimento diferente.

```
1001Leite
1002Biscoito
1003Café
1004Torradas
1005Chá
```

Para que o arquivo seja lido corretamente, é preciso acrescentar a opção TRUNCOVER ao comando INPUT, conforme mostra o programa 8a.

```
/* Programa 8a: Importação de um arquivo separado por colunas
               sem comprimento fixo */
DATA Produtos;
  INFILE 'C:\CursoSAS\PRODUTOS.txt' TRUNCOVER;
  INPUT INPUT codigo 1-4 nome $ 5-15;
RUN;
```

Produtos

codigo	nome
1001	Leite
1002	Biscoito
1003	Café
1004	Torradas
1005	Chá

Além de TRUNCOVER, a opção PAD também pode ser utilizada em situações deste tipo:

```
/* Programa 8b: Importação de um arquivo separado por colunas
               sem comprimento fixo */
DATA Produtos;
  INFILE 'C:\CursoSAS\PRODUTOS.txt' PAD;
  INPUT INPUT codigo 1-4 nome $ 5-15;
RUN;
```

3.4.5 Opção MISSOVER

Durante a leitura de um arquivo, caso o final de uma linha seja alcançado e nem todas as variáveis declaradas no INPUT tenham sido preenchidas, o SAS acabará completando estas variáveis com dados da linha seguinte. Melhor explicando: o conteúdo

de duas linhas do arquivo de entrada será misturado em uma única observação do data set de saída.

Considere o arquivo “PAGAMENTOS.txt”, que contém as parcelas pagas por quatro diferentes clientes que compraram um produto no valor de R\$ 300,00. Neste exemplo, as clientes Amanda e Ines optaram por parcelar a compra em duas parcelas iguais, enquanto Ramon pagou à vista e Nelson preferiu pagar em três parcelas (R\$ 150,00 na primeira parcela e R\$ 75,00 nas duas seguintes).

```
Amanda 150 150
Ramon   300
Nelson  150 75 75
Ines    150 150
```

Ao importar uma base de dados nesse formato, é preciso utilizar a opção **MISSOVER** para atribuir o valor missing para a terceira parcela de Amanda e Ines e para as duas parcelas finais de Ramon.

```
/* Programa 9: leitura de um arquivo com a opção MISSOVER */
DATA Parcelas;
  INFILE 'C:\CursoSAS\PAGAMENTOS.txt' MISSOVER;
  INPUT Nome $ P1 P2 P3;
RUN;
```

Parcelas

Nome	P1	P2	P3
Amanda	150	150	.
Ramon	300	.	.
Nelson	150	75	75
Ines	150	150	.

3.4.6 Opções **FLOWOVER** e **STOPOVER**

As opções **FLOWOVER** e **STOPOVER** também podem ser utilizadas no **INPUT**.

- **FLOWOVER**: corresponde simplesmente ao comportamento padrão do comando **INPUT**, onde o passo **DATA** lê a próxima observação e tenta associar valores a todas as variáveis. Sendo assim, **FLOWOVER** pode ser omitido.
- **STOPOVER**: faz com que o processamento do passo **DATA** seja interrompido em situações onde o final de uma observação é atingido sem que informações para todas as variáveis declaradas tenham sido encontradas.

3.5 A Variável Automática **_INFILE_**

Sempre que dentro de um passo **DATA** for desejado ler uma linha inteira de um arquivo sem quebrá-la em diversas variáveis, utiliza-se a dupla “comando **INPUT**” +

“variável automática _INFILE_”. Considere o arquivo “POEMA.txt”, que armazena um poema de Fernando Pessoa⁴:

```
Tudo Quanto Penso
=====
Tudo quanto penso,
Tudo quanto sou
É um deserto imenso
Onde nem eu estou
Extensão parada
Sem nada a estar ali,
Areia peneirada
Vou dar-lhe a ferroada
Da vida que vivi
```

Fernando Pessoa

O Programa 10 apresenta o código que lê todas as linhas do arquivo “POEMA.TXT” e armazena cada uma delas como uma observação em um data set chamado *Poema*.

```
/* Programa 10: uso da variável _INFILE_ */
DATA Poema;
  LENGTH linha $ 30;
  INFILE 'C:\CursoSAS\POEMA.txt';
  INPUT;
  linha = _INFILE_;
RUN;
```

Poema

linha
Tudo Quanto Penso
=====
Tudo quanto penso,
Tudo quanto sou
É um deserto imenso
Onde nem eu estou
Extensão parada
Sem nada a estar ali,
Areia peneirada
Vou dar-lhe a ferroada
Da vida que vivi
Fernando Pessoa

⁴ Disponível no Portal Domínio Público - <http://www.dominiopublico.gov.br/>

Como funciona o programa? Dentro do passo DATA, o uso de um comando INPUT “puro”, faz com que a linha inteira do arquivo seja copiada para o *buffer* (área de memória de trabalho do programa SAS). O conteúdo desse buffer é referenciado através da variável `_INFILE_`, que é criada de forma automática pelo SAS.

Alguns comentários úteis:

- É importante não confundir o **comando INFILE** (comando para abertura de arquivo) com a **variável automática _INFILE_** (ponteiro para o *buffer* que armazena a linha correntemente processada).
- O comprimento máximo da variável `_INFILE_` é igual ao que for especificado na opção LRECL do comando INFILE. O valor default é 256 (quando LRECL não é especificado) e o valor máximo é 32767 bytes.
- Curiosamente, dentro do passo DATA, a variável automática `_INFILE_` se comporta como se tivesse comprimento variável, equivalente ao comprimento da linha que foi lida (não é necessário usar nenhuma função para remover brancos, a não ser que estes estejam explicitamente presentes no arquivo – o que não era o caso do exemplo do poema).

3.6 Utilizando o Ponteiro de Colunas (@)

Uma forma muito utilizada para a leitura de arquivos separados por colunas é através da utilização do ponteiro de colunas “ @ ” (*single trailing*). Este formato é especialmente indicado nas seguintes situações:

- Quando o número de variáveis a ser lido é muito grande.
- Quando um arquivo possui muitas variáveis, mas apenas algumas delas precisam ser lidas ou deseja-se ler as variáveis em uma ordem diferente da especificada no arquivo.

Considere o arquivo “CARROS.txt”. Neste arquivo, cada observação contém os dados de um carro. As a placa do carro está armazenada nas colunas 1-7 (largura igual a 7), o modelo nas colunas 8-13 (largura 6), o ano nas colunas 14-17 (largura 4) e o preço nas colunas 19- 26 (largura 8).

```
LAB2233Gol      2007   6500.55
KBB9999Palio    2002   4999.99
LCC1234Uno      2015  16000.00
```

O Programa 11a mostra como o arquivo pode ser lido com a utilização da sintaxe baseada em ponteiro de colunas:

```
/* Programa 11a: leitura de arquivo sequencial utilizando o
ponteiro de colunas @ */
DATA Carros;
  INFILE 'C:\CursoSAS\CARROS.txt';
  INPUT
    @1 placa    $ 7.
    @8 modelo   $ 6.
    @14 ano      4.
    @19 preco    8.;
  RUN;
```

Carros

placa	modelo	ano	preco
LAB2233	Gol	2007	6500.55
KBB9999	Palio	2002	4999.99
LCC1234	Uno	2015	16000

Nas linhas 5-9, o comando INPUT emprega o recurso de ponteiro de colunas para determinar as variáveis que farão parte do data set *Carros* e indicar de que maneira elas podem ser obtidas no arquivo texto de origem,. A seguinte sintaxe foi utilizada:

```
@coluna_inicial1  NOME_DA_VARIAVEL1  tipo_e_tamanho1.  
@coluna_inicial2  NOME_DA_VARIAVEL2  tipo_e_tamanho2.  
...  
@coluna_inicialn  NOME_DA_VARIAVELn  tipo_e_tamanhon.  
;
```

Cada variável é especificada em uma linha, bastando indicar a coluna inicial, um nome (que representará o nome da variável no data set) e o tipo e tamanho da variável. Variáveis do tipo caractere precisam ser especificadas com uso da máscara \$. Um ponto e vírgula deve ser utilizado após a declaração da última variável, para indicar o fim do comando INPUT.

A principal vantagem do ponteiro de colunas é que ele oferece flexibilidade para a leitura de arquivos: as variáveis podem ser lidas em qualquer ordem e não existe obrigatoriedade em ler todas as variáveis. O Programa 11b mostra um exemplo onde apenas as variáveis “preço” e “placa” são lidas, nesta ordem, do arquivo de entrada:

```
/* Programa 11b: leitura de apenas algumas variáveis, em ordem  
diferente da estabelecida no arquivo de entrada */  
DATA Carros;  
  INFILE 'C:\CursoSAS\CARROS.txt';  
  INPUT  
    @19 preco    8.  
    @1 placa    $ 7.;  
RUN;
```

Carros

preco	placa
6500.55	LAB2233
4999.99	KBB9999
16000	LCC1234

3.7 FILE e PUT: Os Comandos para Gravação de Arquivos Texto

Os programas anteriormente apresentados envolveram sempre a mesmo tipo de operação: utilizar o passo DATA para importar um ou mais arquivos texto, gerando um data set a partir do conteúdo destes arquivos. Entretanto também é possível utilizar o passo DATA para realizar a operação inversa, ou seja, gravar arquivos texto a partir do conteúdo de um data set. Neste tipo de operação, os dois seguintes comandos devem ser utilizados:

- FILE: para especificar o caminho do arquivo de saída; e
- PUT: para especificar as variáveis que farão parte do arquivo.

```
* Programa 12a: Gravação de arquivo texto;

* Cria o data set "Funcionarios";
DATA Funcionarios;
  INPUT matricula $ nome $ idade sexo $;
  DATALINES;
M01 George 58 M
M02 Jane 32 M
M03 Aldous 40 F
M04 Thomas 28 M
M05 Mary 43 F
  ;
RUN;

* Grava o conteúdo de "Funcionarios" no arquivo SAIDA.txt;
DATA _NULL_;
  SET Funcionarios;
  FILE 'C:\CursoSAS\SAIDA.txt';
  PUT
  @001 matricula $CHAR3.
      nome $CHAR6.
      idade 2.
      sexo $CHAR1.
  ;
RUN;
```

No Programa 12a, o segundo passo DATA (últimas 10 linhas de código) transfere o conteúdo do data set *Funcionarios* (gerado nas linhas 10-13) para o arquivo texto 'SAIDA.txt'. O arquivo gerado, possui o seguinte conteúdo:

```
M01George58M
M02Jane 32M
M03Aldous40F
M04Thomas28M
M05Mary 43F
```

Retornando para o programa, note que na linha 16 a palavra `_NULL_` aparece ao invés do nome de um data set. Este é um recurso disponibilizado pelo SAS para que o programador possa **executar um passo DATA sem gerar um data set de saída na WORK**. Com isto, é possível economizar tempo de processamento. O comando SET simplesmente solicita com o que o SAS leia o data set *Funcionarios* (este comando será

apresentado em detalhes no Capítulo 6). A seguir, o comando FILE instrui o SAS a criar um arquivo para saída. O comando PUT informa o que será gravado e aonde.

ATENÇÃO!

Não se esqueça que o comando PUT também é utilizado para escrever mensagens no log.

3.7.1 Persistindo um Data Set

Em situações onde você desejar persistir (salvar em disco) um data set em uma pasta de seu computador ou uma pasta de rede – ao invés de exportar seu conteúdo para um arquivo texto – será necessário utilizar a seguinte receita:

1. Associar a pasta destino a uma library SAS (ver Seções 2.5 e 2.6).
2. No passo DATA, prefixar o nome do data set como o nome da library.

O Programa 12b mostra como criar o data set *Funcionarios* diretamente na pasta “C:\CursoSAS” ao invés de criá-lo na WORK:

```
* Programa 12b: Persistindo data set "Funcionarios";

* define a library onde o data set será gravado;
libname my_lib 'C:\CursoSAS';

* Cria o data set "Funcionarios",
persistindo-o na library definida;

DATA my_lib.Funcionarios;
  INPUT matricula $ nome $ idade sexo $;
  DATALINES;
M01 George 58 M
M02 Jane 32 M
M03 Aldous 40 F
M04 Thomas 28 M
M05 Mary 43 F
;
RUN;
```

3.8 FILENAME

O comando FILENAME é utilizado para permitir a associação de uma *fileref* com um arquivo físico. A *fileref* nada mais é do que um nome lógico, que pode ser utilizado em qualquer ponto do seu programa para referenciar o arquivo. Veja o exemplo abaixo, que realiza a leitura do arquivo “POEMA.txt” (apresentado na Seção 3.5).

```

/* Programa 13a: uso do comando FILENAME */
FILENAME f 'C:\CursoSAS\POEMA.txt';

DATA Poema;
  LENGTH linha $30;
  INFILE f;
  INPUT;
  linha = _INFILE_;
RUN;

```

O comando FILENAME é um comando do tipo global. Uma vez feita a associação entre o arquivo físico (“C:\CursoSAS\POEMA.txt”) com a fileref (que neste exemplo foi chamada de “f”), esta fileref pode ser utilizada em qualquer passo DATA subsequente. Pode-se inserir a fileref em qualquer ponto de um programa e ela estará disponível em qualquer passo DATA especificado depois de sua definição. Normalmente especificamos a fileref lado de fora do passo DATA, mas também funciona se for feito dentro.

3.8.1 Referenciando Múltiplos Arquivos

Uma mesma fileref pode referenciar mais de um arquivo. Neste caso, durante o processamento do passo DATA, o SAS irá abrir um arquivo após o outro e tratará todos eles como se fossem um só. Esse recurso é bem interessante quando precisa-se trabalhar vários arquivos que possuem o mesmo formato, como é o caso dos três arquivos CSV a seguir:

Arquivo “UFS_NORDESTE.txt”:

```

AL,Alagoas
BA,Bahia
CE,Ceará
MA,Maranhão
PB,Paraíba
PI,Piauí
PE,Pernambuco
RN,Rio Grande do Norte
SE,Sergipe

```

Arquivo “UFS_SUDESTE.txt”:

```

ES,Espírito Santo
MG,Minas Gerais
RJ,Rio de Janeiro
SP,São Paulo

```

Arquivo “UFS_SUL.txt”:

PR,Paraná
RS,Rio Grande do Sul
SC,Santa Catarina

O primeiro arquivo mantém as siglas e nomes das UFs da região Nordeste. O segundo e terceiro contêm as mesmas informações para as regiões Sudeste e Sul, respectivamente. Observe que desta vez os três arquivos CSV não possuem cabeçalho, diferentemente do arquivo apresentado na Seção 3.3. Imagine que fosse preciso fazer um programa onde as informações dos 3 arquivos devam ser submetidas a um mesmo processamento (ex: mesma rotina de críticas). Esta tarefa é trivial no SAS, conforme mostrado no Programa 13b.

```
/* Programa 13b: referenciando múltiplos arquivos */

FILENAME entrada
("C:\CursoSAS\UFS_NORDESTE.csv",
"C:\CursoSAS\UFS_SUDESTE.csv",
"C:\CursoSAS\UFS_SUL.csv");

DATA Ufs;
  LENGTH sigla_uf $2.;
  LENGTH nome_uf $19.;
  INFILE entrada DSD DLM=', ';
  INPUT sigla_uf $ nome_uf $;
RUN;
```

<i>Ufs</i>	
sigla_uf	nome_uf
AL	Alagoas
BA	Bahia
CE	Ceará
MA	Maranhão
PB	Paraíba
PI	Piauí
PE	Pernambuco
RN	Rio Grande do Norte
SE	Sergipe
ES	Espírito Santo
MG	Minas Gerais
RJ	Rio de Janeiro
SP	São Paulo
PR	Paraná
RS	Rio Grande do Sul
SC	Santa Catarina

Segue a explicação. O Programa 13b cria um fileref chamado “entrada”, que aponta para três arquivos físicos: “UFS_NORDESTE.csv”, “UFS_SUDESTE.csv”, “UFS_SUL.csv”. Dentro do programa SAS, os três arquivos são enxergados como um único. É como se o SAS criasse um “arquivo virtual”, contendo as observações dos três arquivos concatenadas (Obs.: as observações são “virtualmente concatenadas” na ordem em que os arquivos são especificados no comando FILENAME).

Para finalizar esta seção, será apresentada uma aplicação prática muito interessante para o conceito recém-apresentado: um programa que cria um único arquivo de saída a partir da concatenação dos três arquivos de entrada. Os comentários no código explicam o funcionamento do programa.

```
/* Programa 13c: gera um único arquivo a partir de 3 arquivos */

*-----;
* -- PASSO 1: especifica quem são os arquivos de entrada ;
*     e quem será o arquivo de saída ;
*-----;

FILENAME entrada
("C:\CursoSAS\UFS_NORDESTE.csv",
"C:\CursoSAS\UFS_SUDESTE.csv",
"C:\CursoSAS\UFS_SUL.csv");

FILENAME saida "C:\CursoSAS\UFS_TRES_REGIOES.txt";

*-----;
* -- PASSO 2: gera arquivo de saída ;
*-----;

DATA _NULL_;

    * -- leitura;
    INFILE entrada;
    INPUT; *lê uma linha e armazena no buffer (_INFILE_);

    * -- escrita;
    FILE saida;
    PUT _INFILE_; *salva a linha no arquivo de saída;
RUN;
```

3.9 Leitura de Múltiplas Observações por Linha (@@)

No processo de leitura de arquivos, o SAS assume por padrão que cada linha do arquivo de entrada corresponde a uma observação. Caso você precise lidar com uma base que possua múltiplas observações representadas em uma única linha (uma forma de representação mais rara, porém muito conveniente em algumas aplicações práticas), terá que fazer uso do especificador “ @@ ” (*double trailing*) no comando INPUT. Para demonstrar a utilização deste especificador, considere o arquivo “JOGOS.txt”, que

armazena informações sobre um total de 20 jogos realizados por um time de futebol no último campeonato nacional. Para cada jogo, registra-se o local da partida ('F'=fora de casa ou 'C'=em casa) e o número de gols marcados na partida. Observe que cada linha armazena informações sobre 5 jogos.

```
F 1 C 3 F 0 C 0 C 5
F 2 C 1 F 0 F 0 C 2
C 4 F 0 F 3 C 1 F 1
C 2 F 1 C 2 F 1 C 3
```

O programa a seguir mostra que a importação desse arquivo é feita de forma trivial com o uso do recurso “@@” no INPUT, resultando em um data set com 20 observações (uma para cada jogo).

```
*Programa 14: leitura de arquivo com várias observações por linha;
DATA Jogos;
INFILE 'C:\CursoSAS\JOGOS.txt';
    INPUT local $ gols @@;
RUN;
```

<i>Jogos</i>	
local	gols
F	1
C	3
F	0
C	0
C	5
F	2
C	1
F	0
F	0
C	2
C	4
F	0
F	3
C	1
F	1
C	2
F	1
C	2
F	1
C	3

3.10 Trabalhando com Dados Numéricos Especiais (Nonstandard)

A linguagem SAS define duas categorias para a representação de dados em arquivos externos: representação padrão (*standard*) ou representação especial (*nonstandard*).

Dados representados no formato padrão podem ser lidos diretamente pelo comando INPUT. Qualquer número inteiro ou sequência de caracteres é considerada uma representação padrão. Um número com ponto decimal (número real) ou prefixado por um sinal de menos (número negativo) também é considerado como representado na forma padrão.

Por sua vez, dados em representação especial são basicamente valores numéricos que contêm caracteres não-numéricos. Alguns exemplos:

- Números prefixados pelo sinal de cifrão (\$). Ex.: \$900
- Números contendo o separador de milhar (em inglês, o separador é a vírgula “ , ”). Ex.: 1,000,000.00 (um milhão).
- Datas (dia, hora, minuto, segundo). Ex.: 25/12/2017.

Abaixo apresenta-se o arquivo CSV “SALARIO.txt”, que armazena o nome e o salário de um grupo de pessoas. Este arquivo possui duas variáveis, sendo uma representada na forma padrão (“nome”) e uma com representação especial (“salário”).

```
Daniel,$5000.50  
Carlos,$3500.99  
Denis,$6500.00
```

Para realizar a leitura de dados especiais no SAS, são utilizados os *informats*. Um informat pode ser entendido como uma espécie de “máscara” que é especificada ao lado de uma variável definida no comando INPUT, indicando o formato desta variável no arquivo de entrada. O Programa 15 ilustra a utilização do informat “COMMA8.” para possibilitar a correta leitura da variável “salário”. Neste informat, a palavra “COMMA” indica ao SAS que a variável é prefixada por \$, enquanto o valor “8” indica que, após o \$, existe um número com um total de no máximo 8 dígitos, incluindo as casas decimais.

```
*Programa 15: leitura de arquivo com dados nonstandard;  
DATA Salarios;  
  INFILE 'C:\CursoSAS\SALARIOS.csv' DSD DLM=' , ' ;  
  INPUT nome $ salario COMMA8. ;  
RUN;
```

Salarios

nome	salario
Daniel	5000.5
Carlos	3500.99
Denis	6500

A seguir são relacionados e comentados alguns exemplos de informat. Em todos os exemplos, *w* indica o máximo de dígitos.

- **COMMAw.**: pode ser utilizado não apenas para a leitura de números prefixados por “\$”, mas também para números contendo vírgulas como separador de milhar, tais como, 1,000,000.
- **PERCENTw.**: converte porcentagens (Ex.: 25%).
- **DDMMYYw.**: para a leitura de datas no formato “dia/mês/ano”. Por exemplo: DDMMYY10 permite a leitura de datas com 10 dígitos, como 25/12/2017. As próximas duas seções apresentam detalhes sobre a importação de dados contendo data e hora de eventos.
- **PDw.**: leitura de números representados no formato decimal compactado (comum em ambiente mainframe).
- **ZDw.**: leitura de números representados no formato decimal zonado (também comum em ambiente mainframe).

3.11 Trabalhando com Valores do Tipo Data (DATE)

No sistema SAS, uma variável do tipo DATE serve para armazenar datas (dia, mês e ano). Internamente, a variável DATE corresponde a uma variável numérica que contém o valor inteiro equivalente ao **número de dias passados desde 01/01/1960**. A faixa de valores que podem ser manipulados vai de 01/01/1582 (valor inteiro -138.061) até 31/12/20000 (valor inteiro 6.589.335). O valor 0 (zero) corresponde a 01/01/1960.

O SAS possui três recursos principais para a manipulação de datas: (i) informat especiais para a leitura de datas; (ii) funções para operações com datas (exemplo: subtração de datas, obtenção do dia da semana, etc) e (iii) formatos especiais para a impressão de datas.

O programa apresentado nesta seção exemplifica a maneira pela qual um valor pode ser lido de um arquivo texto e interpretado como uma data pelo SAS. Ele faz uso do arquivo “TEMPERATURAS.txt”, apresentado a seguir, que contém cinco datas e a temperatura máxima registrada em cada uma dessas datas.

```
01/04/2017 12
02/04/2017 28
03/04/2017 25
04/04/2017 32
05/04/2017 39
```

```
*Programa 16a: leitura de datas;
DATA Temperaturas;
  INFILE 'C:\CursoSAS\TEMPERATURAS.txt';
  INPUT
    @001 dia DDMMYY10.
    @012 graus_c 2.;
RUN;
```

Temperaturas

dia	graus
17623	12
17624	28
17625	25
17626	32
17627	39

No Programa 16a, a variável “dia” é importada com o informat DDMMYY10, que representa uma data no formato “dia/mês/ano” com 10 caracteres (dois para dia, dois para mês, quatro para ano e mais duas barrinhas). Apesar de o programa importar os dados com sucesso, um problema ocorre: as datas são mostradas como números! Isto acontece porque é assim que o SAS as armazena internamente. Por exemplo, 01/04/2017 é mostrado como 17623, ou seja, 17623 dias passados de 01/01/1960.

Para que as datas sejam apresentadas em um formato mais adequado, é preciso aplicar o comando FORMAT à variável “dia”. Este comando serve para mudar a forma como qualquer variável deverá ser mostrada em data sets e relatórios. No programa 16b, apresentado a seguir, a instrução FORMAT DIA DDMMYY10. é usada para especificar o formato de saída do tipo DATE com a máscara “dd/mm/aaaa” para a variável “dia”.

```
*Programa 16b: leitura de datas com formatação de exibição;
DATA Temperaturas;
  INFILE 'C:\CursoSAS\TEMPERATURAS.txt';
  INPUT
    @001 dia DDMMYY10.
    @012 graus_c 2.;

  FORMAT dia DDMMYY10.;
RUN;
```

Temperaturas

dia	graus
01/04/2017	12
02/04/2017	28
03/04/2017	25
04/04/2017	32
05/04/2017	39

IMPORTANTE!

Em muitas situações práticas, você terá que ler uma variável do tipo data a partir de um arquivo texto, mas não precisará efetuar nenhum tipo de operação com esta variável. Neste caso não existe qualquer problema em armazenar o valor desta data em uma variável caractere.

3.12 Trabalhando com Valores do Tipo Hora (TIME)

Na linguagem SAS, as variáveis do tipo TIME podem ser utilizadas para manipular informações do tipo hora/minuto/segundo. Assim como ocorre com as variáveis DATE, as variáveis do tipo TIME são armazenadas internamente como valores inteiros, desta vez correspondendo ao número de segundos passados desde a meia-noite. A seguir será apresentado um exemplo de importação de informações no formato TIME a partir do arquivo “DATA_HORA.txt”.

```
01/01/2016 00:00:00
15/03/2016 19:30:04
05/04/2017 02:25:45
```

O Programa 16 faz a leitura do arquivo quebrando a informação em duas variáveis. Uma delas (“dia”) recebe a porção DATE e a outra (“hora”) a porção TIME. Este programa contém dois comandos FORMAT para permitir que cada variável seja exibida com a formatação adequada no data set de saída

```
*Programa 17: leitura de datas e horas;
DATA Temperaturas;
  INFILE 'C:\CursoSAS\DATA_HORA.txt';
  INPUT
    @001 dia DDMMYY10.
    @012 hora TIME8.;

  FORMAT dia DDMMYY10.
         hora TIME8.;
RUN;
```

Data e Hora

dia	hora
01/01/2016	00:00:00
15/03/2016	19:30:04
05/04/2017	02:25:45

IMPORTANTE!

No próximo capítulo será apresentada uma técnica que permite a importação das informações da data e hora para uma única variável do tipo DATETIME.

4. Transformação de Dados no Passo DATA

O SAS permite que variáveis sejam criadas ou modificadas dentro de um passo DATA. Este capítulo cobre este importante tema, tendo a sua estrutura definida da seguinte forma. As Seções 4.1 a 4.18 apresentam diversas instruções e funções SAS que podem ser utilizadas para alterar ou criar variáveis no passo DATA. A Seção 4.19, última do capítulo, aborda o *Program Data Vector*, importante estrutura de dados interna que regula o funcionamento do passo DATA e que, por isso, deve ser conhecida e entendida por todo programador SAS.

4.1 Criando Variáveis dentro do Passo DATA

Em um programa SAS, a declaração de variáveis pode ser realizada em qualquer linha de um passo DATA. O **tipo** da variável (caractere ou numérico) será **automaticamente** estabelecido como o tipo de valor que for atribuído à variável. Para demonstrar este conceito, considere o arquivo “NOTAS.txt”, que contém as matrículas de quatro alunos e as notas obtidas por cada um em duas diferentes provas.

```
M0012017 9.8 9.5
M0022017 5.3 4.1
M0032017 2.5 8.0
M0042017 7.5 7.5
```

O Programa 18 importa o arquivo de notas e cria três novas variáveis: “media”, “media_pond” e “e_mail”. Explicações são apresentadas em comentários dentro do próprio corpo do programa.

```
/* Programa 18: criação de variáveis no passo DATA */
DATA Medias;
  INFILE 'C:\CursoSAS\NOTAS.txt';
  INPUT matricula $ 1-8 nota1 nota2;

  * CALCULA A MÉDIA;
  media = (nota1 + nota2) / 2;

  * MÉDIA PONDERADA (PESO 2 NA SEGUNDA PROVA);
  media_pond = (nota1 + nota2 * 2) / 3;

  * E-MAIL DO ALUNO - EXEMPLO DE CONCATENAÇÃO DE STRING;
  e_mail = matricula || '@unixyz.edu.br';
RUN;
```

Medias

matricula	nota1	nota2	media	media_pond	e_mail
M0012017	9.8	9.5	9.65	9.6	M0012017@unixyz.edu.br
M0022017	5.3	4.1	4.7	4.5	M0022017@unixyz.edu.br
M0032017	2.5	8.0	5.25	6.16666667	M0032017@unixyz.edu.br
M0042017	7.5	7.5	7.5	7.5	M0042017@unixyz.edu.br

Alguns comentários adicionais:

- Na linha onde a variável “e_mail” é criada, o operador “ || ” foi utilizado como operador de concatenação (a matrícula do aluno é concatenada com a *string* “@unixyz.edu.br”).
- Observe que, no exemplo apresentado, o programador não foi obrigado a declarar as variáveis e seus tipos no início do passo DATA. No entanto, ao longo deste livro serão apresentados alguns exemplos de situações práticas onde é necessário realizar este tipo de definição.

4.2 As Instruções de Desvio IF-THEN-ELSE

Na programação SAS, as instruções de desvio IF-THEN-ELSE são normalmente empregadas para a criação e transformação de variáveis. A seguir, apresentam-se diversos exemplos de sintaxe para a utilização destas instruções.

4.2.1 IF com apenas um comando subordinado

Basta utilizar uma linha de comando:

```
IF condição THEN comando1;
```

4.2.2 IF com mais de um comando subordinado

Neste caso é preciso envolver os comandos subordinados entre as palavras **DO** e **END**;

```
IF condição THEN DO;  
    comando1;  
    comando2;  
    ...  
    comandon;  
END;
```

ATENÇÃO!

A sintaxe da instrução IF requer o uso de ponto-e-vírgula após **ambas** as palavras **DO** e **END**.

4.2.3 IF / ELSE IF / ELSE

Condições podem ser logicamente agrupadas com o uso dos comandos ELSE IF e ELSE.

```

IF condição1 THEN comando1;
ELSE IF condição2 THEN comando2;
ELSE IF condição3 THEN comando3;
ELSE comando4;

```

Caso o IF, algum dos ELSE IFs ou o ELSE final possua mais de um comando subordinado, torna-se necessário colocar os comandos em questão entre as palavras reservadas DO; e END; - conforme mostrado no item 4.2.2.

O Programa 19 processa novamente arquivo “NOTAS.txt” (mostrado na seção anterior), porém criando desta vez a variável “situacao”. Veja que o conteúdo de “situação” leva em consideração o valor calculado da média do aluno e, se necessário, de sua média ponderada. No exemplo, o operador de comparação “>= ” (maior ou igual) é utilizado nas instruções IF e ELSE IF (informações adicionais são apresentadas na Seção 4.3).

```

/* Programa 19: instruções de desvio IF, ELSE IF e ELSE */
DATA Medias;
  INFILE 'C:\CursoSAS\NOTAS.TXT';
  INPUT matricula $ 1-8 nota1 nota2;

  * CALCULA A MÉDIA;
  media = (nota1 + nota2) / 2;

  /* VERIFICA SE ALUNO FOI APROVADO
     ELE SERÁ APROVADO SE A MÉDIA FOR MAIOR OU IGUAL A 6
     OU SE TIVER MÉDIA MAIOR OU IGUAL A 4
     E MÉDIA PONDERADA MAIOR OU IGUAL A 6
  */
  IF media >= 6.0 THEN
    situacao = 'APR';
  ELSE IF media >= 4.0 THEN DO;
    media_pond = (nota1 + nota2 * 2) / 3;
    IF media_pond >= 6.0 THEN
      situacao = 'APR';
    ELSE
      situacao = 'REP';
  END;
  ELSE situacao = 'REP';
RUN;

```

Medias

matricula	nota1	nota2	media	situacao	media_pond
M0012017	9.8	9.5	9.65	APR	.
M0022017	5.3	4.1	4.7	REP	4.5
M0032017	2.5	8.0	5.25	APR	6.16666667
M0042017	7.5	7.5	7.5	APR	.

