

FORMAÇÃO
MAINFRAME

COBOL

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

- 1. Introdução**
- 2. Formato do Fonte Cobol**
 - 2.1. Área de numeração seqüencial**
 - 2.2. Área de indicação**
 - 2.3. Área A**
 - 2.4. Área B**
- 3. Estrutura de um Programa Cobol**
 - 3.1. IDENTIFICATION DIVISION**
 - 3.2. ENVIRONMENT DIVISION**
 - 3.2.1. CONFIGURATION SECTION**
 - 3.2.2. INPUT-OUTPUT SECTION**
 - 3.3. DATA DIVISION**
 - 3.3.1. Especificação dos dados**
 - 3.3.1.1. Nível**
 - 3.3.1.2. Nome do dado Índice**
 - 3.3.1.3. Formato do dado**
 - 3.3.1.4. Valor inicial**
 - 3.3.2. Estrutura da DATA DIVISION**
 - 3.3.2.1. FILE SECTION**
 - 3.3.2.1.1. FILE DESCRIPTION (FD)**
 - 3.3.2.2. WORKING-STORAGE SECTION**
 - 3.3.2.3. LINKAGE SECTION**
 - 3.3.2.3.1. Utilização do parâmetro PARM**
 - 3.4. PROCEDURE DIVISION**
 - 3.4.1. Movimentação de dados**
 - 3.4.2. Cálculos aritméticos**
 - 3.4.2.1. Adição**
 - 3.4.2.2. Subtração**
 - 3.4.2.3. Multiplicação**
 - 3.4.2.4. Divisão**
 - 3.4.2.5. Resolução de fórmulas**
 - 3.4.3. Comandos condicionais**
 - 3.4.3.1. Formato dos comandos condicionais**

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

3.4.3.2. NEXT SENTENCE

3.4.4.3. CONTINUE

3.4.4.4. EXIT

3.4.3.5 Condições concatenadas

3.4.3.6 Condições compostas

3.4.4. Alterações/Desvios do fluxo do programa

3.4.4.1. Uso do comando GO TO

3.4.4.2. Uso do comando PERFORM

4. Arquivos

4.1. Registros

4.2. Abertura de arquivos

4.3. Leitura/gravação de arquivos

4.4. Fechamento de arquivos

4.5. Entrada e Saída de baixo volume

5. Encerramento do programa

6. Formatos especiais de dados

6.1. Sinal de campos numéricos.

6.2. Formatos binários

6.3. Cláusula USAGE

6.3.1. USAGE DISPLAY

6.3.2. USAGE COMP

6.3.3. USAGE COMP-3.

6.4. Tabelas

6.4.1. Níveis das tabelas

7. Impressão

7.1. Opção AFTER POSITIONING

7.2. Máscaras de Edição de Campos

7.3. Supressão de zeros

7.4. Tipos de Máscaras de Edição - OUTRAS

7.5. BLANK WHEN ZERO

8. Ordenação de Arquivos

8.1. Sort intrínseco

8.2. Sort extrínseco

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

9. Arquivos de acesso aleatório (VSAM)

9.1. Manipulação de Arquivos VSAM

9.1.1. INPUT-OUTPUT SECTION para VSAM

9.1.2. FILE STATUS

9.1.3. PROCEDURE DIVISION para VSAM

9.1.4. Abertura do arquivo.

9.1.5. READ

9.1.6. READ NEXT

9.1.7. READ PREVIOUS

9.1.8. START

9.1.9. WRITE.

9.1.10. REWRITE

9.1.11. DELETE

9.1.12. Fechamento de arquivos

10. Comunicação entre programas

11. Comandos Diversos

11.1. EVALUATE

11.2. REDEFINES

11.3. INSPECT

11.4. STRING

11.5. UNSTRING

11.6. TABELAS INDEXADAS

11.6.1. SET

11.6.2. SEARCH

11.6.2.1. Pesquisa seqüencial

11.6.2.2. Pesquisa binária

11.7. ALTER

11.8. GO TO DEPENDING ON

11.9. Nomes condicionais

11.10. Copy

12. Diagrama de Bloco

12.1. O que é um Diagrama de Bloco

12.2. Simbologias

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

1. INTRODUÇÃO

O Cobol é uma linguagem de alto nível criada no ano de 1959. A palavra Cobol é uma abreviação de: **Common Business Oriented Language**.

Como seu nome indica, o objetivo desta linguagem é permitir o desenvolvimento de aplicações comerciais para pessoas sem conhecimento profundo de computadores. Por isso a linguagem Cobol usa frases normais da língua inglesa, e a estrutura de um programa Cobol se assemelha a um texto com divisões, parágrafos e frases em inglês.

Depois de escrito o programa Cobol (ou programa fonte), é necessário traduzí-lo para a linguagem interna do computador (linguagem de máquina), convertendo-se então em um programa objeto. Esta conversão é feita pelo próprio computador, usando o programa compilador de Cobol.

Damos em seguida a definição de alguns termos importantes para o desenvolvimento do curso:

- Byte: conjunto de 8 bits (pontos magnéticos) que formam uma posição de memória.
- ASCII: tabela usada em micros para conversão do valor binário dos bits de um byte em um caracter.
- EBCDIC: tabela da IBM para conversão do valor binário dos bits de um byte em um caracter.
- Programa fonte: Texto de um programa escrito em uma linguagem de alto nível (ex.: Cobol).
- Programa objeto: Programa convertido para a linguagem interna do computador (linguagem máquina).
- Compilador: Conversor de programa fonte em programa objeto.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

2. FORMATO – FONTE COBOL

Todo programa escrito na linguagem Cobol possui algumas regras a serem seguidas.

Uma destas regras se refere ao formato das linhas de comando (instruções) dentro do seu editor de fonte.

Uma linha de comando Cobol pode ter até 80 caracteres, conforme o formato abaixo:

Seqüência	I	Área A	Área B	Comentário
1	6	7	8 11 12 72	73 80

Colunas de 1 a 6: Área de numeração seqüencial

Coluna 7: Área de indicação

Colunas de 8 a 11: Área A

Colunas de 12 a 72: Área B

Colunas de 73 a 80: Comentários

2.1. ÁREA DE NUMERAÇÃO SEQUENCIAL

Normalmente consiste em seis dígitos em ordem crescente que normalmente são utilizados para numerar as linhas do fonte. Segundo as regras no **ANS85** pode-se também colocar comentários nesta área. Além disso, podemos colocar um asterisco na coluna 1 (um) ou qualquer outro caractere com valor **ASCII** menor do que 20 (**ESPAÇO**), fazendo com que a linha inteira seja considerada como um comentário.

Pode-se também deixar esta área em branco.

2.2. ÁREA DE INDICAÇÃO

Um hífen (-) nesta posição indica que existe uma continuação de um texto que foi iniciado na linha anterior.

Um asterisco (*) nesta posição indica que toda a linha deve ser tratada como um comentário.

Uma barra (/) nesta posição, além de marcar a linha como comentário fará com que ao se imprimir este fonte, haverá um salto de página após esta linha.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

2.3. Área A

Posição a partir do qual se escreve nome de parágrafos.

2.4. Área B

Posição a partir da qual se escrevem as instruções Cobol.

Veja o exemplo de código fonte a seguir:

NUM	I	A	AREA B	COMEN.
000110	*		ESTE E UM PARAGRAFO	*PROG1*
000120		PARA	GRAFO-1.	*PROG1*
000130			ADD VALOR TO LUCRO.	*PROG1*

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

3. ESTRUTURA – PROGRAMA COBOL

De maneira semelhante a um livro com seus capítulos, parágrafos e itens, um programa Cobol é formado por uma hierarquia de Divisions, Sections, parágrafos e instruções.

Como regra de sintaxe, toda declaração de Division, Section, parágrafo ou instrução deve ser terminado por ponto final (.).

O código Cobol possui quatro divisões que devem ser utilizadas nesta ordem:

IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

DATA DIVISION.

PROCEDURE DIVISION.

Como foi explicado no item anterior (Formato do fonte) a declaração das **DIVISIONS** devem se iniciar na área A do texto (coluna 8).

3.1. IDENTIFICATION DIVISION

Esta é a divisão de identificação do programa. Não contém sections, mas somente alguns parágrafos preestabelecidos e opcionais. O único parágrafo obrigatório é o **PROGRAM-ID** (Nome do programa). O nome do programa deve ser uma palavra com até 8 caracteres (letras ou números), começando por uma letra.

Esta divisão possui a seguinte estrutura:

IDENTIFICATION	DIVISION.
PROGRAM-ID.	nome-programa.
AUTHOR.	comentário.
DATE-WRITTEN.	comentário.
DATE-COMPILED.	comentário.
SECURITY.	comentário.
REMARKS.	comentário.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

PROGRAM-ID (Program Identification) - deverá ser informado o nome do programa através do qual ele será identificado.

AUTHOR - cláusula opcional onde pode ou não constar o nome do autor do programa.

DATE-WRITTEN - cláusula opcional onde o desenvolvedor coloca a data em que o programa foi codificado.

DATE-COMPILED - cláusula opcional onde o compilador insere a data da compilação do programa.

SECURITY - cláusula opcional onde podem ou não constar informações sobre o acesso ao programa.

REMARKS - cláusula opcional onde normalmente se colocam observações sobre o programa. Como por exemplo, o histórico de suas atualizações e seus respectivos autores, ou a data de criação e objetivo do programa, etc.

IMPORTANTE:

Se houver a necessidade da inclusão de mais de uma linha com textos explicativos, deve-se utilizar um "*" (asterisco) na coluna 7 (sete). Desta forma a nova linha será tratada como comentário pelo compilador.

3.2. ENVIRONMENT DIVISION

Esta divisão descreve o equipamento envolvido no programa. Possui duas sections e sua estrutura é a seguinte:

ENVIRONMENT	DIVISION.
CONFIGURATION	SECTION.
SOURCE-COMPUTER.	comentário.
OBJECT-COMPUTER.	comentário.
SPECIAL-NAMES.	
INPUT-OUTPUT	SECTION.
FILE-CONTROL.	
I-O CONTROL.	

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

3.2.1. CONFIGURATION SECTION

Esta seção destina-se a uma descrição geral do ambiente do computador. Ela é composta por três parágrafos: **SOURCE-COMPUTER**, **OBJECT-COMPUTER** e **SPECIAL-NAMES** conforme mostrado acima.

SOURCE-COMPUTER - identifica o computador onde foi confeccionado o programa (opcional).

OBJECT-COMPUTER - identifica o computador do ambiente de produção (opcional).