4.3 Comparação de Comparação e Lógicos

4.3.1 Operadores de Comparação

Os operadores de comparação (também chamados de operadores relacionais) são utilizados junto às instruções de desvio – como IF-THEN-ELSE – com o intuito de permitir a especificação de uma comparação entre o conteúdo de duas variáveis ou a comparação de uma constante com o conteúdo de uma variável. Sempre que o resultado da comparação for FALSO, o valor 0 será retornado. Caso contrário, o valor 1 será retornado.

Em uma instrução de desvio, comparações envolvendo valores numéricos podem especificadas com o uso de **símbolos** ou **mnemônicos**. Já as comparações de valores alfanuméricos podem ser realizadas apenas com o uso dos símbolos. A Tabela 1 apresenta a relação de operadores de comparação da linguagem SAS. O operador IN é o único que não possui um símbolo associado, podendo, no entanto, ser utilizado com variáveis do tipo caractere ou numéricas.

Tabela 1. Operadores relacionais

Símbolo	Mnemônico	Significado
=	EQ	Igual
^=, ~= ou !=	NE	Diferente
>	GT	Maior
<	LT	Menor
>=	GE	Maior ou igual
<=	LE	Menor ou igual
	IN	Pesquisa em um conjunto de valores

Considerando as informações da Tabela 1, tem-se, por exemplo, que as seguintes instruções são equivalentes:

- IF x < y THEN z = 1
- IF x LT y THEN z=1.

É importante observar que um teste envolvendo a comparação de uma constante do tipo caractere requer a utilização de aspas simples ou duplas em volta da constante. Exemplo:

- IF nome="SAS" THEN k=2.

4.3.2 Valores Lógicos

Embora a Linguagem SAS não possua um tipo lógico (booleano), o tipo numérico pode ser utilizado para representar valores VERDADEIRO ou FALSO, conforme mencionado na seção anterior. Quando uma condição é avaliada, o valor 0 (zero) é tratado como o valor lógico FALSO e o valor 1 (um) ou qualquer outro valor diferente de zero é tratado como VERDADEIRO. Veja o exemplo a seguir. Ao final da execução as variáveis MSG1 e MSG3 conterão o valor 'ENTROU NESSE IF', enquanto MSG2 permanecerá com o conteúdo missing.

```

/* Programa 20: examinando a avaliação de valores lógicos */
DATA Logico;
  X=1; Y=0; Z=-999;
  LENGTH msg1 $ 30 msg2 $ 30 msg3 $ 30;
  IF (X) THEN msg1='ENTROU NESSE IF';
  IF (Y) THEN msg2='NÃO ENTROU NESSE IF';
  IF (Z) THEN msg3='ENTROU NESSE IF';
RUN;

```

Logico

X	Y	Z	msg1	msg2	msg3
1	0	-999	ENTROU NESSE IF		ENTROU NESSE IF

4.3.3 Operadores Lógicos

A Tabela 2 apresenta os operadores lógicos disponíveis na linguagem SAS, que também podem ser especificados com o uso de símbolos ou mnemônicos. Estes operadores combinam o resultado de duas ou mais condições especificadas no comando IF (ou em outro comando que envolva teste lógico).

Tabela 2. Operadores lógicos

Símbolo	Mnemônico	Significado
&	AND	Operador E
ou !	OR	Operador OU
^, ~ ou ¬	NOT	Operador NÃO

Com relação ao uso de símbolos ou mnemônicos, observe que as duas instruções a seguir são equivalentes:

- IF idade = 18 AND sexo= “M” THEN alistar_exercito = 1;
- IF idade = 18 & sexo = “M” THEN alistar_exercito = 1;

A Tabela 3 apresenta a ordem de precedência aplicada quando vários operadores lógicos estão presentes em um mesmo comando IF. O operador NOT tem precedência maior, seguido do operador AND e do operador OR, respectivamente.

Tabela 3. Regras de precedência para operadores lógicos

Operador	Ordem de Avaliação
NOT	1
AND	2
OR	3

Estas regras de precedência podem ser sobrepostas através do uso de parênteses. Considere o seguinte exemplo: suponha uma data set que armazena informações sobre apartamentos à venda. Considere que um programador deseje, por algum motivo, marcar os apartamentos que possuam três quartos e que estejam localizados nos bairros “CENTRO” e “BARRA”. Neste caso, o comando IF abaixo **não atenderia** ao desejo do programador, pois acabaria por marcar o conjunto de apartamentos errado:

```
IF bairro = 'CENTRO' OR bairro='BARRA' AND quartos=3 THEN
    marcado=1;
ELSE
    marcado=0;
```

Como o operador AND tem precedência maior do que a do operador OR, o SAS testará a condição especificada em **negrito** primeiro. Ou seja, o teste será interpretado da seguinte maneira: “Na observação corrente, faça marcado=1 se o apartamento estiver localizado no bairro da 'BARRA' e possuir 3 quartos; ou se o apartamento estiver localizado no 'CENTRO'”. Sendo assim, todos os apartamentos do “CENTRO” seriam marcados com 1, independente de possuírem três quartos ou não.

Observe agora o código a seguir, onde os parênteses são empregados para alterar a precedência e, desta forma, marcar o conjunto de observações correto:

```
IF (bairro = 'CENTRO' OR bairro='BARRA') AND quartos=3 THEN
    marcado=1;
ELSE
    marcado=0;
```

Desta vez, o teste será interpretado pelo SAS da seguinte maneira: “Na observação corrente faça marcado=1, se apartamento estiver localizado no 'CENTRO' ou 'BARRA' e se o mesmo possuir três quartos”.

4.4 Comparações Envolvendo o Valor Missing

Comparações numéricas que envolvam o valor missing merecem ser tratadas com cuidado especial. Neste tipo de operação, o seguinte critério é utilizado pelo SAS: quando um valor está armazenado como missing em uma variável numérica, ele será tratado como equivalente a **menos infinito**. Ou seja: um valor menor do que qualquer outro valor que poderia estar armazenado na variável.

Supondo que você tenha um valor numérico qualquer armazenado em uma variável “n”, os resultados das diferentes comparações envolvendo esta variável serão sempre os apresentados na Tabela 4.

Tabela 4. Comparações envolvendo o valor missing

comparação	resultado
. = n	resulta em FALSO
. ~= n	resulta em VERDADEIRO
. < n	resulta em VERDADEIRO (menos infinito é sempre menor do que n)
. <= n	resulta em VERDADEIRO (menos infinito é sempre menor ou igual a n)
. > n	resulta em FALSO
. >= n	resulta em FALSO

IMPORTANTE!

Qualquer operação aritmética envolvendo o missing (por exemplo, $. + 1$), sempre resultará em missing. O mesmo não é válido para funções e procedures que realizam algum tipo de computação sobre conjuntos de valores (ex: calcular a média dos valores de uma determinada variável de um data set), onde normalmente os valores missing são descartados do cálculo (essa situação será apresentada em maiores detalhes no capítulo sobre PROCs).

Caso o seu objetivo seja o de testar se o conteúdo de uma variável está missing, a forma de efetuar o teste é a seguinte: para variáveis do tipo caractere, o teste consiste em uma comparação contra um espaço em branco entre aspas (' ' ou “ ”) e para variáveis numéricas a comparação é realizada contra o ponto (.). Por exemplo, supondo que X e Y representem variáveis do tipo caractere e numérico, respectivamente, os comandos IF necessários para avaliar se cada uma possui valor missing são dados por:

- IF X = ' ' THEN ...
- IF Y = . THEN ...

4.5 Operadores Aritméticos e de String

A Tabela 5 relaciona os operadores aritméticos da linguagem SAS.

Tabela 5. Operadores aritméticos

Símbolo	Definição	Exemplo	Resultado
+	adição	x+10	Adiciona o valor 10 a x.
-	subtração	y - 3	Subtrai o valor 3 de y
*	multiplicação	x * 3	Multiplifica x por 3
/	divisão	y / 5	Divide y por 5
**	exponenciação	z ** 3	Eleva z à terceira potência

Para a realização de outros tipos de operações matemáticas, é necessário fazer uso de alguma função matemática ou aritmética disponibilizada pelo Base SAS. Por exemplo, para calcular a raiz quadrada de um número, utiliza-se a função SQRT. A Seção 4.13 introduz as principais funções do Base SAS.

4.6 Operação de Data Subsetting

Considere que seja necessário criar um data set SAS que contenha apenas um subconjunto de observações de um arquivo de entrada. Por exemplo: a partir de um arquivo de alunos, montar um data set contendo apenas os alunos com média acima de 6,0. Este processo é conhecido no mundo SAS como *subsetting*. Ele também é realizado com a utilização da instrução IF, conforme mostra o Programa 21a:

```

/* Programa 21a: data subsetting */
DATA Aprovados;
  INFILE 'C:\CursoSAS\NOTAS.TXT';
  INPUT MATRICULA $ 1-8 NOTA1 NOTA2;
  * CALCULA A MÉDIA;
  MEDIA = (NOTA1 + NOTA2) / 2;

  * MANTÉM APENAS QUEM OBTVEVE A MÉDIA MÍNIMA EM "Aprovados";
  IF MEDIA >= 6.0;
RUN;

```

Aprovados

matricula	nota1	nota2	media
M0012017	9.8	9.5	9.65
M0042017	7.5	7.5	7.5

Observe que o *data set* aprovados conterá apenas dois registros, com os dados dos dois alunos que obtiveram a média superior à 6.0.

A sintaxe do comando para subsetting parece, a princípio, um pouco estranha. Afinal, qualquer programador que não conhece SAS observaria este programa e pensaria: “se (IF) a média for maior ou igual a 6.0 então (THEN) o quê??? Será que este IF não está incompleto?!”. Não, não está incompleto, é simplesmente a “sintaxe econômica” adotada pelo SAS para implementar a operação de data subsetting.

IMPORTANTE!

A operação de data subsetting com o comando IF é feita da seguinte forma:

IF condição;

Funcionamento:

se a condição resultar em VERDADEIRO, o SAS continua normalmente com o passo DATA.

se a condição for FALSA:

1. Nenhum outro comando será executado para aquela observação;
2. A observação não será adicionada ao data set que está sendo criado;
3. O SAS passará para a próxima observação

O comando DELETE também pode ser usado na operação de subsetting. Enquanto o IF sozinho indica observações a serem incluídas, um IF com DELETE serve para indicar as observações a serem excluídas. Veja o exemplo apresentado no Programa 21b:

```

/* Programa 21b: data subsetting com a instrução DELETE */
DATA Aprovados;
  INFILE 'C:\CursoSAS\NOTAS.TXT';
  INPUT MATRICULA $ 1-8 NOTA1 NOTA2;
  * CALCULA A MÉDIA;
  MEDIA = (NOTA1 + NOTA2) / 2;

  * EXCLUI DE "Aprovados" QUEM NÃO OBTEVE A MÉDIA 6,0;
  IF MEDIA < 6.0 THEN DELETE;
RUN;

```

Em resumo, as duas instruções para data subsetting apresentadas abaixo são equivalentes:

- IF MEDIA >= 6.0;
- IF MEDIA < 6.0 THEN DELETE;

4.7 Conhecendo as Data Set Options

As *data set options* permitem com que o programador possa definir determinadas ações específicas enquanto um data set está sendo lido ou criado. Existem quatro diferentes tipos, apresentados na subseções a seguir.

4.7.1 KEEP

A opção KEEP é utilizada para permitir com que seja indicada a lista de variáveis que serão gravadas no data set de saída. Se uma variável não estiver na lista ela não será levada para o data set de saída.

```

/* Programa 22: data set option KEEP */
/* o data set de saída conterá apenas as variáveis
   "matricula" e "media" */

DATA MediasFinais (KEEP = matricula media);
  INFILE 'C:\CursoSAS\NOTAS.TXT';
  INPUT MATRICULA $ 1-8 notal1 nota2;
  * CALCULA A MÉDIA;
  media = (notal1 + nota2) / 2;
RUN;

```

MediasFinais

matricula	media
M0012017	9.65
M0022017	4.7
M0032017	5.25
M0042017	7.5

O Programa 22 solicita a criação de um data set denominado *MediasFinais* que conterá apenas duas variáveis, “matricula” e “media”. Ou seja: as variáveis “nota1” e “nota2” desta vez não farão parte do data set gerado no passo DATA.

A opção KEEP deve ser colocada entre parênteses, seguida do sinal de igualdade. A relação de variáveis a serem **mantidas** deve ser especificada utilizando um **espaço em branco** como separador. A sintaxe é apresentada a seguir:

- DATA nome_do_dataset (KEEP = lista de variáveis mantidas);

4.7.2 DROP

A opção DROP é utilizada com o mesmo propósito do KEEP. A diferença é que a lista associada a essa opção indica as variáveis que **não** serão gravadas no data set de saída. O Programa 23 mostra como a data set option pode ser utilizada para gerar um data set idêntico ao gerado pelo Programa 21.

```
/* Programa 23: a data option DROP */
/* o data set de saída conterá apenas as variáveis
MATRICULA e MEDIA */

DATA MediasFinais (DROP = nota1 nota2);
  INFILE 'C:\CursoSAS\NOTAS.TXT';
  INPUT MATRICULA $ 1-8 nota1 nota2;
  * CALCULA A MÉDIA;
  media = (nota1 + nota2) / 2;
RUN;
```

A sintaxe da opção DROP é apresentada a seguir:

- DATA nome_do_dataset (DROP = lista de variáveis excluídas);

4.7.3 WHERE

A opção WHERE é utilizada com o objetivo de realizar a operação de subsetting (exclusão de observações, operação apresentada na Seção 4.4). Uma observação será gravada no data set de saída apenas quando a condição especificada na cláusula WHERE for verdadeira. No exemplo a seguir, o data set *Aprovados* conterá apenas as matrículas e médias finais dos alunos aprovados. Observe que foi possível combinar as opções KEEP e WHERE.

```
/* Programa 24: a data option WHERE */
/* o data set de saída conterá apenas os alunos aprovados;

DATA Aprovados (KEEP = matricula media
                WHERE = (media >= 6.0));

  INFILE 'C:\CursoSAS\NOTAS.TXT';
  INPUT matricula $ 1-8 nota1 nota2;
  * CALCULA A MÉDIA;
  media = (nota1 + nota2) / 2;
RUN;
```

Aprovados

matricula	media
M0012017	9.65
M0042017	7.5

4.7.4 RENAME

A opção RENAME pode ser utilizada para que mudar o nome de uma variável no data set de saída. A mudança de nome ocorre apenas no momento em que a observação está sendo gravada. Por este motivo, o programa **deve** referenciar os nomes antigos das variáveis dentro do passo DATA.

```
/* Programa 25: a data option RENAME */  
*a variável "media" será renomeada para "media_final" no data set  
de saída;
```

```
DATA Medias (KEEP = matricula media  
              RENAME = (media = media_final));
```

```
INFILE 'C:\CursoSAS\NOTAS.TXT';  
INPUT matricula $ 1-8 nota1 nota2;  
* CALCULA A MÉDIA;  
media = (NOTA1 + NOTA2) / 2;  
RUN;
```

Medias

matricula	media_final
M0012017	9.65
M0022017	4.7
M0032017	5.25
M0042017	7.5

4.8 Comando RETAIN

Durante a execução do laço implícito do passo DATA, o SAS **atribui o valor missing para todas as variáveis no início de cada iteração**. Estes valores podem até ser modificados com o uso do comando INPUT ou através de comandos de atribuição, mas ficarão novamente com o valor missing no início da iteração seguinte. Para melhor compreender esta característica curiosa do passo DATA, considere o seguinte exemplo:

* Programa 26a: a variável "contador" não muda de valor nas diferentes iterações;

```
DATA Sem_Retain;
  contador = 0; *declara "contador" e inicializa com zero;
  INFILE 'C:\CursoSAS\NOTAS.TXT';
  INPUT matricula $ 1-8 nota1 nota2;
  media = (NOTA1 + NOTA2) / 2;
  contador = contador + 1;
RUN;
```

Sem Retain

contador	matricula	nota1	nota2	media
1	M0012017	9.8	9.5	9.65
1	M0022017	5.3	4.1	4.7
1	M0032017	2.5	8.0	5.25
1	M0042017	7.5	7.5	7.5

Veja que, no programa, a variável "contador" foi declarada no início do passo DATA, recebendo o valor zero (assim o seu tipo foi automaticamente estabelecido como numérico). Na última linha do passo DATA, incrementa-se à variável em uma unidade. No entanto, isto não é suficiente para que as mudanças no valor de "contador" sejam acumuladas entre os diferentes passos, conforme mostra o data set resultante..

O comando RETAIN pode ser utilizado para modificar este comportamento. Se uma variável for declarada com o uso deste comando, seu valor ao final de uma iteração será levado para a iteração seguinte. O Programa 26b demonstra o uso do RETAIN.

* Programa 26b: declara "contador" iniciando com zero e usando RETAIN;

```
RETAIN contador 0;

INFILE 'C:\CursoSAS\NOTAS.TXT';
INPUT matricula $ 1-8 nota1 nota2;
media = (NOTA1 + NOTA2) / 2;
contador = contador + 1;
RUN;
```

Com Retain

contador	matricula	nota1	nota2	media
1	M0012017	9.8	9.5	9.65
2	M0022017	5.3	4.1	4.7
3	M0032017	2.5	8.0	5.25
4	M0042017	7.5	7.5	7.5

IMPORTANTE!

No exemplo apresentado, o comando RETAIN foi utilizado na primeira linha do passo DATA e a variável declarada possuía um valor inicial especificado. No entanto, o SAS permite com que o RETAIN seja utilizado em qualquer linha e também não requer a especificação de um valor inicial.

4.9 Comando LENGTH

O comando LENGTH – que já foi apresentado na Seção 3.3 – serve para especificar o comprimento de uma variável caractere no momento de sua declaração. Este comando será revisto agora, por ser o comando requerido para **impedir com que variáveis declaradas com o uso dos comandos IF-THEN-ELSE contenham valores truncados**.

Se você criar uma variável através de um comando de atribuição, o comprimento da variável será determinado pelo valor da primeira ocorrência dela. Considere o seguinte exemplo, que mais uma vez utiliza a base de dados “TEMPERATURAS.txt” (esta base foi utilizada na nos exemplos da Seção 3.10), contendo cinco datas e a temperatura máxima registrada em cada uma dessas datas.

```
01/0/2017 12
02/04/2017 28
03/04/2017 25
04/04/2017 32
05/04/2017 39
```

Considere o seguinte programa, que cria a variável derivada “classe” baseada em diferentes faixas de valores de temperatura.

```
* Programa 27a: variável “classe” com valores truncados;
DATA Temperaturas;
  INFILE 'C:\CursoSAS\TEMPERATURAS.txt';
  INPUT
    @001 dia DDMMYY10.
    @012 graus_c 2.;

  FORMAT dia DDMMYY10.;

  IF graus_c <= 20 THEN
    classe = 'FRIO';
  ELSE IF graus_c > 20 AND graus_c <= 29 THEN
    classe = 'AGRADAVEL';
  ELSE IF graus_c > 29 AND graus_c <= 34 THEN
    classe = 'CALOR';
  ELSE
    classe = 'CALOR ABSURDO';
RUN;
```

Temperaturas

dia	graus	classe
01/04/2017	12	FRIO
02/04/2017	28	AGRA
03/04/2017	25	AGRA
04/04/2017	32	CALO
05/04/2017	39	CALO

A variável “classe” recebe o valor ‘FRIO’ durante o processamento da primeira observação. Como esta palavra possui 4 letras, o SAS atribui comprimento 4 para “classe”. Qualquer outro valor para esta variável, que seja atribuído em uma iteração posterior do passo DATA, será truncado em 4 caracteres, conforme mostrou o resultado.

Para resolver este problema, basta declarar explicitamente a variável “classe” utilizando o comando LENGTH para estabelecer o comprimento desta variável. Isto é feito no programa a seguir.

```
* Programa 27b: variável "classe" definida corretamente;
DATA Temperaturas;
  INFILE 'C:\CursoSAS\TEMPERATURAS.txt';
  INPUT dia $ 1-10 graus_c;

  *declara "classe" como string de 15 posições;
  LENGTH classe $ 15;

  IF graus_c <= 20 THEN
    classe = 'FRIO';
  ELSE IF graus_c > 20 AND graus_c <= 29 THEN
    classe = 'AGRADAVEL';
  ELSE IF graus_c > 29 AND graus_c <= 34 THEN
    classe = 'CALOR';
  ELSE
    classe = 'CALOR ABSURDO';
RUN;
```

Temperaturas

dia	graus	classe
01/04/2017	12	FRIO
02/04/2017	28	AGRADAVEL
03/04/2017	25	AGRADAVEL
04/04/2017	32	CALOR
05/04/2017	39	CALOR ABSURDO

4.11 Desvio Utilizando a Estrutura SELECT-WHEN

A estrutura SELECT-WHEN é utilizada para executar diferentes grupos de comandos em função do valor de uma expressão. A sintaxe do comando é apresentada a seguir:

```
SELECT <(variável)>;  
    WHEN1 (expressão1)  
        comandos1;  
    WHENn (expressãon)  
        comandosn;  
    OTHERWISE  
        comandoso;  
END;
```

A estrutura SELECT-WHEN funciona da seguinte maneira. A expressão associada ao primeiro WHEN é avaliada. Se a avaliação resultar em VERDADEIRO, o programa executa os comandos subordinados a este WHEN e depois “sai do SELECT” (o fluxo do programa é desviado para a instrução localizada imediatamente após o END do SELECT). Se a avaliação do primeiro WHEN resultar em FALSO, então a expressão associada ao segundo WHEN será avaliada. Se a avaliação for VERDADEIRO, executam-se os comandos associados ao segundo WHEN e o fluxo de execução “sai do SELECT”. E da mesma maneira são avaliados todos os WHENs. A seguir um exemplo simples de uso da estrutura SELECT para determinar a região de uma UF durante a importação do arquivo “UFs.csv”

```
AL,Alagoas  
BA,Bahia  
CE,Ceará  
ES,Espírito Santo  
MA,Maranhão  
MG,Minas Gerais  
PB,Paraíba  
PE,Pernambuco  
PI,Piauí  
PR,Paraná  
RJ,Rio de Janeiro  
RN,Rio Grande do Norte  
RS,Rio Grande do Sul  
SC,Santa Catarina  
SE,Sergipe  
SP,São Paulo
```

```

* Programa 28: utilizando SELECT-WHEN;
DATA UFs;
  INFILE 'C:\CursoSAS\UFS.csv' DLM=',' DSD;
  LENGTH sigla $ 2 nome $ 20 regiao $10;
  INPUT sigla $ nome $;

  SELECT (sigla);
    WHEN ('RJ','SP','MG','ES') REGIAO = 'SUDESTE';
    WHEN ('RS','SC','PR') REGIAO = 'SUL';
    WHEN ('MA','PI','CE','RN','PB','PE','AL','SE','BA')
                                                REGIAO = 'NORDESTE';
    OTHERWISE REGIAO = 'N/A';

  END;
RUN;

```

UFs

sigla	nome	regiao
AL	Alagoas	NORDESTE
BA	Bahia	NORDESTE
CE	Ceará	NORDESTE
ES	Espírito Santo	SUDESTE
MA	Maranhão	NORDESTE
MG	Minas Gerais	SUDESTE
PB	Paraíba	NORDESTE
PE	Pernambuco	NORDESTE
PI	Piauí	NORDESTE
PR	Paraná	SUL
RJ	Rio de Janeiro	SUDESTE
RN	Rio Grande do Norte	NORDESTE
RS	Rio Grande do Sul	SUL
SC	Santa Catarina	SUL
SE	Sergipe	NORDESTE
SP	São Paulo	SUDESTE

É possível montar a estrutura SELECT-WHEN sem associar uma variável ao SELECT. O exemplo a seguir demonstra esta característica. Trata-se do programa para obtenção de classes de temperaturas (apresentado na seção anterior), agora modificado para utilizar a estrutura SELECT-WHEN.

```

* Programa 29: utilizando SELECT-WHEN (2);
DATA Temperaturas;
  INFILE 'C:\CursoSAS\TEMPERATURAS.txt';
  INPUT dia $ 1-10 graus_c;
  LENGTH classe $ 15;

  SELECT;
    WHEN (graus_c <= 20) CLASSE = 'FRIO';
    WHEN (graus_c > 20 AND GRAUS_C <=29) classe = 'AGRADÁVEL';
    WHEN (graus_c > 29 AND GRAUS_C <=34) classe = 'CALOR';
    OTHERWISE classe = 'CALOR ABSURDO';
  END;
RUN;

```

Temperaturas

dia	graus	classe
01/04/2017	12	FRIO
02/04/2017	28	AGRADAVEL
03/04/2017	25	AGRADAVEL
04/04/2017	32	CALOR
05/04/2017	39	CALOR ABSURDO

Por fim, o próximo exemplo demonstra que é possível montar expressões compostas por condições e variáveis distintas na cláusula WHEN. O exemplo também descreve o fluxo de execução da estrutura SELECT-WHEN.

```

* Programa 30: utilizando SELECT-WHEN (3);
DATA Exemplo;
  salario=500; filhos=3;

  SELECT;
    WHEN (salario <= 600 AND filhos > 1 ) bonus=salario * 0.20;
    WHEN (filhos > 2) bonus = salario * 0.05;
    OTHERWISE bonus=0;
  END;
RUN;

```

Exemplo

salario	filhos	bonus
500	3	100

As variáveis “salario” e “filhos” são avaliadas contra o primeiro WHEN. Note que o resultado da avaliação é VERDADEIRO e, por isso, a instrução `bonus= salario * 0.20` é executada. A seguir o controle do programa é automaticamente passado para o primeiro comando depois do END; do SELECT (melhor explicando: o SELECT é encerrado). Por

esta razão as expressões WHEN (FILHOS > 2) e OTHERWISE não são avaliadas pelo Programa 29.

4.12 Comando OUTPUT

Em seu processamento normal, o SAS irá adicionar uma observação ao data set de saída sempre **ao final** de cada iteração (após o último comando) do laço implícito do passo DATA. No entanto, é possível mudar este comportamento com o uso do comando OUTPUT, conforme mostra o Programa 31.

```
Programa 31: comando OUTPUT;  
DATA ExemploOutput;  
  X=1; Y=2; Z = X * Y;  
  OUTPUT;  
  
  X=2; Y=2; Z = X * Y;  
  OUTPUT;  
  
  X=2; Y=3; Z = X * Y; K=1000;  
  OUTPUT;  
RUN;
```

ExemploOutput

X	Y	Z	K
1	2	2	.
2	2	4	.
2	3	6	1000

Como resultado da execução, será criado um data set com 3 observações. Cada observação conterá valores diferentes para as variáveis X, Y, Z e K, que correspondem aos valores armazenados nestas variáveis no momento anterior à execução do comando OUTPUT. Note que o valor de K é missing nas duas primeiras observações.

4.13 Funções do Base SAS

Uma função é um componente da linguagem SAS que recebe zero, um ou mais argumentos, executa algum tipo de processamento, e retorna um valor. O valor retornado pode ser do tipo numérico ou string. O pacote Base SAS oferece cerca de 400 funções, de diferentes categorias, incluindo aritméticas, matemáticas, para conversão de tipos, para processamento de strings, para processamento de datas e outras. Estas funções podem ser utilizadas dentro:

- de um passo DATA;

- da PROC REPORT e de algumas PROCs estatísticas (assunto do Capítulo 5)
- de uma instrução SQL (Capítulo 6).
- de uma macro (Capítulo 6)

Será apresentado agora um exemplo de programa simples que cria uma variável dentro do passo DATA cujo valor é computado por uma função. Esse programa faz uso do arquivo “INTEIROS.txt”, que contém 3 números inteiros:

```
64
100
144
```

O programa a seguir realiza a leitura do arquivo “INTEIROS.txt” e gera um data set de saída que armazena o número lido do arquivo (variável “n”) e a sua raiz quadrada (variável “raiz”). A função SQRT é utilizada para computar a raiz quadrada de “n”.

```
*Programa 32: utilizando uma função;
DATA Raizes;
  INFILE 'C:\CursoSAS\INTEIROS.txt';
  INPUT n;

  raiz = SQRT(n);
RUN;
```

Raizes

n	raiz
64	8
100	10
144	12

As subseções a seguir apresentam algumas das funções mais utilizadas pelos programadores SAS. Cada subseção abrange uma categoria específica.

4.13.1 Funções Aritméticas e Matemáticas

A Tabela 6 relaciona exemplos de funções aritméticas e matemáticas do SAS.

Tabela 6. Funções aritméticas e matemáticas

Função	Significado	Código Exemplo
ABS(n)	Retorna o valor absoluto de <i>n</i> .	X=-9; Y=ABS(X); *retorna 9;
EXP(n)	Retorna o valor da função exponencial.	X1=0; Y1=EXP(X1); *retorna 1; X2=1.0; Y2=EXP(X2); *retorna 2.7182818285;

INT(<i>n</i>)	Retorna a porção inteira de um número real.	X1=2.1; Y1=INT(X1); *retorna 2; X2=5.9; Y2=INT(X2); *retorna 5;
LOG(<i>n</i>)	Retorna o logaritmo natural (base <i>e</i> .)	X1=1.0; Y1=LOG(X1); *retorna 0; X2=10.0; Y2=LOG(X2); *retorna 2.302585093;
LOG10(<i>n</i>)	Retorna o logaritmo na base 10.	X = 100; Y=LOG10(X); *retorna 2;
LOG2(<i>n</i>)	Retorna o logaritmo na base 2.	X = 1024; Y=LOG2(X); *retorna 10;
MAX(<i>n</i> ₁ , ..., <i>n</i> _{<i>k</i>})	Retorna o maior valor entre <i>k</i> valores passados como argumento.	MAX(0, -1, 1); *retorna 1;
MEAN(<i>n</i> ₁ , ..., <i>n</i> _{<i>k</i>})	Retorna a média de <i>k</i> valores passados como argumento. Valores missing são ignorados.	MEAN(2, ., 1, 5); *retorna 2.66667;
MIN(<i>n</i> ₁ , ..., <i>n</i> _{<i>k</i>})	Retorna o menor valor entre <i>k</i> valores passados como argumento.	MIN(0, -1, 1); *retorna -1;
MOD(<i>n</i> , <i>d</i>)	Retorna o resto da divisão de <i>n</i> por <i>d</i> .	MOD(11,3); *retorna 2;
N(<i>n</i> ₁ , ..., <i>n</i> _{<i>k</i>})	Retorna a quantidade valores diferentes de missing em uma lista.	N(2, ., 1, 5); *retorna 3;
NMISS(<i>n</i> ₁ , ..., <i>n</i> _{<i>k</i>})	Retorna a quantidade valores missing em uma lista	NMISS(2, ., 1, 5); *retorna 1;
ROUND(<i>n</i> , <i>m</i>)	Arredonda <i>n</i> para o múltiplo mais próximo de <i>m</i> ou para o inteiro mais próximo caso <i>m</i> seja omitido.	ROUND(9.8); *retorna 10; ROUND(5.1); *retorna 5; ROUND(1234.56,100); *retorna 1200;
SIGN(<i>n</i>)	Retorna o sinal de <i>n</i> ou zero.	X1=-90; Y1=SIGN(X1); *retorna -1; X2=200; Y2=ABS(X2); *retorna 1;
SQRT(<i>n</i>)	Retorna a raiz quadrada de <i>n</i> .	SQRT(9); *retorna 3;
SUM(<i>n</i> ₁ , ..., <i>n</i> _{<i>k</i>})	Retorna a soma de <i>k</i> valores passados como argumento. Valores missing são ignorados.	SUM(2, ., 1, 5); *retorna 8;

4.13.2 Funções para Conversão de Tipos

As funções PUT e INPUT (não confundir com os comandos de mesmo nome) são as principais funções de conversão do SAS, merecendo um destaque especial nesta seção. A função INPUT recebe como entrada uma variável caractere e retorna o seu valor convertido para numérico. É preciso especificar no segundo argumento da função o número

de posições desejadas para o valor convertido. A função PUT faz o contrário: recebe um número e converte para caractere. Também é preciso especificar no segundo argumento o tamanho do valor de saída. O Programa 33 apresenta exemplos de utilização de ambas as funções.

```
* Programa 33: funções PUT e INPUT;
DATA Conversao;
  ano_str = '2017'; *variável string;
  x_num = 1; *variável numérica;

  *ano_num: valor de "ano_str" convertido para
              number de 4 posições;
  ano_num = INPUT(ano_str, 4.);

  *x_str: valor de "x_num" convertido para
           string de 1 posição;
  x_str = PUT(x_num,1.);

  *x_str2: valor de "x_num" convertido para
            string de 4 posições;
  *      a máscara "z4." Faz com que a string seja;
  *      completada com zeros à esquerda;
  x_str2 = PUT(x_num,z4.);
RUN;
```

Conversao

ano_str	x_num	ano_num	x_str	x_str2
2017	1	2017	1	0001

4.13.3 Funções para Processamento de Strings

O pacote Base SAS oferece uma rica coleção de funções para processamento de strings. Algumas das mais utilizadas são apresentadas na Tabela 7.

Tabela 7. Funções para processamento de strings

Função	Significado	Código Exemplo
COMPRESS(s, c)	Remove da string <i>s</i> todas as ocorrências do(s) caractere(s) especificado(s) em <i>c</i> .	X='abacaxi'; Y = COMPRESS(X,'ai'); *retorna 'bcx';
FIND(s, sub)	Retorna a posição da primeira ocorrência da substring <i>sub</i> dentro da string <i>s</i> ou zero caso a substring não seja encontrada.	LENGTH X \$10; X='Mississippi'; Y=FIND(X,'is'); *retorna 2; X='BAHIA'; Y=FIND(X,'h'); *retorna 0;

FIND(s, sub, m)	Função FIND acrescida de um ou mais modificadores. Estes podem ser usados como parâmetros para procurar a partir de uma posição específica de <i>s</i> , ou para ignorar maiúsculas/minúsculas ('i').	LENGTH X \$10; X='Mississippi'; Y=FIND(X,'is',3); *retorna 5, pois começou a procurar da posição 3. X='BAHIA'; Y=FIND(X,'h','i'); *retorna 3.
LENGTHC(s)	Retorna o comprimento de <i>s</i> contando os espaços em branco à direita ou esquerda.	LENGTH X \$10; X='Rio'; Y = LENGTHC(X); *retorna 10;
LENGTHN(s)	Retorna o comprimento de <i>s</i> sem contar espaços em branco à direita ou esquerda.	LENGTH X \$10; X='Rio'; Y = LENGTHN(X); *retorna 3;
LOWCASE(s)	Converte <i>s</i> para minúsculo.	LOWCASE('AEIOU'); *retorna 'aeiou';
PROPCASE(s)	Converte os caracteres iniciais de <i>s</i> para maiúsculo.	PROPCASE('minas gerais'); *retorna 'Minas Gerais';
SCAN(s, pos, d)	Quebra a string <i>s</i> em partes, de acordo com o delimitador <i>d</i> , retornando a parte associada ao índice <i>pos</i> .	X='ABCDEFGH'; Y=SCAN(X,1,'E'); *retorna 'ABCD' (primeira substring delimitada por 'E');
STRIP(s)	Remove espaços em branco à esquerda e à direita de <i>s</i> .	X=' Rio '; Y=STRIP(X); *retorna 'Rio';
SUBSTR(s, pos, k)	Extraí uma substring a partir da string <i>s</i> , começando da posição <i>pos</i> e com tamanho de <i>k</i> caracteres	X='Namorar'; Y=SUBSTR(X,2,4); *retorna 'amor';
TRANWRD(s, a, b)	Na string <i>s</i> , troca todas as ocorrências da substring <i>a</i> pela substring <i>b</i> .	X='abacaxi'; Y = TRANWRD(X,'xi','te'); *retorna 'abacate';
TRIM(s)	Remove espaços em branco à direita de <i>s</i> .	X=' Rio '; Y=TRIM(X); *retorna ' Rio';
UPCASE(s)	Converte <i>s</i> para maiúsculo.	UPCASE('aeiou'); *retorna 'AEIOU';

4.13.4 Funções para Tratamento de Valores Missing

Conforme mostrado em seções anteriores, o valor missing possui uma semântica especial no SAS, exigindo cuidados especiais sempre que toma parte em operações aritméticas e de comparação. A Tabela 8 apresenta duas funções úteis para o tratamento deste tipo de valor.

Tabela 8. Funções para tratamento de valores missing

Função	Significado	Código Exemplo
COALESCE(v, b)	Caso <i>v</i> esteja missing, troca o seu valor por <i>b</i> . Caso contrário, mantém o valor original de <i>v</i> .	X1=1000; X2 =.; Y1=COALESCE(X1, 9); *retorna 1000; Y2=COALESCE(X2, 9); *retorna 9;
MISSING(v)	Retorna 1 (VERDADEIRO) caso a variável <i>v</i> esteja missing ou 0 (FALSO) caso contrário. A variável <i>v</i> pode ser do tipo numérico ou string.	X1=1000; X2 =.; Y1=MISSING(X1); *retorna 0; Y2=MISSING(X2); *retorna 1;

4.13.5 Comentários Finais

Esta seção apresentou uma introdução às funções do Base SAS. Conforme mencionado no início da seção, este pacote oferece centenas de funções, muitas delas possuindo mais de uma forma de utilização (uma mesma função que se comporta de maneira distinta dependendo dos tipos e quantidades de parâmetros especificados). Desta forma, cobrir totalmente o assunto está fora do escopo deste livro. Para obter uma referência completa das funções do Base SAS, consulte o manual de referência “SAS 9.4 Functions and CALL Routines”. É importante registrar que em diversas outras seções deste livro (como é o caso das duas próximas seções), outras funções ainda serão introduzidas.

4.14 Trabalhando com Variáveis do Tipo DATETIME

Na Seção 3.12, foi apresentado um exemplo em que as informações do arquivo “DATA_HORA.txt” (reproduzido abaixo) foram importadas para duas variáveis distintas, uma do tipo DATE (para armazenar o dia/mês/ano) e outra do tipo TIME (para armazenar a hora/minuto/segundo).

```
01/01/2016 00:00:00
15/03/2016 19:30:04
05/04/2017 02:25:45
```

Entretanto, é possível armazenar ambas as informações em uma variável só, caso o tipo DATETIME seja utilizado. Assim como ocorre com as variáveis DATE e TIME, as variáveis DATETIME também são armazenadas internamente como valor inteiro. Neste caso o valor armazenado representa o número de segundos passados desde a meia-noite de 01/01/1960. Valores positivos representam datas posteriores a este dia e valores negativos representam datas anteriores.

O Programa 34 apresenta um exemplo de importação de informações diretamente para o formato DATETIME. O procedimento precisa ser realizado em duas etapas. Primeiro quebra-se a informação em duas variáveis, uma para a porção DATE e a outra

para a porção TIME (mesmo processo realizado pelo programa da Seção 3.11). Em seguida uma função chamada DHMS deve ser utilizada para combinar as duas porções em uma única variável do tipo DATETIME.

```
*Programa 34: leitura de DATETIMES;

*PRIMEIRA ETAPA: LÊ PORÇÃO DATA E PORÇÃO HORA EM VARIÁVEIS
                  SEPARADAS
DATA Data_e_Hora;
  INFILE 'C:\CursoSAS\DATA_HORA.txt';
  INPUT
    @001 dia    DDMMYY10.
    @012 hora   TIME8.;

*SEGUNDA ETAPA: JUNTA AS DUAS PORÇÕES COM O USO DA FUNÇÃO DHMS;
data_final = DHMS(dia,0,0,hora);

FORMAT dia DDMMYY10.
        hora TIME8.
        data_final DATETIME.;

RUN;
```

Data e Hora

dia	hora	data_final
01/01/2016	00:00:00	01JAN16:00:00:00
15/03/2016	19:30:04	15MAR16:19:30:04
05/04/2017	02:25:45	05APR17:02:25:45

No programa acima, a sintaxe DHMS(VAR_DATE,0,0,VAR_TIME) foi utilizada para unir a porção DATE com a porção TIME, seguindo um exemplo indicado no manual do SAS, mas existem outras formas de utilização da função DHMS.

IMPORTANTE!

As variáveis do tipo DATE, TIME e DATETIME são sempre armazenadas internamente como variáveis numéricas. Elas podem até ser exibidas com formato de data, através da utilização do comando FORMAT, mas na realidade são gravadas como números.

4.15 Funções e Operações para a Manipulação de Datas

Esta seção aborda uma miscelânea de funções do Base SAS para a manipulação de variáveis do tipo DATE, TIME e DATETIME. Os primeiros exemplos são apresentados no

Programa 34, que também mostra algumas das operações permitidas com variáveis destes tipos. As explicações são fornecidas em comentários no próprio corpo do programa.

```
*Programa 35: funções e operações para datas;
DATA Funcoes_Data;

    *Função Today(): retorna a data corrente no servidor SAS;
    * o resultado é armazenado como tipo DATE e não DATETIME;
    hoje = Today();

    *É possível somar e subtrair dias de uma data;
    ontem = hoje - 1; *subtrai um dia da data de hoje;
    amanha = hoje + 1; *soma um dia da data de hoje;

    *Também podemos extrair o dia, mês e ano de uma variável;
    *DATE usando as funções Day, Month e Year, respectivamente;
    dia = Day(hoje);
    mes = Month(hoje);
    ano = Year(hoje);

    *Função DateTime():retorna a data/hora corrente no servidor SAS;
    agora = DateTime();

    *Função DatePart(): retorna a porção DATE de um DATETIME;
    *Função TimePart(): retorna a porção TIME de um DATETIME;
    pData = DatePart(agora);
    pHora = TimePart(agora);

    *Também é possível extrair as horas, minutos e segundos;
    * de uma variável TIME, usando as funções Hour, Minute e
      Second;
    hor    = Hour(pHora);
    min = Minute(pHora);
    seg = Second(pHora);

    FORMAT hoje DDMMYY10.
           ontem DDMMYY10. amanha DDMMYY10.
           dia 2. mes 2. ano 4.
           agora DATETIME.
           pData DDMMYY10.
           pHora TIME8.
           ;

RUN;
```

Supondo que o programa tenha sido executado no dia 31/05/2017 às 18:01:52, o seguinte data set seria produzido:

Funcoes Data

hoje	ontem	amanha	dia	mes	ano	agora	pData	pHora	hor	min	seg
31/05/ 2017	30/05/ 2017	01/06/2017	31	5	2017	31MAY 17:01:52	31/05/ 2017	18:01: 52	18	1	51.87

A seguir, apresenta-se um programa que introduz duas importantes funções que trabalham com o conceito de **intervalo** entre datas: DATDIF e INTCK.

```
*Programa 36: Funções DATDIF e INTCK;
DATA Intervalos;
  * declara e inicializa duas variáveis do tipo DATE;
  * as datas são declaradas entre aspas, com o uso do
  * formato (timestring) 'ddmmaaaa'd;
  d1 = '01jan2017'd;
  d2 = '15feb2017'd;

  * função DATDIF: retorna o número de dias contidos entre
    duas datas;
  *parâmetros:
    'actual' -> considera o número de dias real do mês
    '30' -> considera que cada mês possui 30 dias;
  d3 = DATDIF(d1,d2,'actual');
  d4 = DATDIF(d1,d2,'30');

  *função INTCK: permite a especificação de uma
    unidade de medida para o intervalo a ser calculado;
  d5 = INTCK('DAY ',d1,d2); *núm. de dias entre d1 e d2;
  d6 = INTCK('WEEK ',d1,d2); *núm. de semanas entre d1 e d2;
  d7 = INTCK('MONTH ',d1,d2); *núm. de meses entre d1 e d2;

  FORMAT d1 d2  DDMMYY10.;

RUN;
```

<i>Intervalos</i>						
d1	d2	d3	d4	d5	d6	d7
01/01/2017	15/02/2017	45	44	45	6	1

4.16 A Estrutura de Repetição DO-WHILE

Embora o passo DATA possua um laço implícito, existem muitas situações práticas onde é necessário programar algum tipo de estrutura de repetição dentro do mesmo para que seja possível montar o data set adequado ao problema que queremos resolver. Considere, por exemplo, o seguinte problema. Imagine que a partir do arquivo de entrada “NUMEROS.txt”, deseja-se criar um data set com duas variáveis: o número contido no arquivo e um indicador que informe se este número é um **número primo** ou não.

```
10
2
13
```

100
9
41
1000
1013
500
7

Um algoritmo que pode ser usado para determinar se um número n é primo é assumir a hipótese de que ele é primo e depois tentar quebrar esta hipótese, montando um laço (estrutura de repetição de comandos) que realize a divisão de n por todos os números contidos entre 2 e a raiz quadrada de n . Se em alguma iteração do laço o resto da divisão for igual a zero, então identifica-se que n não é primo. A resolução deste problema pode ser implementada na linguagem SAS com o uso da estrutura de repetição DO-WHILE, como mostra o Programa 37.

```
* Programa 37: DO-WHILE
DATA Primos (KEEP=n primo);
  * leitura dos números;
  INFILE 'C:\CursoSAS\NUMEROS.txt';
  INPUT n;

  * rotina que determina se o número é primo;
  primo = 'SIM'; *parto da hipótese de que é primo;
  i = 2;
  raiz = Round(Sqrt(n));
  DO WHILE (i <= raiz);
    IF MOD(n,i) = 0 THEN DO; *se o resto deu 0 não é primo;
      primo = 'NÃO';
      LEAVE; *quebra o laço DO-WHILE "na marra";
    END; *{IF};
    i = i + 1;
  END; *{WHILE};
RUN;
```

Primos

n	primo
10	NÃO
2	SIM
13	SIM
100	NÃO
9	NÃO
41	SIM
1000	NÃO
1013	SIM
500	NÃO
7	SIM

Um breve comentário sobre o funcionamento do programa. A opção (KEEP=n primo) faz com que o data set gerado contenha apenas as variáveis “n” e “primo”. As variáveis auxiliares “i” e “raiz” não são levadas para o data set.

A estrutura DO-WHILE opera de maneira absolutamente idêntica a de qualquer outra linguagem de programação. Trata-se de uma estrutura de repetição com **teste no início**, que é mantida **enquanto a condição especificada ao lado da palavra WHILE for satisfeita**. Em nosso exemplo, o programa “só entra no laço” quando o valor da raiz de “n” for maior do que 2 (valor inicial de “i”). Dentro do laço, “i” é incrementado em 1 a cada iteração, até que atinja a condição de saída. A sintaxe da estrutura DO-WHILE é resumida a seguir:

```
DO WHILE (condição);  
  Comando1;  
  ...  
  Comandon;  
END;
```

4.17 A Estrutura de Repetição DO

A linguagem SAS também permite a implementação de laços com repetição por um número fixo de iterações (a mesma operação realizada pelo comando FOR realiza em outras linguagens). Veja o exemplo do Programa 38

```
* Programa 38: DO (laço com número fixo de iterações);  
DATA TABUADA_DE_MULTIPLICACAO;  
  DO n=1 to 9 by 1;  
    DO i = 1 to 9 BY 1;  
      valor = n*i;  
      OUTPUT;  
    END;  
  END;  
RUN;
```

Este programa monta um data set que contém a tabuada completa de multiplicação dos números de 1 a 9. Primeiro é montado um laço externo que varia “n” (variável de controle) de 1 a 9. Dentro deste laço existe um outro laço que varia “i” entre 1 e 9. Subordinado a este laço interno estão dois comandos. O primeiro cria a variável “valor”, que recebe o resultado da multiplicação de “n” por “i”. O segundo é o comando OUTPUT para forçar a escrita de uma observação no data set de saída. A sintaxe da estrutura é descrita a seguir:

```
DO var_de_controle = valor_inicial TO valor_final <BY incremento>;
```

O incremento é opcional. Se não for especificado, a variável de controle será incrementada em 1. Valores negativos podem ser utilizados quando for desejado decrementar a variável de controle.

4.18 A Estrutura de Repetição DO-UNTIL

A estrutura DO-UNTIL executa um grupo de comandos repetidas vezes até que uma condição especificada seja VERDADEIRA. A condição é checada **no final** de cada iteração do laço. Veja um exemplo no Programa 39.

```
*Programa 39: DO-UNTIL;  
DATA ExemploUntil;  
  x = 1;  
  DO UNTIL (x < 10);  
    OUTPUT;  
    x = x + 1;  
  END;  
RUN;
```

ExemploUntil

x
1

Quando se usa a estrutura DO-UNTIL, o laço é executado pelo menos uma vez, já que a condição que mantém o laço é testada apenas no final. Por este motivo, o data set produzido pelo programa acima armazena uma observação. A sintaxe do DO-UNTIL é apresentada abaixo:

```
DO UNTIL (condição);  
  Comando1;  
  ...  
  Comandon;  
END;
```

ATENÇÃO!

Infelizmente, a sintaxe da estrutura DO-UNTIL não é muito didática. O funcionamento da estrutura se tornaria muito mais claro se a palavra UNTIL fosse escrita no final do laço, como ocorre em outras linguagens de programação.

4.19 Program Data Vector (PDV)

Para encerrar este capítulo, será introduzido um importante conceito do SAS, denominado **Program Data Vector** (PDV).

O PDV pode ser definido como uma **estrutura lógica** que existe para auxiliar os programadores SAS. Ele pode ser imaginado um array em memória que contém todas as variáveis criadas em um passo DATA. O PDV armazena os dados referentes a **uma única observação** de cada vez (ou seja, a observação que está sendo processada na iteração corrente). É a partir das informações do PDV que os data sets são gravados, sempre observação por observação.

Além das variáveis criadas no passo DATA, o PDV sempre contém duas **variáveis automáticas**: **_N_** e **_ERROR_**. A primeira armazena o número da iteração que está sendo processada no momento, enquanto a segunda indica se um erro ocorreu durante o processamento do passo DATA. Para melhor compreender o funcionamento do PDV, serão apresentados dois exemplos. Ambos são baseados na utilização do arquivo “ARQ_XY.csv”, cujo conteúdo é apresentado a seguir:

```
2,9
5,10
20,3
```

4.19.1 Exemplo 1 – Funcionamento Padrão do PDV

Considere o programa SAS abaixo:

```
*Programa 40: Funcionamento Padrão do PDV;
DATA Dummy;
  INFILE 'C:\CursoSAS\ARQ_XY.csv' DLM=',';
  INPUT X Y;
  Z = X * Y;
RUN;
```

Este programa realiza a leitura das variáveis numéricas X e Y a partir do arquivo “ARQ_XY.csv” e cria a variável derivada $Z = X * Y$, gerando um data set denominado *Dummy*. Observe que o laço automático do passo DATA deste programa realizará três iterações quando executado. Examinaremos a forma pela qual o SAS preenche o PDV com informações em cada uma das iterações.

PDV antes da execução da primeira iteração

Em um programa SAS cada passo é compilado e executado de maneira independente e sequencial. As variáveis são adicionadas ao PDV na ordem em que são vistas pelo compilador. O tipo e o comprimento das variáveis são determinados em tempo de compilação de acordo com a primeira referência que for feita à variável dentro do código. No programa exemplo a primeira variável que aparece no código é “X”, no comando INPUT. Esta variável é do tipo numérico, que por default ocupa 8 bytes com representação em ponto flutuante. Em seguida, o compilador verá as variáveis “Y” e “Z”, nesta ordem. Desta forma, após a compilação do passo DATA e antes do início da execução de sua primeira iteração, o PDV possuirá o formato apresentado na Figura 14. Todas as variáveis estão com o conteúdo missing.

X	Y	Z	_N_	_ERROR_

Figura 14. PDV antes da execução da primeira iteração

Execução da primeira iteração

No início da iteração “_N_” receberá o valor 1. Após a leitura da primeira observação seguir “X” e “Y” e serão preenchidas e o valor de “Z” será calculado e atribuído. No fim da iteração o PDV estará configurado da forma mostrada na Figura 15.

X	Y	Z	_N_	_ERROR_
2	9	18	1	0

Figura 15. PDV no final da primeira iteração

A variável “_ERROR_” conterá o valor 0, pois nenhum erro ocorreu durante o processamento. Ao longo da existência do PDV, o conteúdo da variável é sempre binário: 0 quando não houve em nenhuma iteração e 1 se um ou mais erros ocorrerem.

No fim da execução da primeira iteração, uma observação é adicionada ao data set *Dummy*, contendo os valores armazenados no PDV para as variáveis “X”, “Y” e “Z”. O conteúdo das variáveis automáticas “_ERROR_” e “_N_” **nunca** é copiado para o data set de saída. Se você, por algum motivo, desejar gravar o conteúdo de uma destas variáveis, precisará criar uma variável auxiliar e realizar uma atribuição (ex: NUM = _N_).

Execução da segunda iteração

No início da execução da segunda iteração, o conteúdo das variáveis “X”, “Y” e “Z” é anulado (todas recebem missing). A variável “_N_” é **incrementada automaticamente** e o conteúdo de “_ERROR_” não é mudado. No final desse passo, o conteúdo do PDV é o apresentado na Figura 16. Mais uma observação será gravada em *Dummy*.

X	Y	Z	_N_	_ERROR_
5	10	50	2	0

Figura 16. PDV no final da segunda iteração

Execução da terceira iteração

Funciona de maneira idêntica à segunda iteração. Ao final da execução da terceira iteração o PDV terá o conteúdo mostrado na Figura 17. A última observação será gravada no data set de saída.

X	Y	Z	_N_	_ERROR_
20	3	60	3	0

Figura 17. PDV no final da última iteração

Resultado Final

Após a gravação da última observação – que representa o fim da execução do passo DATA – o PDV é destruído. O data set *Dummy* terá sido inteiramente produzido (seu conteúdo é reproduzido abaixo).

<i>Dummy</i>		
X	Y	Z
2	9	18
5	10	50
20	3	60

O compilador SAS verificará então se existe um novo passo (DATA ou PROC) a ser compilado e executado no Programa 39. Como este programa não possui nenhum outro passo, sua execução será encerrada.

4.19.2 Exemplo 2 – Funcionamento do PDV com Variável RETAIN

O Programa 41, listado a seguir, lê o mesmo arquivo de entrada do exemplo anterior. No entanto, desta vez o data set é criado com a opção DROP e uma variável denominada “ACUMULA_Y” é declarada com o uso do comando RETAIN.

```
*Programa 41: Funcionamento do PDV com variável RETAIN ;
DATA Dummy2 (DROP = Y);
  RETAIN ACUMULA_Y 0;
  INFILE 'C:\CursoSAS\ARQ_XY.csv' DLM=' , ';
  INPUT X Y;
  ACUMULA_Y = ACUMULA_Y + Y;
RUN;
```

Execução da primeira iteração

A Figura 18 apresenta o conteúdo do PDV ao final da primeira iteração. Note que, desta vez, a primeira variável do PDV é “ACUMULA_Y”, pois ela foi a primeira “vista” pelo compilador.

ACUMULA_Y	X	Y	_N_	_ERROR_
9	2	9	1	0

Figura 18. PDV no final da primeira iteração

Como sabemos, o PDV é usado para popular os data sets de saída. Neste exemplo as variáveis “ACUMULA_Y” e “X” serão gravadas, mas “Y” não será levada para o data set DUMMY2, pois faz parte da lista da opção DROP deste data set. Conclui-se então que: **todas as variáveis de um data set de saída sempre estão PDV, mas nem todas as variáveis do PDV são gravadas no data set.**

Execução da segunda iteração

No início da segunda iteração as variáveis “X” e “Y” terão o conteúdo anulado, porém o valor de “ACUMULA_Y” será preservado do passo anterior. Isto ocorre, evidentemente, porque “ACUMULA_Y” foi declarada com o uso do comando RETAIN. Ao final da segunda iteração o conteúdo do PDV é o representado na Figura 19.

ACUMULA_Y	X	Y	_N_	_ERROR_
19	5	10	2	0

Figura 19. PDV no final da segunda iteração

Resultado Final

A seguir é mostrado o conteúdo final do data set *Dummy2* gerado pelo Programa 40. Veja que “Y” não faz parte do mesmo e que “ACUMULA_Y” é a sua primeira variável.

Dummy2

ACUMULA_Y	X
9	2
19	5
22	20

5. Introdução às PROCs do Base SAS

A linguagem SAS dispõe de apenas dois tipos de passos em que blocos de código podem ser dispostos: DATA e PROC. O código de um passo DATA é normalmente utilizado com a finalidade de realizar a integração, carga e transformação de dados, conforme visto nos capítulos anteriores. Por sua vez, o passo PROC é o local onde o programador pode fazer a chamada de uma *procedure* (ou simplesmente PROC) do SAS. O pacote Base SAS oferece mais de 50 diferentes PROCs. Grande parte destina-se a produção de estatísticas básicas e relatórios a partir de dados contidos em data sets. No entanto, também existem as PROCs “utilitárias”, que servem para alterar ou criar data sets.

Este capítulo cobre as principais PROCs do Base SAS. Ele está dividido da seguinte forma. A Seção 5.1 conceitua o passo PROC, através da descrição de suas principais características. As Seções 5.2 e 5.3 apresentam, respectivamente, a PROC SORT e os conceitos de BY Variable, BY Value e BY Group, tópicos que precisam ser preliminarmente conhecidos para a programação de PROCs. Em seguida, as Seções 5.4 a 5.9 introduzem as PROCs para produção de relatórios e as PROCs analíticas do Base SAS, abordando também alguns outros conceitos relevantes sobre estes temas. Por fim, as Seções 5.10 a 5.15 introduzem algumas das PROCs utilitárias do Base SAS.

5.1 O que é uma PROC?

Em alguns livros de SAS, uma metáfora interessante é utilizada para explicar o conceito de PROC. Trata-se da “metáfora do formulário”, que diz que utilizar uma PROC é algo semelhante a preencher um formulário: alguém projetou o formulário e tudo que precisamos fazer é preencher os campos em branco (Figura 20).

PROC xyz	
DATA =	_____
BY	_____
TITLE	_____
FORMAT	_____

Figura 20. Metáfora do formulário

Cada PROC possui o seu próprio conjunto de campos – na realidade, **parâmetros** ou **opções** – a ser preenchido. Alguns parâmetros são obrigatórios e outros opcionais. Felizmente, muitas PROCs compartilham os mesmos parâmetros, o que faz com que o aprendizado se torne facilitado.

Todo passo PROC começa com a palavra-reservada PROC seguida pelo nome da procedure, como FREQ ou PRINT. Os parâmetros são colocados após o nome da procedure.

5.1.1 Parâmetro DATA

Este parâmetro, disponível em praticamente todas as PROCs, informa ao SAS qual data set será utilizado como entrada para a procedure. No exemplo a seguir a PROC PRINT utiliza o data set chamado *Cientes*:

```
PROC PRINT DATA = Clientes;
```

Observações importantes:

- Curiosamente, o parâmetro DATA é **opcional**. Se for omitido o SAS utilizará o último data set criado.
- Ainda em relação a este tema, é importante observar que o último data set criado não significa o último data set utilizado no programa.

5.2 PROC SORT

A PROC SORT tem o propósito de realizar a ordenação de um data set por uma ou mais variáveis. Esta é uma das PROCs mais importantes do SAS, pois as operações de junção (MERGE) e modificação (MODIFY) de data sets, a serem apresentadas no próximo capítulo, requerem data sets ordenados. Além disso, diversas procedures para análise de dados são capazes de produzir estatísticas ou relatórios agrupados pelas variáveis de ordenação. A sintaxe básica da PROC SORT é:

```
PROC SORT DATA = nome_do_dataset;  
BY lista_de_variáveis;
```

As variáveis associadas ao comando BY são chamadas de **BY Variables**. O Programa 42 apresenta um exemplo de utilização da PROC SORT em que um data set com o nome e a idade de diversos clientes é ordenado pela variável “nome”.

```
*Programa 42: PROC SORT;  
DATA Clientes;  
  INPUT nome $ idade;  
  DATALINES;  
  MARIA 33  
  ANA 18  
  PEDRO 45  
  MARIA 33  
  CARLA 29  
  ROBERTO 50  
  AMANDA 28  
;  
RUN;  
  
PROC SORT DATA = Clientes; BY nome;  
RUN;
```

Como resultado, *Clientes* terá as suas observações ordenadas pela variável “nome”, conforme mostrado a seguir.

Clientes

nome	idade
AMANDA	28
ANA	18
CARLA	29
MARIA	33
MARIA	33
PEDRO	45
ROBERTO	50

A opção OUT pode ser usada para especificar a saída para um outro data set. Se você modificar o programa anterior de acordo com o código apresentado abaixo verificará que o data set *Clientes* não estará ordenado por nome e que será gerado um outro data set chamado *Clientes_Ord*, este sim ordenado.

```
PROC SORT DATA = Clientes OUT = Clientes_Ord;
  BY nome;
RUN;
```

A opção NODUPKEY pode ser utilizada para eliminar registros duplicados (o caso de “MARIA”). Já a cláusula DESCENDING pode ser utilizada junto ao comando BY para que a ordenação seja realizada na forma descendente.

```
PROC SORT DATA = Clientes OUT = Clientes_Desc NODUPKEY;
  BY DESCENDING nome;
RUN;
```

Clientes_Desc

nome	idade
ROBERTO	50
PEDRO	45
MARIA	33
CARLA	29
ANA	18
AMANDA	28

Se for preciso ordenar por mais de uma variável, basta especificá-las utilizando o espaço como separador. No exemplo abaixo, a ordenação está sendo realizada prioritariamente pela variável “V1” (ascendente), depois por “V2” (também ascendente) e, por fim, por “V3” na ordem descendente.

```
PROC SORT DATA = Teste;
  BY V1 V2 DESCENDING V3;
RUN;
```

5.3 BY Variables, BY Values e BY Groups

Na PROC SORT, o comando BY informa as variáveis que serão utilizadas para a ordenação. No vocabulário do SAS, estas variáveis são chamadas de **BY Variables**. Além do conceito de BY Variable existem outros dois conceitos extremamente importantes associados ao comando BY:

- **BY Value:** representa o valor armazenado em uma BY Variable.
- **BY Group:** representa um conjunto de observações que compartilham um mesmo BY Value.

A Figura 21 apresenta um exemplo que ilustra os três conceitos de forma prática, em um data set contendo informações de UFs e cidades.

	UF	CIDADE
1	MG	MURIAÉ
2	MG	JUIZ DE FORA
3	MG	BELO HORIZONTE
4	MG	UBERLÂNDIA
5	RJ	RIO DE JANEIRO
6	RJ	NITERÓI
7	RJ	PETRÓPOLIS
8	RJ	SÃO GONÇALO
9	RJ	ITAPERUNA
10	RS	PORTO ALEGRE
11	RS	SÃO LEOPOLDO

Figura 21. BY Variable, BY Groups e BY Values

Observe que o data set está ordenado pela variável “UF”, mas não está ordenado por “CIDADE”. Logo, a BY Variable é “UF”. Esta variável divide o data set em três BY Groups distintos, caracterizados pelos BY Values “MG” (primeiro BY Group), “RJ” (segundo BY Group) e “RS” (terceiro BY Group).

Apesar de simples, os conceitos apresentados são muito importantes dentro da programação SAS. Isto porque vários comandos e PROCs do SAS exploram ou até mesmo funcionam baseados na utilização de BY Variables, BY Values e BY Groups. Alguns exemplos a serem apresentados neste livro são PROC MEANS, PROC REPORT, MERGE, MODIFY, entre outros.