SPECIAL-NAMES - tem comandos pré-definidos em Cobol, para especificar alfabeto, moeda ou separador de decimal (vírgula ou ponto). Porém todos os comandos são opcionais. O separador de decimais é o mais freqüentemente, indicando que vai se usar a vírgula para separar as casas decimais. A sintaxe correta é:

SPECIAL-NAMES.

DECIMAL-POINT

IS COMMA.

3.2.2 INPUT-OUTPUT SECTION.

Esta seção destina-se a configuração dos arquivos que o programa utilizará. Ela possui dois parágrafos: **FILE-CONTROL** e **I-O-CONTROL**.

FILE-CONTROL - neste parágrafo devemos especificar cada um dos arquivos que o programa irá utilizar, de forma individualizada.

I-O-CONTROL - contém vários parágrafos opcionais para especificar opções de uso de fitas magnéticas. Como este dispositivo caiu em desuso, este parágrafo não é mais usado atualmente.

No parágrafo **FILE-CONTROL**, usamos uma instrução **SELECT** para cada arquivo descrito. A sintaxe correta é:

SELECT *nome-do-arquivo* **ASSIGN TO** *dispositivo-do-computador*.

No exemplo abaixo, mostraremos a **ENVIRONMENT DIVISION** de um programa que irá acessar um arquivo chamado **CLIENTES**.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

```
....  
ENVIRONMENT          DIVISION.  
CONFIGURATION        SECTION.  
SPECIAL-NAMES.  
    DECIMAL-POINT    IS COMMA.  
INPUT-OUTPUT         SECTION.  
FILE-CONTROL.  
    SELECT CLIENTES ASSIGN TO DA-S-CLIENTES.  
....
```

O programador pode usar qualquer palavra com até 30 caracteres como *nome-de arquivo*, mas uma vez definido, este deverá ser usado igualmente em todos os pontos do programa quando se referir a este arquivo – nome interno.

O formato da cláusula *dispositivo-do-computador* varia conforme o computador (micro, mainframe, etc), mas no caso do mainframe usa-se o formato mostrado no exemplo, composto de 3 segmentos, separados por hífen:

Tipo de dispositivo:

UR - para dispositivos de registro fixo (impressoras, cartão).

UT - para dispositivos de registro variável (fitas).

DA - para dispositivos de acesso aleatório (discos).

Modo de acesso:

S – Sequencial.

D – Direto (Randômico).

I – Indexado.

Nome externo do arquivo – nome pelo qual o operador do computador reconhece o arquivo. O nome externo geralmente está associado aos comandos JCL na execução do programa.

Para arquivos abertos para leitura (veja **OPEN INPUT** e **OPEN I-O**), pode-se especificar a cláusula **OPTIONAL** no **SELECT**. Com esta cláusula, se ao tentarmos abrir o arquivo e este não existir, ele é automaticamente criado vazio.

Sintaxe do **SELECT** com **OPTIONAL**:

SELECT OPTIONAL *nome-do-arquivo* **ASSIGN TO** *dispositivo*.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

3.3. DATA DIVISION

A **DATA DIVISION** é a divisão do programa onde são definidos os dados, incluindo todas as variáveis e constantes necessárias, assim como o layout dos registros dos arquivos.

A definição de um dado deve estar no seguinte formato:

Nível *nome-do-dado* **formato** **valor-inicial.**

3.3.1. ESPECIFICAÇÃO DOS DADOS

3.3.1.1. NÍVEL

O nível é um número que varia de 01 a 99 e que define a hierarquia do dado em relação aos outros dados.

Podem-se agrupar os dados e concatená-los formando hierarquias como por exemplo:

```
01  WRK-DATA-NASCIMENTO.  
    03  WRK-DIA-NASCIMENTO.  
        03  WRK-MES-NASCIMENTO.  
            03  WRK-ANO-NASCIMENTO.
```

Neste exemplo, **DIA MÊS** e **ANO-NASCIMENTO** são partes integrantes do item de grupo **DATA-NASCIMENTO**. Os níveis 01 e 03, neste exemplo, são usados para determinar esta hierarquia.

Os níveis de 50 a 99 têm uso específico, reservado para futuras expansões do Cobol. Em mainframe usam-se dois níveis nesta faixa:

Nível 77 - usado quando o dado não tem hierarquia (item independente).

Nível 88 - usado para nomes condicionais.

IMPORTANTE:

De todos estes níveis, somente o nível 01 e o nível 77 podem ser codificados na margem A do programa (coluna 8). Os demais níveis devem ser codificados a partir da margem B (coluna 12).

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

REGRA:

Na codificação de programa, é exigido que toda a hierarquia de um item de grupo inicie com um item de nível 01. A partir dele, cada nível subordinado deve ter o mesmo número, por exemplo, se para o segundo nível for escolhido o número 3 todos os itens do segundo nível devem ter nível 03, e se o terceiro nível tiver o número 10, todos os itens do terceiro nível devem ter nível 10, e assim sucessivamente.

3.3.1.2. NOMEAÇÃO DO DADO

Pode-se utilizar qualquer palavra – de no máximo 30 caracteres -- incluindo letras, números e hífen, sendo que pelo menos uma deve ser letra.

Este nome deverá ser usado em todos os pontos do programa a que se referir a este dado. Se não for necessário realizar uma referência no programa deste dado, o nome poderá ser omitido. Porém, para o **COBOL** ele é obrigatório. Neste caso, usa-se então a palavra **FILLER** que identifica dados anônimos.

Por exemplo:

01 **WRK-INICIO-PGM** – este dado não será utilizado no restante do programa.

Poderia substituir por:

01 FILLER

3.3.1.3. FORMATO DO DADO

O formato dos dados é especificado pela palavra reservada **PICTURE**, ou pela sua abreviação **PIC**.

A cláusula **PICTURE** é usada para descrição de informações sobre itens, tais como: tamanho, sinal, tipo (numérico, alfanumérico ou alfabético).

Tipos possíveis:

ALFABÉTICO - é representado por letras mais o espaço, e o caractere usado é a letra **"A"**.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Por exemplo:

77 WRK-DADO1 PICTURE IS AAA VALUE 'ABC'.

77 WRK-DADO2 PIC IS AAA VALUE 'ABC'.

77 WRK-DADO3 PIC A(3) VALUE 'BCD'.

ALFANUMÉRICO - é representado por letras, números e caracteres. O caractere usado é a letra **"X"**. A representação de dados não pode exceder a 120 caracteres.

Por exemplo:

01 WRK-DADO1 PIC XXX VALUE 'ANO'.

01 WRK-DADO2 PIC X(04) VALUE 'KKKK'.

NUMÉRICO - usa-se para representação exclusiva de itens numéricos. Os caracteres usados são: **"9"**, **"V"**, **"P"**, **"S"**.

"9" = é utilizado para indicar a posição do campo que contém um dígito de "0" a "9".

"V" = é usado para mostrar a posição da vírgula decimal. O ponto decimal, se colocado, não faz parte do item.

"P" = representa um dígito numérico zero (0).

"S" = indica a presença de sinal, deve ser colocado antes do "9"

A quantidade de caracteres contido no dado é especificado no formato repetindo-se as letras acima.

Por exemplo, se o item **WRK-QUANT-PROD** tem 5 algarismos, seu formato deve ser:

77 WRK-QUANT-PROD PIC 99999.

OU

77 WRK-QUANT-PROD PIC 9(05).

Quando um item numérico tiver casas decimais, não se carrega na memória o separador decimal (vírgula).

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

O item **WRK-VALOR-PROD** por exemplo, tem um valor de 2,35. O armazenamento na memória fica como 235, mas no programa é preciso saber em que posição estaria a vírgula que desapareceu (vírgula implícita -- definida no formato pela letra V). Portanto, teríamos que definir o dado conforme abaixo:

77	WRK-VALOR-PROD	PIC 99999V99.
	OU	
77	WRK-VALOR-PROD	PIC 9(05)V99.

IMPORTANTE:

Em um item de grupo, o nível principal não deve ser definido com a cláusula **PICTURE**. Somente os níveis pertencentes a ele devem ser definidos (itens elementares).

Por exemplo:

01	WRK-FUNCIONARIO. – Sem definição de formato	
03	WRK- NASCIMENTO. – Sem definição de formato	
05	WRK-MES	PIC 99.
05	WRK-ANO	PIC 99.

O Cobol suporta itens numéricos com até 18 algarismos e itens alfanuméricos com até 32768 caracteres (dependendo do sistema operacional).

Existem ainda formatos especiais da PIC para itens a serem impressos ou exibidos na tela, que serão vistos mais adiante.

3.3.1.4. VALOR INICIAL DO DADO

Esta cláusula é opcional. Seu objetivo é definir um valor para o item quando o programa se inicia. Se ela for omitida, o item correspondente terá valores imprevisíveis. No caso de um item que irá conter uma totalização, por exemplo, é conveniente que ele seja **INICIALIZADO** com o valor zero.

A sintaxe para o valor inicial é:

77	WRK-QUANT-PROD	PIC 99999 VALUE literal.
-----------	-----------------------	---------------------------------

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Para os dados numéricos a inicialização deve ser:

```
77 WRK-IDADE-MINIMA PIC 99 VALUE 18.
```

Isto significa que quando o programa for iniciado, a variável **WRK-IDADE-MINIMA** conterá o valor **18**.

Para os dados alfanuméricos a inicialização deve ser inserida entre aspas:

```
77 WRK-NOME-RUA PIC X(20) VALUE 'RUA FIDALGA'.
```

Isto significa que quando o programa for iniciado, a variável **WRK-NOME-RUA** conterá o valor **RUA FIDALGA**.

Os literais alfanuméricos podem ter no máximo 120 caracteres.

Podem-se ainda usar como valor inicial as **CONSTANTES FIGURATIVAS: ZEROS, SPACES, HIGH-VALUES** ou **LOW-VALUES**.

O **HIGH-VALUES** – maior valor possível assumido pela variável - e **LOW-VALUES** – menor valor possível assumido pela variável - só poderão ser utilizados em variáveis do tipo alfanuméricas ou item de grupo (também consideradas alfanuméricas).

Este tipo de inicialização é largamente utilizada para programas que executam balance-line (batimento entre arquivos).

IMPORTANTE:

LOW-VALUES é diferente de espaços -- indica que o conteúdo da variável na memória deve ter os seus bytes com todos os bits desligados.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Exemplos de inicialização em hexadecimal:

INICIALIZAÇÃO COM	TAMANHO VARIÁVEL	FORMATO HEXADECIMAL
ZEROS	PIC 9(03)	FFF
		000
SPACES	PIC X(03)	444
		000
LOW-VALUES	PIC X(03)	000
		000
HIGH-VALUES	PIC X(03)	FFF
		FFF

3.3.2. ESTRUTURA - DATA DIVISION

Na **DATA DIVISION** é especificado o layout do registro de cada arquivo – entrada/saída utilizados no programa. Também serve para declaração das áreas de trabalho e constantes necessárias para o processamento dos dados.

A **DATA DIVISION** é composta por quatro seções:

FILE	SECTION.
WORKING-STORAGE	SECTION.
REPORT	SECTION.
LINKAGE	SECTION.

3.3.2.1. FILE SECTION

A **FILE SECTION** é usada para definir o layout / conteúdo do registro do arquivo que o programa irá ler / gravar.

Vimos anteriormente, que na **INPUT-OUTPUT SECTION (ENVIRONMENT DIVISION)**, que para cada arquivo a ser tratado no programa deverá haver uma instrução **SELECT** especificando e definindo um nome para o arquivo. Na **FILE SECTION** precisamos definir a estrutura de cada um destes arquivos. Isto é feito através do parágrafo **FILE DESCRIPTION (FD)**.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

3.3.2.1.1. FILE DESCRIPTION (FD)

É a descrição do arquivo.

FD NOME-DO-ARQUIVO

BLOCK CONTAINS (quantidade de blocos)

RECORD CONTAINS (tamanho do registro lógico)

RECORDING MODE (formato do arquivo)

LABEL RECORD (formato do label)

DATA RECORD (nome do registro).

Parâmetros:

BLOCK CONTAINS - Especifica o tamanho do registro físico.

Ex.: **BLOCK CONTAINS 9999 RECORDS**



Se for colocado zeros (0), assume informações do cartão **"DD"** - **JCL**.

Se não for colocado **"RECORDS"**, assume **"CHARACTERS"**.

RECORD CONTAINS - Especifica o tamanho do registro lógico.

Ex.: **RECORD CONTAINS 9999 CHARACTERS**



Se esta cláusula for colocada, é feita uma conferência pelo compilador, somando a quantidade de bytes da definição do registro.

RECORDING MODE – Especifica o formato do arquivo

Ex.: **RECORDING MODE IS X**

Onde **"X"** poderá ser definido com:

(F) -> comprimento fixo

(V) -> comprimento variável

(U) -> indefinido

(S) -> estendido (spanned)

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Se não for colocada a cláusula **"RECORDING MODE"**, o compilador determinará pelo cartão **"DD"** – **JCL** ou catálogo.

LABEL RECORD - Especifica o formato do label.

Ex.: **LABEL RECORD IS XXXXXXXX**

Onde **"XXXXXXXX"** poderá ser definido com:

STANDARD -> padrão

OMITTED -> omitido

Quando omitido assume **"LABEL STANDARD"**.

Para impressora, leitora de cartões, perfuradoras, usar **"OMITTED"**, pois não possuem **"LABELS"**. Os demais casos usar **"STANDARD"**.

DATA RECORD - Serve apenas como documentação, identificando os registros do arquivo pelo nome.

Ex.: **DATA RECORD IS NOME-DO-DADO-1** ou **DATA RECORD ARE NOME-DADO-1, NOME-DADO-2, ...**

CLÁUSULA FILLER

É usada para um item elementar ou um item de grupo, e nunca será refenciado. Pode ser usada na **"DATA DIVISION"** e suas **"SECTIONS"**.

Exemplo:

```
01  REGISTRO.  
    02  FILLER                PIC X(100).
```

CLÁUSULA VALUE

É usada para definir um valor inicial para um item da **"WORKING-STORAGE SECTION"**. Não pode ser usada na **"FILE SECTION"**.

Exemplo:

```
01  CABEC01.  
    02  FILLER                PIC X(10)      VALUE SPACES.  
    02  FILLER                PIC X(06)      VALUE 'FOLHAS'.  
    02  CB01-DATA             PIC X(10)      VALUE SPACES.
```

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

O arquivo **CLIENTES** definido na **INPUT-OUTPUT SECTION**, seria detalhado aqui como:

DATA	DIVISION.
FILE	SECTION.
FD CLIENTE	
RECORDING MODE	IS F
LABER RECORDING	IS STANDARD
BLOCK CONTAINS	0 RECORDS
RECORD CONTAINS	80 CHARACTERS
DATA-RECORD	IS REG-CLIENTES.
01 REG-CLIENTES.	
03 COD-CLIENTE	PIC 9(8).
03 NOME-CLIENTE	PIC X(20).
03 ENDER-CLIENTE	PIC X(40).
03 VALOR-CLIENTE	PIC 9(10)V99.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

3.3.2.2. WORKING-STORAGE SECTION

A **WORKING-STORAGE SECTION** é usada para definir todas as variáveis necessárias ao funcionamento do programa.

Não há parágrafos nesta seção, e os dados podem ser definidos como itens de grupo (**níveis 01 a 49**), ou itens elementares (**nível 77**).

IMPORTANTE:

Não poderá haver quebra na seqüência da declaração de níveis. Primeiramente, devem ser definidas todas as variáveis elementares (**nível 77**) e após os itens de grupo (**nível 01 a 49**). Portanto, após o uso do nível **01** não é recomendável voltar a declarar variáveis com nível **77**.

3.3.2.3. LINKAGE SECTION

Usada para recebimento de parâmetros -- estabelecer comunicação -- entre dois programas ou recebimento de parâmetros via **JCL (PARM)**.

3.3.2.3.1. UTILIZAÇÃO DO PARM

Para acessar dados informados através do **PARM**, declarar na **LINKAGE SECTION** uma estrutura com:

- Uma variável **PIC S9(04) COMP** que recebe o tamanho dos dados.
- Em continuação, outra variável que irá conter, em tempo de execução, os dados informados através do **PARM**; ela deve ter tamanho suficiente para receber todos os dados.

LINKAGE SECTION.

01	LKG-DATA-VIA-PARM.	
03	LKG-DATA-PARM-TAM	PIC S9(04) COMP.
03	LKG-DATA-1	PIC X(10).
03	LKG-DATA-2	PIC X(08).

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

```
*=====*
PROCEDURE DIVISION                                USING LKG-DATA-VIA-PARM.
*=====*

DISPLAY LKG-DATA-PARM-TAM.
DISPLAY LKG-DATA-1 ' ' LKG-DATA-2.
```

Se o programa for executado com **PARM='01/01/200331/12/03'** ele exibirá

00018

01/01/2003

31/12/03

Note que o tamanho da variável colocado em **LKG-DATA-PARM-TAM** inclui somente o tamanho dos dados em si, sem incluir os **2 bytes** de **LKG-DATA-PARM-TAM**.

3.4. PROCEDURE DIVISION

A última divisão controla a execução do programa, e é onde colocaremos os comandos oriundos do algoritmo planejado pelo programador.

Os comandos (instruções) do Cobol são formados por um único verbo da língua inglesa, seguido dos parâmetros necessários. As instruções podem ser reunidas em parágrafos, e estes em seções, definidas pelo programador com o fim de tornar o programa mais fácil de ser entendido.

Por exemplo, dentro de uma seção principal, posso definir as macro-seções, responsáveis por executar todos os procedimentos necessários para se atingir o objetivo final do programa:

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

0000-ROTINA-PRINCIPAL **SECTION.**

PERFORM 1000-ENTRADA. – SEGUNDA SEÇÃO DO PROGRAMA

PERFORM 2000- PROCESSAMENTO. – TERCEIRA SEÇÃO DO PROGRAMA

PERFORM 3000-SAIDA. – QUARTA SEÇÃO DO PROGRAMA

0000-99-FIM. **EXIT.**

3.4.1. MOVIMENTAÇÃO DO DADO

Para copiar dados de uma variável para outra se usa a instrução **MOVE**.

Sintaxe:

MOVE *variável-1* **TO** *variável-2.*

REGRAS:

- A variável receptora (variável-2) não pode ser uma literal.

Exemplo – utilização incorreta:

MOVE WRK-VALOR **TO 'RUA DIREITA'.**

- Tanto a variável receptora quanto a de origem deverão ter o mesmo formato, ou seja, deve-se mover conteúdo numérico para numérico, alfanumérico para alfanumérico.

Exemplo – utilização incorreta:

MOVE WRK-VALOR **TO NOME-CLIENTE.**

Quando a variável receptora tiver um tamanho menor que a variável emissora (de origem) do dado, ocorre um **TRUNCAMENTO** do dado para que este possa se adaptar ao novo item. Não é gerado nenhum aviso quando isto ocorre.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Quando a variável receptora tiver um tamanho maior que a emissora (de origem), a variável receptora irá automaticamente ser preenchida com espaços se ela for alfanumérica, ou com zeros à esquerda se ela for numérica.

Se a variável de origem movida for alfanumérica, o truncamento acontece na parte direita do campo (os dados são movidos da esquerda para a direita).

Se a variável de origem movida for numérica, o truncamento é à esquerda (os números são movidos da direita para a esquerda).

3.4.2. CÁLCULOS ARITMÉTICOS

As instruções para efetuar cálculos aritméticos são: **ADD, SUBTRACT, MULTIPLY, DIVIDE e COMPUTE.**

Quando o cálculo envolver variáveis/números com casas decimais, é preciso observar que o Cobol adapta o resultado ao formato que foi definido para a variável de resultado, ocasionando truncamento ou surgimento de zeros adicionais na parte decimal do resultado.

Por exemplo, se tivermos na WORKING-STORAGE a definição das variáveis abaixo:

77	WRK-QUANT	PIC 99V99	VALUE 35,12.
77	WRK-ENTRADA	PIC 99V9	VALUE 12,5.
77	WRK-SALDO	PIC 9999.	

Se somarmos **WRK-QUANT** e **WRK-ENTRADA** e colocarmos o resultado em **WRK-SALDO**, esta ficará com 47, pois esta variável não está prevendo o uso de casas decimais.

3.4.2.1. ADIÇÃO

Para efetuarmos a soma de valores utilizamos o comando **ADD**.

Formato 1:

ADD *valor-1 valor-2* **TO** *acum-1 acum2*

Neste formato, a soma dos valores das variáveis, valor-1 e valor-2, são acrescidos aos valores já existentes nas variáveis acum-1 e acum-2.

As variáveis, valor-1 e valor-2, podem ser literais numéricas. Já as variáveis, acum-1 e acum-2, só poderão ser variáveis numéricas – receberão o valor acumulado.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Por exemplo:

```
77 WRK-VAL-1          PIC 9(02)  VALUE 5.  
77 WRK-VAL-2          PIC 9(02)  VALUE 10.  
77 WRK-VAL-3          PIC 9(02)  VALUE 12.
```

ADD WRK-VAL-1 WRK-VAL-2 TO WRK-VAL-3.