5.4 PROC PRINT

A PROC PRINT é a mais simples e, possivelmente, mais utilizada PROC do SAS. Seu objetivo é produzir um relatório contendo observações de um data set. Nesta seção, serão apresentadas diversas formas de utilização desta PROC a partir do data set *Pessoas*, gerado através do código SAS abaixo:

```
* Programa 43: Cria data set para exemplos da PROC PRINT;
DATA Pessoas;

  INPUT id $ nome $ idade salario uf $;
  DATALINES;
1 Alex      50 5000 RJ
2 Carlos    21 2000 RJ
3 Jenifer   55 3500 SP
4 Glaucia  37 6500 SP
5 Antony    18 2000 RJ
6 Isabel    42 4500 MG
7 Andres    33 3500 SP
;
RUN;
```

Pessoas

id	nome	idade	salario	uf
1	Alex	50	5000	RJ
2	Carlos	21	2000	RJ
3	Jenifer	55	3500	SP
4	Glaucia	37	6500	SP
5	Antony	18	2000	RJ
6	Isabel	42	4500	MG
7	Andres	33	3500	SP

O código abaixo mostra a forma mais básica para gerar um relatório com a PROC PRINT: basta escrever PROC PRINT DATA=*nome_do_dataset*. Com isto, será gerado um relatório contendo todas as observações e todas as variáveis do data set especificado. No EG a saída pode ser gerada como HTML, PDF, RTF e outros formatos.

```
*Gera um relatório básico ;
PROC PRINT DATA=Pessoas;
RUN;
```

Obs.	id	nome	idade	salario	uf
1	1	Alex	50	5000	RJ
2	2	Carlos	21	2000	RJ
3	3	Jenifer	55	3500	SP
4	4	Glaucia	37	6500	SP
5	5	Antony	18	2000	RJ
6	6	Isabel	42	4500	MG
7	7	Andres	33	3500	SP

Veja que além das observações, o relatório também imprime o número da observação (coluna “Obs.”, a primeira do relatório).

A PROC PRINT aceita alguns comandos de configuração que mudam o formato padrão de produção do relatório. Estes parâmetros serão apresentados nas subseções a seguir.

5.4.1 Comando VAR

Possibilita a seleção de um subconjunto de variáveis do data set e a especificação da ordem em que as mesmas deverão aparecer no relatório.

```
*Relatório conterá apenas "nome", "uf" e "salario" e "Obs.";
PROC PRINT DATA=Pessoas;
  VAR nome uf salario;
RUN;
```

Obs.	nome	uf	salario
1	Alex	RJ	5000
2	Carlos	RJ	2000
3	Jenifer	SP	3500
4	Glaucia	SP	6500
5	Antony	RJ	2000
6	Isabel	MG	4500
7	Andres	SP	3500

5.4.2 Comandos NOOBS e N

O comando NOOBS serve para remover a coluna “Obs.” no relatório produzido. Já o comando N faz com que o total de linhas seja impresso no final do relatório. Importante: ao contrário do comando VAR, os comandos NOOBS e N devem ser especificados na mesma linha onde especifica-se o PROC PRINT, conforme mostra o exemplo abaixo.

```
*Relatório conterá apenas "nome", "uf" e "salario";
PROC PRINT DATA=Pessoas NOOBS N;
  VAR nome uf salario;
RUN;
```

nome	uf	salario
Alex	RJ	5000
Carlos	RJ	2000
Jenifer	SP	3500
Glaucia	SP	6500
Antony	RJ	2000
Isabel	MG	4500
Andres	SP	3500

N=7

5.4.3 Comando BY

Produz uma seção separada para cada BY Group. Para que o comando funcione, é preciso que o data set esteja ordenado pela(s) variável(is) indicadas no comando BY, pois apenas assim os BY Group poderão ser identificados pelo SAS. Caso o dataset não esteja ordenado, utilize a PROC SORT antes da PROC PRINT, conforme mostra o exemplo a seguir, onde um relatório com seções separadas para a variável “uf” é produzido.

*Relatório conterá seções separadas para a variável UF;

```
PROC SORT DATA=Pessoas; BY uf;
```

```
PROC PRINT DATA=Pessoas NOOBS;
```

```
  BY uf;
```

```
RUN;
```

uf=MG

id	nome	idade	salario
6	Isabel	42	4500

uf=RJ

id	nome	idade	salario
1	Alex	50	5000
2	Carlos	21	2000
5	Antony	18	2000

uf=SP

id	nome	idade	salario
3	Jenifer	55	3500
4	Glaucia	37	6500
7	Andres	33	3500

5.4.4 Comando SUM

Este comando faz com que uma linha extra, contendo o somatório de uma variável numérica indicada, seja colocada no relatório. Caso utilizado em conjunto com o comando BY, produzirá um subtotal para cada seção e apresentará o total geral na última linha de resultados (seção referente ao último BY Group).

```
*Seções separadas para cada UF e somatório do salário;  
PROC SORT DATA=Pessoas; BY uf;  
  
PROC PRINT DATA=Pessoas NOOBS;  
  BY uf;  
  SUM salario;  
RUN;
```

uf=MG

id	nome	idade	salario
6	Isabel	42	4500

uf=RJ

id	nome	idade	salario
1	Alex	50	5000
2	Carlos	21	2000
5	Antony	18	2000
uf			9000

uf=SP

id	nome	idade	salario
3	Jenifer	55	3500
4	Glaucia	37	6500
7	Andres	33	3500
uf			13500
			27000

5.4.5 Comando WHERE

Permite a especificação de uma condição para remover linhas do relatório. No exemplo a seguir, o comando WHERE é utilizado para que sejam listados apenas os funcionários com salário superior a R\$ 4000,00.

```
*Listar apenas os funcionários com salário acima de R$4000;  
PROC PRINT DATA=Pessoas NOOBS;  
  VAR nome uf salario;  
  WHERE salario > 4000;  
RUN;
```


nome	uf	salario
Alex	RJ	5000
Glaucia	SP	6500
Isabel	MG	4500

5.4.6 Comando ID

Permite com que uma ou mais variáveis sejam utilizadas como chave de identificação da observação.

```
*Observações identificadas por "nome";
PROC PRINT DATA=Pessoas;
  ID nome;
RUN;
```

nome	id	idade	salario	uf
Alex	1	50	5000	RJ
Carlos	2	21	2000	RJ
Jenifer	3	55	3500	SP
Glaucia	4	37	6500	SP
Antony	5	18	2000	RJ
Isabel	6	42	4500	MG
Andres	7	33	3500	SP

5.4.7 TITLE

O comando TITLE faz com que o SAS coloque um título no topo de cada página do relatório de saída. É muito importante esclarecer que TITLE é, na verdade, um **comando global** e não um comando específico da PROC PRINT.

```
*Observações identificadas por "nome";
PROC PRINT DATA=Pessoas;
  TITLE "Listagem de Pessoas";
  ID nome;
RUN;
```

Listagem de Pessoas

Obs.	id	nome	idade	salario	uf
1	1	Alex	50	5000	RJ
2	2	Carlos	21	2000	RJ
3	3	Jenifer	55	3500	SP
4	4	Glaucia	37	6500	SP
5	5	Antony	18	2000	RJ
6	6	Isabel	42	4500	MG
7	7	Andres	33	3500	SP

Quando você não especificar um título para um relatório, o SAS utiliza **automaticamente** o último valor atribuído TITLE na sessão (se nenhuma atribuição foi realizada, TITLE estará com valor missing e, por conseguinte, os relatórios ficarão sem título, conforme ocorreu nos primeiros exemplos). Para anular o valor do TITLE, basta fazer uma chamada ao comando sem especificar nenhuma frase entre aspas:

```
TITLE;
```

IMPORTANTE!

Se o valor do TITLE não for anulado após o seu uso, acabará sendo impresso em todos os relatórios que forem chamados posteriormente a sua definição. Isso significa não apenas os relatórios produzidos pela PROC PRINT, mas também por outras PROCs.

5.5 PROC FREQ

Esta é uma das PROCs analíticas mais utilizadas pelos programadores SAS. Sua função é produzir tabelas de frequências para uma ou mais variáveis de um data set. A sintaxe básica da PROC FREQ é:

```
PROC FREQ DATA = nome_do_dataset;  
  TABLES conjunto_de_variáveis1;  
  ...  
  TABLES conjunto_de_variáveisn;
```

Esta seção apresenta exemplos obtidos a partir do data set *Brinquedos*, definido no Programa 44. Considere que este data set armazena informações a respeito de um conjunto de brinquedos vendidos por uma loja hipotética. As variáveis armazenadas são o “nome do brinquedo”, “nome do fabricante” e “idade mínima recomendada”.

```

* Programa 44: Cria data set para exemplos da PROC FREQ;
DATA Brinquedos;
  LENGTH nome $ 30. fabricante $ 30. idade 3.;

  INFILE DATALINES DLM=',';
  INPUT nome $ fabricante $ idade;
  DATALINES;
Boneca,Gaussian Toys,5
Quebra-cabeças,Gaussian Toys,7
Kit Dinossauros,Probability Kids Inc.,5
Minigame,Gaussian Toys,7
Carrinho,Gaussian Toys,5
Kit Pequeno Estatístico,Probability Kids Inc.,10
Ursinho de Pelúcia,Probability Kids Inc.,5
Livro Super-Heróis,Gaussian Toys,10
;
RUN;

```

Brinquedos

nome	fabricante	idade
Boneca	Gaussian Toys	5
Quebra-cabeças	Gaussian Toys	7
Kit Dinossauros	Probability Kids Inc.	5
Minigame	Gaussian Toys	7
Carrinho	Gaussian Toys	5
Kit Pequeno Estatístico	Probability Kids Inc.	10
Ursinho de Pelúcia	Probability Kids Inc.	5
Livro Super-Heróis	Gaussian Toys	10

5.5.1 Tabelas Unidimensionais

O código a seguir gera uma tabela de frequências para a variável “fabricante”.

```

*Tabela de frequência - análise unidimensional;
PROC FREQ DATA=Brinquedos;
  TABLES fabricante;
RUN;

```

THE FREQ PROCEDURE

fabricante	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Gaussian Toys	5	62.50	5	62.50
Probability Kids Inc.	3	37.50	8	100.00

As colunas apresentadas no resultado gerado são:

- **coluna 1:** valores distintos assumidos pela variável “fabricante” no data set. São apenas dois: “Gaussian Toys” e “Probability Kids Inc.”;
- **coluna 2 – Frequency:** contagem da frequência de cada valor;
- **coluna 3 – Percent:** frequência relativa (contagem da frequência / total de observações não-missing)
- **coluna 4 – Cumulative Frequency:** frequência acumulada;
- **coluna 5 – Cumulative Percent:** frequência relativa acumulada;

A PROC FREQ pode ser utilizada sem o comando TABLES. Nesse caso, o SAS irá gerar uma tabela de frequências para cada variável do data set (no caso do data set Brinquedos, três tabelas seriam geradas).

```
*PROC FREQ sem TABLES;
PROC FREQ DATA=Brinquedos; RUN;
```

5.5.2 Tabelas Bidimensionais

O próximo exemplo gera uma tabela de frequência bidimensional com o cruzamento das variáveis “fabricante” e “idade”.

```
*Tabela de frequência - análise bidimensional;
PROC FREQ DATA=Brinquedos;
  TABLES fabricante * idade;
RUN;
```

THE FREQ PROCEDURE

Table of fabricante by idade						
		idade				
		5	7	10	Total	
fabricante						
Gaussian Toys	Frequency	2	2	1	5	
	Percent	25.00	25.00	12.50	62.50	
	Row Pct	40.00	40.00	20.00		
	Col Pct	50.00	100.00	50.00		
Probability Kids Inc.	Frequency	2	0	1	3	
	Percent	25.00	0.00	12.50	37.50	
	Row Pct	66.67	0.00	33.33		
	Col Pct	50.00	0.00	50.00		
Total	Frequency	4	2	2	8	
	Percent	50.00	25.00	25.00	100.00	

Em cada célula, o SAS imprime a frequência, o percentual, o percentual para a linha e o percentual para a coluna. Observe, por exemplo, a célula referente ao fabricante “Gaussian Toys” e idade igual a 5 anos (Frequency), que contém o valor 2. Este valor refere-se ao total de brinquedos fabricados pela “Gaussian Toys” que são indicados para crianças de 5 anos (se você examinar o data set, verá que são os brinquedos “Boneca” e “Carrinho”). Isto representa 25.00% do total de observações da base de dados, conforme mostra o valor da célula logo abaixo (Percent). Na mesma coluna, linha seguinte, é possível verificar que 40.00% dos brinquedos Fabricados pela “Gaussian Toys” são destinados a crianças de 5 anos (Row Pct.). Por fim, a linha logo abaixo (Col. Pct) mostra que dentre o total de brinquedos para crianças de 5 anos, 50% são fabricados pela “Gaussian Toys”.

5.5.3 Opções do Comando TABLES

Existe uma variedade de opções que podem ser utilizadas junto ao comando TABLES:

- **BY**: computa uma tabela de frequência distinta para cada BY Variable.
- **MISSING**: trata o valor missing como uma categoria (possível valor) da variável de tabulação.
- **NOROW**, **NOCOL**, **NOPERCENT** podem ser utilizadas para suprimir, respectivamente, as estatísticas por linha, coluna e as frequências relativas.
- **OUT**: possibilita a geração de um data set contendo a frequência e a frequência relativa da(s) variável(is) especificadas no comando TABLES.
- **WEIGHT**: permite especificar uma variável que armazene o peso associado a cada observação, caso a mesma exista no data set.

As opções devem ser especificadas na mesma linha do comando TABLES, após o símbolo “ / ” (barra). O exemplo a seguir mostra como a PROC FREQ pode ser utilizada para gerar um data set com a frequência e frequência relativa da variável “fabricante”. A opção NOPERCENT é utilizada para suprimir a frequência relativa do relatório (no entanto, veja que os valores percentuais continuam presentes no data set gerado).

```
*Tabela de frequência - análise unidimensional;
PROC FREQ DATA=Brinquedos;
  TABLES fabricante / NOPERCENT OUT = freq_fabric;
RUN;
```

THE FREQ PROCEDURE

fabricante	Frequency	Cumulative Frequency
Gaussian Toys	5	5
Probability Kids Inc.	3	8

Freq Fabric

fabricante	COUNT	PERCENT
Gaussian Toys	5	62.5
Probability Kids Inc.	3	37.5

5.5.4 Trabalhando com Mais de Duas Variáveis

Os exemplos apresentados mostraram tabelas unidimensionais e bidimensionais. No entanto, também é possível gerar tabelas de frequência envolvendo o cruzamento de mais de 2 variáveis. Por exemplo, para gerar uma tabela de frequências cruzadas para as variáveis “V1”, “V2” e “V3”, basta utilizar: TABLES V1 * V2 * V3;

5.5.5 Gerando Diversas Tabelas de Frequência

É possível gerar várias tabelas de frequência em um único PROC FREQ, bastando para isso utilizar vários comandos TABLE:

```
*Tabelas de frequência;  
PROC FREQ DATA=Brinquedos;  
    TABLES fabricante;  
    TABLES fabricante * idade;  
RUN;
```

5.5.6 Opção ORDER

Esta opção permite com que a tabela de frequência seja ordenada por algum dos seguintes critérios:

- **data**: ordena os valores de acordo com a sua posição no data set.
- **formatted**: ordena os valores de acordo com os seus formatos (veja a seção 4.6).
- **freq**: ordena pela frequência, de forma descendente.

O critério deve ser definido na própria linha PROC FREQ (e não na linha do comando TABLES), como mostra o próximo exemplo:

```
*Tabela ordenada pela frequência - variável "idade";  
PROC FREQ DATA=Brinquedos order = freq;  
    TABLES idade;  
RUN;
```

5.5.7 Opções Estatísticas

Estas opções possibilitam a produção de estatísticas para variáveis categóricas. Um exemplo muito utilizado é **chisq**, que fornece teste chi-squared para independência entre duas variáveis. Para uma referência completa sobre estas opções, consulte o manual “Base SAS procedures guide”:

```
*Uso da opção estatística CHISQ;  
PROC FREQ DATA=Brinquedos;  
    TABLES fabricante * idade /chisq;  
RUN;
```

5.6 Combinando PROC FREQ e PROC FORMAT para Tratar Variáveis Contínuas

Considere o arquivo “IDADES.csv”, que armazena os nomes e idades de diversas pessoas:

```
Ângela, 67
João, 25
Maria, 28
César, 30
Gilberto, 42
Mariano, 25
Rita, 18
Nina, 32
Renan, 45
Ilza, 50
Raul, 29
```

Suponha que deseja-se produzir uma tabela de frequências considerando três faixas etárias: “0-30 anos”, “30-49 anos” e “50 ou mais”. Em uma situação como esta, a PROC FORMAT pode ser utilizada em conjunto com a PROC FREQ, conforme mostra o Programa 45. As explicações são apresentadas no corpo do programa (atenção para a sintaxe exigida pelos comandos).

```
* Programa 45: PROC FORMAT + PROC FREQ;
DATA Clientes;
  INFILE 'C:\CursoSAS\IDADES.csv' DLM=', ';
  INPUT nome $ idade;
RUN;

* Cria o formato "idade_fmt", com o uso da PROC FORMAT;
PROC FORMAT;
  value idade_fmt 0-29 =      '<30'
                  30-49 =    '30-49'
                  50-high =  '>=50';
RUN;

* O formato "idade_fmt" pode então ser aplicado à
* variável "idade" na PROC FREQ;
PROC FREQ;
  TABLE idade;
  FORMAT idade idade_fmt.;
RUN;
```

THE FREQ PROCEDURE

idade	Frequency	Percent	Cumulative Frequency	Cumulative Percent
<30	5	45.45	5	45.45
30-49	4	36.36	9	81.82
>=50	2	18.18	11	100.00

5.7 PROC MEANS

Esta procedure computa a média, variância, desvio padrão e outras estatísticas para variáveis numéricas de um data set. Sua sintaxe mais comum é dada por:

```
PROC MEANS DATA = nome_do_dataset;
  BY lista_de_by_variáveis;
  VAR lista_de_variáveis;
```

Onde:

- **BY** lista_de_by_variáveis: usado para a execução da análise de maneira separada para combinação de valores das variáveis indicadas. O data set precisa estar ordenado para que o comando seja usado.
- **VAR** lista_de_variáveis: especifica as variáveis numéricas que serão analisadas.

Considere uma empresa hipotética que realiza palestras sobre as linguagens Java, Python e SAS. Suponha que a empresa promove de 3 a 5 eventos por mês. As palestras sobre cada linguagem são realizadas no mesmo horário por três professores diferentes. O arquivo “PALESTRAS.txt”, listado a seguir, apresenta a audiência de todas as palestras realizadas nos meses de fevereiro e março de 2017. Neste arquivo, a primeira informação é o dia do evento, seguido da quantidade de pessoas que assistiram às palestras de Java, Python e SAS, respectivamente. Note que no dia 22/03 não houve palestra sobre SAS.

```
01/02/2017  80  78  82
13/02/2017 102  72  96
25/02/2017 100  79 115
08/03/2017  87  87  79
15/03/2017  97  70  78
22/03/2017 109  81  .
30/03/2017  90  78  79
```

Considere que a empresa deseja obter estatísticas básicas a respeito da audiência das palestras em diferentes meses. Para resolver este problema, ela poderia utilizar o código apresentado pelo Programa 46.


```

*Programa 46: PROC MEANS;
DATA PALESTRAS (DROP=DIA NUM_MES);
  INFILE '\\wsasprd01v\ibge\di\codes\bigdata\PALESTRAS.txt';

  LENGTH mes $10;

  INPUT
  @01 dia DDMMYY10.
  @12 java 3.
  @16 python 3.
  @20 sas 3.
  ;

  num_mes = MONTH(dia);
  SELECT (num_mes);
    WHEN (2) MES = 'FEVEREIRO';
    WHEN (3) MES = 'MARÇO';
  END;
RUN;

*ORDENA POR MES;
PROC SORT; BY mes;

*EXECUTA A PROC MEANS;
PROC MEANS DATA=Palestras;
  BY mes;
  VAR java python sas;
  TITLE 'ESTATÍSTICAS DAS PALESTRAS';
RUN;

```

Neste programa, primeiro cria-se um data set chamado *Palestras* que contém as variáveis nome do mês (derivada do dia) e a audiência das palestras de Java, Python e SAS. Em seguida o data set é ordenado pelo mês. Veja que a PROC SORT foi chamada sem a especificação da opção DATA=. Conforme mencionado na Seção 5.1, isto não é necessário, pois desejamos a ordenação pelo ultimo data set criado, que é exatamente o data set *Palestras*.

No ultimo passo do programa, é feita uma chamada para a execução da PROC MEANS (a opção DATA= também não precisava ter sido especificada, já que desejamos que a PROC MEANS analise o último data set gerado). Em seguida, o comando BY é usado para informar ao SAS que desejamos a estatística por mês. Já o comando VAR, indica as variáveis a serem analisadas. Finalizando, a opção TITLE é usada para a especificação de um título. Como resultado da execução do procedimento, o seguinte relatório é produzido.

ESTATÍSTICAS DAS PALESTRAS

The MEANS Procedure

mes=FEVEREIRO

Variable	N	Mean	Std Dev	Minimum	Maximum
java	3	94.0000000	12.1655251	80.0000000	102.0000000
python	3	76.3333333	3.7859389	72.0000000	79.0000000
sas	3	97.6666667	16.5630110	82.0000000	115.0000000

mes=MARÇO

Variable	N	Mean	Std Dev	Minimum	Maximum
java	4	95.7500000	9.7766729	87.0000000	109.0000000
python	4	79.0000000	7.0710678	70.0000000	87.0000000
sas	3	78.6666667	0.5773503	78.0000000	79.0000000

Os comandos BY e VAR são opcionais. Se você não especificar nenhum deles, o SAS computará estatísticas para todas as variáveis numéricas, levando em conta todas as observações. O comando CLASS ser utilizado no lugar do BY para a produção de um relatório mais compacto.

5.7.1 Opções

No exemplo apresentado, a PROC MEANS calculou, para cada variável, o número de valores não nulos, a média, desvio padrão, valor máximo e valor mínimo. Isso ocorreu porque a procedure foi chamada sem nenhuma **opção**. No entanto, o SAS disponibiliza opções que possibilitam a escolha dos índices que serão calculados. Algumas delas são relacionadas a seguir:

- **N**: número de valores não-missing.
- **NMISS**: número de valores missing.
- **SUM**: somatório.
- **MIN**: valor mínimo.
- **MAX**: valor máximo.
- **MEAN**: média .
- **STD**: desvio padrão.
- **VAR**: variância.

A seguir apresenta-se a sintaxe para a especificação das opções.

```
PROC MEANS DATA=PALESTRAS N NMISS SUM;
  BY MES;
  VAR java python sas;
  TITLE 'ESTATÍSTICAS DAS PALESTRAS';
RUN;
```

Também é possível exportar os resultados para um data set, utilizando a sintaxe a seguir:

```

PROC MEANS DATA = nome_do_dataset;
  BY lista_de_by_variáveis;
  VAR lista_de_variáveis;
  OUTPUT OUT = data_set;
  estatística1 (variáveis_origem1) = variáveis_destino1;
  ...
  estatístican (variáveis_origemn) = variáveis_destinon;

```

A seguir um exemplo que demonstra a utilização da funcionalidade:

```

PROC MEANS DATA=Palestras;
  BY MES;
  VAR java python sas;
  OUTPUT OUT = Resultados
  MEAN(java python sas) = M_java M_pyth M_sas
  SUM(java python sas) = S_java S_pyth S_sas;
  TITLE 'ESTATÍSTICAS DAS PALESTRAS';
RUN;

```

mes	_TYPE_	_FREQ	M_java	M_pyth	M_sas	S_java	S_pyth	S_sas
		—						
FEVEREIRO	0	3	94	76.33	97.67	282	229	293
MARÇO	0	4	95.75	79	78.67	383	316	236

Neste exemplo, é solicitada a geração de um data set chamado *Resultados* que conterá o resultado da análise da PROC MEANS. Duas estatísticas serão gravadas. A primeira é a média de audiência dos livros de Java, Python e SAS, com o uso da opção **MEAN**. Os nomes das variáveis destino foram especificados como “M_java”, “M_pyth” e “M_sas”. A outra estatística é a soma total de público para cada palestra, com o uso da opção **SUM**. As variáveis destino foram chamadas de “S_java”, “S_pyth” e “S_sas”. A seguir apresenta-se o data set produzido. As variáveis _TYPE_ e _FREQ_ são colocadas automaticamente no resultado pelo SAS.

5.8 PROC TABULATE

Como o seu nome diz, a PROC TABULATE destina-se a produzir tabulações. Os resultados gerados por essa procedure podem ser obtidos com o uso de outras PROCs, como PRINT, FREQ e MEANS. No entanto, muitos programadores SAS optam por utilizar a PROC TABULATE porque consideram que ela produz tabulações mais “bonitas” a partir de um código mais simples. A sintaxe básica da PROC TABULATE é apresentada a seguir:

```

PROC TABULATE DATA = nome_do_dataset;
  CLASS lista_de_variáveis;
  TABLE var_pagina, var_linha, var_coluna;

```

O comando CLASS informa ao SAS quais variáveis farão parte da tabulação. O comando TABLE, por sua vez, especifica como a tabela será organizada (quais variáveis serão usadas nas dimensões página, linha e coluna – veja que a sintaxe exige que os nomes das variáveis estejam separados por vírgula, de maneira oposta ao que ocorre nas outras PROCs do SAS). É possível utilizar vários comandos TABLE dentro de um mesmo PROC TABULATE, porém cada um deles poderá trabalhar com um máximo de três dimensões.

O Programa 47 ilustra um exemplo de utilização da PROC TABULATE a partir de um data set contendo dados de impressoras. As variáveis deste data set são: “modelo” (número do modelo), “colorida” (1=sim, 0=não), “tipo” (‘laser’ ou ‘ink-jet’) e faixa de preço (‘100-200’, ‘>200’).

```
* Programa 47: PROC TABULATE;
DATA Impressoras;
  INPUT modelo $ cor tipo $ preco $;
  DATALINES;
I001 1 ink-jet >200
I002 0 laser 100-200
I003 0 ink-jet 100-200
I004 1 ink-jet 100-200
I005 0 laser >200
I006 1 ink-jet 100-200
I007 1 laser >200
I008 0 ink-jet 100-200
I009 0 ink-jet 100-200
I010 1 laser >200
  ;
RUN;

PROC TABULATE DATA=Impressoras;
  CLASS cor tipo preco;
  TABLE cor, tipo, preco;
  TITLE 'Total de Impressoras por cor, tipo e preço';
RUN;
```

Total de Impressoras por cor, tipo e preço

	cor 0	
	preco	
	100-200	>200
	N	N
tipo		
ink-jet	3	.
laser	1	1

Total de Impressoras por cor, tipo e preço

cor 1		
	preço	
	100-200	>200
	N	N
tipo		
ink-jet	2	1
laser	.	2

5.8.1 Produzindo Estatísticas Adicionais

Além das variáveis, cada dimensão também pode apresentar estatísticas adicionais, tais como:

- **ALL**: adiciona um total por página, linha ou coluna aos resultados.
- **N**: número de valores não-missing.
- **NMISS**: número de valores missing.
- **SUM**: somatório.
- **MIN**: valor mínimo.
- **MAX**: valor máximo.
- **MEAN**: média aritmética.
- **STDDEV**: desvio padrão.

Veja o exemplo a seguir:

```
PROC TABULATE DATA=Impressoras;
  CLASS cor tipo preço;
  TABLE cor ALL, tipo ALL, preço ALL;
  TITLE 'Total de Impressoras por cor, tipo e preço';
RUN;
```

Total de Impressoras por cor, tipo e preço

cor 0			
	preço		
	100-200	>200	All
	N	N	N
tipo			
ink-jet	3	.	3
laser	1	1	2
All	4	1	5

Total de Impressoras por cor, tipo e preço

	cor 1		
	preço		
	100-200	>200	All
	N	N	N
tipo			
ink-jet	2	1	3
laser	.	2	2
All	2	3	5

5.8.2 Concatenando, Agrupando e Cruzando

Dentro de um TABLE, é possível realizar as seguintes operações com variáveis:

- **Concatenação:** basta separar os nomes das variáveis com espaço ao invés de vírgula. Exemplo: TABLE cor tipo, preço
- **Agrupamento:** coloca-se o nome das variáveis entre parênteses. Exemplo: TABLE (cor tipo), , preço
- **Cruzamento:** implementado com o uso do asterisco. Exemplo: TABLE cor * tipo, preço.

```
PROC TABULATE DATA=Impressoras;
  CLASS cor tipo preço;
  TABLE cor, tipo, preço;
  TITLE 'Total de Impr.: cor concatenado com tipo x preço';
RUN;
```

Total de Impr.: cor concatenado com tipo x preço

	preço	
	100-200	>200
	N	N
cor		
0	4	2
1	2	3
tipo		
ink-jet	5	1
laser	1	3

5.9 PROC CORR

A PROC CORR é utilizada para medir a relação entre duas variáveis. Mais especificamente, ela computa o coeficiente de correlação entre um par de variáveis, medida que expressa numericamente a intensidade e a “direção” (positiva ou negativa) da correlação. Basicamente, quando não há correlação entre as variáveis, elas terão uma correlação de zero. Se elas forem perfeitamente positivamente correlacionadas, a medida retornará 1.0. E se forem perfeitamente negativamente correlacionadas, retornará -1.0. Na maioria das situações práticas, o valor estará compreendido entre -1.0 e 1.0. A sintaxe básica da PROC CORR é dada por:

```
PROC CORR DATA = nome_do_dataset;  
  VAR V1;  
  WITH V2;
```

Onde V1 e V2 são as variáveis as quas deseja-se examinar a correlação. Para exemplificar o uso desta PROC, considere o data set *Animais* que contém a altura e peso de um grupo de animais de uma dada espécie em um zoológico.

<i>Animais</i>		
id	altura	peso
A1	49	81
A2	50	88
A3	53	87
A4	55	99
A5	60	91
A6	55	89
A7	60	95
A8	50	90

O Programa 48 mostra como a PROC CORR pode ser utilizada para medir a correlação entre as variáveis “altura” e “peso”.

```
* Programa 48: PROC CORR;  
DATA Animais;  
  INPUT id $ altura peso;  
  DATALINES;  
  A1 49 81  
  A2 50 88  
  A3 53 87  
  A4 55 99  
  A5 60 91  
  A6 55 89  
  A7 60 95  
  A8 50 90  
  ;  
RUN;
```

```
PROC CORR DATA=Animais;
  VAR altura; WITH peso;
RUN;
```

Ao executar o programa, a PROC CORR exibirá um relatório apresentando estatísticas descritivas das variáveis “altura” e “peso”, e ao final do relatório exibirá o valor do coeficiente de correlação de Pearson entre as mesmas:

...	
Pearson Correlation Coefficients, N=8	
Prob > R under Ho: Rho=0	
	altura
peso	0.6100
	0.0015

O resultado indica que as duas variáveis possuem uma correlação positiva 0.6100. O valor 0.0015 expressa o grau de certeza do valor obtido (significância estatística).

5.10 PROC REPORT

Como o nome diz, esta PROC destina-se a produção de relatórios a partir de data sets SAS. Esta seção apresenta diversos exemplos de utilização da PROC REPORT sobre um data set com dados de países (gerado pelo Programa 48, listado a seguir). O data set possui cinco variáveis: sigla do país, nome do país, continente (‘A’=América ou ‘E’=Europa), extensão territorial (em km2) e população.

```
* Programa 49: PROC REPORT;
DATA Países;
  INPUT sigla $ nome $ continente $ extensao populacao;
  DATALINES;
BRA Brasil A 8515767 204450649
CUB Cuba A 109890 11389562
FRA França E 549190 64395345
HUN Hungria E 93030 9855023
ITA Itália E 301340 59797685
MEX México A 1964380 127017224
NOR Noruega E 323780 5210967
PER Peru A 1285220 31376670
PRT Portugal E 92090 10349803
URY Uruguai A 176220 3431555
;
RUN;
```


5.10.1 Forma Básica

A PROC REPORT é uma procedure bastante sofisticada; no entanto sua sintaxe básica é bem simples:

```
PROC REPORT DATA = nome_do_dataset;  
  COLUMN lista_de_variáveis;
```

O comando COLUMN pode indicar as variáveis que farão parte do relatório e ordem em que as mesmas serão apresentadas. Se omitido, todas as variáveis serão listadas, na ordem em que estão dispostas no data set. Execute o código a seguir para verificar

```
PROC REPORT DATA = Países;  
  COLUMN sigla extensao;  
RUN;
```

sigla	extensao
BRA	8515767
CUB	109890
FRA	549190
HUN	93030
ITA	301340
MEX	1964380
NOR	323780
PER	1285220
PRT	92090
URY	176220

Observe que, nesta forma básica, a PROC REPORT gera um relatório similar ao gerado pela PROC PRINT.