Resultado após a execução da instrução acima:

WRK-VAL-3 = 27 (VALOR FINAL)

Formato 2:

ADD *valor-1 valor-2* **GIVING** *total*.

Neste formato a soma dos valores, valor-1 e valor-2, não são acumulados na variável total. A variável total é zerada antes de receber a soma, perdendo assim o valor acumulado no passado.

Por exemplo:

ADD WRK-VAL-1 WRK-VAL-2 GIVING WRK-VAL-3.

Resultado após a execução da instrução acima:

WRK-VAL-3 = 15 (VALOR FINAL)

Parâmetros opcionais -- podem ser adicionados a qualquer dos formatos acima:

ROUNDED - usado somente quando os operandos não forem números inteiros.

Se as variáveis que recebem o resultado do **ADD** não tiverem casas decimais suficientes, normalmente ocorre truncamento e o resultado perde as decimais. Com a cláusula **ROUNDED** é feito um **ARREDONDAMENTO** do resultado (valores menores que .5 são truncados, e os maiores são arredondados para cima).

ON SIZE ERROR – usado para os casos em que houver estouro do campo receptor do resultado.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Pode-se colocar nesta instrução uma mensagem de erro, parar o programa ou desviar para um parágrafo especial de tratamento de erro.

Por exemplo:

```
ADD WRK-VAL-1          TO WRK-VAL-2  
ON SIZE ERROR
```

```
DISPLAY 'ESTOUROU O CAMPO DE RESULTADO'
```

```
STOP RUN.
```

3.4.2.2. SUBTRAÇÃO

As subtrações são efetuadas com o comando **SUBTRACT**. De maneira análoga ao **ADD**, há dois formatos básicos:

Formato 1:

```
SUBTRACT valor-1 valor-2 FROM acum-1 acum2 ....
```

Neste formato, a soma dos valores das variáveis, *valor-1* e *valor-2*, são subtraídos aos valores já existentes nas variáveis *acum-1* e *acum-2*.

As variáveis, *valor-1* e *valor-2*, podem ser literais numéricas. Já as variáveis, *acum-1* e *acum-2*, só poderão ser variáveis numéricas – receberão o valor subtraído.

Formato 2:

```
SUBTRACT valor-1 valor-2 FROM acum GIVING total.
```

Neste formato a soma dos valores das variáveis, *valor-1* e *valor-2*, são subtraídos de *acum*, e o resultado é colocado em *total*. Os valores das variáveis, *valor-1*, *valor-2* e *acum*, não são alterados.

A variável *total* é zerada antes de receber o resultado.

Parâmetros opcionais -- podem ser adicionados a qualquer dos formatos acima:

ROUNDED

ON SIZE ERROR

Estes parâmetros têm funcionamento idêntico aos do comando **ADD**.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

3.4.2.3. MULTIPLICAÇÃO

O comando utilizado para operações de multiplicação é o **MULTIPLY**.
Há também 2 (dois) formatos:

Formato 1:

MULTIPLY *valor-1* **BY** *valor-2*.

Este formato multiplica a variável *valor-1* pela variável *valor-2*, e o resultado fica em *valor-2*.

IMPORTANTE:

Na multiplicação e divisão, o resultado fica sempre no último operando.

Assim, se quisermos multiplicar a variável **WRK-QUANT** por 3 (três) devemos executar a instrução:

MULTIPLY 3 BY WRK-QUANT.

Como consequência da regra do resultado, o último operando da multiplicação deve ser sempre uma variável. Portanto, seria incorreto executar a instrução abaixo:

MULTIPLY WRK-QUANT BY 3.

Formato 2:

MULTIPLY *valor-1* **BY** *valor-2* **GIVING** *valor-3*.

Neste formato o resultado da multiplicação é colocado em *valor-3* e os valores das variáveis, *valor-1* e *valor-2*, não são alterados.

Parâmetros opcionais -- podem ser adicionados a qualquer dos formatos acima:

ROUNDED

ON SIZE ERROR

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Estes parâmetros têm funcionamento idêntico aos do comando ADD.

3.4.2.4. DIVISÃO

O comando utilizado para operações de divisão é o DIVIDE.

Há também 3 (três) formatos:

Formato 1:

DIVIDE *valor-1* **INTO** *valor-2*.

Este formato divide a variável *valor-2* pela variável *valor-1* e o resultado é movido para *valor-2*. Assim como na multiplicação, o resultado da divisão fica sempre no último operando. Assim, se quisermos dividir a variável WRK-QUANT por 3 (três) devemos executar a instrução:

DIVIDE 3 **INTO WRK-QUANT.**

Como consequência da regra do resultado, o último operando da divisão deve ser sempre uma variável. Portanto, seria incorreto executar a instrução abaixo:

DIVIDE WRK-QUANT **BY 3.**

Formato 2:

DIVIDE *valor-1* **INTO** *valor-2* **GIVING** *valor-3*.

Neste formato o resultado da divisão é colocado em *valor-3* e os valores das variáveis, *valor-1* e *valor-2*, não são alterados.

Formato 3:

DIVIDE *valor-1* **BY** *valor-2* **GIVING** *valor-3*.

Neste formato o resultado da divisão de *valor-1* por *valor-2* é colocado em *valor-3* e os valores das variáveis, *valor-1* e *valor-2*, não são alterados.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Parâmetros opcionais -- podem ser adicionados a qualquer dos formatos acima:

ROUNDED

ON SIZE ERROR

Estes parâmetros têm funcionamento idêntico aos do comando **ADD**.

REMAINDER *valor-4*

Este parâmetro só pode ser usado com os formatos 2 e 3, e quando os operandos envolvidos forem números inteiros.

Sua função é carregar o resto da divisão em *valor-4*.

Sintaxe deste parâmetro junto ao formato 3:

DIVIDE *valor-1*

BY *valor-2*

GIVING *valor-3*

REMAINDER *valor-4*.

3.4.2.5. RESOLUÇÃO DE FORMULAS

O comando **COMPUTE** é utilizado para resolver fórmulas complexas.

Sintaxe:

COMPUTE *valor* = *fórmula*.

Fórmula pode ser qualquer expressão aritmética contendo os seguintes operandos:

- +** **Adicionar**
- **Subtrair**
- *** **Multiplicar**
- /** **Dividir**
- **** **Exponenciação**
- ()** **Parênteses**

Por exemplo, a fórmula $A = \sqrt{B^2 + C^2}$, deverá ser codificada:

COMPUTE A = (B ** 2 + C ** 2) ** (.5).

IMPORTANTE:

A exponenciação com expoentes decimais (no exemplo ****(.5)**) não funciona em todos os sistemas operacionais.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

3.4.3. COMANDOS CONDICIONAIS

Comandos condicionais são comandos executados dependendo de uma condição. A instrução que coordena a condição nos comandos condicionais é o **IF**. As condições testadas pela instrução IF são divididas em 3 categorias, e estão relacionadas na tabela abaixo:

Categorias	Símbolo	Notação alternativa
Relacionais	=	IS EQUAL
	>	IS GREATER THAN
	<	IS LESS THAN
Teste de sinal	Não tem	IS POSITIVE
	Não tem	IS NEGATIVE
	Não tem	IS ZERO
Teste de classe	Não tem	IS NUMERIC
	Não tem	IS ALPHABETIC

Testes de condição – exemplos:

IF WRK-CODIGO	IS EQUAL 5
IF WRK-VALOR	IS GREATER THAN ZEROS
IF WRK-SALDO	IS NEGATIVE
IF WRK-CODIGO	IS NUMERIC

Pode-se associar a palavra NOT com qualquer tipos de teste, criando-se assim testes de condição negativa. Por exemplo:

IF WRK-CODIGO	IS NOT EQUAL 5
IF WRK-VALOR	IS NOT GREATER THAN ZEROS
IF WRK-SALDO	IS NOT NEGATIVE
IF WRK-CODIGO	IS NOT NUMERIC

3.4.3.1. FORMATO DOS COMANDOS

Existem 2 (dois) formatos de comandos condicionais:

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Formato 1:

```
IF  condição-1
      Instruções-1
      Instruções-2
END-IF.
```

Neste formato, se a condição condição-1 for satisfeita, todas as instruções definidas internamente (instruções-1 e instruções-2) serão executadas.

Se condição-1 não for verdadeira, o programa não executa as instruções definidas internamente e retoma a execução a partir do ponto final (após o **END-IF**).

Com respeito à distribuição do texto do fonte, é necessário salientar os seguintes aspectos:

- O recuo maior usado para escrever instruções-1 (endentação) é importante tanto esteticamente quanto para um melhor entendimento do fonte.
- Procurar sempre escrever uma instrução por linha.
- O ponto final é fundamental em comandos condicionais, pois indica o fim das instruções sob influência do teste.

Exemplos de uso de comandos condicionais:

```
.....
77 WRK-VALOR                PIC 9(4)    VALUE 50.
.....
IF WRK-VALOR                GREATER 100
      DISPLAY 'O VALOR É MUITO ALTO'
      MOVE ZEROS            TO WRK-VALOR
END-IF.

MOVE WRK-VALOR            TO WRK-SAIDA.
```

No exemplo acima a variável **WRK-SAIDA** será formatada com valor 50 porque a condição do **IF** não foi atendida.

Se o ponto final estivesse após o comando **DISPLAY**, a instrução **MOVE ZEROS TO WRK-VALOR** seria executada sempre, independente do **IF**. Neste caso a variável **WRK-SAIDA** será formatada com zeros.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Formato 2:

```
IF  condição-1  
    Instruções-1  
ELSE  
    Instruções-2  
END-IF.
```

Instruções-3.

Neste formato, quando a *condição-1* for satisfeita, as *instruções-1* serão executadas, caso contrário, as *instruções-2* serão executadas.

O ponto final marca o fim do comando condicional (**IF/ELSE/END-IF**).

As instruções-3, que estão após o ponto final, serão sempre executadas independente da *condição-1*.

Exemplo incorreto de comando condicional:

```
IF  WRK-VALOR              GREATER 1000  
    MOVE 'VALOR ACIMA DO LIMITE' TO WRK-MENSAGEM.  
ELSE  
    MOVE 'VALOR ABAIXO DO LIMITE' TO WRK-MENSAGEM  
END-IF.  
  
MOVE WRK-MENSAGEM        TO WRK-SAIDA.
```

Finaliza o comando
IF

A instrução **ELSE** ficará fora da influência do **IF**, devido ao ponto final colocado após o primeiro comando **MOVE**. O compilador emitirá uma mensagem de erro de sintaxe.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

3.4.3.2. NEXT SENTENCE

O comando **NEXT SENTENCE** pode ser usado associado aos comandos condicionais quando não há nenhuma instrução a ser executada.

Por exemplo:

```
IF  WRK-VALOR          GREATER 10
  NEXT SENTENCE
ELSE
  ADD 10                TO WRK-VALOR
END-IF.
```

O **NEXT SENTENCE** direciona a execução para a primeira instrução após o ponto final.

3.4.3.3. CONTINUE

Não efetua processamento, servindo somente para contexto.

Usado em geral quando o teste sendo efetuado não tem processamento específico para a condição satisfeita.

Exemplos:

```
MOVE SPACES TO TEXTO.
IF TEXTO EQUAL SPACES
  CONTINUE
ELSE
  DISPLAY "NAO PASSA POR AQUI".

IF CAMPOA GREATER THAN CAMPOB
  CONTINUE
ELSE
  COMPUTE CAMPOC = CAMPOA - CAMPOB.

TESTA-SIGLA-REGIAO-SUL.
  IF SIGLA = "SP" CONTINUE
```

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

```
ELSE
    IF SIGLA = "PR" CONTINUE
ELSE
    IF SIGLA = "SC" CONTINUE
ELSE
    IF SIGLA = "RS" CONTINUE
ELSE
    PERFORM ERRO1 THRU SAI-ERRO1.
SIGLA-OK.  EXIT.
```

3.4.4.4. EXIT

É um ponto comum de finalização para uma série de procedimento(s).

NOME-PARAGRAFO. EXIT.

A cláusula **"EXIT"** deve ser precedida por um nome de parágrafo e deve ser única cláusula do parágrafo.

Em um programa poderá ter vários **EXIT's** associados com **PERFORM's**.

3.4.4. INITIALIZE

Com este comando efetuamos a inicialização de um área de trabalho, ele tem a finalidade de **limpar/inicializar** uma determinada área de trabalho ou de um determinado arquivo, com um único dado, previamente determinado, **sem** ter que usarmos o comando **MOVE**, usando apenas o comando **INITIALIZE**.

Por exemplo:

INITIALIZE AREA-TRABALHO

REPLACING NUMERIC DATA BY 7

DISPLAY 'TROCANDO OS CAMPO NUMERICOS POR 7 = ' AREA-TRABALHO.

INITIALIZE AREA-TRABALHO

REPLACING ALPHANUMERIC DATA BY '%'

DISPLAY 'TROCANDO OS CAMPO ALFA POR % = ' AREA-TRABALHO.

INITIALIZE AREA-TRABALHO

DISPLAY 'LIMPANDO OS CAMPOS PELO DEFAULT = ' AREA-TRABALHO.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

3.4.5. CONDIÇÕES CONCATENADAS

As condições concatenadas acontecem quando no lugar das instruções subordinadas a um teste de condição coloca-se um segundo teste de condição, originando assim uma cadeia de testes subordinados um ao outro. Estas cadeias de testes são teoricamente indefinidas, porém o aumento de testes concatenados aumenta a complexidade da lógica.

Nas condições concatenadas é importante observar o uso da endentação, como forma de aumentar sua clareza.

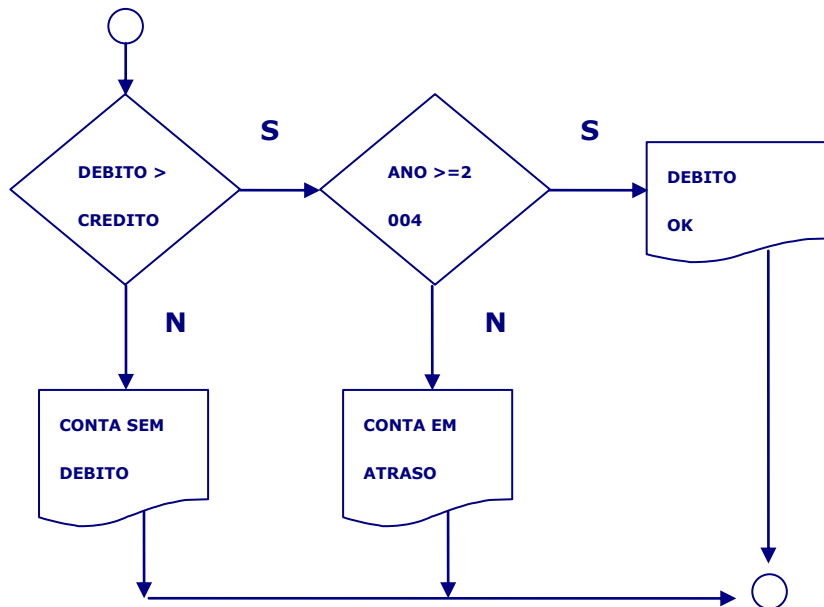
No exemplo abaixo, mostramos condições concatenadas:

```
IF  WRK-DEBITO          GREATER WRK-CREDITO
  IF  WRK-ANO           NOT LESS 2004
    MOVE 'DEBITO OK'      TO WRK-MENSAGEM
  ELSE
    MOVE 'CONTA EM ATRASO' TO WRK-MENSAGEM
  END-IF
ELSE
  MOVE 'CONTA SEM DEBITO'  TO WRK-MENSAGEM
END-IF.
```

O fluxograma que representa a condição acima é:

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação



3.4.6. CONDIÇÕES COMPOSTAS

Condições compostas são testes de condições ligados com as cláusulas **AND** ou **OR**.

Regras:

AND - a condição resultante de duas ou mais condições ligadas pela cláusula **AND** é verdadeira somente se todas as condições componentes forem verdadeiras.

OR - a condição resultante de duas ou mais condições ligadas pela cláusula **OR** é verdadeira se pelo menos uma das condições componentes for verdadeira.

Por exemplo:

```
IF WRK-NOTA          GREATER 5 AND
   WRK-FREQUENCIA    GREATER 80
   MOVE 'ALUNO APROVADO' TO WRK-MENSAGEM
END-IF.

IF WRK-IDADE         GREATER 18 OR
   WRK-ACOMPANHANTE  GREATER ZEROS
   MOVE 'ENTRADA PERMITIDA' TO WRK-MENSAGEM
END-IF.
```

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Quando houver necessidade de se usar vários testes de condição unida por cláusulas **AND** e **OR** misturados, é aconselhável o uso de parênteses para agrupar as condições. Sem o uso de parênteses, as condições são resolvidas da esquerda para a direita, primeiro todas as condições **AND** e depois as condições **OR**. Estas regras podem conduzir a erros se não forem avaliadas com muito cuidado. o uso de parênteses direciona a seqüência de avaliação das condições, simplificando a lógica.

Obs.: O Cobol resolve primeiro as expressões dentro dos parênteses.

Confira os dois exemplos abaixo e descubra a diferença entre eles:

IF (A = B OR C = D) AND E = F ...

IF A = B OR (C = D AND E = F) ...

3.4.7. ALTERAÇÕES/DESVIOS DO FLUXO DO PROGRAMA

A execução da lógica do programa, ou algoritmo, é feita pelas instruções que são codificadas na **PROCEDURE DIVISION**. O Cobol executa as instruções sequencialmente, de cima para baixo, somente saltando as instruções subordinadas aos testes condicionais com resposta falsa. Os comandos de alteração/desvio de fluxo permitem interromper esta seqüência de execução.

Existem dois comandos utilizados para controle de fluxo de programa:

GO TO

PERFORM

Para permitir o uso de desvios na seqüência das instruções de um programa, é necessária a inclusão de **PARÁGRAFOS** em pontos convenientes do programa. As duas instruções acima desviam a execução do programa para estes **PARÁGRAFOS**.

3.4.7.1. USO DO COMANDO GO TO

O comando **GO TO** é usada para desvios simples e imperativos.

No exemplo abaixo, o comando **GO TO** está associado a um comando condicional.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

```
IF WRK-TEMPO-DE-CASA          GREATER 10
    MOVE 50                    TO WRK-BONUS
    GO                          TO DESVIO-1
END-IF.

MOVE ZEROS                     TO WRK-BONUS.
```

DESVIO-1.

```
ADD WRK-BONUS                  TO WRK-SALARIO.
```

IMPORTANTE:

O uso do comando **GO TO** deve ser feito com critério. Nos últimos anos, a utilização do **GO TO** se restringe a desvios para o início ou final de uma seção.

Diz-se que, um programa bem estruturado não necessita de uso de **GO TO**.

3.4.7.2. USO DO COMANDO PERFORM

O comando **PERFORM** desvia a execução do programa para um parágrafo ou seção específicos, executa as instruções deste parágrafo / seção e retoma a execução do programa na instrução seguinte ao comando **PERFORM**, isto é, reinicia o fluxo interrompido pelo **PERFORM**.

Por exemplo:

```
MOVE WRK-CODIGO                TO WRK-CAMPO-TESTE.

PERFORM 2100-CALCULAR-DIGITO.

IF WRK-DIG-CALCULADO            EQUAL WRK-DIGITO-TESTE
    MOVE 'OK'                   TO WRK-MENSAGEM
ELSE
    MOVE 'DIGITO NÃO CONFERE'    TO WRK-MENSAGEM
END-IF.
```

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

No programa deverá existir um parágrafo / seção chamada "**2100-CALCULAR-DIGITO**", ou o compilador acusará erro de codificação.

```
*-----*
2100-CALCULAR-PRINCIPAL          SECTION.
*-----*

      MOVE ZEROS                  TO WRK-DIG-CALCULADO.

      COMPUTE .....

      MOVE WRK-DIG-RETORNO        TO WRK-DIG-CALCULADO.

*-----*
2100-99-FIM.                     EXIT.
*-----*
```

A instrução **PERFORM** pode ser usada para executar mais de um parágrafo se tiver a cláusula **THRU**:

Sintaxe:

PERFORM *parágrafo-1* **THRU** *parágrafo-2*.

Neste caso, o programa desvia para parágrafo-1, executa todas as instruções / parágrafos que encontrar até atingir o parágrafo-2, executa o parágrafo-2 e retorna.

Há ainda a opção de usar o **PERFORM** para executar parágrafo/seções repetidas vezes – podem-se usar repetições definidas ou indefinidas.

A sintaxe - **PERFORM** com repetições definidas:

PERFORM *parágrafo* **n TIMES.**

Onde **n** é um literal numérico (número), ou seja, a quantidade de vezes que o parágrafo / seção será executado.

Para repetições indefinidas é necessário fornecer uma condição para ser testada. A instrução **PERFORM** executará o parágrafo / seção até que a condição testada seja satisfeita.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

A sintaxe – **PERFORM** com repetições indefinidas:

PERFORM *parágrafo* **UNTIL** *condição.*

Existe ainda uma forma mais sofisticada de **PERFORM** com repetição indefinida que controla uma variável que pode ser usada no parágrafo executado.