5.10.2 Comando DEFINE

Este comando possibilita a especificação de opções para uma variável. A opção ORDER, por exemplo, pode ser utilizada para ordenar linhas pela variável especificada. No exemplo a seguir, os resultados são ordenados por continente.

```
PROC REPORT DATA = Países;  
  COLUMN sigla continente extensao;  
  DEFINE continente / ORDER;  
RUN;
```

sigla	continente	extensao
BRA	A	8515767
CUB		109890
MEX		1964380
PER		1285220
URY		176220
FRA	E	549190
HUN		93030
ITA		301340
NOR		323780
PRT		92090

Com o uso da opção GROUP, podem ser produzidos relatórios com resultados agregados por uma ou mais variáveis. No exemplo a seguir, o relatório apresenta o somatório da extensão territorial dos países para cada continente.

```
PROC REPORT DATA = Países;
  COLUMN continente extensao;
  DEFINE continente/ GROUP;
RUN;
```

continente	extensão
A	12051477
E	1359430

A opção ACROSS também serve para a geração de resultados agrupados. A diferença é que ela produz uma contagem de frequência ao invés de gerar um somatório.

```
PROC REPORT DATA = Países;
  COLUMN continente;
  DEFINE continente/ ACROSS;
RUN;
```

continente	
A	E
5	5

O comando DEFINE também permite a modificação do cabeçalho de uma coluna, conforme mostra o exemplo a seguir.

```

PROC REPORT DATA = Países;
  COLUMN sigla continente extensao;
  DEFINE sigla / 'sigla.país';
  DEFINE continente / ORDER 'cont.país';
  DEFINE extensao / 'ext.terr.';
RUN;

```

sigla.país	cont.país	ext.terr
BRA	A	8515767
CUB		109890
MEX		1964380
PER		1285220
URY		176220
FRA	E	549190
HUN		93030
ITA		301340
NOR		323780
PRT		92090

5.10.2 Comando BREAK

Este comando é utilizado para a definição de quebras de seção em um relatório. O comando funciona adicionando uma quebra para cada valor único de uma variável especificada. É possível também definir uma ou mais “opções de quebra”:

- **OL**: desenha uma linha após a quebra;
- **UL**: desenha uma linha antes da quebra;
- **PAGE**: pula uma página;
- **SKIP**: insere uma linha em branco.
- **SUMMARIZE**: insere uma linha com o somatório das variáveis numéricas.

```

PROC REPORT DATA = Países;
  COLUMN continente sigla nome extensao populacao;
  DEFINE continente / ORDER;
  BREAK AFTER continente / OL SUMMARIZE;
RUN;

```

continente	sigla	nome	extensão	população
A	BRA	Brasil	8515767	204450649
	CUB	Cuba	109890	11389562
	MEX	México	1964380	127017224
	PER	Peru	1285220	31376670
	URY	Uruguai	176220	3431555
<i>A</i>			<i>12051477</i>	<i>377665660</i>
E	FRA	França	549190	64395345
	HUN	Hungria	93030	9855023
	ITA	Itália	301340	59797685
	NOR	Noruega	323780	5210967
	PRT	Portugal	92090	10349803
<i>E</i>			<i>1359430</i>	<i>149608823</i>

Importante: o comando BREAK só é permitido sobre variáveis definidas (DEFINE) com a opção ORDER.

5.11 Introdução à Tecnologia ODS – Output Delivery System

Como o seu próprio nome diz, ODS (*output delivery system* – algo como “sistema de entrega de saída”) é uma tecnologia incorporada ao sistema SAS utilizada para definir o formato de saída dos relatórios produzidos em um programa. Com o uso dessa tecnologia, torna-se possível exportar os resultados e tabulações produzidos pelas PROCs apresentadas neste capítulo para diferentes formatos, tais como PDF, RTF, HTML, PS e XML.

O Programa 50 mostra como exportar os resultados de uma PROC PRINT para um relatório PDF, chamado “Relatorio_Pessoas.pdf”

* Programa 50: Cria data set para exemplos da PROC PRINT;
DATA Pessoas;

```

INPUT id $ nome $ idade salario uf $;
DATALINES;
1 Alex      50 5000 RJ
2 Carlos    21 2000 RJ
3 Jenifer   55 3500 SP
4 Glaucia  37 6500 SP
5 Antony    18 2000 RJ
6 Isabel    42 4500 MG
7 Andres    33 3500 SP
;
RUN;
```

```
ods pdf file='C:\CursoSAS\Relatorio_Pessoas.pdf';

PROC PRINT DATA=Pessoas;
RUN;

ods _all_ close;
```

A receita para a utilização do ODS é a seguinte. Antes da chamada da PROC (fora do passo DATA e fora do passo PROC), você deve definir o formato e o nome do arquivo de saída, utilizando a seguinte sintaxe:

```
ods formato file=nome_do_arquivo_de_saída;
```

Onde *formato* é utilizado para indicar o formato do arquivo (pdf, html, rtf, etc.). Após a chamada da PROC, você deverá idealmente encerrar a saída ODS utilizando:

```
ods _all_ close;
```

5.12 PROC TRANSPOSE

A PROC TRANSPOSE é utilizada para “girar” um data set, transformando observações em variáveis e variáveis em observações. Considere, por exemplo, o data set *Profs*, que armazena o nome de três professores e os nomes das matérias lecionadas pelos mesmos. Note que existem duas observações associadas ao primeiro professor e três associadas aos dois outros professores. Observe ainda que o data set está ordenado pelo nome do professor (primeiro critério) e nome da matéria (segundo critério).

Profs

nome	materia
Ana	Programação Python
Ana	Programação SAS
Jeff	Big Data
Jeff	Estatística
Jeff	SQL
Tomas	Big Data
Tomas	Estatística
Tomas	Programação SAS

Esta base está no chamado “formato vertical”: contém poucas variáveis (duas) e uma quantidade bem maior de observações (oito). Com o uso da PROC TRANSPOSE, é possível passar essa base para o “formato horizontal”:

Profs_Horizontal

nome	mat1	mat2	mat3
Ana	Programação Python	Programação SAS	
Jeff	Big Data	Estatística	SQL
Tomas	Big Data	Estatística	Programação SAS

Agora a base possui três observações, uma para cada professor, e quatro variáveis, “nome” (nome do professor), “mat1”, “mat2” e “mat3”, para armazenar separadamente o nome de cada matéria lecionada pelo professor (como a primeira professora leciona apenas uma matéria, o valor de “mat3” correspondente a sua observação fica missing). O Programa 51 mostra como essa transformação é implementada com o uso da PROC TRANSPOSE.

```
* Programa 51: PROC TRANSPOSE;
DATA Profs;
  LENGTH nome $ 6. materia $ 20.;
  INFILE DATALINES DLM=',';
  INPUT nome $ materia $;
  DATALINES;
  Ana, Programação Python
  Ana, Programação SAS
  Jeff, Big Data
  Jeff, SQL
  Jeff, Estatística
  Tomas, Big Data
  Tomas, Estatística
  Tomas, Programação SAS
  ;
RUN;

PROC SORT DATA=Profs; BY nome materia;

PROC TRANSPOSE DATA=Profs OUT=ProfsHorizontal PREFIX=mat;
  VAR materia;
  BY nome;
RUN;
```

ProfsHorizontal

nome	_NAME_	mat1	mat2	mat3
Ana	materia	Programação Python	Programação SAS	
Jeff	materia	Big Data	Estatística	SQL
Tomas	materia	Big Data	Estatística	Programação SAS

O programa funciona da seguinte maneira:

- O comando VAR indica quais as variáveis a serem transpostas. No exemplo, apenas a variável “materia”.
- O comando BY serve para indicar as BY Variables, ou seja, as variáveis de grupo que serão mantidas após a transposição. Neste caso, a variável “nome”. O data set

original deverá estar ordenado por esta variável ou ocorrerá erro na execução do programa.

- Na opção DATA, especifica-se o nome do data set original e em OUT o nome do novo data set.
- Por fim, a opção PREFIX é utilizada para definir qual será o nome da variável transposta no novo data set. No exemplo, ela foi definida como “mat”, gerando as então as variáveis “mat1”, “mat2” e “mat3” no data set horizontal. Se esta opção for omitida, o SAS irá gerar as variáveis como COL1, COL2, etc.

Veja que o dataset *ProfsHorizontal* foi gerado com uma variável extra denominada “_NAME_”, contendo o nome da variável transposta (“materia”) em todas as observações. Para eliminar esta variável, basta fazer uso da data set option DROP junto à opção OPTION:

```
PROC TRANSPOSE DATA=Profs OUT=ProfsHorizontal (DROP=_NAME_)
                PREFIX=mat;
```

5.12 PROC APPEND

Esta PROC é utilizada para adicionar as observações de um data set SAS no final de outro data set. O data set que **recebe** os dados é chamado de **base**. É necessário que os dois data sets tenham a mesma definição, ou seja, as mesmas variáveis.

```
* Programa 52a: PROC APPEND;
```

```
* CRIA O DATASET "Vendas1" COM AS VENDAS DE JANEIRO;
```

```
DATA Vendas1;
  INPUT produto $ mes $ quant;
  DATALINES;
  CAFÉ JAN 1200
  SUCO JAN 350
  CHÁ JAN 245
  ;
RUN;
```

```
* CRIA DATASET "Vendas2" COM AS VENDAS DE FEVEREIRO;
```

```
DATA Vendas2;
  INPUT produto $ mes $ quant;
  DATALINES;
  CAFÉ FEV 1512
  SUCO FEV 487
  CHÁ FEV 300
  GUARANÁ FEV 408
  ;
RUN;
```

```
* CONCATENA AS OBSERVAÇÕES DE "Vendas2" NO FINAL DE "Vendas1"
  (ou seja: Vendas1 = Vendas1 União Vendas2);
```

```
PROC APPEND BASE=Vendas1 DATA=Vendas2;
RUN;
```

Ao final da execução, *Vendas1* conterá as seguintes observações:

<i>Vendas1</i>		
produto	mes	quant
CAFÉ	JAN	1200
SUCO	JAN	350
CHÁ	JAN	245
CAFÉ	FEV	1512
SUCO	FEV	487
CHÁ	FEV	300
GUARANÁ	FEV	408

5.13.1 Concatenando Diversos Data Set

Infelizmente só é possível realizar a concatenação de “2 em 2” (usando dois datasets). Isto quer dizer que se for preciso concatenar 3, 4 ou mais data sets, torna-se necessário escrever várias PROC's APPEND no código, conforme mostra o Programa 52b.

```
* Programa 52b: PROC APPEND envolvendo 3 datasets;

* CRIA O DATASET "Vendas1" COM AS VENDAS DE JANEIRO;
DATA Vendas1;
  INPUT produto $ mes $ quant;
  DATALINES;
  CAFÉ JAN 1200
  SUCO JAN 350
  CHÁ JAN 245
  ;
RUN;

* CRIA DATASET "Vendas2" COM AS VENDAS DE FEVEREIRO;
DATA Vendas2;
  INPUT produto $ mes $ quant;
  DATALINES;
  CAFÉ FEV 1512
  SUCO FEV 487
  CHÁ FEV 300
  GUARANÁ FEV 408
  ;
RUN;

* CRIA DATASET "Vendas3" COM AS VENDAS DE MARÇO;
DATA Vendas3;
  INPUT PRODUTO $ MES $ QUANT;
  DATALINES;
  CAFÉ MAR 1499
  SUCO MAR 490
```



```

CHÁ MAR 287
GUARANÁ MAR 371
;
RUN;

* CONCATENA "Vendas2" NO FINAL DE "Vendas1";
PROC APPEND BASE=Vendas1 DATA=Vendas2;
RUN;

* AGORA CONCATENA "Vendas3" NO FINAL DE "Vendas1";
PROC APPEND BASE=Vendas1 DATA=Vendas3;
RUN;

```

Vendas1

produto	mes	quant
CAFÉ	JAN	1200
SUCO	JAN	350
CHÁ	JAN	245
CAFÉ	FEV	1512
SUCO	FEV	487
CHÁ	FEV	300
GUARANÁ	FEV	408
CAFÉ	MAR	1499
SUCO	MAR	490
CHÁ	MAR	287
GUARANÁ	MAR	371

5.13.2 Concatenação com o Comando SET

Muito comumente os programadores SAS utilizam a concatenação via comando SET (mais detalhes sobre este comando serão apresentados no próximo capítulo). Este método oferece duas vantagens básicas: (i) é possível especificar um nome para o data set de saída; e (ii) permite a concatenação de mais de 2 data sets. O código a seguir pode ser usado como alternativa ao exemplo anterior para realizar a concatenação dos dados de Janeiro, Fevereiro e Março.

```

* GERA O DATA SET "Vendas" COM O CONTEÚDO DE "Vendas1",
* "Vendas2" E "Vendas3" CONCATENADO;
DATA Vendas;
    SET Vendas1 Vendas2 Vendas3;
RUN;

```

Entretanto, a concatenação com o SET apresenta uma desvantagem em relação à PROC APPEND: quando se usa o comando SET, o SAS processa todas as observações em todos os data sets para produzir o data set final. Já a PROC APPEND não faz isso! Ela não

faz uma varredura completa no data set BASE, apenas adiciona diretamente as observações do outro data set nele. Ou seja, as observações do data set BASE não sofrem nenhum tipo de processamento. Por isso, o uso da PROC APPEND costuma ser vantajoso sem situações onde o data set BASE é muito volumoso.

5.14 PROC COMPARE

Esta procedure **compara o conteúdo** de dois data sets, variável por variável. A comparação é baseada no uso de uma variável chave (na realidade, uma BY Variable) – por isso os dois data sets a serem comparados precisam estar ordenados por esta variável. Como resultado do processo de comparação, o SAS gera um relatório que conterá um resumo das divergências encontradas e listará as observações divergentes (ex: observações que possuem a mesma chave, porém conteúdo diferente ou uma observação que esteja presente em um dos arquivos, mas não esteja presente no outro).

Como exemplo, considere as duas bases de dados apresentadas abaixo. Suponha que a primeira contém uma relação de empresas que fizeram parte de uma pesquisa hipotética denominada “PESQEMP” no ano de 2016. De maneira análoga, considere que a segunda base de dados contém a relação de empresas investigadas pela mesma pesquisa no ano de 2017. Para ambos os arquivos, existem três variáveis: raiz do CNPJ da empresa (com oito dígitos – chave de identificação), razão social e pessoal ocupado (número de funcionários, abreviaremos como “po”).

Arquivo “PESQEMP2016.txt”:

11111111	Empresa XYZ	115
44444444	Organizacao Alfa	10
55555555	Industria Teste	1200
77777777	Padaria do Joaquim	14
99999999	Colégio Barra	17

Arquivo “PESQEMP2017.txt”:

11111111	Empresa XYZ	115
44444444	Organizacao Beta	10
55555555	Industria Teste	1200
66666666	Bolos Caseiros LTDA	9
99999999	Colégio Barra	30

Veja que de 2016 para 2017 as empresas de CNPJ “11111111” e “55555555” continuaram com as mesmas informações. Já a empresa de CNPJ “44444444” mudou de razão social, porém mantendo o mesmo “po”. A empresa de CNPJ “77777777” fez parte apenas da pesquisa de 2016, enquanto a empresa de CNPJ “66666666” fez parte apenas da pesquisa de 2017. Por fim, a empresa de CNPJ “99999999” registrou um aumento de “po” de um ano para o outro (de 17 para 30).

O Programa 53 realiza a importação dos dois arquivos para dois data sets distintos. Em seguida os datasets são ordenados por CNPJ e, por fim, aplica-se, a PROC COMPARE para a obtenção do relatório de divergências entre os data sets. A sintaxe básica da PROC COMPARE é bem simples: basta escolher um data set como **base** e o outro como

compare. Deve-se também indicar a variável chave com o uso da opção “**id**” (em nosso caso é o “**cnpj**”).

```
*PROGRAMA 53: PROC COMPARE;

* importa os datasets;
DATA Pesq2016;
  INFILE 'C:\CursoSAS\PESQEMP2016.txt';
  INPUT
    @1 cnpj $8.
    @9 razao $20.
    @30 po 4.;
RUN;

DATA Pesq2017;
  INFILE 'C:\CursoSAS\PESQEMP2017.txt';
  INPUT
    @1 cnpj $8.
    @9 razao $20.
    @30 po 4.;
RUN;

* ordena pela chave;

* OBS: embora em nosso exemplo, os arquivos de entrada
* já estejam ordenados, mantivemos o PROC SORT
* apenas por questões didáticas;
PROC SORT DATA=pesq2016;
  BY cnpj;

PROC SORT DATA=pesq2017;
  BY cnpj;

* realiza a comparação com o PROC COMPARE;
PROC COMPARE BASE=pesq2016
  COMPARE=pesq2017;
  ID cnpj;
RUN;
```

O relatório gerado pela procedure mostrará todas as divergências do data set base em relação ao data set de comparação. Este relatório é dividido em diversas seções, que são apresentadas e brevemente comentadas a seguir.

- **Data Set Summary:** esta seção indica os nomes dos data sets que foram comparados e a data de criação, número de variáveis e total de observações de cada um.

<p>The COMPARE Procedure Comparison of WORK.PESQ2016 with WORK.PESQ2017 (Method=EXACT)</p> <p>Data Set Summary</p>
--

Dataset	Created	Modified	NVar	NObs
WORK.PESQ2016	05APR17:17:11:03	05APR17:17:11:03	3	5
WORK.PESQ2017	05APR17:17:11:03	05APR17:17:11:03	3	5

- **Variables Summary:** nesta seção, o relatório indica o total de variáveis em comum e quantidade de variáveis chave (id).

Variables Summary				
Number of Variables in Common: 3.				
Number of ID Variables: 1.				

- **Observation Summary:** esta é uma das seções mais interessantes do relatório, onde um “resumão” sobre as divergências entre as observações de cada data set é apresentado.

Observation Summary			
Observation	Base	Compare	ID
First Obs	1	1	cnpj=11111111
First Unequal	2	2	cnpj=44444444
Last Unequal	5	5	cnpj=99999999
Last Obs	5	5	cnpj=99999999

Number of Observations in Common: 4.

Number of Observations in WORK.PESQ2016 but not in WORK.PESQ2017: 1.

Number of Observations in WORK.PESQ2017 but not in WORK.PESQ2016: 1.

Total Number of Observations Read from WORK.PESQ2016: 5.

Total Number of Observations Read from WORK.PESQ2017: 5.

Number of Observations with Some Compared Variables Unequal: 2.

Number of Observations with All Compared Variables Equal: 2.

- **Values Comparison Summary:** outra seção muito interessante, que exibe um resumo sobre as diferenças encontradas para cada variável.

Values Comparison Summary					
Number of Variables Compared with All Observations Equal: 0.					
Number of Variables Compared with Some Observations Unequal: 2.					
Total Number of Values which Compare Unequal: 2.					
Maximum Difference: 13.					
All Variables Compared have Unequal Values					
Variable	Type	Len	Ndif	MaxDif	
razao	CHAR	20	1		
po	NUM	8	1	13.000	

- **Parte Final:** no final do relatório, o SAS realiza uma comparação dos data sets registro por registro, listando todos os casos onde algum tipo de divergência foi encontrada. Caso exista um número muito grande de divergências, o SAS mostrará apenas os primeiros casos.

The COMPARE Procedure					
Comparison of WORK.PESQ2016 with WORK.PESQ2017					
(Method=EXACT)					
Value Comparison Results for Variables					
<hr/>					
cnpj		Base Value	Compare Value		
		razao	razao		
<hr/>					
44444444		Organizacao Alfa	Organizacao Beta		
<hr/>					
<hr/>					
cnpj		Base po	Compare po	Diff.	% Diff
<hr/>					
99999999		17.0000	30.0000	13.0000	76.4706
<hr/>					

Em problemas que envolvem processos de carga ou migração de dados, a PROC COMPARE costuma ser extremamente útil, pois nos permite comparar de forma simples e rápida os dados dos arquivos fonte com os que foram migrados para os arquivos destino. Nesta seção apresentamos apenas a utilização básica da procedure (que já é super útil e suficiente para a maior parte das situações práticas). Para obter informações sobre o uso avançado e customização da PROC COMPARE, consulte o documento “Base SAS 9.4 Procedures Guide”.

5.15 PROC CONTENTS

Em sua forma usual, a PROC CONTENTS é utilizada para produzir um relatório contendo metadados associados a um data set (informações relacionadas a sua definição) e também um resumo sobre conteúdo correntemente armazenado no mesmo. Ela é especialmente útil em situações onde você precisa trabalhar com um data set cuja definição e/ou conteúdo lhe é desconhecido, uma vez que ela fornece um resumo instantâneo contendo essas informações. Alguns exemplos de dados extraídos pela PROC CONTENTS são:

- Nome e data de criação do data set;
- Número de observações;
- Número de variáveis;
- Data da última atualização;
- Nome, tipo, tamanho, formato e informat associado a cada variável;

O Programa 54 demonstra o uso da PROC CONTENTS (Obs.: o programa utiliza o arquivo “PESQEMP2017.txt”, apresentado na seção anterior).

```

*PROGRAMA 54: PROC CONTENTS;
DATA Pesq2017;
  INFILE 'C:\CursoSAS\PESQEMP2017.txt';
  INPUT
    @1 cnpj $8.
    @9 razao $20.
    @30 po 4.;
RUN;

PROC CONTENTS DATA=pesq2017;
RUN;

```

The CONTENTS Procedure

Data Set Name	WORK.PESQ2017	Observations	5
Member Type	DATA	Variables	3
Engine	V9	Indexes	0
Created	Ter, 13 de Jun	Observation Length	40
Last Modified	Ter, 13 de Jun	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS64		
Encoding	wlatin1		

Engine/Host Dependent Information	
Data Set Page Size	4096
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	101
Obs in First Data Page	5
Number of Data Set Repairs	0
Filename	X:\WORK\user\pesq2017.sas7bdat

Alphabetic List of Variables and Attributes				
	#	Variable	Type	Len
	1	cnpj	Char	8
	3	po	Num	8
	2	razao	Char	20

6. Programação Avançada

Este capítulo aborda as técnicas avançadas de programação na Linguagem SAS. As seções abordam cinco assuntos distintos:

- Combinação de data sets. Os tópicos introdutórios sobre o assunto são apresentados nas Seções 6.1 a 6.3, enquanto a operação de combinação propriamente dita é coberta nas Seções 6.4 a 6.9.
- Modificação de data sets. Tópico abordado nas Seções 6.10 e 6.11.
- Arrays. Tema da Seção 6.12.
- Macros e macro variáveis. Seções 6.13 a 6.18.
- PROC SQL: Seções 6.19 a 6.25.

6.1 SET

O comando SET permite com que você leia um data set SAS armazenado em alguma library (pode ser a WORK ou qualquer outra), podendo modificá-lo de diversas formas: adicionando ou eliminando variáveis, modificando observações, etc. A sintaxe básica deste comando é a seguinte:

```
DATA novo_data_set;  
  SET data_set_existente;
```

Para demonstrar a utilização do comando, considere o seguinte exemplo. Suponha que exista uma pasta de rede denominada “\\X01\bases”, que pode ser acessada através do sistema SAS. Considere que esta pasta contenha um data set SAS chamado “FILMES.sas7sdat” com o conteúdo apresentado a seguir:

FILMES			
titulo	ano	pais	duracao
Monty Python em Busca do Cálice Sagrado	1975	GBR	91
Noel: Poeta da Vila	2006	BRA	99
As Pontes de Madison	1995	USA	135
Meu Melhor Amigo	2006	FRA	94
La La Land	2016	USA	128
Intocáveis	2011	FRA	112
O Filho da Noiva	2001	ARG	.
Vidas Secas	1963	BRA	103
Hair	1979	USA	121
Edukators	2004	DEU	127

O código a seguir, mostra como esse data set pode ser gerado e persistido no compartilhamento “\\X001\bases” (mude o nome do compartilhamento ou da pasta para algum local onde a sua conta tenha direito de escrita para poder rodar o exemplo).

```
* Programa 55a: Geracao do Data Set FILMES
libname libBases '\\X001\bases';
```

```
DATA libBases.FILMES;
  LENGTH titulo $ 50;
  INFILE DATALINES DLM=',';
  INPUT titulo $ ano pais $ duracao;

  DATALINES;
  Monty Python em Busca do Cálice Sagrado,1975,GBR,91
  Noel: Poeta da Vila,2006,BRA,99
  As Pontes de Madison,1995,USA,135
  Meu Melhor Amigo,2006,FRA,94
  La La Land,2016,USA,128
  Intocáveis,2011,FRA,112
  O Filho da Noiva,2001,ARG,.
  Vidas Secas,1963,BRA,103
  Hair,1979,USA,121
  Edukators,2004,DEU,127
  ;
RUN;
```

O programa a seguir cria um novo data set chamado *Filmes2*, desta vez na WORK do SAS, a partir dos dados do data set *FILMES*. O data set conterá apenas o título do filme e uma variável derivada, denominada “duracao_horas”, armazenando a duração do filme em horas, ao invés de minutos.

```
*Programa 55b: SET;
libname libBases 'X:\\bases';

DATA Filmes2 (KEEP=titulo duracao_horas);
  SET libBases.FILMES;
  duracao_horas = duracao / 60;
RUN;
```

<i>Filmes2</i>	
titulo	duracao_horas
Monty Python em Busca do Cálice Sagrado	1.5166666667
Noel: Poeta da Vila	1.65
As Pontes de Madison	2.25
Meu Melhor Amigo	1.5666666667
La La Land	2.1333333333
Intocáveis	1.8666666667
O Filho da Noiva	.
Vidas Secas	1.7166666667
Hair	2.0166666667
Edukators	2.1166666667

Em resumo: o comando SET permite com que, de forma simples, um novo data set seja criado a partir das informações presentes em outro data set.

IMPORTANTE!

O número de iterações executadas pelo laço automático de um passo DATA será igual ao número de observações do data set indicado no comando SET.

6.2 Comandos *KEEP*, *DROP*, *RENAME* e *WHERE*

Os comandos *KEEP*, *DROP*, *RENAME* e *WHERE* **não devem ser confundidos** com as **opções** de data set que possuem o mesmo nome, que foram apresentadas na Seção 4.7. Estes comandos aparecem dentro do código do passo DATA; já as opções de data set aparecem entre parênteses ao lado dos comandos SET e DATA. Para demonstrar a forma de utilização destes comandos, considere novamente o arquivo “NOTAS.txt”, apresentado no Capítulo 4. Recorde que este arquivo contém as matrículas de quatro alunos e as notas obtidas por cada um em duas diferentes provas.

```
M0012017 9.8 9.5
M0022017 5.3 4.1
M0032017 2.5 8.0
M0042017 7.5 7.5
```

O Programa 56 mostra a utilização do comando *DROP*, para gerar um data set contendo apenas as variáveis “matricula” e “media” (variável derivada). O uso dos comandos *KEEP*, *RENAME* e *WHERE* é feito de forma similar.

```
*Programa 56: comando DROP;
*O data set gerado conterá apenas as variáveis
"matricula" e "media";
DATA MediasFinais;
  INFILE 'C:\CursoSAS\NOTAS.TXT';
  INPUT matricula $ 1-8 nota1 nota2;

  * CALCULA A MÉDIA;
  media = (nota1 + nota2) / 2;

  DROP nota1 nota2; *uso do comando DROP eliminando 2 variáveis;
RUN;
```

MediasFinais

matricula	media
M0012017	9.65
M0022017	4.7
M0032017	5.25
M0042017	7.5

Embora os comandos KEEP, DROP, RENAME e WHERE e as opções de mesmo nome tenham funções praticamente idênticas, **existem diferenças com relação ao modo como o SAS executa os comandos e as opções**. Este assunto será discutido em seções posteriores deste capítulo.

6.3 Variáveis Temporárias FIRST e LAST

As **variáveis temporárias FIRST e LAST** são variáveis numéricas que podem ser adicionadas e gerenciadas automaticamente no PDV caso o programador assim deseje. Da mesma forma que ocorre com qualquer tipo de variável automática (por exemplo, `_N_` e `_ERROR_`), elas **não** são transferidas para o data set de saída. Estas variáveis **só podem ser utilizadas em data sets ordenados** e funcionam da seguinte forma:

- **FIRST**: recebe automaticamente o valor **VERDADEIRO** (1) se a observação correntemente processada no passo DATA representa o **primeiro registro de um BY Group**. Caso contrário armazena o valor FALSO (0).
- **LAST**: recebe automaticamente o valor **VERDADEIRO** (1) se a observação correntemente processada no passo DATA representa o **último registro de um BY Group**. Caso contrário armazena o valor FALSO (0).

As variáveis temporárias costumam facilitar em muito o desenvolvimento de certas rotinas. O exemplo abaixo demonstra esta situação, utilizando o arquivo “CAPITAIS.csv”. Este arquivo contém a relação das capitais dos estados brasileiros localizados nas regiões Sul, Sudeste e Norte. Para cada capital, indica-se o seu nome (colunas 1-14), região (colunas 16-23) e a população estimada de acordo com o IBGE⁵ (colunas 24-31).

Belém	Sudeste	1446042
Belo Horizonte	Sudeste	2513451
Boa Vista	Norte	326419
Curitiba	Sul	1893977
Florianópolis	Sul	477798
Macapá	Norte	465495
Manaus	Norte	2094391
Palmas	Norte	279856
Porto Alegre	Sul	1481019
Porto Velho	Norte	511219
Rio Branco	Norte	377057
São Paulo	Sudeste	12038175
Rio de Janeiro	Sudeste	6498837
Vitória	Sudeste	359555

No Programa 57 demonstra-se como FIRST e LAST podem ser utilizadas para a criação de um data set contendo o nome da região e o somatório da população de suas capitais. A explicação do programa é realizada através de comentários no corpo do mesmo.

⁵<https://cidades.ibge.gov.br/>

```

*Programa 57: variáveis temporárias FIRST e LAST;
DATA Capitais;
  INFILE 'C:\CursoSAS\CAPITAIS.txt';
  INPUT
    @001 nome    $14.
    @016 regioao $8.
    @024 populacao 8.
  ;
RUN;

*ordena o data set por regioao ("regiao" é a BY Variable!);
PROC SORT DATA=Capitais; BY regioao;

*o data set PopulacaoRegiao conterá a populacao total por região;
DATA PopulacaoRegiao(KEEP = regioao pop_total);
  SET Capitais; BY regioao;
  RETAIN pop_total;

  *se estou na primeira capital da região,
  inicializo o valor da populacao total;
  IF FIRST.regiao THEN pop_total = 0;

  *acumula a soma das rendas;
  pop_total = pop_total + populacao;

  *se estou na última capital da regioao,
  gravo a populacao total em PopulacaoRegiao;
  IF LAST.regiao THEN OUTPUT;
RUN;

```

PopulacaoRegiao

regiao	pop_total
Norte	4054437
Sudeste	22856060
Sul	3852794

IMPORTANTE!

Para que as variáveis FIRST e LAST possam ser usadas dentro de um passo DATA é preciso que duas condições sejam atendidas:

1. Os dados de entrada devem ser ordenados por alguma BY Variable.
2. Deve-se realizar uma chamada ao comando BY dentro do passo DATA, onde a BY Variable será informada.