Sintaxe:

PERFORM *parágrafo* **VARYING** *variável* **FROM** *valor-1* **BY** *valor-2*
UNTIL *condição.*

Exemplo:

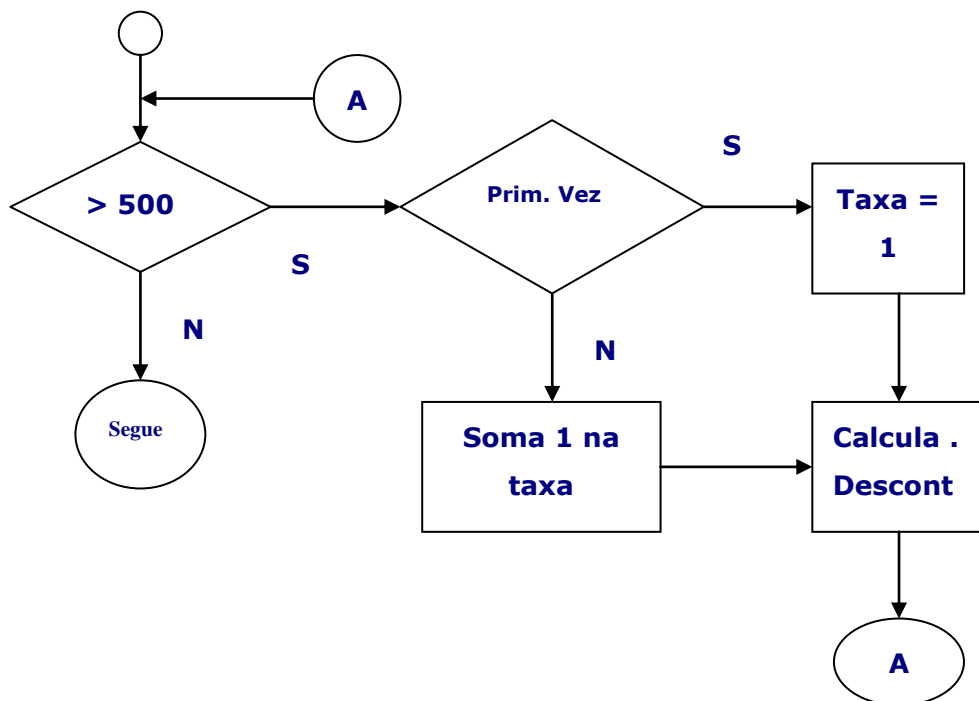
```
PERFORM 2200-CALCULAR-DESCONTO  
                VARYING TAXA      FROM 1 BY 1  
                UNTIL    DESCONTO NOT GREATER 500.
```

Neste exemplo, o **PERFORM** inicializa a variável **TAXA** com o valor inicial 1. Em seguida, testa a condição **DESCONTO NOT GREATER 500**, e se não satisfeita executa a seção **2200-CALCULAR-DESCONTO**. Realiza este procedimento até que a condição (**UNTIL**) seja atendida. A variável taxa será incrementada de 1 em 1 a cada teste.

Use o diagrama de blocos abaixo para entender esta instrução:

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação



Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

4. ARQUIVO

Todas as instruções que vimos até aqui manipulavam as variáveis na memória do computador. Mas a memória é volátil (quando se desliga o computador, perde-se todo seu conteúdo). Além disso, é um recurso limitado, e quando nosso programa finaliza, ele precisa ser apagado para que outros programas possam ser executados. Todos os dados que estão na memória e que não queremos perder quando o programa é encerrado devem ser gravados em **ARQUIVOS**, como discos magnéticos, fitas magnéticas, etc.

Um programa Cobol trata todos os tipos de arquivos basicamente da mesma maneira:

- Os arquivos devem ser declarados na **INPUT-OUTPUT SECTION (ENVIRONMENT DIVISION)**, com a instrução **SELECT**.
- O layout/conteúdo do arquivo deve ser descrito na **FILE SECTION (DATA DIVISION)** - instrução **FD**.
- Na **PROCEDURE DIVISION** é necessário programar as instruções para manipulação dos arquivos.

4.1. REGISTRO

Para armazenar informações em arquivos, inicialmente é necessário definir um arquivo para cada tipo de informação. Desta maneira teremos um arquivo para funcionários, um arquivo para produtos, um arquivo para notas fiscais, etc.

Uma vez definido a informação para um arquivo, devemos definir os dados que queremos salvar neste arquivo. Por exemplo, para um arquivo de cadastro de funcionários poderíamos gravar as seguintes informações:

CODIGO DO FUNCIONARIO

NOME

DATA DE ADMISSÃO

CARGO

SALARIO

O arquivo de funcionários será então uma repetição seqüencial dos dados acima para cada um dos funcionários da empresa. Cada uma das informações acima recebe

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

genericamente o nome de **CAMPO**. Por exemplo, o **CODIGO DO FUNCIONÁRIO** é um **CAMPO** pertencente ao arquivo de funcionários.

Um ou um conjunto de campos que define um funcionário recebe o nome de **REGISTRO**. O conjunto de todos os registros dos funcionários, constitui o **Arquivo de Funcionários**, que será identificado por um **DDNAME**.

4.2. ABERTURA DE ARQUIVO

Todo arquivo **DEVE OBRIGATORIAMENTE** estar aberto para a manipulação de seus registros.

O comando **OPEN** realiza a abertura do arquivo quando da execução do programa. Este comando abre o contacto com o dispositivo físico do arquivo, e reserva na memória áreas necessárias para a troca de dados entre o computador e o dispositivo externo.

A sintaxe para o comando é:

OPEN *tipo arquivo nome-do-arquivo.*

Onde:

Tipo de arquivo: inserir **INPUT** para arquivos de entrada, ou seja, somente operação de leitura será executada; **OUTPUT** para arquivos de saída, ou seja, são arquivos que receberão informações (gravação) ou de I-O para arquivos de entrada/saída.

Nome do arquivo: **DDNAME** do arquivo. Deve ser o mesmo utilizado na cláusula **SELECT**.

Exemplos:

OPEN INPUT	CADFUNC
	EMPRESA.
OUTPUT	CADSAIDA.
I-O	ARQLOG.

4.3. LEITURA E GRAVAÇÃO DE ARQUIVO

LEITURA

Existem 3 (três) comandos para leitura/gravação de registros dos arquivos.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

READ *nome-do-arquivo*
AT END *instruções-1.*
OU
READ *nome-do-arquivo*

Formato mais usual

Esta instrução lê um registro do arquivo e carrega seus dados nas variáveis definidas na **FD (FILE SECTION)**. A cada novo comando **READ** carrega o próximo registro do arquivo e assim sucessivamente – arquivo seqüencial.

A clausula **AT END** é opcional, e faz com que o programa execute as *instruções-1* quando foram lidos todos os registros do arquivo ou se o arquivo estiver vazio.

GRAVAÇÃO

WRITE *nível 01 do registro definido na FD.*

OU

WRITE *nível 01 do registro definido na FD FROM nível 01 do registro auxiliar.*

Esta instrução grava no arquivo todos os dados carregados no layout do registro na **FD**. De maneira diferente do comando **READ**, que usa como operando o nome do arquivo, o comando **WRITE** usa o nome do registro.

REGRAVAÇÃO

REWRITE *nível 01 do registro definido na FD.*

Este comando é usado para atualizar (alterar) o conteúdo de um registro já existente em um arquivo. Por razões físicas, este comando só pode ser usado com discos magnéticos.

4.4. FECHAMENTO DE ARQUIVO

Antes do término do programa, os arquivos abertos precisam ser fechados e desta forma, liberar os dispositivos físicos e no caso de gravações garantir a integridade dos dados.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Sintaxe:

CLOSE *nome-do-arquivo*.

4.5. ENTRADA E SAÍDA DE BAIXO VOLUME

Quando um programa precisa receber uma informação com quantidade pequena de bytes -- poucos caracteres como por exemplo, a data do processamento, ou emitir uma saída pequena como por exemplo, uma mensagem de erro, pode-se usar as instruções **ACCEPT** e **DISPLAY**.

Estas instruções não precisam da definição de arquivos de entrada e saída. Como em muitas linguagens, o Cobol usa os conceitos de Entrada Padrão e Saída Padrão para especificar dispositivos que o programa não define. Para o Mainframe, a Entrada Padrão geralmente é o arquivo de comandos **JCL** usado para disparar o **JOB**, e a Saída Padrão é a impressora que emite o resumo da execução do **JOB**.

As instruções **ACCEPT** e **DISPLAY** operam sobre as entrada/saída padrão, ou sobre a Console do computador se for especificado.

Sintaxe do comando **ACCEPT**:

ACCEPT *variável receptora* **FROM CONSOLE.**

Onde:

Variável receptora - deve ser definida na **WORKING-STORAGE** e ter formato compatível com a informação que será recebida.

FROM CONSOLE - clausula opcional. Se omitida, os dados serão obtidos através de comandos via JCL. Se especificado, o programa é interrompido nesta instrução, uma mensagem de dados requeridos aparece na console do computador, e somente quando o operador digita os dados na console e tecla <ENTER> o programa continua.

O comando **DISPLAY** tem o formato:

DISPLAY *variável-1 variável-2* **UPON CONSOLE.**

OU

DISPLAY *variável-1*

Este comando exibirá na saída padrão os valores das variáveis, *variável-1* e *variável-2*, ou sobre a Console quando a clausula **UPON CONSOLE** for especificada.

Variável-1 e *variável-2* podem ser campos da **WORKING-STORAGE** ou literais.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

5. ENCERRAMENTO DO PROGRAMA

Para finalizar um programa utiliza-se os comandos **STOP RUN** ou **GOBACK**.

O comando **GOBACK** deve ser usado para parar o programa chamado (módulo) e retornar ao programa chamador para dar continuidade a sua execução.

IMPORTANTE:

O comando **STOP RUN** não pode ser utilizado para programas de processamento **ONLINE**. Diferentemente do **GOBACK** que pode ser utilizado em programas **BATCH** e **ONLINE**.

Abaixo um exemplo de um programa completo que lê uma fita de movimentação de produtos e imprime um relatório com o resumo das quantidades acumuladas por produto.

```
*-----*
IDENTIFICATION                               DIVISION.
*-----*
PROGRAM-ID. TREI0001.
AUTHOR. FULANO DE TAL.
*-----*
*               T C S   T R A I N I N G   -   F O R M A C A O               *
*-----*
*   PROGRAMA TESTE   *
*-----*
ENVIRONMENT                               DIVISION.
*-----*
*-----*
CONFIGURATION                               SECTION.
*-----*
*-----*
INPUT-OUTPUT                               SECTION.
*-----*
```

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

FILE-CONTROL.