6.4 MERGE - Introdução

MERGE é o comando utilizado no passo DATA para **combinar** o conteúdo de dois data sets. Neste tipo de operação, normalmente você terá dois data sets SAS com dados relacionados que você desejará combinar em um novo data set (ex: pedido com produto, empresa com filial, pessoa com domicílio, paciente com prontuário, funcionário com projeto, etc.). Os data sets combinados possuirão algum tipo de **relacionamento** efetivado por **uma ou mais variáveis em comum** (variáveis de ligação) A Figura 22 resume os passos necessários para a execução do MERGE entre data sets.

1. **Ordenar** os *data sets* pela **variável em comum**, caso eles não estejam ordenados. Para isto, utilize a **PROC SORT**.
2. **Criar um novo passo DATA** que se encarregará de produzir o *data set* que armazenará o resultado do MERGE.
3. **Incluir o comando MERGE** nesse passo, especificando a relação de *data sets* que serão combinados.
4. **Incluir o comando BY** para informar ao SAS quais são as variáveis em comum.

Figura 22. Passos para a implementação de uma rotina de MERGE em um Programa SAS

A sintaxe básica do comando MERGE é apresentada abaixo.

```
DATA ds_saída;  
  MERGE d1 d2;  
  BY variáveis_em_comum;
```

As subseções a seguir apresentarão diversos exemplos práticos de utilização do comando MERGE.

6.5 MERGE – Exemplo Básico

Nesta seção será apresentada a operação de MERGE entre um data set com dados de EMPRESAS e um data set com os dados das unidades locais (UL's) destas empresas⁶. Os data sets serão montados a partir dos arquivos “EMPRESAS.txt” e “ULS.txt”, cujos conteúdos são apresentados a seguir.

O arquivo de empresas possui 3 variáveis:

- Raiz do CNPJ (colunas 1-8);
- Razão Social (9-48) ;
- Natureza Jurídica Simplificada (coluna 49; 1=Órgão Público; 2=Empresa Privada).

⁶Uma UL corresponde basicamente a um endereço de atuação da empresa.

11111111	MARIA GONÇALVES ME	2
66666666	LIMPSEV SERVIÇOS DE LIMPEZA	2
22222222	INDUSTRIA DE CALÇADOS RIO DO OURO	2
44444444	SUPERMERCADO DO BOM PREÇO	2
33333333	COMÉRCIO DE ROUPAS BOA VISTA	2
55555555	PREFEITURA DE NEW JERSEY	1
88888888	MERCADINHO ABCD	2

O arquivo de UL's possui 4 variáveis:

- Raiz do CNPJ (colunas 1-8);
- Sufixo do CNPJ (colunas 9-12);
- Nome Fantasia (13-33) ;
- Pessoal Ocupado (ou seja, total de funcionários, colunas 34-37).

222222220001	RIO DO OURO I	20
222222220002	RIO DO OURO II	5
222222220003	RIO DO OURO III	15
333333330001	LOJA RIO DE JANEIRO	8
333333330002	LOJA NITERÓI	4
444444440001	SEDE ADMINISTRATIVA	112
444444440003	LOJA PRINCIPAL	101
444444440004	LOJA RECIFE	48
444444440005	LOJA PORTO ALEGRE	50
444444440010	LOJA VITÓRIA	.
555555550001		1800
666666660001	LIMPSEV	12

Neste exemplo, a variável em comum é “Raiz do CNPJ”. O Programa 58, apresentado a seguir, realiza a importação dos dois arquivos para depois criar um novo data set chamado *Todas*, resultante do MERGE entre as empresas e UL's.

```

/*-----
Programa 58: MERGE - EXEMPLO 1 (BÁSICO);
PRODUZ O DATA SET "TODAS" RESULTANTE DO MERGE ENTRE UM
DATA SET DE EMPRESAS E UM DATA SET DE UL'S
-----*/

* PASSO 1 - IMPORTA ARQUIVO DE EMPRESAS PARA UM DATA SET SAS;
DATA Emps;
  INFILE 'C:\CursoSAS\EMPRESAS.txt';
  INPUT
    @001 raiz $8.
    @009 razao $40.
    @049 natjur 1.
  ;
RUN;

* PASSO 2 - IMPORTA ARQUIVO DE UL'S PARA UM DATA SET SAS;
DATA Uls;
  INFILE 'C:\CursoSAS\ULS.txt';

```

```

INPUT
@001 raiz      $8.
@009 sufixo    $4.
@013 fantasia $21.
@034 po        4.
;
RUN;

* PASSOS 3 e 4 - ORDENAÇÃO DOS DATA SETS PELA VARIÁVEL
  EM COMUM ("raiz");
PROC SORT DATA = Emps;
  BY raiz;
RUN;

PROC SORT DATA = Uls;
  BY raiz;
RUN;

* PASSO 5 - REALIZA O MERGE E
              GRAVA O RESULTADO NO DATA SET Todas;
DATA Todas;
  MERGE Emps Uls;
  BY raiz;
RUN;

```

O programa será agora analisado. O primeiro passo cria um data set denominado *Emps* com dados oriundos do arquivo “EMPRESAS.txt”. De maneira análoga, no segundo passo o data set *Uls* é criado a partir de “ULS.txt”.

O objetivo é realizar o MERGE utilizando a variável “raiz” que é a variável comum entre os dois data sets. Para que o MERGE possa ser feito, é preciso que os data sets estejam ordenados por esta variável em comum. Por este motivo, nos passos 3 e 4 utilizou-se a PROC SORT. Embora o arquivo de UL’s já estivesse ordenado, foi feita a chamada da PROC SORT da mesma forma, pois assim, caso o conteúdo de “ULS.txt” seja alterado no futuro, o funcionamento do programa SAS continuará garantido.

O quinto e último passo do programa é o que, de fato, cria o data set *Todas*, resultante do MERGE entre os data sets *Emps* e *Uls*. As tabelas a serem combinadas são especificadas ao lado do comando MERGE, enquanto a variável em comum é indicada ao lado do comando BY.

O data set *Todas* é gerado com 14 observações e 6 variáveis, conforme mostrado abaixo.

<i>Todas</i>					
raiz	razao	natjur	sufixo	fantasia	po
11111111	MARIA GONÇALVES ME	2			.
22222222	INDUSTRIA DE CALÇADOS RIO DO OURO2	2	0001	RIO DO OURO I	20
22222222	INDUSTRIA DE CALÇADOS RIO DO OURO	2	0002	RIO DO OURO II	5
22222222	INDUSTRIA DE CALÇADOS RIO	2	0003	RIO DO OURO	15

	DO OURO			III	
33333333	COMÉRCIO DE ROUPAS BOA VISTA	2	0001	LOJA RIO DE JANEIRO	8
33333333	COMÉRCIO DE ROUPAS BOA VISTA	2	0002	LOJA NITERÓI	4
44444444	SUPERMERCADO DO BOM PREÇO	2	0001	SEDE ADMINISTRATIVA	112
44444444	SUPERMERCADO DO BOM PREÇO	2	0003	LOJA PRINCIPAL	101
44444444	SUPERMERCADO DO BOM PREÇO	2	0004	LOJA RECIFE	48
44444444	SUPERMERCADO DO BOM PREÇO	2	0005	LOJA PORTO ALEGRE	50
44444444	SUPERMERCADO DO BOM PREÇO	2	0010	LOJA VITÓRIA	.
55555555	PREFEITURA DE NEW JERSEY	1	0001		1800
66666666	LIMPSERV SERVIÇOS DE LIMPEZA	2	0001	LIMPSERV	12
88888888	MERCADINHO ABCD	2			.

Veja que as empresas com Raiz do CNPJ iguais a “11111111” e “88888888” não possuem UL’s associadas, mas mesmo assim foram gravadas no data set. Você entenderá porque isso aconteceu ao estudar a próxima seção.

6.6 MERGE – Como Funciona?

O conceito chave para o entendimento da forma de funcionamento do comando MERGE está, na realidade, associado ao **comando BY** e a maneira como ele **altera o comportamento do PDV** durante a execução dos comandos MERGE, SET e MODIFY (este último, ainda não apresentado).

Basicamente, o comando BY controla a maneira pela qual os valores das variáveis do PDV são passados para missing. Durante o processamento de um BY Group, o sistema SAS **retém os valores das variáveis do PDV** até que tenha copiado a última observação do BY Group em qualquer um dos data sets.

Para que seja possível entender este conceito, é preciso apresentar um exemplo, que mostrará o funcionamento do MERGE passo-a-passo. Suponha que desejamos realizar o MERGE entre os data sets de Empresas e UL’s cujo conteúdo é mostrado na Figura 7.5.

Empresas

raiz	razão
22222222	Emp A

ULs

raiz	sufixo	uf
22222222	0001	RJ

77777777	Emp B
88888888	Emp C
99999999	Emp D

22222222	0002	SP
77777777	0001	MG
99999999	0010	RJ

Figura 23. Data sets que serão combinados com o MERGE

Os dois *data sets* já estão ordenados pela variável comum, “raiz”. Recordando a lição anterior, os comandos necessários para a efetivação do MERGE são:

```
DATA Resultado;
  MERGE Empresas Uls;
  BY raiz;
RUN;
```

A simulação do processo de execução deste trecho de código pelo SAS é apresentada a seguir.

6.6.1. Criação do PDV

Após a compilação do código, o SAS criará o PDV mostrado na Figura 24. Observe que o PDV contém **todas variáveis dos data sets EMPRESAS e ULS**. Note ainda que a **variável comum RAIZ aparece apenas uma vez no PDV**.

raiz	razão	sufixo	uf	_N_	_ERROR_

Figura 24. PDV antes da execução da primeira iteração

6.6.2. Execução da Primeira Iteração

A primeira iteração começa com a leitura da primeira observação do data set *Empresas* (o primeiro especificado na lista de data sets do comando MERGE), fazendo com que as variáveis correspondentes sejam preenchidas no PDV (Figura 25).

raiz	razão	sufixo	uf	_N_	_ERROR_
22222222	Emp A			1	0

Figura 25. PDV após a leitura da primeira observação de *Empresas*

O SAS irá ler agora a primeira observação que pertence ao mesmo BY Group no data set *Uls*. O valor da “raiz” será **sobrescrito** no PDV. Os valores das variáveis “sufixo” e “uf”, que pertencem ao data set *Uls* serão preenchidos no PDV. O valor da “razão” não será alterado, pois essa variável não faz parte do data set *Uls*. Ao final do processo o PDV estará configurado da maneira mostrada na Figura 26.

raiz	razão	sufixo	uf	_N_	_ERROR_
22222222	Emp A	0001	RJ	1	0

Figura 26. PDV após a leitura da primeira observação de *Empresas* e da primeira observação de *Uls*

Neste momento o conteúdo do PDV é gravado no data set de saída (*Resultado*) e o SAS passa para próxima iteração.

6.6.3. Execução da Segunda Iteração

Na primeira ação desta iteração, o SAS verifica se ainda existe mais alguma observação que pertença ao mesmo BY Group (“raiz”= ‘22222222’) nos dois data sets. Olhe novamente para a Figura 23. Veja que não existe mais nenhuma observação para o BY Group no data set *Empresas*. No entanto, ainda existe no data set *Uls*. Por esta razão, o SAS executa as seguintes ações:

1. Manter o conteúdo das variáveis do PDV (exceto as variáveis automáticas). Veja **que este comportamento é diferente do comportamento padrão do PDV, apresentado na Seção 4.19.**
2. **Não** realizar a leitura de uma nova observação em *Empresas*.
3. Realizar a leitura de uma nova observação em *Uls*, alterando o conteúdo das variáveis “raiz”, “sufixo” e “uf”.

Como resultado, a configuração do PDV ao final desta iteração será a mostrada na Figura 27. Estes serão os dados gravados no data set *Resultado*.

raiz	razão	sufixo	uf	_N_	_ERROR_
22222222	Emp A	0002	SP	2	0

Figura 27. PDV após o final da segunda iteração

6.6.4. Execução da Terceira Iteração

No início da iteração, novamente, a primeira ação do SAS é examinar se existe mais alguma observação associada ao BY Group (“raiz”= ‘22222222’) em algum dos dois data sets envolvidos na operação de MERGE. Desta vez a resposta é negativa. Então, por este motivo, o SAS anula as variáveis do PDV (Figura 28).

raiz	razão	sufixo	uf	_N_	_ERROR_
				3	0

Figura 28. PDV no início da terceira iteração

O SAS lê a próxima observação em *Empresas* e preenche as variáveis correspondentes no PDV. Esta observação pertence ao BY Group (“raiz”= ‘77777777’). A seguir o SAS verifica se a próxima observação em *Uls* está associada ao mesmo BY Group. Como o resultado da verificação é VERDADEIRO, a observação é lida de *Uls* e as variáveis correspondentes são atualizadas no PDV (Figura 29). A iteração é finalizada com a gravação de mais uma observação no data set *Resultado*.

raiz	razão	sufixo	uf	_N_	_ERROR_
77777777	Emp B	0001	MG	3	0

Figura 29. PDV após o final da terceira iteração

6.6.5. Execução da Quarta Iteração

O SAS inicia examinando se existe mais alguma observação associada ao BY Group “raiz”= ‘77777777’ nos dois data sets. O resultado da verificação é FALSO, e, com

isso, são anulados os valores das variáveis “raiz”, “razão”, “sufixo” e “uf” no PDV (Figura 30).

raiz	razão	sufixo	uf	_N_	_ERROR_
				4	0

Figura 30. PDV no início da quarta iteração

A próxima observação de *Empresas* é lida, atualizando as variáveis “raiz” e “razão” no PDV. Esta observação pertence ao BY Group (“raiz” = “88888888”). A seguir, o SAS verifica se a próxima observação em *Uls* está associada ao mesmo BY Group. A comparação retorna FALSE e por isso o SAS não importa a observação de ULS. O PDV fica configurado da forma apresentada na Figura 31. Veja que “sufixo” e “uf” ficam com valor missing. Estes serão os dados gravados no data set *Resultado*.

raiz	razão	sufixo	uf	_N_	_ERROR_
88888888	Emp C			4	0

Figura 31. PDV ao final da quarta iteração

6.6.6. Execução da Quinta Iteração (FINAL)

Esta última iteração funciona de maneira análoga, não apresentando nenhuma novidade. Ao final da iteração o PDV estará com a configuração mostrada na Figura 32 e o processo de geração do data set *Resultado* será finalizado.

raiz	razão	sufixo	uf	_N_	_ERROR_
99999999	Emp D	0010	RJ	5	0

Figura 32. PDV após o final da última iteração

O conteúdo final de *Resultado* é mostrado a seguir.

<i>Resultado</i>			
raiz	razão	sufixo	uf
22222222	EmpA	0001	RJ
22222222	EmpA	0002	SP
77777777	EmpB	0001	MG
88888888	EmpC		
99999999	EmpD	0010	RJ

A título de curiosidade, apresenta-se o programa utilizado na simulação.

```
*Programa 59: Estudo do MERGE e PDV;
DATA Empresas;
  INPUT raiz $ razao $;
  DATALINES;
  22222222 EmpA
```

```

77777777 EmpB
88888888 EmpC
99999999 EmpD
;
RUN;

DATA Uls;
  INPUT raiz $ sufixo $ uf $;
  DATALINES;
22222222 0001 RJ
22222222 0002 SP
77777777 0001 MG
99999999 0010 RJ
;
RUN;

DATA Resultado;
  MERGE Empresas Uls;
  BY raiz;
RUN;

```

6.7 MERGE – Opção IN

Quando dois data sets são combinados, é possível utilizar a opção IN para acompanhar qual o data set origem que contribuiu para a criação de cada observação no data set de saída. Isto permite com que o funcionamento padrão do MERGE seja modificado, possibilitando ao programador controlar as observações que serão transportadas para o data set resultante da combinação.

A opção IN pode ser imaginada como uma espécie de **etiqueta de fabricação**, como mencionado em alguns livros sobre SAS. Para que o conceito fique claro, altere a rotina de MERGE do programa da Seção 6.5 da forma mostrada a seguir e, depois, execute novamente o programa.

```

* PASSO 5: alteração - GRAVA APENAS AS EMPRESAS QUE POSSUEM UL(S);
DATA Com_ULS;
  MERGE Emps (IN=E) Uls (IN=U);
  BY raiz;
  IF E AND U THEN OUTPUT;
RUN;

```

Nesta modificação:

- O comando MERGE EMPS (IN=E) ULS (IN=U) faz com que os data sets *Emps* e *Uls* recebam, respectivamente, os rótulos E e U.
- Já a linha de comando IF E AND U THEN OUTPUT atua como um IF de **subsetting** que dá a seguinte ordem ao SAS: “grave a observação corrente no data set de saída **apenas se** ela for resultante da contribuição dos data sets E e U).

Sendo assim, as empresas que não possuem UL’s não serão gravadas no data set de saída, que ficará com 12 observações ao invés das 14 mostradas na Seção 6.5.

Com Uls

raiz	razao	natjur	sufixo	fantasia	po
22222222	INDUSTRIA DE CALÇADOS RIO DO OURO2	2	0001	RIO DO OURO I	20
22222222	INDUSTRIA DE CALÇADOS RIO DO OURO	2	0002	RIO DO OURO II	5
22222222	INDUSTRIA DE CALÇADOS RIO DO OURO	2	0003	RIO DO OURO III	15
33333333	COMÉRCIO DE ROUPAS BOA VISTA	2	0001	LOJA RIO DE JANEIRO	8
33333333	COMÉRCIO DE ROUPAS BOA VISTA	2	0002	LOJA NITERÓI	4
44444444	SUPERMERCADO DO BOM PREÇO	2	0001	SEDE ADMINISTRATIVA	112
44444444	SUPERMERCADO DO BOM PREÇO	2	0003	LOJA PRINCIPAL	101
44444444	SUPERMERCADO DO BOM PREÇO	2	0004	LOJA RECIFE	48
44444444	SUPERMERCADO DO BOM PREÇO	2	0005	LOJA PORTO ALEGRE	50
44444444	SUPERMERCADO DO BOM PREÇO	2	0010	LOJA VITÓRIA	.
55555555	PREFEITURA DE NEW JERSEY	1	0001		1800
66666666	LIMPSERV SERVIÇOS DE LIMPEZA	2	0001	LIMPSERV	12

Agora, realize mais uma alteração no programa:

```
* PASSO 5 alteração - GRAVA APENAS AS EMPRESAS
  QUE NÃO POSSUEM UL(S);
DATA Sem_ULs;
  MERGE Emps (IN=E) Uls (IN=U);
  BY raiz;
  IF E AND NOT U THEN OUTPUT;
RUN;
```

Esta alteração fará com que apenas as observações geradas **somente** pelo data set *Emps* sejam gravadas na saída. Estas observações correspondem às empresas que não possuem UL's.

Sem_ULs

raiz	razao	natjur	sufixo	fantasia	po
11111111	MARIA GONÇALVES ME	2			.
88888888	MERCADINHO ABCD	2			.

No passo DATA também é possível especificar a criação de mais de um data set. No exemplo a seguir, serão gerados três data sets a partir do comando MERGE (*Todas*, *Com_ULS* e *Sem_ULS*). A instrução OUTPUT se encarregará de colocar as observações adequadas em cada data set.

* PASSO 5 - REALIZA O MERGE E GRAVA O RESULTADO EM 3 DATA SETS;

```
DATA Todas Sem_ULs Com_ULs;
  MERGE Emps (IN=E) Uls (IN=U);
  BY raiz;
```

```
*este data set contém apenas as empresas com UL(s);
IF E AND U THEN OUTPUT Com_ULs;
```

```
*este data set contém apenas as empresas sem UL;
ELSE IF E AND NOT U THEN OUTPUT Sem_ULs;
```

```
*este data set contém todas as empresas;
OUTPUT Todas;
```

```
RUN;
```

6.8 MERGE – Problemas mais Comuns

Esta seção apresenta uma relação dos problemas mais comumente vivenciados pelos programadores SAS com relação ao uso do comando MERGE com BY.

As Variáveis BY precisam ter o MESMO NOME.

Para que seja possível realizar o MERGE entre dois data sets é preciso que as variáveis BY em ambos os data sets possuam o mesmo nome. Muitas vezes você precisará utilizar a opção RENAME para resolver esta questão (ver Seção 4.7).

As Variáveis BY precisam ter o MESMO TIPO.

Não é possível realizar a combinação, quando as Variáveis BY possuem o mesmo nome, mas tipos diferentes (ou seja, se uma é do tipo caractere e a outra do tipo numérico).

Os Data Sets devem estar ORDENADOS pelas Variáveis BY

Se você especificar um data set que não esteja ordenado no comando BY, o SAS não realizará a operação de MERGE e apresentará a seguinte mensagem de erro: “ERROR: BY variables are not properly sorted on *nome_do_dataset*”.

Data Sets com OBSERVAÇÕES DUPLICADAS afetam o Data Set de Saída

Se você combinar data sets que possuem observações duplicadas, estas serão transportadas para o data set de saída. Observe o exemplo da Figura 33.

Razoes

raiz	razão
22222222	Emp X
22222222	Emp X
88888888	Emp Y

Receitas

raiz	classe_receita
22222222	A
88888888	C

Figura 33 – Data sets que serão combinados com o MERGE

Considere que se deseja combinar o data set *Razoes* (que possui o CNPJ e a razão social das empresas) com o data set *Receitas* (que possui o CNPJ e a classe de receita). Imagine que por um erro provocado em outro programa, o data set *Razoes* possui uma observação duplicada, a de “cnpj” = ‘22222222’. O resultado do MERGE entre os dois arquivos acabaria por levar o registro duplicado, conforme mostra a Figura 34.

Resultado

raiz	razão	classe_receita
22222222	Emp X	A
22222222	Emp X	A
88888888	Emp Y	C

Figura 34 – Data sets com observação duplicada

Problema das Variáveis comuns aos dois data sets

Se você tentar combinar data sets que possuem variáveis comuns (**com nomes iguais**), o SAS levará reservará apenas um “espaço” no PDV para as duas variáveis. Isto é um problema para o caso de variáveis que **não** são BY Variables. Observe o exemplo apresentado no Programa 60.

```
*Programa 60: variáveis de nomes iguais,
mas que não são BY Variables;
DATA Um;
  INPUT ID $ X;
  DATALINES;
a 1
b 2
c 3
;
RUN;

DATA Dois;
  INPUT ID $ X;
  DATALINES;
a 11
b 22
c 33
;
RUN;

DATA Comum;
  MERGE Um Dois; BY ID;
RUN;
```

Comum

ID	X
a	11
b	22
c	33

Uma variável comum aparecerá apenas uma vez no data set de saída. No exemplo acima, o data set *Um* irá popular o PDV primeiro com os conteúdos de “ID” e “X”. Como o data set *Dois* possui observações do mesmo BY Group, o conteúdo de “ID” e o conteúdo de “X” serão sobrescritos. No resultado final, acabarão prevalecendo as informações do data set *Dois*, já que ele foi o especificado por último na lista do MERGE.

Problema das Variáveis BY com TAMANHOS DIFERENTES

Nesta situação, é possível realizar o MERGE. Porém o comprimento adotado será o comprimento da variável no primeiro data set subordinado a lista do MERGE. Com isto, poderão ocorrer problemas de truncamento de valores.

6.9 MERGE sem BY

Os exemplos das seções anteriores mostraram casos de **Match MERGE**, ou seja, combinação de data sets com o uso do BY. No entanto, também é possível realizar o MERGE sem o uso do comando BY (embora, na prática, poucas vezes surgem problemas onde exista necessidade para esta operação). Neste caso o SAS simplesmente irá gerar a primeira observação do data set de saída combinando as primeiras observações de todos os data sets subordinados ao MERGE. A segunda observação no data set de saída será resultante da combinação das segundas observações de todos os data sets de entrada. E assim por diante. Veja o seguinte exemplo:

```
*Programa 61: MERGE sem BY;
DATA Um;
  INPUT X;
  DATALINES;
1
2
3
;
RUN;

DATA Dois;
  INPUT NOME $;
  DATALINES;
EDUARDO
ANA
;
RUN;
```

```
DATA Sem_BY;
  MERGE Um_Dois;
RUN;
```

<i>Sem BY</i>	
X	nome
1	Eduardo
2	Ana
3	

6.10 MODIFY - Introdução

O comando MODIFY é utilizado para **modificar um data set** dentro do passo DATA, **sem que seja preciso criar um novo data set**. Dependendo do problema a ser resolvido, pode funcionar de forma mais eficiente do que outros tipos de atualização, como por exemplo a atualização via comando SET.

Para demonstrar a utilidade do comando MODIFY, veja o Programa 62a, que cria o data set *F_XY* e o salva no local apontado pela library libBases. O data set possui três variáveis: “K”, “X” e “Y”. Neste exemplo, “K” é uma constante com o valor 5 e “Y” representa o valor de $X * K$.

```
*Programa 62a: cria data set usado para explicar MODIFY;
libname libBases '\\X001\bases';
```

```
DATA libBases.F_XY;
  INPUT K X;
  Y = K * X;
  DATALINES;
  5 10
  5 20
  5 30
  ;
RUN;
```

<i>F_XY</i>		
K	X	Y
5	10	50
5	20	100
5	30	150

Imagine que o desenvolvedor deste programa deixou o data set *F_XY* gravado por alguns dias e depois resolveu recriá-lo, fazendo uma modificação no valor de “K”. Suponha que “K” passou a valer 5.5 ao invés de 5. Ele poderia executar esta tarefa fazendo uso do comando SET:


```
*Programa 62b: modificando o dataset F_XY com o comando SET;
libname libBases '\\X001\bases';

DATA libBases.F_XY;
  SET libBases.F_XY;
  K = 5.5;
  Y = K * X;
RUN;
```

<i>F_XY</i>		
K	X	Y
5.5	10	55
5.5	20	110
5.5	30	165

Ao executar o programa, o SAS lerá cada observação de *F_XY* para o PDV. As modificações em “K” e “Y” serão feitas no PDV e os dados serão sempre gravados em um **data set temporário**. Após a última iteração do passo DATA, o SAS substituirá o data set *F_XY* pelo data set temporário.

Com o uso do MODIFY o processo de atualização é diferente, pois o SAS **modificará diretamente o data set, sem criar nenhuma estrutura temporária**. Com isto a atualização torna-se mais eficiente.

```
*Programa 62c: modificando o data set com o comando MODIFY;
libname libBases '\\X001\bases';

DATA libBases.F_XY;
  MODIFY libBases.F_XY;
  K = 5.5;
  Y = K * X;
RUN;
```

Na próxima seção, o comando MODIFY é examinado em detalhes.

6.11 MODIFY com o Comando BY

Existem **quatro formas** de usar o comando MODIFY:

- sozinho;
- com o comando BY;
- com chave (*key*); e
- com o comando POINT.

Nesta seção abordaremos a utilização do MODIFY com o comando BY, por ser a que possui maior aplicação prática nas empresas. O uso do MODIFY com BY é similar ao do MERGE. Dois data sets ordenados por uma ou mais variáveis são combinados. A diferença é que, neste caso, teremos a combinação entre um **data set de transações** com um **data set alvo** (ou master). O data set alvo terá o conteúdo modificado pelo de transações.

Durante o confronto entre os data sets será preciso utilizar um destes três **“comandos de ação”** (action statements):

- **OUTPUT:** cria uma nova observação no data set alvo, a partir de uma observação do data set de transações.
- **REPLACE:** atualiza uma observação no data set alvo, com os dados de uma observação do data set de transações.
- **REMOVE:** exclui uma observação no data set alvo.

O Programa 63 mostra um programa que usa o comando MODIFY, para alterar um data set com dados de empresas a partir de um data set de transações. Os comentários sobre o funcionamento do programa são apresentados logo em seguida.

```
*Programa 63: MODIFY - aplicando arquivo de transações a
um data set;
* CRIA O DATA SET ALVO;
DATA Emp_Alvo;
  INPUT cnpj razao $ 9-45 natjur cnae;
  DATALINES;
  11111111 MARIA GONÇALVES ME                2135 62040
  66666666 LIMPERSV SERVIÇOS DE LIMPEZA      2143 81117
  22222222 INDUSTRIA DE CALÇADOS RIO DO OURO 2062 15335
  44444444 SUPERMERCADO DO BOM PREÇO         2062 47113
  33333333 COMÉRCIO DE ROUPAS BOA VISTA      2062 47814
  55555555 PREFEITURA DE NEW JERSEY        1031 86101
  88888888 MERCADINHO ABCD                  2135 47113
  ;
RUN;

* CRIA DATA SET DE TRANSAÇÕES;
DATA Emp_Trans;
  INPUT cnpj razao $ 9-45 natjur cnae;
  DATALINES;
  11111111 MARIA CORREA LTDA                2135 62040
  77777777 MERCADO EFGH                    2062 47113
  55555555 PREFEITURA DE NOVA JÉRSEI      1031 70107
  ;
RUN;

* ORDENA POR CNPJ;
PROC SORT DATA = Emp_Alvo; BY cnpj; RUN;
PROC SORT DATA = Emp_Trans; BY cnpj; RUN;

* MODIFICA O DATA SET "Emp_Alvo" USANDO
  O CONTEÚDO DO DATA SET DE TRANSAÇÕES;
DATA Emp_Alvo;
  MODIFY Emp_Alvo Emp_Trans;
  BY cnpj;
  SELECT (_IORC_);
    WHEN (%SYSRC(_SOK)) REPLACE;
    WHEN (%SYSRC(_DSENMR)) DO;
      OUTPUT;
      _ERROR_ = 0;
    END;
  END;
RUN;
```

Ao final do processamento, o data set *Emp_Alvo* possuirá a seguinte configuração:

<i>Emp_Alvo</i>			
cnpj	razao	natjur	cnae
11111111	MARIA GONÇALVES ME	2135	62040
22222222	INDUSTRIA DE CALÇADOS RIO DO OURO	2062	15335
33333333	COMÉRCIO DE ROUPAS BOA VISTA	2062	47814
44444444	SUPERMERCADO DO BOM PREÇO	2062	47113
55555555	PREFEITURA DE NOVA JÉRSEI	1031	70107
66666666	LIMPSERV SERVIÇOS DE LIMPEZA	2143	81117
88888888	MERCADINHO ABCD	2135	47113
77777777	MERCADO EFGH	2062	47113

Segue a explicação sobre o funcionamento do Programa 63. Os dois primeiros passos do programa são bem simples: no primeiro cria-se o data set alvo (*Emp_Alvo*) e no segundo o data set de transações que será usado para alterá-lo. A seguir os dois data sets são ordenados por CNPJ.

O quinto passo é o que é realmente interessante, pois contém o comando MODIFY responsável por alterar o conteúdo de *Emp_Alvo*. A instrução: ‘MODIFY EMP_ALVO EMP_TRANS;’ diz ao SAS que *Emp_Alvo* será modificado por *Emp_Trans*. O MODIFY requer que os data sets sejam indicados nessa ordem, primeiro o master e depois o data set de transações. A instrução: ‘BY cnpj;’ indica que a variável “cnpj” será utilizada como chave de comparação. A seguir é montado um SELECT que avalia o valor da variável *_IORC_* (Input/Output Return Code) que é outra **variável automática** do passo DATA (assim como *_N_* e *_ERROR_*). Ela contém um **valor numérico** que indica o status da última ação de entrada/saída executada. O SAS disponibiliza a macro %SYSRC para converter este valor numérico para um mnemônico mais fácil de ser memorizado pelo programador. A Tabela 9 relaciona os códigos e seus mnemônicos.

Tabela 9. *_IORC_* - código e mnemônicos

MNEMÔNICO	CÓDIGO NUMÉRICO	SIGNIFICADO
<i>_SOK</i>	0	A observação do data set de transações encontrou uma observação correspondente no data set alvo.
<i>_DSENMR</i>	1230013	A observação do data set de transações não encontrou uma observação correspondente no data set alvo durante o processamento do BY GROUP.
<i>_DSENMTR</i>	1230014	A observação do data set de transações não encontrou uma observação correspondente no data set alvo durante o processamento do BY GROUP e não é a primeira vez que isso acontece para o BY GROUP em questão.

Retornando ao nosso programa. Cada transação do arquivo de transações é aplicada contra o arquivo alvo.