SELECT ENTRADA

FILE-STATUS

SELECT SAIDA

FILE-STATUS

ASSIGN TO TAPE

IS WRK-FS-ENTRADA.

ASSIGN TO PRINTER

IS WRK-FS-SAIDA.

DATA

DIVISION.

FILE

SECTION.

* INPUT - ARQUIVO CONTENDO OS REGISTROS ENTRADA - LRECL = 0046 *

* ORG. SEQUENCIAL *

FD ENTRADA

RECORDING MODE

IS F

LABER RECORD

IS STANDARD

BLOCK CONTAINS

0 RECORDS.

01 REG-ENTRADA.

03 COD-PROD

PIC X(06).

03 NOME-PROD

PIC X(30).

03 QUANT-PROD

PIC 9(06)V99.

* OUTPUT - ARQUIVO CONTENDO OS REGISTROS ENTRADA - LRECL = 0052 *

* ORG. SEQUENCIAL *

FD SAIDA

RECORDING MODE

IS F

LABER RECORD

IS STANDARD

BLOCK CONTAINS

0 RECORDS.

01 REG-SAIDA.

03 COD-PROD

PIC X(06).

03 FILLER

PIC X(04).

03 NOME-PROD

PIC X(30).

03 FILLER

PIC X(04).

03 QUANT-PROD

PIC 9(06)V99.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

```
*-----*
WORKING-STORAGE          SECTION.
*-----*

77  FILLER                PIC X(32)      VALUE
   '** INICIO DA WORKING-STORAGE – TREI0001 **'.

77  WRK-FS-ENTRADA        PIC X(02)      VALUE SPACES.
77  WRK-FS-SAIDA          PIC X(02)      VALUE SPACES.
77  WRK-COD-ANT           PIC X(06)      VALUE SPACES.
77  WRK-QUANT-PROD        PIC 9(06)V99   VALUE ZEROS.
77  WRK-NOME-ANT          PIC X(30)      VALUE SPACES.
77  FILLER                PIC X(32)      VALUE
   '** FINAL DA WORKING-STORAGE – TREI0001 **'.
```


Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

```
*-----*
PROCEDURE                                DIVISION.
*-----*
*-----*
0000-ROTINA-PRINCIPAL                    SECTION.
*-----*

    PERFORM 1000-INICIALIZAR.
    PERFORM 2000-TRATAR-ARQ-VAZIO.
    PERFORM 3000-PROCESSAR-REG UNTIL WRK-FS-ENTRADA EQUAL '10'.
    PERFORM 3100-GRAVAR-REG-SAIDA.
    PERFORM 5000-FINALIZAR.
*-----*

0000-99-FIM.                            EXIT.
*-----*

1000-INICIALIZAR                        SECTION.
*-----*

    OPEN INPUT  ENTRADA
        OUTPUT SAIDA.
    PERFORM 1100-TRATAR-FS-ENTRADA.
    PERFORM 1200-TRATAR-FS-SAIDA.
*-----*

1000-99-FIM.                            EXIT.
*-----*
*-----*

2000-TRATAR-ARQ-VAZIO                    SECTION.
*-----*

    PERFORM 2100-LER-ARQ-ENTRADA.
    IF  WRK-FS-ENTRADA                    EQUAL '10'
        DISPLAY '***** TREI0001 *****'
        DISPLAY '*          ARQUIVO ENTRADA VAZIO          *'
        DISPLAY '*          PROCESSAMENTO ENCERRADO          *'
        DISPLAY '*                                           *'
        DISPLAY '***** TREI0001 *****'
    PERFORM 5000-FINALIZAR
END-IF.
```

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

```
*-----*
2000-99-FIM.                                EXIT.
*-----*
*-----*
2100-LER-ARQ-ENTRADA                        SECTION.
*-----*

    READ ENTRADA.

    IF  WRK-FS-ENTRADA                        EQUAL '10'
      GO                                     TO 2100-99-FIM
    END-IF.

    MOVE COD-PROD OF REG-ENTRADA
      TO WRK-COD-ANT.
    MOVE NOME OF REG-ENTRADA TO WRK-NOME-ANT.

    PERFORM 1100-TRATAR-FS-ENTRADA.

*-----*
2100-99-FIM.                                EXIT.
*-----*
*-----*
3000-PROCESSAR-REG                          SECTION.
*-----*

    IF  COD-PROD OF REG-ENTRADA
      NOT EQUAL WRK-COD-ANT
      PERFORM 3100-GRAVAR-REG-SAIDA
      MOVE ZEROS                                TO WRK-QUANT-PROD
    END-IF.

    ADD QUANT-PROD OF REG-ENTRADA
      TO WRK-QUANT-PROD.

    PERFORM 2100-LER-ARQ-ENTRADA.
```

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

```
*-----*
3000-99-FIM.                                EXIT.
*-----*
*-----*
3100-GRAVAR-REG-SAIDA                      SECTION.
*-----*

      MOVE SPACES                          TO REG-SAIDA.
      MOVE WRK-COD-ANT                     TO COD-PROD OF REG-SAIDA.
      MOVE WRK-NOME-ANT                    TO NOME-PROD OF REG-SAIDA.
      MOVE WRK-QUANT-PROD                  TO QUANT-PROD OF REG-SAIDA.

      WRITE REG-SAIDA.

      PERFORM 1200-TRATAR-FS-SAIDA.

*-----*
3100-99-FIM.                                EXIT.
*-----*
*-----*
4000-FINALIZAR                            SECTION.
*-----*

      CLOSE ENTRADA
           SAIDA.

      PERFORM 1100-TRATAR-FS-ENTRADA.
      PERFORM 1200-TRATAR-FS-SAIDA.

      STOP RUN.

*-----*
4000-99-FIM.                                EXIT.
*-----*
```

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

6. FORMATOS ESPECIAIS DE DADOS

6.1. SINAL – CAMPOS NUMÉRICOS

Já vimos que para declarar uma variável numérica na DATA DIVISION, usamos o caractere 9 juntamente com a cláusula PIC.

Para prever o sinal (positivo ou negativo), deve-se acrescentar a letra S após a PIC, conforme abaixo:

77 WRK-VAR-1

PIC S9999.

6.2. FORMATO BINÁRIO

Quando usamos os formatos descritos nos parágrafos anteriores para descrever variáveis na **DATA DIVISION**, o Cobol armazena cada caractere das variáveis definidas em 1 byte da memória. Foi visto na introdução do curso que um byte é um conjunto de 8 bits. O computador associa cada combinação de bits ligado/desligado destes 8 bits com uma letra (ou caractere) do nosso alfabeto.

Assim, no caso do mainframe, que utiliza o padrão **EBCDIC**, podemos representar estes 2 exemplos:

Caracter	Bits na memória	Valor decimal	Valor hexadecimal
A	1 1 0 0 0 0 0 1	191	C1
5	1 1 1 1 0 1 0 1	245	F5

Acontece porém que o computador não foi projetado para fazer cálculos com este formato. Ele usa o valor binário puro dos bits para cálculos, onde cada bit da direita para a esquerda tem o valor da potência de 2 correspondentes, como no esquema:

Bits	8	7	5	4	3	2	1
Valor	128	64	32	8	4	2	1

Assim o caractere 5 do formato **EBCDIC** acima seria interpretado pelo computador nas operações de cálculo como 245.

Os programadores Cobol não precisariam se preocupar com este detalhe, porque o compilador Cobol coloca no programa objeto gerado, instruções adicionais para, na hora do

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

cálculo, converter todas as variáveis do formato **EBCDIC** para binário, fazer o cálculo, e converter o resultado do formato binário para **EBCDIC**, e devolver o resultado convertido para a memória.

Os inconvenientes de trabalhar com variáveis numéricas em **EBCDIC** são:

- A ineficiência causada pelas conversões de formato.
- O formato **EBCDIC** ocupa espaço maior na memória.

6.3. CLÁUSULA USAGE

A cláusula **USAGE** é utilizada definir o formato da variável na memória.

Existem 3 (três) opções de formato:

USAGE DISPLAY.

USAGE COMP.

USAGE COMP-3.

6.3.1. USAGE DISPLAY

Especifica que a variável será formatada em **EBCDIC**. Este formato é assumido quando se omite a cláusula **USAGE**.

6.3.2. USAGE COMP

Usado para especificar que a variável deve ter formato binário. Só é usado para variáveis numéricas.

O Cobol formata números com **USAGE COMP** somente em 3 formatos binários que são os formatos disponíveis na **CPU** do mainframe para cálculos:

- Um formato com 2 bytes, ou 16 bits (1 bit de sinal e 15 bits de valores binários);
- Um formato de 4 bytes, ou 32 bits (1 bit de sinal e 31 bits de valores binários);
- Um formato com 8 bytes, ou 64 bits (1 bit de sinal e 63 bits de valores).

A escolha destes formatos é feita de acordo com o tamanho do campo especificado pelo programador, segundo a tabela abaixo:

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

DE	ATE	BYTES	VALORES
S9(1)COMP	S9(4) COMP	2	-32768 a + 32768
S9(5)COMP	S9(9) COMP	4	-2.147.483.648 a + 2.147.483.648
S9(10)COMP	S9(18)COMP	8	valores com ate 18 algarismos

6.3.3. USAGE COMP-3

Este é um formato binário usado exclusivamente em mainframes IBM para variáveis numéricas. Neste formato, usam-se grupos de 4 bits para representar 1 número, e por isso 1 byte consegue armazenar 2 números. Os valores dos bits no byte aparecem como:

Bits	8	7	6	5	4	3	2	1
Valor	8	4	2	1	8	4	2	1

Neste formato o sinal é obrigatório, usando também 4 bits, com configurações conforme tabela abaixo:

Bits	8	4	2	1	Valor decimal	Valor hexa	Sinal
	1	0	1	0	10	A	+
	1	0	0	1	11	B	-
	1	1	0	0	12	C	+
	1	1	0	1	13	D	-
	1	1	1	0	14	E	+
	1	1	1	1	15	F	+

Todas as outras combinações de bits são inválidas para sinal.

Mostraremos abaixo o número 245 usando os 3 (três) formatos em Cobol:

Cobol	Configuração dos bits	Hexa
77 VAR-1 PIC 999 VALUE 245	1111 0010 1111 0100 1111 0101	F2 F4 F5
77 VAR-1 PIC 999 COMP VALUE 245	0000 0000 1111 0101	00 F5
77 VAR-1 PIC S999 COMP-3 VALUE 245	0010 0100 0101 1100	24 5C

Como regra geral, o Cobol não suporta números com mais de 18 algarismos.