- Se o “cnpj” de uma transação é encontrado no arquivo alvo, o programa avaliará a condição WHEN (%SYSRC(_SOK)) como TRUE e executará a instrução REPLACE. Esta instrução atualiza o registro de “cnpj” correspondente no arquivo alvo.
- Se o “cnpj” de uma transação **não é** encontrado no arquivo alvo, o programa avaliará a condição WHEN (%SYSRC(_SOK)) como FALSE. Em seguida avaliará a condição WHEN (%SYSRC(_DSENMR)) como TRUE e executará dois comandos: OUTPUT e _ERROR_ = 0. O comando OUTPUT é o que, de fato, insere um novo registro no arquivo alvo. Já o comando _ERROR_ = 0 é usado porque o SAS automaticamente coloca o valor 1 nesta variável quando uma transação do arquivo de transações não é encontrada no arquivo alvo. O SAS, por default, considera que não achar a chave de uma transação no arquivo alvo é um “erro” e acaba gerando várias mensagens no log. Então temos que fazer _ERROR_ = 0 para avisar ao SAS que o que aconteceu não foi um erro e sim algo possível de ocorrer em nosso cenário.

O LOG do SAS mostra a quantidade de registros inseridos, excluídos e atualizados: *“NOTE: The data set WORK.EMP_ALVO has been updated. There were 2 observations rewritten, 1 observations added and 0 observations deleted”.*

Para encerrar a seção, apresenta-se a seguir, um exemplo de trecho de programa que utiliza o MODIFY para apagar registros.

```
* APAGA REGISTROS DO DATA SET "Emp_Alvo";
* USANDO O CONTEÚDO DO DATA SET DE TRANSAÇÕES;
DATA Emp_Alvo;
  MODIFY Emp_Alvo Emp_Trans;
  BY cnpj;
  IF _IORC_ = %SYSRC(_SOK) THEN REMOVE;
RUN;
```

6.12 Arrays

A Linguagem SAS suporta a utilização de arrays (vetores e matrizes) dentro do passo DATA, com o objetivo principal de **simplificar a forma como podemos nos referir a um grupo de variáveis relacionadas**. Uma vez criado, um array pode ser manipulado de maneira semelhante a feita por qualquer outra linguagem de programação.

Para demonstrar a forma de declarar e utilizar arrays no SAS, considere o arquivo “VENDAS.csv”, mostrado abaixo, que registra o **valor das vendas de 6 produtos**, efetuadas por três vendedores de uma loja de departamento hipotética no mês de abril.

```
M10, JOÃO, 12000.55, 12400.98, 15999.99, 13300.21, 13800.12, 9200.00
M11, PAULO, 15111.12, 15232.15, 11100.15, 12380.00, 12322.15, 17812.15
M13, ANTONIO, 9000.00, 12800.35, 13500.35, 10200.00, 10800.13, 11999.99
```

Considere um programa que será utilizado para calcular o salário a ser pago para cada vendedor. A seguinte regra será utilizada: todo vendedor ganha R\$ 1.000,00 somados ao valor de uma comissão. Se o valor das vendas de um produto for inferior à R\$ 12.000,00, o vendedor recebe comissão de 2% referente ao produto em questão, caso contrário a comissão é de 3%. O programa SAS a seguir gera um data set com os valores das comissões para cada vendedor, utilizando um array no processo de cálculo.

```
*Programa 64: Trabalhando com um Array;
DATA FolhaSalarial (KEEP = matricula nome salario);

    *ETAPA 1 - LEITURA DO ARQUIVO "VENDAS_ABRIL.TXT";
    INFILE 'C:\CursoSAS\VENDAS.csv' DLM=';' DSD;
    INPUT matricula $ nome $ vendaProd1-vendaProd6;

    *ETAPA 2 - DECLARAÇÃO DO ARRAY "VENDAS";
    ARRAY vendas{6} vendaProd1-vendaProd6;

    *ETAPA 3 - CALCULA A COMISSÃO A SER RECEBIDA
    USANDO O ARRAY "vendas";
    * APÓS O FIM DO LOOP O SALÁRIO PODERÁ SER CALCULADO;

    salario = 1000; *SALÁRIO INICIAL;
    comissao = 0;

    DO i = 1 to 6;
        IF vendas{i} < 12000 THEN
            comissao = comissao + vendas{i} * 0.02;
        ELSE
            comissao = comissao + vendas{i} * 0.03;
        END;

        salario = salario + comissao;

    FORMAT salario dollar9.2; *FORMATO PARA EXIBIÇÃO DO SALÁRIO;

RUN;
```

Na primeira etapa do programa ocorre a leitura do arquivo "VENDAS.csv". A primeira variável do arquivo corresponde à matrícula do vendedor e a segunda ao seu nome. A seguir os valores das vendas dos 6 produtos são lidos com o uso da seguinte sintaxe: vendaProd1-vendaProd6. Este tipo de sintaxe, dentro do comando INPUT, serve para fazer com que o SAS declare automaticamente no PDV seis variáveis com os nomes "vendaProd1", "vendaProd2", "vendaProd3", "vendaProd4", "vendaProd5" e "vendaProd6". É importante deixar claro que o uso deste recurso no comando INPUT - que ainda não havia sido mostrado em outras lições – **não** representa a declaração de um array. Na realidade **estão sendo declaradas 6 variáveis** distintas.

A segunda etapa do programa consiste na declaração do array "vendas", com dimensão de seis posições:

```
array VENDAS{6} VENDAPROD1-VENDAPROD6;
```

No momento da declaração do *array* é feita uma **correspondência** entre o mesmo e as variáveis VENDPROD1-VENDPROD6 (Figura 35).

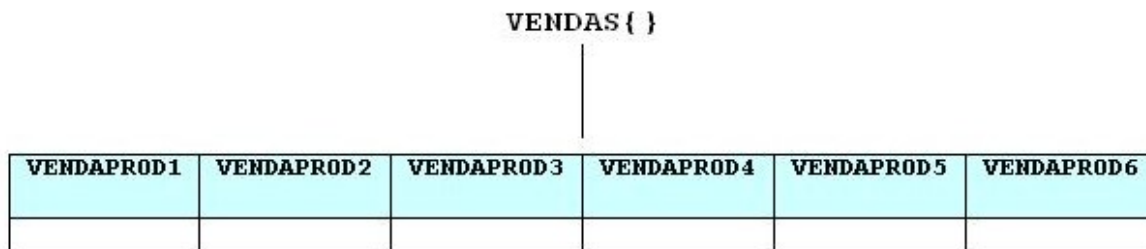


Figura 35 – Correspondência entre o *array* “vendas” e as variáveis vendaProd1-vendaProd6 no Program Data Vector

A correspondência das seis variáveis com o array torna possível o acesso ao conteúdo de qualquer variável através da referência a um índice do array. É exatamente isto que é feito na Etapa 3 do Programa 64. Veja que o cálculo da comissão para cada funcionário é realizado através de um loop que analisa o valor das vendas de cada um dos produtos, acumulando os resultados na variável “comissao”. O data set produzido pelo programa é apresentado abaixo (veja que o comando FORMAT faz com que o valor dos salários seja exibido de uma forma mais elegante).

FolhaSalarial

matricula	nome	salario
M10	JOÃO	\$3,209.06
M11	PAULO	\$3,407.73
M12	ANTONIO	\$2,629.02

A sintaxe para a declaração de arrays SAS é apresentada abaixo:

```
array nome_do_array{N} elementos;
```

Onde:

- **{N}**: determina o número de elementos do array. Também é possível usar () ou [].
- **elementos**: corresponde às variáveis às quais o array irá se referir. Se elas não existirem o SAS irá criá-las. É importante saber que o array não é armazenado no PDV, mas as variáveis de referência sempre o são. Os comandos DROP e KEEP podem ser usados sobre as variáveis de referência para que elas não sejam copiadas para o data set de saída.

6.13 Macro Variáveis

As variáveis criadas em um passo DATA possuem **escopo local**. Isto quer dizer que as variáveis “nasceram” e “morrem” dentro do passo em que foram declaradas e, por isso, não podem ter o seu conteúdo levado para um passo DATA ou PROC subsequente.

Felizmente, a linguagem SAS dispõe de um recurso chamado **macro variável** para contornar este problema. Resumidamente, pode-se dizer que uma macro variável é uma estrutura que possui as seguintes características:

- Possui **escopo global**, ou seja, pode ser enxergada por qualquer passo DATA ou PROC de um programa;
- Possui o nome prefixado por &.

O comando %LET é o responsável pela criação de macro variáveis. Sua sintaxe é:

```
%LET nome_da_macro_variável = valor;
```

Alguns exemplos:

- %LET PI = 3.14;
- %LET EMPRESA = Empresa XYZ;

Observe que não é necessário utilizar aspas para representar um valor que possui espaços (isso só é requerido quando o valor possui espaços em branco à esquerda).

Todo programa que possui uma macro variável ou uma **macro** possui **um passo adicional de compilação**, executado pelo **macro processador do SAS**. Antes do SAS compilar e executar cada passo do programa, ele passa todos os processamentos de macro para o macro processador. Este, por sua vez, “resolve” as macros, transformando-as em código SAS. Melhor explicando: o que o processador SAS faz é, simplesmente, substituir, dentro do código de um passo (DATA ou PROC), as referências a uma macro pelo texto que estiver a ela associado (ex: antes de compilar o SAS troca todas as referências de &PI por 3.14). É por este motivo que os manuais de referência dizem que “uma variável macro é sempre do tipo caractere” (na verdade ela armazena um texto que será usado para substituir referências, mesmo que esse texto seja um número como 3.14!). A seguir um exemplo que mostra a aplicação de macro variáveis.

```
*Programa 65: Macro Variáveis;
%LET PI = 3.14;
%LET CALC1 = ÁREA DO CÍRCULO;
%LET CALC2 = VOLUME DA ESFERA;
```

```
*monta data set com área do círculo de raios entre 1 e 10;
DATA Circulo;
  DO raio=1 to 10 by 1;
    area = &PI * (raio ** 2);
    msg = "&CALC1 DE RAO " || PUT(raio,2.);
    OUTPUT;
  END;
RUN;
```

```
*monta data set com volume da esfera de raios entre 1 e 10;
DATA Esfera;
  DO raio=1 to 10 by 1;
    area = 4/3 * &PI * (raio ** 3);
    msg = "&CALC2 DE RAO " || PUT(raio,2.);
    OUTPUT;
  END;
RUN;
```

Inicialmente são declaradas três macro variáveis fora dos passos DATA do programa. O objetivo é **utilizá-las como constantes** ao longo do programa. No processo de compilação **de cada passo**, desta vez haverá uma etapa inicial feita pelo macro processador, para substituir as referências às macros pelos seus valores. Melhor explicando: antes da compilação do passo ‘DATA Circulo’ (primeiro passo DATA do programa), o macro processador substituirá as referências às variáveis “&PI”, “&CALC1” pelos valores ‘3.14’ e ‘ÁREA DO CÍRCULO’, respectivamente. Depois disso é que o passo “DATA Circulo” será, de fato, compilado e executado pelo SAS. A seguir a mesma coisa será feita com relação ao passo “DATA Esfera” (segundo passo DATA do programa). Primeiro o macro processador substituirá as referências às variáveis “&PI”, “&CALC2” pelos valores ‘3.14’ e ‘VOLUME DA ESFERA’, respectivamente. Então compilará e executará o passo DATA Esfera. Os dois data sets gerados são apresentados a seguir.

Circulo

raio	area	msg
1	3.14	ÁREA DO CÍRCULO DE RAO 1
2	12.56	ÁREA DO CÍRCULO DE RAO 2
3	28.26	ÁREA DO CÍRCULO DE RAO 3
4	50.24	ÁREA DO CÍRCULO DE RAO 4
5	78.5	ÁREA DO CÍRCULO DE RAO 5
6	113.04	ÁREA DO CÍRCULO DE RAO 6
7	153.86	ÁREA DO CÍRCULO DE RAO 7
8	200.96	ÁREA DO CÍRCULO DE RAO 8
9	254.34	ÁREA DO CÍRCULO DE RAO 9
10	314	ÁREA DO CÍRCULO DE RAO 10

Esfera

raio	area	msg
1	4.188666666667	VOLUME DA ESFERA DE RAO 1
2	33.493333333333	VOLUME DA ESFERA DE RAO 2
3	113.04	VOLUME DA ESFERA DE RAO 3
4	267.94666667	VOLUME DA ESFERA DE RAO 4
5	523.33333333	VOLUME DA ESFERA DE RAO 5
6	904.32	VOLUME DA ESFERA DE RAO 6
7	1436.02666667	VOLUME DA ESFERA DE RAO 7
8	2143.57333333	VOLUME DA ESFERA DE RAO 8
9	3052.08	VOLUME DA ESFERA DE RAO 9
10	4186.66666667	VOLUME DA ESFERA DE RAO 10

ATENÇÃO!

Não esqueça que as referências às macro variáveis são sempre resolvidas pelo processador de macros antes da execução de um passo DATA ou PROC.

6.14 Macro Variáveis Dinâmicas – CALL SYMPUT

No exemplo da seção anterior, as macro variáveis foram declaradas fora dos passos DATA e utilizadas como constantes. No entanto, uma aplicação bem mais interessante para as macro variáveis é feita pela rotina CALL SYMPUT. Esta rotina **cria uma variável macro dinamicamente**, ou seja, a partir do valor de uma variável declarada dentro de um passo DATA. Com isso, **torna-se possível transferir informações de um passo DATA para outro**. A sintaxe da rotina CALL SYMPUT é apresentada a seguir:

```
CALL SYMPUT ("nome_da_macro_variável", valor);
```

Veja que o nome da macro variável a ser criada deve estar entre aspas. No parâmetro valor, podemos especificar tanto uma constante literal, um número ou uma variável do passo DATA. O próximo exemplo demonstra o uso da rotina CALL SYMPUT. A explicação encontra-se no próprio código do programa.

```
*Programa 66: CALL SYMPUT;
*CRIAR DATA SET COM NOME E SALARIO DE EMPREGADOS;
DATA Empregados;
  INPUT nome $ 1-10 salario;
  DATALINES;
  Maria      1500
  Roberto    500
  Carlos     5000
  João       3000
  Bosco     2500
  ;
RUN;

*ORDENA O DATA SET;
PROC SORT; BY salario;

*OBTÉM O MAIOR SALÁRIO E ARMAZENA NA MACRO VARIÁVEL "maior_sal";
DATA _NULL_;
  SET Empregados;
  BY salario;
  IF LAST.salario THEN CALL SYMPUT("maior_sal", salario);
RUN;

*ALTERA O DATA SET EMPREGADOS CRIANDO UMA VARIÁVEL;
```

```

*ADICIONANDO VARIÁVEL COM A DIFERENÇA ENTRE O MAIOR SALÁRIO;
*E O SALÁRIO DO FUNCIONÁRIO;
DATA Empregados;
  SET Empregados;
  diferenca = &maior_sal - salario;
RUN;

```

<i>Empregados</i>		
nome	salario	diferenca
Maria	500	4500
Roberto	1500	3500
Carlos	2500	2500
João	3000	2000
Bosco	5000	0

A documentação completa sobre a rotina CALL SYMPUT e outras rotinas de macro pode ser encontrada no manual de referência “SAS 9.4 Macro Language: Reference”.

6.15 Macros

Uma macro do SAS é um trecho de código que aceita parâmetros (de entrada e saída), pode ser armazenado em um arquivo separado e compartilhado por diversos programas SAS. Ou seja, uma macro **é o recurso do SAS que permite a reutilização de código**.

As macros parecem um pouco com as funções e procedures das outras linguagens de programação. No entanto a maneira com que o SAS as compila e executa é completamente diferente do processo usado nas outras linguagens. Na realidade os SAS faz com as macros a mesma coisa que faz com as macro variáveis: **usa o macro processador para substituir o texto da macro por código SAS** antes de compilar e executar um passo DATA ou PROC. A sintaxe para a declaração de macros é apresentada a seguir:

```

%MACRO nome_da_macro (Param1, ... , Paramn);
comando1;
...
comandon;
%MEND nome_da_macro;

```

Todos os parâmetros funcionam como parâmetros de entrada e saída (IN OUT). O exemplo a seguir mostra uma macro usada para converter uma temperatura em graus Celsius para graus Fahrenheit. O Programa 68 gera uma tabela de equivalência de temperaturas, variando de -100°C a +100°C.

*Programa 67: Criação de MACRO;

```

*macro ConvTemp: converte temperatura em graus Celsius para
Fahrenheit;
%macro ConvTemp(C, F);
    &F = &C * 1.8 + 32;
%mend ConvTemp;

*este passo DATA usa a macro ConvTemp para gerar tabela de
equivalência de valores de temperatura em graus Celsius e
Fahrenheit;
DATA TABELA_TEMPERATURAS;
    DO Celsius=-100 TO 100 BY 10;
        %ConvTemp(Celsius, Fahrenheit);
        OUTPUT;
    END;
RUN;

```

TABELA TEMPERATURAS

Celsius	Fahrenheit
-100	-148
-90	-130
-80	-112
-70	-94
-60	-76
-50	-58
-40	-40
-30	-22
-20	-4
-10	14
0	32
10	50
20	68
30	86
40	104
50	122
60	140
70	158
80	176
90	194
100	212

Algumas considerações importantes. Dentro do código da macro, os parâmetros devem ser referenciados com o uso do caractere & antes de seu nome (no exemplo &C e &F). Para que a macro possa ser usada, é preciso chamá-la, no passo DATA ou PROC utilizando o caractere % (no exemplo, %ConvTemp(Celsius, Fahrenheit))

6.16 Macro – Como Funciona?

Embora, aparentemente, uma macro pareça funcionar como uma sub-rotina ou procedure comum de linguagens como Python, JAVA ou C#, na verdade o seu processo de compilação e execução é completamente diferente. Na linguagem SAS a chamada a uma macro implica simplesmente na **substituição da referência a macro pelo texto a ela associado** dentro do passo DATA ou PROC. Esta substituição é realizada pelo macro processador, da mesma maneira que é feito o tratamento de macro variáveis.

De uma maneira simples, pode-se dizer que um passo DATA ou PROC que possua uma chamada a macro é executado pelo SAS em três etapas:

1. O macro processador substitui a referência à macro pelo texto a ela associado.
2. O compilador SAS compila o passo DATA ou PROC.
3. O passo é executado.

Para que isto fique mais claro, apresentaremos um exemplo “etapa-por-etapa”, que explica o processo de execução do passo DATA do programa abaixo (é o mesmo apresentado na seção anterior).

```
%macro ConvTemp(C, F);  
    &F = &C * 1.8 + 32;  
%mend ConvTemp;  
  
DATA TABELA_TEMPERATURAS;  
    DO Celsius=-100 TO 100 BY 10;  
        %ConvTemp(Celsius, Fahrenheit);  
    OUTPUT;  
END;  
RUN;
```

O passo “DATA TABELA_TEMPERATURAS” possui uma chamada de macro e, por isso, a primeira etapa para viabilizar a sua execução consistirá na substituição de “%ConvTemp(Celsius, Fahrenheit)” pelo texto associado a esta chamada. Isto é feito pelo macro processador. Ao final da execução desta etapa, o código do passo “DATA TABELA_TEMPERATURAS” terá sido modificado da maneira indicada abaixo. Note que os parâmetros são substituídos da maneira adequada.

```
DATA TABELA_TEMPERATURAS;  
    DO Celsius=-100 TO 100 BY 10;  
        Fahrenheit = Celsius * 1.8 + 32;  
    OUTPUT;  
END;  
RUN;
```

A partir desse ponto o SAS compila e executa o programa da maneira que já foi descrita no Capítulo 4, ou seja, utilizando normalmente o PDV.

Uma recurso muito interessante oferecido pelo SAS é a possibilidade de criar **bibliotecas de macros**. Uma macro pode ser armazenada em um arquivo e usada por diversos programas. Para que um programa possa fazer uso de uma macro genérica, basta utilizar o comando %INCLUDE. Por exemplo, para incluir o arquivo de macros “lib_macros.sas” armazenado em “C:\CursoSAS\macros”, dentro de um programa, é necessário dar o seguinte include:

```
%include 'C:\CursoSAS\macros\lib_macros.sas';
```

6.17 Sistema SAS e os SGBD Relacionais

Conforme brevemente introduzido no Capítulo 1, um SGBD relacional é um software sofisticado que permite a definição, manipulação e compartilhamento de bancos de dados entre vários usuários e aplicações. Entre os exemplos conhecidos de SGBDs, é possível citar Oracle, MySQL, Microsoft SQL Server, DB2 e PostgreSQL.

Tipicamente, os dados dos sistemas de produção de uma empresa (sistema de RH, sistema de vendas, sistema de fornecedores, etc.), encontram-se armazenados em SGBDs ao invés de serem mantidos em arquivos texto ou planilhas Excel. Por este motivo é importante conhecer forma utilizada para o acesso à banco de dados relacionais nos programas SAS. Felizmente, o SAS adota um abordagem bastante intuitiva para resolver essa questão, que baseia-se em dois conceitos principais:

- Dentro de um programa SAS, um banco de dados é simplesmente **enxergado como uma library**.
- O comando **LIBNAME** é utilizado para indicar os **dados de conexão** do banco para o SAS.

A sintaxe básica para especificação da LIBNAME possui **três partes**, que são apresentadas e descritas a seguir:

```
LIBNAME Libref engine dados_da_conexão;
```

- **Libref**: o parâmetro *libref* representa o **nome** pelo qual você irá referenciar a sua conexão de banco de dados dentro dos programas SAS. Para que você possa acessar os objetos do banco, precisará adicionar a libref como prefixo seguida de um ponto (“.”). Por exemplo, se a sua libref chama-se “libVendas” (conexão com o banco de dados de vendas) e o banco possui uma tabela chamada T_CLIENTES, você precisará acessar o objeto utilizando a sintaxe libVendas.T_CLIENTES.
- **engine**: no Sistema SAS, o acesso a bancos de dados relacionais é gerenciado através das interfaces SAS/ACCESS (ver Seção 1.1.2). Ou seja, para a conexão funcionar, é preciso que o ambiente SAS possua a interface SAS/ACCESS específica para o SGBD que mantém os dados, já que **o módulo Base SAS não possui funcionalidades para conexão com SGBDs**. No exemplo apresentado na Seção 1.1.2, foi mostrado um ambiente SAS contendo a interface *SAS/ACCESS for Oracle*, que permite o acesso nativo ao SGBD Oracle. Considerando este exemplo, o parâmetro “engine” seria o local para realizar esta especificação.

- **dados_da_conexão:** Este parâmetro varia de acordo com o SGBD a ser acessado. Quando o acesso é feito ao SGBD Oracle, por exemplo, é necessário colocar o servidor, usuário e senha nos dados de conexão. Já em uma conexão ao DB2 é preciso especificar o banco e um ID com direito de acesso.

Continuando com o exemplo do SGBD Oracle, a sintaxe básica para a configuração de uma library de acesso a um banco de dados mantido neste SGBD é apresentada abaixo:

```
LIBNAME Libref oracle path=servidor user=xxx pw=xxx schema=xx;
```

Explicando os parâmetros:

- **oracle:** engine que especifica a utilização da interface *SAS/ACCESS for Oracle*.
- **path:** nome do servidor onde o Oracle está instalado.
- **user:** nome do usuário.
- **pw:** senha.
- **schema:** esquema (dono - *owner*) dos objetos. Este parâmetro não é obrigatório, porém é interessante de ser utilizado em casos onde o usuário acessa objetos de outro esquema.

Um exemplo é apresentado a seguir:

```
libname libVendas oracle user=bd_vendas pw=123 path=ORASERVER01;
```

Este exemplo hipotético cria uma conexão para o banco ‘bd_vendas’ no servidor chamado ‘ORASERVER01’. A senha deste banco é ‘123’.

A conexão será materializada na sessão SAS através da criação de uma library chamada “libVendas”. Qualquer tabela armazenada neste banco poderá ser acessada, bastando prefixar o nome da tabela por “libVendas” (ex: libVendas.T_CLIENTES). Os comandos e PROCs mostrados neste livro poderão ser normalmente aplicados sobre as tabelas, pois as mesmas são enxergadas como data sets na sessão SAS. Ou seja: você poderá utilizar diretamente a PROC FREQ, PROC MEANS, PROC REPORT, ou qualquer outra PROC diretamente sobre as tabelas do banco de dados. Poderá ainda recuperar e alterar dados armazenados neste banco com o uso de comandos como SET, MERGE, MODIFY, etc. Poderá exportar dados das tabelas para arquivos texto, utilizando FILE e PUT. Enfim, tudo que você pode fazer com um data set, poderá fazer com qualquer tabela do SGBD!

Uma **boa prática de programação** que pode ser adotada em projetos do EG, consiste em **criar um nó de código apenas para manter a configuração das libraries** que serão utilizadas nos programas. Para tal, siga os passos abaixo:

1. Criação do Projeto “prjSQL”: Crie um novo projeto no EG (FILE / NEW / PROJECT) e salve com o nome “prjSQL”.

2. Criação do “nó de código” para as libraries: em nossos exercícios utilizaremos 3 libraries. Crie um novo nó de código (FILE / NEW / PROGRAM), atribua a este nó o nome de **MyLibraries** e digite as instruções abaixo.

```
*libname para conexão com o banco de dados de vendas (Oracle)
```

```
libname libVendas oracle user=bd_vendas pw=123 path=ORASERVER01;

*libname dos exemplos do livro;
libname libCURSO 'C:\CursoSAS';
```

IMPORTANTE!

Embora o exemplo apresentado nesta seção tenha sido relacionado ao Oracle, a conexão com outros SGBDs é feita de forma similar.

6.18 PROC SQL - Introdução

Os bancos de dados relacionais são manipulados através de uma linguagem padrão, desenvolvida especialmente para o ambiente relacional, denominada SQL (*Structured Query Language*). Muitos pesquisadores da área de banco de dados consideram-na a principal responsável pela imensa popularidade conquistada pelos SGBD's relacionais nos últimos 20 anos.

A SQL também está disponível na linguagem SAS, através da PROC SQL. Uma característica muito interessante desta PROC é o fato de que ela permite com que comandos SQL s executados não apenas sobre tabelas de SGBD's relacionais, mas também sobre data sets SAS. Isto quer dizer que você pode importar um arquivo texto para um data set e depois realizar consultas sobre o mesmo usando a linguagem SQL. As principais instruções SQL ANSI são suportadas, tais como SELECT, INSERT, UPDATE, DELETE, CREATE e DROP. A PROC SQL faz parte do Base SAS, então é possível utilizá-la sem a necessidade de licenciar nenhum módulo adicional.

A sintaxe básica da procedure é apresentada a seguir. Ela começa com a declaração "PROC SQL;" e termina com "QUIT;". Entre estas duas declarações, uma ou mais instruções SQL podem ser utilizadas.

```
PROC SQL;
  Instrução_SQL1;
  ...
  Instrução_SQLn;
QUIT;
RUN;
```

Nas próximas subseções, serão apresentados alguns exemplos de utilização do comando SELECT (o mais importante da SQL) sobre dois data sets: *Profissao* e *Funcionario*. O primeiro data set contém uma relação de "ids" (códigos) e "nomes" de profissões, enquanto o segundo data set armazena a "matrícula", "nome", "idade", "sexo" e o "id da profissão" de diferentes funcionários de uma empresa.

Profissao

id	nome
1	Engenheiro

2	Desenvolvedor
3	Analista de Dados
4	Minerador de Dados
5	Matemático

Funcionarios

mat	nome	idade	sexo	id_prof
M01	George	58	M	5
M02	Jane	32	F	3
M03	Aldous	40	M	3
M04	Thomas	28	M	1
M05	Mary	43	F	.

Observe que as variáveis “id_prof” do data set *Funcionarios* e “id” de *Profissao*, servem como variável de ligação entre ambos os data sets. Note ainda que no exemplo apresentado existe uma funcionária sem profissão cadastrada (“M05-Mary”) e também duas profissões que não estão associadas a nenhum funcionário (“2-Desenvolvedor” e “4-Minerador de Dados”). O Programa 68 apresenta o código necessário para criação destes data sets.

```
*PROC SQL: criação dos data sets para os exemplos da PROC SQL;
DATA Profissao;
  INPUT id  nome $18.;

  DATALINES;
  1 Engenheiro
  2 Desenvolvedor
  3 Analista de Dados
  4 Minerador de Dados
  5 Matemático
  ;
RUN;

DATA Funcionarios;
  INPUT mat $ nome $ idade sexo $ id_prof;

  DATALINES;
  M01 George 58 M 5
  M02 Jane 32 M 3
  M03 Aldous 40 F 3
  M04 Thomas 28 M 1
  M05 Mary 43 F .
  ;
RUN;
```

6.18.1 SELECT *

O exemplo abaixo mostra o comando SQL para retornar todas as linhas (observações) e colunas (variáveis) do data set *Funcionarios*. Este data set contém cinco colunas: “mat”, “nome”, “idade”, “sexo” e “id_prof”. Para exibir todas as colunas de um data set ou tabela de SGBD, deve-se utilizar a palavra-chave **SELECT** com um “*” (asterisco). A tabela a ser consultada deve ser indicada após a palavra chave **FROM**.

```
PROC SQL;
  SELECT * FROM Funcionarios;
  QUIT;
RUN;
```

mat	nome	idade	sexo	id_prof
M01	George	58	M	5
M02	Jane	32	F	3
M03	Aldous	40	M	3
M04	Thomas	28	M	1
M05	Mary	43	F	.

Note que um comando SELECT dentro de uma PROC SQL produz um relatório, ou seja, seu funcionamento é similar ao da PROC PRINT.

6.18.2 Seleção de colunas específicas de uma tabela

Para seleccionar colunas específicas de uma tabela, os nomes das colunas em questão devem ser separados por vírgulas, logo após a palavra-chave SELECT. O exemplo a seguir mostra o comando para recuperar apenas o nome e idade dos funcionários.

```
PROC SQL;
  SELECT nome, idade FROM Funcionarios;
  QUIT;
RUN;
```

nome	idade
George	58
Jane	32
Aldous	40
Thomas	28
Mary	43

6.18.3 Restringindo o conjunto de linhas retornadas (WHERE)

No próximo exemplo, a instrução SELECT recupera o nome e idade dos funcionários com “id da profissão” igual a 3 (ou seja: os funcionários cuja profissão é ‘analista de dados’). Para tal, é preciso utilizar a cláusula WHERE com a condição id_prof = 3.

```

PROC SQL;
  SELECT nome, idade FROM Funcionarios
  WHERE id_prof = 3;

  QUIT;
RUN;

```

nome	idade
Jane	32
Aldous	40

Mais detalhadamente, a instrução SQL acima solicita com que o SAS exiba o valor das colunas “nome” e “idade” do data set “Funcionarios” considerando apenas as linhas onde a coluna “id_prof” possui valor igual a 3. Como só existem duas linhas que atendem a esta condição, apenas dois resultados são retornados no conjunto resposta.

É importante observar que a cláusula WHERE suporta a definição de condições complexas, concatenadas por operadores lógicos (AND, OR, NOT). Ou seja, a forma como uma condição WHERE é montada equivale a mesma forma utilizada no comando IF (ver Seções 4.2 a 4.4)

6.18.4 Ordenando os resultados (ORDER BY)

É possível especificar que os resultados sejam ordenados por uma ou mais colunas, através do uso da cláusula ORDER BY. O exemplo seguinte mostra uma consulta que retorna os funcionários ordenados por idade.

```

PROC SQL;
  SELECT * FROM Funcionarios
  ORDER BY idade;

  QUIT;
RUN;

```

mat	nome	idade	sexo	id_prof
M04	Thomas	28	M	1
M02	Jane	32	F	3
M03	Aldous	40	M	3
M05	Mary	43	F	.
M01	George	58	M	5

6.19 PROC SQL – Junção de Tabelas

As junções são utilizadas em instruções SELECT para recuperar dados de duas tabelas e “juntá-los” com o objetivo de produzir uma saída combinada. As tabelas

envolvidas em uma junção precisarão possuir uma variável em comum. Em nosso exemplo, o “id da profissão” é a variável em comum entre os data sets *Profissao* e *Funcionario*. Ao contrário do que ocorre no comando MERGE, a SQL não exige que a variável de ligação possua o mesmo nome nos dois data sets que serão “mergeados”.

A SQL possui diversos tipos de JOIN, que são cobertos nas subseções a seguir.

6.19.1 INNER JOIN

A junção interna (INNER JOIN) seleciona todas as linhas de ambas as tabelas contanto que haja coincidência nos valores das colunas especificadas na **condição de junção**. No exemplo abaixo, o INNER JOIN é utilizado para obter a lista de todos os funcionários e os nomes de suas profissões (Obs.: o comando TITLE foi utilizado no exemplo apenas com o intuito de dar um título para o relatório produzido).

```
PROC SQL;
  TITLE 'INNER JOIN';

  SELECT F.*, P.*
  FROM Funcionarios F INNER JOIN Profissao P
  ON (F.id_prof = P.id)
  ORDER BY mat;

  QUIT;
RUN;
```

INNER JOIN

F.mat	F.nome	F.idade	F.sexo	F.id_prof	P.id	P.nome
M01	George	58	M		5	5 Matemático
M02	Jane	32	F		3	3 Analista de Dados
M03	Aldous	40	M		3	3 Analista de Dados
M04	Thomas	28	M		1	1 Engenheiro

Para realizar a junção interna entre duas tabelas em SQL, basta seguir sempre a mesma receita:

1. Especificar os nomes das tabelas envolvidas na junção na cláusula FROM. Cada tabela poderá receber um “apelido”, utilizado para simplificar a referência às mesmas. Em nosso exemplo, *Funcionarios* recebeu o apelido “F” e *Profissao* o apelido “P”.
2. Especificar o tipo de junção INNER JOIN entre as duas tabelas.
3. Para efetivarmos a junção entre as duas tabelas, é preciso escrever a palavra-chave ON e depois especificar entre parênteses a condição de junção entre as duas tabelas. Isso significa simplesmente, indicar qual é o atributo que “liga” as duas tabelas, ou seja, qual é a coluna que está presente nas duas tabelas. Nesse caso, é o “id da profissão”, que se chama “id” em *Profissao* e que se chama “id_prof” em *Funcionarios*.

A junção final ficou então definida como:

```
FROM Funcionarios F INNER JOIN Profissao P ON (F.id_prof = P.id)
```

Observe que no comando SELECT, os campos selecionados são agora precedidos pelo apelido de cada tabela, para indicar a que tabela pertence cada coluna (tabela “F” ou tabela “P”). Observe ainda que a funcionária “M05-Mary” não aparece no resultado final, pois ela não possui um código de profissão associado (está com o valor missing armazenado na coluna “id_prof”). Outro comentário importante é que tanto faz especificar “ON (F.id_prof = P.id)” ou “ON (P.id = F.id_prof)” na condição de junção.

Do exemplo apresentado, é possível concluir que o INNER JOIN possui o mesmo propósito do comando MERGE com IN, no entanto oferecendo a vantagem de **não** exigir que a coluna de ligação possua o mesmo nome nas duas tabelas.

6.19.2 LEFT JOIN

A operação de LEFT JOIN (ou LEFT OUTER JOIN) retorna todas as linhas da tabela à esquerda, mesmo que não exista casamento (valor equivalente) na tabela à direita. Em nosso exemplo, a funcionária “M05-Mary” não possui uma profissão cadastrada. Se desejarmos produzir um relatório contendo todos os funcionários com seus respectivos nomes de profissão, incluindo os funcionários sem profissão cadastrada, é preciso fazer uso do LEFT JOIN, conforme mostra o exemplo abaixo:

```
PROC SQL;
TITLE 'LEFT JOIN';

SELECT F.*, P.*
FROM Funcionarios F LEFT JOIN Profissao P
ON (F.id_prof = P.id)
ORDER BY mat;

QUIT;
RUN;
```

LEFT JOIN

F.mat	F.nome	F.idade	F.sexo	F.id_prof	P.id	P.nome
M01	George	58	M		5	5 Matemático
M02	Jane	32	F		3	3 Analista de Dados
M03	Aldous	40	M		3	3 Analista de Dados
M04	Thomas	28	M		1	1 Engenheiro
M05	Mary	43	F		.	.

Para realizar a LEFT JOIN entre duas tabelas em SQL a receita é a mesma da utilizada para o INNER JOIN. O único detalhe a ser mudado é que, na cláusula FROM, a tabela a qual desejamos levar todas as linhas para o resultado final deve ser especificada à esquerda. Em nosso exemplo, desejávamos todos os funcionários no resultado final, independentemente de terem profissão ou não, por isso a tabela *Funcionarios* foi especificada à esquerda da palavra LEFT JOIN, enquanto *Profissao* ficou à direita. Já com

relação a condição de junção (ON) a ordem não faz diferença, ou seja, tanto faz especificar “ON (F.id_prof = P.id)” ou “ON (P.id = F.id_prof)”.

Sendo assim, para listar todos os cargos e funcionários, mesmo os cargos que não tenham um funcionário associado, devemos especificar *Profissao* à esquerda.

```
PROC SQL;
  TITLE 'LEFT JOIN (2)';

  SELECT F.*, P.*
  FROM Profissao P LEFT JOIN Funcionarios F
  ON (F.id_prof = P.id)
  ORDER BY mat;

  QUIT;
RUN;
```

LEFT JOIN (2)

F.mat	F.nome	F.idade	F.sexo	F.id_prof	P.id	P.nome
M01	George	58	M		5	5 Matemático
M02	Jane	32	F		3	3 Analista de Dados
M03	Aldous	40	M		3	3 Analista de Dados
M04	Thomas	28	M		1	1 Engenheiro
					.	2 Desenvolvedor
					.	4 Minerador de Dados

6.19.3 RIGHT JOIN

A junção do tipo RIGHT JOIN (ou RIGHT OUTER JOIN) retorna todas as linhas da tabela à direita, mesmo que não exista casamento (valor equivalente) na tabela à esquerda. Desta forma, para produzir um relatório com todos os funcionários com seus respectivos nomes de profissão, incluindo os funcionários sem profissão cadastrada, seria necessário montar o comando SELECT da forma indicada abaixo:

```
PROC SQL;
  TITLE 'RIGHT JOIN';

  SELECT F.*, P.*
  FROM Profissao P RIGHT JOIN Funcionarios F
  ON (F.id_prof = P.id)
  ORDER BY mat;

  QUIT;
RUN;
```

RIGHT JOIN

F.mat	F.nome	F.idade	F.sexo	F.id_prof	P.id	P.nome
M01	George	58	M		5	5 Matemático
M02	Jane	32	F		3	3 Analista de Dados
M03	Aldous	40	M		3	3 Analista de Dados
M04	Thomas	28	M		1	1 Engenheiro
M05	Mary	43	F		.	.

Como o LEFT JOIN e o RIGHT JOIN funcionam da mesma maneira (a única diferença é que no primeiro caso iremos especificar a tabela a qual desejamos todas as linhas no resultado à esquerda e no segundo caso ela ficará à direita), na prática o LEFT JOIN acabou se tornando mais adotado pelos programadores.

6.19.4 FULL JOIN

A junção do tipo FULL JOIN (ou FULL OUTER JOIN) junta em uma única operação os resultados do LEFT JOIN e do RIGHT JOIN. Ela retorna todas as linhas da tabela à esquerda e todas as linhas da tabela a direita devidamente combinadas, de acordo com a especificação da condição de junção. Caso existam linhas na tabela esquerda que não tenham equivalência com a tabela direita, elas serão levadas para o resultado final. E caso existam linhas na tabela direita que não tenham equivalência na tabela esquerda, elas também serão levadas para o resultado final.

```

PROC SQL;
    TITLE 'FULL JOIN';

    SELECT F.*, P.*
    FROM Profissao P FULL JOIN Funcionarios F
    ON (F.id_prof = P.id)
    ORDER BY mat;

    QUIT;
RUN;

```

FULL JOIN

F.mat	F.nome	F.idade	F.sexo	F.id_prof	P.id	P.nome
M01	George	58	M		5	5 Matemático
M02	Jane	32	F		3	3 Analista de Dados
M03	Aldous	40	M		3	3 Analista de Dados
M04	Thomas	28	M		1	1 Engenheiro
M05	Mary	43	F		.	.
					.	2 Desenvolvedor
					.	4 Minerador de Dados

Observe que tanto a funcionária sem profissão (linha 5 do resultado), como as profissões que não estão associadas a nenhum funcionário (linhas 6 e 7 do resultado) foram listadas pelo SELECT.

6.20 PROC SQL – CREATE TABLE ... AS SELECT

Uma das aplicações mais populares da PROC SQL consiste na criação de data sets SAS a partir do resultado de um comando SELECT. Para este fim, a estrutura CREATE TABLE ... AS SELECT deve ser utilizada.

No exemplo a seguir, o data set *Resultado* será criado e populado na área de WORK do SAS. Esta data set contém o nome de cada funcionário e o nome da profissão. A instrução AS foi utilizada renomear as colunas no novo data set.

```
PROC SQL;

CREATE TABLE Resultado AS
SELECT  F.nome AS nome_funcionario,
        P.nome AS nome_profissao
FROM Funcionarios F LEFT JOIN Profissao P
ON (F.id_prof = P.id)
ORDER BY mat;

QUIT;
RUN;
```

Resultado

nome_funcionario	nome_profissao
George	Matemático
Jane	Analista de Dados
Aldous	Analista de Dados
Thomas	Engenheiro
Mary	

6.21 MERGE x SQL JOIN

Tanto o uso do comando MERGE para combinar tabelas, como a utilização da PROC SQL objetivando o mesmo propósito, **representam abordagens que possuem vantagens e desvantagens. Dependendo do tipo de combinação a ser feita**, o uso do MERGE pode ser mais adequado do que o do SQL com JOIN e vice-versa. A Tabela 10 apresenta os principais “prós” e “contras” de cada uma das abordagens.

Tabela 10 – MERGE x SQL Join: Prós e Contras

	MERGE	SQL JOIN (PROC SQL)
PRÓS	<ul style="list-style-type: none"> - Permite a utilização de recursos dentro do DATA STEP, que facilitam a transformação de variáveis. Ex.: FIRST/LAST, IF-THEN, loops, etc. - Para quem não sabe nada sobre MERGE e nada sobre SQL, é mais fácil aprender o MERGE. 	<ul style="list-style-type: none"> - Permite combinar data sets que não estão ordenados. - Permite a junção entre 3 ou mais tabelas em uma única instrução, usando variáveis BY diferentes Ex: <i>Profissao</i> com <i>Funcionario</i> com <i>Departamento</i>.
CONTRAS	<ul style="list-style-type: none"> - É preciso ordenar o data set por alguma variável BY. - Não é possível fazer um MERGE que combine tabelas por variáveis BY diferentes. A rotina tem que ser dividida em 2 passos DATA. 	<ul style="list-style-type: none"> - Não é possível utilizar os recursos de programação do DATA STEP.

6.22 PROC SQL – Produzindo Resultados Agregados

A instrução SELECT suporta a consultas complexas não somente com o uso de junções, mas também com funções para a produção de resultados agregados. Estas instruções serão apresentadas a partir de exemplos que exploram o data set *Projetos*. Este data set armazena os projetos correntemente conduzidos por uma empresa de reforma de edifícios. O data set possui quatro variáveis: “codigo” (código do projeto), tipo (‘A’=Reforma Estrutural ou ‘B’=Manutenção de Fachada), “local” (UF onde obra está sendo realizada) e “custo” (orçamento da obra em R\$). Observe que o projeto ‘P5’ ainda não tem o orçamento definido (está com valor missing armazenado na variável “custo”).

<i>Projetos</i>			
codigo	tipo	local	custo
P1	A	RJ	500000
P2	A	DF	900000
P3	B	RJ	150000
P4	A	DF	1000000
P5	B	RJ	.
P6	A	SP	850000

O Programa 70 apresenta o código para criação do data set:


```

*Programa 70: Criação do data set Projetos;
DATA Projetos;
  INPUT codigo $ tipo $ local $ id_prof;

  DATALINES;
  P1 A RJ 500000
  P2 A DF 900000
  P3 B RJ 150000
  P4 A DF 1000000
  P5 B RJ .
  P6 A SP 850000
  ;
RUN;

```

6.22.1 COUNT

A função COUNT determina o número de ocorrências de valores não-missing de uma variável em um data set.

```

PROC SQL;

  SELECT COUNT(codigo), COUNT(custo) FROM Projetos;

  QUIT;
RUN;

```

COUNT(codigo)	COUNT(custo)
6	5

O valor 6 indica que este é o número de observações onde a variável “projeto” não possui valor missing. Analogamente, o valor 5 indica que existem cinco observações onde em que a variável “orcamento” não está missing.

6.22.2 COUNT(*)

A estrutura COUNT(*) é utilizada para contar o número total de observações em um data set (ao invés de contar o número de observações onde uma variável específica não possui valor missing):

```

PROC SQL;

  SELECT COUNT(*) FROM Projetos;

  QUIT;
RUN;

```

6.22.3 MIN, MAX, AVG e SUM

Para uma determinada variável de um data set, as funções MIN, MAX e AVG retornam, respectivamente, o valor mínimo, máximo e médio da mesma. Já a função SUM retorna a somatório dos valores da variável em todas as observações. Os valores missing são ignorados por essas funções.

```
PROC SQL;
```

```
    SELECT MIN(custo) ,
           MAX(custo) ,
           AVG(custo) ,
           SUM(custo)
    FROM Projetos;
```

```
    QUIT;
RUN;
```

MIN(custo)	MAX(custo)	AVG(custo)	SUM(custo)
150000	1000000	680000	3400000

6.22.4 GROUP BY

A cláusula GROUP BY pode ser combinada com as funções de grupo para permitir a produção de resultados agregados por uma ou mais variáveis. O exemplo abaixo obtém os valores mínimo, máximo, médio e o somatório do orçamento por tipo de projeto. Note que ao contrário do que ocorre com diversas PROCs do SAS, para que o comando funcione não é necessário que o data set esteja ordenado pela(s) variável(is) indicadas na cláusula GROUP BY.

```
PROC SQL;
```

```
    SELECT tipo,
           MIN(orçamento) ,
           MAX(orçamento) ,
           AVG(orçamento) ,
           SUM(orçamento)
    FROM Projetos
    GROUP BY tipo;
```

```
    QUIT;
RUN;
```

tipo	MIN(custo)	MAX(custo)	AVG(custo)	SUM(custo)
A	500000	1000000	812500	3250000
B	150000	150000	150000	150000

É possível compor grupos formados por mais de uma variável. O exemplo a seguir, produz o somatório do orçamento por tipo de projeto e local. Os resultados são ordenados por “tipo” e “local”.

```
PROC SQL;

    SELECT tipo,
           local,
           SUM(orcamento)
    FROM Projetos
    GROUP BY tipo, local;

QUIT;
RUN;
```

tipo	local	SUM(custo)
A	DF	1900000
A	RJ	500000
B	SP	850000
B	RJ	150000

Caso você queira eliminar algum grupo do resultado final, torna-se necessário utilizar a cláusula HAVING. No exemplo abaixo, são eliminados todos os grupos em que o somatório do orçamento é inferior à 600000.

```
PROC SQL;

    SELECT tipo,
           local,
           SUM(orcamento)
    FROM Projetos
    GROUP BY tipo, local
    HAVING SUM(orcamento) > 600000
    ORDER BY tipo, local;

QUIT;
RUN;
```

tipo	local	SUM(custo)
A	DF	1900000
B	SP	850000

Uma cobertura abrangente sobre a linguagem SQL podem ser obtidas no manual de referência “SAS 9.4 SQL Procedure: user’s guide”.

6.23 Dicas Finais

Chegamos ao fim! Este livro introduziu alguns dos principais conceitos relacionados ao desenvolvimento de programas com o uso da linguagem SAS. O enfoque principal foi a apresentação de técnicas para o acesso, integração, transformação e atualização de dados disponibilizados em arquivos e bancos relacionais. O livro também abordou as PROCs para produção de relatórios e para o estudo estatístico descritivo de data sets. Ao longo dos capítulos, descreveu-se a forma de funcionamento de algumas das principais estruturas da linguagem SAS, como o Program Data Vector, o comando MERGE e os BY Groups, entre outros. Para que você possa conhecer mais sobre a linguagem (e outros recursos do Sistema SAS) serão relacionados alguns *sites* que representam importantes fontes de informação.

- **SAS 9.4 Help and Documentation:** esta é a documentação oficial do Sistema SAS release 9.4, que também é conhecida como **SAS OnLineDOC**. O link para acesso é: support.sas.com/documentation/94/. Pelo fato de o SAS representar um software muito sofisticado, por vezes a navegação por este site torna-se um verdadeiro desafio. A quantidade de informações disponibilizadas é muito grande e nem sempre é fácil achar o que realmente desejamos.
- **SAS Global Forum:** conferência anual que objetiva a troca de experiências entre usuários SAS e a publicação de artigos sobre diversos temas relacionados ao produto. Alguns exemplos: artigos sobre o uso do comando MODIFY, comparações entre a PROC SQL e o MERGE, uso do EG, aplicações do SAS em diversas áreas, etc. A cada ano são produzidos mais de 200 artigos. Anteriormente a conferência era chamada de SUGI. O link do SAS Global Fórum disponibiliza gratuitamente todos os artigos publicados desde a conferência de 1997 (SUGI 22). Os papers publicados em nos diferentes anos do congresso podem ser acessados livremente em: <https://support.sas.com/events/sasglobalforum/previous/online.html>. Ao consultar as referências deste livro, você poderá observar que a grande maioria consiste em artigos publicados nesta conferência.
- **Outras Conferências:** além do SAS Global Forum, existem diversas conferências anualmente promovidas por grupos de usuários/programadores SAS. Alguns exemplos são a MISUG e PharmaSUG (o “SUG” vem de “SAS Users Group”).
- **SAS Technical Papers:** fornecem detalhes técnicos sobre como você pode executar uma tarefa. Dentre as várias categorias de papers existe uma dedicada inteiramente ao Base SAS. Link para acesso: <http://support.sas.com/resources/papers/index.html>.
- **Sites de Universidades:** o Sistema SAS é utilizado em diversas universidades em todo o mundo. Muitas delas disponibilizam tutoriais bastante interessantes sobre o SAS.
- **FALE COM O AUTOR:** caso você deseje esclarecer dúvidas sobre tópicos deste livro ou sobre o uso da linguagem SAS em geral, sinta-se livre para entrar em contato com o autor: Eduardo Corrêa Gonçalves (eduardo.ence@gmail.com).

REFERÊNCIAS BIBLIOGRÁFICAS

BEE, B. Jazz it up a little with formats. In: SAS Global Forum, 2014, Washington D. C. **Anais...** Washington D. C.: SAS, 2014, paper 1495-2014.

BILENAS, J. V. Making sense of PROC TABULATE. In: SUGI 30, 2005, Filadélfia. **Anais...** Filadélfia: SAS, 2005, paper 243-30.

BLOOM, J. Everything you wanted to know about MERGE but were afraid to ask. **SAS Technical Support**. Disponível em: < <https://support.sas.com/techsup/technote/ts644.pdf> > TS-DOCS, TS-664, 2000.

BOST, C. J. For coders only: the SAS enterprise guide experience. In: SAS Global Forum, 2014, Washington D. C. **Anais...** Washington D. C.: SAS, 2014, paper 1441-2014.

BUCHECKER, M. Kicking and screaming your way to SAS enterprise guide. In: SAS Global Forum, 2016, Las Vegas. **Anais...** Las Vegas: SAS, 2016, paper 9000-2016.

BURLEW, M. M. **SAS macros made easy**. 3 ed. Cary: SAS Press, 2014.

CARPENTER, A. L. Are you missing out? Working with missing values to make the most of what is not there. In: PharmaSUG, 2014, San Diego. **Anais...** San Diego: PharmaSUG, 2014, paper TT02.

CHENG, W. Using the SAS windowing environment for program development. In: PharmaSUG, 2004, San Diego. **Anais...** San Diego: PharmaSUG, 2004, paper TT13.

CODY, R. An introduction to SAS character functions. In: SUGI 31, 2006, São Francisco. **Anais...** São Francisco: SAS, 2006, paper 247-31.

CODY, R. **An Introduction to SAS University Edition**. Cary: SAS Press, 2015.

CODY, R. A survey of some of the most useful SAS functions. In: SAS Global Forum, 2017, Orlando. **Anais...** Orlando: SAS, 2017, paper 269-2017.

CRAWFORD, P. More hidden Base SAS features to impress your colleagues. In: SAS Global Forum, 2016, Las Vegas. **Anais...** Las Vegas: SAS, 2016, paper 2120-2016.

CURTIS, M. Improve database transaction processing using MODIFY, the most under-appreciated data step file handling statement. In: SUGI 31, 2006, São Francisco. **Anais...** São Francisco: SAS, 2006, paper 264-31.

DEGUIRE, Y. The FILENAME statement revisited. In: SAS Global Forum, 2007, Orlando. **Anais...** Orlando: SAS, 2007, paper 007-2007.

DITOMMASO, D. Taking control of macro variables. In: SUGI 28, 2003, Seattle. **Anais...** Seattle: SAS, 2003, paper 95-28.

DROOGENDYK, I. Joining data: data step MERGE or SQL?. In: SAS Global Forum, 2008, San Antonio. **Anais...** San Antonio: SAS, 2008, paper 178-2008.

EBERHARDT, P. Can you read into SAS for me? Using INFILE and INPUT to load data into SAS. In: SAS Global Forum, 2016, Las Vegas. **Anais...** Las Vegas: SAS, 2016, paper 11840-2016.

ETL-Tools.Info. SAS tutorial: generation of a sample business datawarehouse scenario. **ETL-Tools.Info**, 2008. Disponível em: <http://etl-tools.info/en/sas_tutorial/tutorial.htm>. Acesso em: 02 abr. 2017.

FIRST, S. The SAS INFILE and FILE statements. In: SAS Global Forum, 2008, San Antonio. **Anais...** San Antonio: SAS, 2008, paper 166-2008.

FOLEY, M. J. Missing values: everything you ever wanted to know In: SESUG 2005, Portsmouth. **Anais...** Portsmouth: SESUG, 2005, paper TU-06.

GALSTER, D. SAS tutorial. **Dept. of Math and Statistics**, South Dakota State University, 2006. Disponível em: <http://learn.sdstate.edu/Dwight_Galster/>. Acesso em: 13 jun. 2017.

GLASS, R.; HADDEN, L. S. Document and enhance your SAS code, datasets, and catalogs with SAS functions, macros, and SAS metadata. In: SAS Global Forum, 2016, Las Vegas. **Anais...** Las Vegas: SAS, 2016, paper 8300-2016.

GO, I. C.; TAVAKOLI, A. S. Getting out of the PROC PRINT comfort zone to start using PROC REPORT. In: SESUG 2013 Conference, St. Pete Beach. **Anais...** St. Pete Beach: SAS, 2013, paper PO-4.

HADDEN, L. S. Build your metadata with PROC CONTENTS and ODS output. In: SAS Global Forum, 2014, Washington D. C. **Anais...** Washington D. C.: SAS, 2014, paper 1549-2014.

HARRINGTON, T. J. An introduction to SAS PROC SQL. In: SUGI 27, 2002, Orlando. **Anais...** Orlando: SAS, 2002, paper 70-27.

HAWORTH, L. Anyone can learn PROC TABULATE. In: SUGI 27, 2002, Orlando. **Anais...** Orlando: SAS, 2002, paper 60-27.

HECHT, D. PROC PRINT and ODS: teaching an old proc new tricks. In: SAS Global Forum, 2011, Las Vegas. **Anais...** Las Vegas: SAS, 2011, paper 270-2011.

HOLLAND, P.. Frequent asked questions about SAS software. **Holland Numerics Inc.**, 2008. Disponível em: <<http://www.hollandnumerics.co.uk/sasfaq/SASFAQ1.HTM>>. Acesso em: 30 mai. 2017.

HORSTMAN, J. M. Let the CAT out of the bag: string concatenation in SAS 9. In: SAS Global Forum, 2016, Las Vegas. **Anais...** Las Vegas: SAS, 2016, paper 11666-2016.

HORSTMAN, J. M. The fantastic four: running your report using the TABULATE, TRANSPOSE, REPORT or SQL procedure. In: SAS Global Forum, 2016, Las Vegas. **Anais...** Las Vegas: SAS, 2016, paper 6124-2016.

HORWITZ, L. A guide to SAS for it organization. In: SAS Global Forum, 2014, Washington D. C. **Anais...** Washington D. C.: SAS, 2014, paper SAS103-2014.

HOWARD, N. Introduction to SAS functions. In: SUGI 24, 1999, Miami Beach. **Anais...** Miami Beach: SAS, 1999, paper 57.

HOWARD, N. How SAS thinks or why the data step does what it does. In: SUGI 29, 2004, Montreal. **Anais...** Montreal: SAS, 2004, paper 252-29.

HU, W. Top ten reasons to use PROC SQL. In: SUGI 29, 2004, Montreal. **Anais...** Montreal: SAS, 2004, paper 042-29.

JOHNSON, J. The use and abuse of the program data vector. In: SAS Global Forum, 2012, Orlando. **Anais...** Orlando: SAS, 2012, paper 255-2012.

KARP, A.HS. Working with SAS date and time functions. In: SAS Global Forum, 2010, Seattle. **Anais...** Seattle: SAS, 2010, paper 134-2010.

KELLEY, F. J. Getting started with SAS/Access for Oracle. In: SESUG 2005, Portsmouth. **Anais...** Portsmouth: SESUG, 2005, paper ETL05-05.

KULIGOWSKI, A. T. In the search of lost card. In: SUGI 30, 2005, Filadélfia. **Anais...** Filadélfia: SAS, 2005, paper 263-30.

KULIGOWSKI, A. T. Using INFILE and INPUT statements to introduce external data into the SAS system. In: SESUG 2007, Hilton Head. **Anais...** Hilton Head: SESUG, 2007, paper HW07.

KULIGOWSKI, A. T.; MENDEZ, L. An introduction to SAS arrays. In: SAS Global Forum, 2016, Las Vegas. **Anais...** Las Vegas: SAS, 2016, paper 9000-2016.

LAFLEER, K. P. Hands-on SAS macro programming essentials for new users. In: SAS Global Forum, 2015, Dallas. **Anais...** Dallas: SAS, 2015, paper 6406-2015.

LAUDERDALE, K. PROC SQL – the dark side of SAS? In: SAS Global Forum, 2007, Orlando. **Anais...** Orlando: SAS, 2007, paper 071-2007.

MASON, P. Tricky aspects of the data step. In: SUGI 29, 2004, Montreal. **Anais...** Montreal: SAS, 2004, paper 121-29.

MILLER, E. One filp, two flip, fat file, flat file. In: SAS Global Forum, 2012, Orlando. **Anais...** Orlando: SAS, 2012, paper 039-2012.

MORGAN, D. Demystifying date and time intervals. In: SAS Global Forum, 2015, Dallas. **Anais...** Dallas: SAS, 2015, paper 2460-2015.

- PILI, S.; ALZOUBI, L.; NASSEN, K. Using the contents of PROC CONTENTS to perform multiple operations across a SAS data library. In: SUGI 27, 2002, Orlando. **Anais...** Orlando: SAS, 2002, paper 84-27.
- POPE, L. PROC PRINT to be proud of. In: VIEWS Conference, 2003, London. **Anais...** London: LexJansen, 2003, tutorial 05.
- SAS Institute Inc. About SAS: giving you the power to know[®] since 1976. **SAS**, 2017. Disponível em: <https://www.sas.com/en_us/company-information.html#history>. Acesso em: 02 abr. 2017.
- SAS Institute Inc. **Base SAS[®] 9.4 procedures guide**. 7 ed. Cary: SAS Institute Inc., 2017.
- SAS Institute Inc. **SAS/ACCESS[®] 9.4 for relational databases**: reference. 9 ed. Cary: SAS Institute Inc., 2016.
- SAS Institute Inc. **SAS/ACCESS[®] 9.4 interface to PC files**: reference. 4 ed. Cary: SAS Institute Inc., 2016.
- SAS Institute Inc. **SAS fundamentals**: a programming approach course notes. Cary: SAS Institute Inc., 1996.
- SAS Institute Inc. **SAS[®] 9.4 functions and call routines**: reference. 5 ed. Cary: SAS Institute Inc., 2016.
- SAS Institute Inc. **SAS[®] 9.4 macro language**: reference. 5 ed. Cary: SAS Institute Inc., 2016.
- SAS Institute Inc. **SAS[®] 9.4 language**: concepts. 6 ed. Cary: SAS Institute Inc., 2016.
- SAS Institute Inc. **SAS[®] 9.4 SQL procedure**: user's guide. 4 ed. Cary: SAS Institute Inc., 2016.
- SCERBO, M.; LAJINESS, M. PROC FREQ and MEANS: to stat or not to stat. In: SUGI 30, 2005, Filadélfia. **Anais...** Filadélfia: SAS, 2005, paper 263-30.
- SCHREIER, H. Now _INFILE_ is an automatic variable – so what?. **NESUG**, 2001. Disponível em: <<http://www.lexjansen.com/nesug/nesug01/cc/cc4018bw.pdf>>. Acesso em: 13 mai. 2017.
- SHEN, D.; LU, Z. Computation of correlation coefficient and its confidence interval in SAS. In: SUGI 31, 2006, São Francisco. **Anais...** São Francisco: SAS, 2006, paper 170-31.
- SLAUGHTER, S. J.; DELWICHE, L. D. SAS macro programming for beginners. In: SUGI 29, 2004, Montreal. **Anais...** Montreal: SAS, 2004, paper 243-29.
- SLAUGHTER, S. J.; DELWICHE, L. D. **The little SAS Book for Enterprise Guide 4.2**. Cary: SAS Press, 2010.

SLAUGHTER, S. J.; DELWICHE, L. D. **The little SAS Book**: a primer. 5 ed. Cary: SAS Press, 2012.

STROUPE, J. Adventures in arrays: a beginning tutorial. In: SAS Global Forum, 2007, Orlando. **Anais...** Orlando: SAS, 2007, paper 273-2007.

VIRGILE, B. Changing the shape of your data: PROC TRANSPOSE vs. arrays. In: SUGI 24, 1999, Miami Beach. **Anais...** Miami Beach: SAS, 1999, paper 60.

WHITLOCK, I. How to think through the SAS data step. In: SAS Global Forum, 2007, Orlando. **Anais...** Orlando: SAS, 2007, paper 208-2007.

WHITLOCK, I. SAS macro design issues. In: SUGI 30, 2005, Filadélfia. **Anais...** Filadélfia: SAS, 2005, paper 67-27.

WILLIAMS, C. S. PROC COMPARE – worth another look! In: SAS Global Forum, 2010, Seattle. **Anais...** Seattle: SAS, 2010, paper 149-2010.

ZIRBELL, D. Learn the basics of PROC TRANSPOSE In: SAS Global Forum, 2009, Washington, D.C. **Anais...** Washington, D.C.: SAS, 2009, paper 060-2009.

Sobre o Autor

Eduardo Corrêa Gonçalves cursou Doutorado em Ciência da Computação pela UFF (2015) com período sanduíche na University of Kent, no Reino Unido. Também cursou Mestrado (2004) e Graduação (1999) em Ciência da Computação pela UFF. Possui certificação Oracle Database SQL Certified Expert (OCE). Atualmente, trabalha como desenvolvedor de banco de dados no Instituto Brasileiro de Geografia e Estatística (IBGE) e também atua como professor colaborador na Escola Nacional de Ciências Estatísticas (ENCE-IBGE). Suas áreas de interesse são: Algoritmos e Banco de Dados.

Currículo Lattes: <http://buscatextual.cnpq.br/buscatextual/visualizacv.do?id=K4137241P3>