6.4 TABELAS

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Alguns algoritmos exigem a definição de uma mesma variável várias vezes, aumentando o trabalho de codificação do programa correspondente tanto na **DATA DIVISION**, como também as instruções resultantes na **PROCEDURE DIVISION**. Por exemplo, em um algoritmo para acumular as vendas do ano separadas por mês, precisamos definir 12 campos de total na **DATA DIVISION**, e a **PROCEDURE DIVISION** deverá ter 12 testes do mês da venda para decidir em que total deve ser feito a soma. Ex.:

DATA DIVISION.

03 TOTAL-01 PIC 9(8)V99.

03 TOTAL-02 PIC 9(8)V99.

03 TOTAL-12 PIC 9(8)V99.

PROCEDURE DIVISION.

IF MÊS = 01

ADD VENDAS TO TOTAL-01

ELSE IF MÊS = 02

ADD VENDAS TO TOTAL-02

ELSE

ELSE IF MÊS = 12

ADD VENDAS TO TOTAL-12.

A linguagem Cobol possui um recurso para resolver este problema. Na **DATA DIVISION** a variável será definida somente uma vez, acompanhada da cláusula **OCCURS** que definirá quantas vezes a variável deve ser repetida. A sintaxe da definição do item com **OCCURS** é:

Nível variável PIC picture OCCURS n TIMES.

O total mensal do exemplo acima ficará.

01 TOTAIS-GERAIS.

03 TOTAL-MENSAL PIC 9(8)V99 OCCURS 12 TIMES.

A cláusula **OCCURS** só pode ser usada em variáveis de nível **02 a 49**.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Quando uma variável de uma tabela (definida com **OCCURS**) for usada na **PROCEDURE DIVISION**, ela precisa ser acompanhada de um indexador (subscrito) que definirá qual ocorrência da tabela está sendo referido. Este subscrito deve estar dentro de parênteses e pode ser um literal numérico ou uma variável numérica com valores inteiros. Por ex.:

```
ADD VENDAS TO TOTAL-MENSAL(5).
```

Neste caso a soma esta sendo feita no quinto mês (Maio).

A codificação do algoritmo do exemplo acima ficará reduzido agora a:

DATA DIVISION.

01 TOTAIS-GERAIS.

```
03 TOTAL-MENSAL PIC 9(8)V99 OCCURS 12 TIMES.
```

PROCEDURE DIVISION.

.....

```
ADD VENDAS TO TOTAL-MENSAL (MÊS-VENDA).
```

6.4.1 NÍVEIS DAS TABELAS

Em Cobol podemos definir uma entrada de uma tabela como uma nova tabela, e assim sucessivamente até um nível de 3 tabelas. Por exemplo, para obter o total de vendas separado por estado, e em cada estado por tipo de produto, e para cada produto por mês de venda, montaremos a **DATA DIVISION** como abaixo:

DATA DIVISION.

01 TOTAIS-VENDA.

```
03 VENDAS-ESTADO OCCURS 27 TIMES.
```

```
05 VENDAS-PRODUTO OCCURS 5 TIMES.
```

```
07 VENDAS-MÊS PIC 9(8)V99 OCCURS 12 TIMES.
```

Este código montará na memória uma tabela com 3 níveis de 1620 totais (27 estados X 5 produtos X 12 meses). Para acessar um total desta tabela será necessário um conjunto de 3 subscritores:

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

PROCEDURE DIVISION.

.....

ADD VENDAS TO VENDAS-MÊS (COD-ESTADO, COD-PRODUTO, MÊS-VENDA).

Os subscritos dentro do parênteses devem estar na mesma seqüência da definição das tabelas (mesma hierarquia).

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

7. IMPRESSÃO

O Cobol trata a impressão de relatórios de maneira similar à gravação de arquivos, ou seja, enviar uma linha de relatório para a impressora é idêntico a gravar um registro em um arquivo. Por isso precisamos definir o relatório na **INPUT-OUTPUT SECTION (ENVIRONMENT DIVISION)** com a instrução **SELECT**, precisamos definir o conteúdo das linhas de impressão na **FILE SECTION (DATA DIVISION)** com a instrução **FD**, e na **PROCEDURE DIVISION** devemos usar as instruções **OPEN**, **WRITE** e **CLOSE** para controlar a impressão.

No entanto, obviamente existem diferenças entre um arquivo e um relatório, e os seguintes detalhes devem ser observados em um programa:

- Diferentemente dos arquivos, onde todos os registros tem o mesmo lay-out, em um relatório as linhas de detalhe podem ser diferentes (incluindo sub-totais, títulos de grupos etc.). Além disso, sempre existe um cabeçalho em cada folha ou ainda linhas de rodapé.
- O programa precisa controlar a mudança de página. Para isto normalmente usa-se a variável **LINAGE-COUNTER**, que é incrementada automaticamente a cada comando **WRITE** do relatório. Quando esta variável atinge o numero de linhas disponíveis na folha, o programa deve comandar o salto para a nova folha imprimir as linhas de cabeçalho.
- É comum haver totalizações em vários níveis (sub-totais, totais gerais etc.). Estes totais são emitidos quando muda a identificação de grupo dentro dos registros lidos. Por exemplo, em um relatório de vendas com totais por mês o programa deve comparar o mês do registro lido com o mês do registro anterior para verificar se são diferentes. Nestas mudanças de identificador de grupo (geralmente conhecido como **QUEBRA**), o programa deve emitir uma linha de total, e acertar convenientemente as variáveis de totalização.
- O comando **WRITE** tem um formato próprio para impressões mostrado abaixo:

WRITE *registro* **BEFORE ADVANCING** *variável* **LINES**

OU

WRITE *registro* **AFTER ADVANCING** *variável* **LINES**

OU

WRITE *registro* **BEFORE ADVANCING PAGE**

OU

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

WRITE *registro* **AFTER ADVANCING PAGE**

As cláusulas **BEFORE ADVANCING** ou **AFTER ADVANCING** são opcionais e servem para comandar a mudança de linha na impressora. A palavra **ADVANCING** pode ser omitida. Uma instrução como a seguinte:

WRITE LINHA-DETALHE BEFORE 2 LINES.

Imprime a linha detalhe **ANTES** de saltar 2 linhas em branco na impressora, ou seja, a linha é impressa e **DEPOIS** há um salto de 2 linhas em branco. O **ADVANCING PAGE** provoca um salto para o início de nova página.

7.1 OPÇÃO AFTER POSITIONING

Esta opção deve ser declarada como caracter alfanumérico (**PICTURE X**).

Por exemplo, na impressão do cabeçalho de um relatório:

FD IMPRESSAO

LABEL RECORD IS OMITTED
RECORDING MODE IS F
BLOCK CONTAINS 0 RECORDS.

01 REG-RELATORIO.

03 CARRO PIC X(01).

02 FILLER PIC X(132) VALUE SPACES.

PROCEDURE

DIVISION

WRITE RELATORIO

FROM CABEC1 AFTER POSITIONING CARRO .

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Abaixo encontra-se o valores possíveis que podem ser atribuídos ao salto do **CARRO**:

Branco	` `	Espacejamento simples
Zero	`0'	Espacejamento duplo
Menos	`-'	Espacejamento triplo
Mais	`+'	Supressão do espacejamento
`1' a `9'		Salto do canal 1 a 9
`A', `B', `C'		Salto do canal 10, 11 e 12

- No **POSITIONING**, o máximo de linhas que se pode pular são 3 (três);
- Não deve-se utilizar **"AFTER"** e **"BEFORE"** no mesmo programa.

7.2. MÁSCARAS DE EDIÇÃO DE CAMPOS

Utiliza-se para itens que devem ser impressos ou enviados para tela de programas online. São definidos na **"WORKING-STORAGE SECTION"**.

O formato é representado por qualquer combinação dos seguintes caracteres:

(9), (V), (P), (.), (Z), (*), (CR), (DB), (,), (0), (B), (\$), (+), (-)

(9), (V), (P) - são usados de maneira semelhante ao uso dos itens numéricos;

(.) - o ponto decimal, quando usado, é inserido na posição indicada;

(Z) - indica a supressão de zeros não significativos;

(*) - é usado como proteção de um número impresso;

(CR), (DB) - significam CR (crédito) e DB (débito). Usa-se somente quando o número for negativo. Caso o número seja positivo, não aparecerá nada;

(,), (0), (B) - são símbolos de edição que são inseridos na impressão;

(\$), (+), (-) - são impressos na posição indicada.

OBS: O símbolo **(-)** não deve ser o próximo sinal após o ponto. Caso isso ocorrer, definir mais um **(-)**.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Exemplo:

01 CAMPO PIC ---.---.---,99.

Definição	Picture	Valor Real	Na memória
9(04)	9999	502	502
9V9(2)	9V99	1,25	125
9(03)	999PP	43700	437
S9(02)	S99	-21	21-
9(05)	99.999	10.987	10987
9(04)V99	Z.ZZZ,99	25,50	002550
9(03)	ZZZ		000
9(03)	**9	422	422
9(03)	***	***	000
S9(03)V99	999,99CR	800,00CR	80000(-)
9(04)	990099	110025	1125
9(06)	99B99B99	12 13 15	121315
9(03)	\$999	\$371	371
S9(02)	-99	-15	15(-)
S9(02)	-99	16	16
S9(02)	+99	15	15(-)
S9(02)	+99	+16	16(+)
S9(02)	99-	15-	15(-)
S9(04)	-,--9	-12	0012(-)

OBS: caso tenha que utilizar o caracter ` , para se fazer entendido ao Cobol, define-se dentro das aspas duplas.

Exemplo:

77 ASPA PIC X(01) VALUE " ' ".

A cláusula **PICTURE (ou PIC)** tem alguns formatos próprios para fazer edição de variáveis numéricas no momento de uma impressão.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Observação: Máscaras de edição também podem ser usadas para mensagens exibidas na **SYSOUT / CONSOLE** e na formatação de campos alfanuméricos.

7.3. SUPRESSÃO DE ZEROS

Tomemos como exemplo a seguinte variável definida na **WORKING-STORAGE**:

03 VALOR PIC 9(5)V99 VALUE 2.

A instrução: **WRITE VALOR**

Terá como resultado: **0000200**

Que é a imagem desta variável na memória do computador (**VALOR** está definido com 5 posições inteiras e 2 variáveis, com uma vírgula implícita). Para suprimir os zeros a esquerda das variáveis numéricas, e forçarmos a impressão da virgula, usamos a letra Z nas posições onde o zero deve ser suprimido, e a definição do **VALOR** no registro de impressão (**FD**) será:

03 VALOR PIC ZZZZ9,99.

E agora a instrução **WRITE** produzirá: **2,00**

7.4. TIPOS DE MASCARA DE EDIÇÃO - OUTRAS

Os seguintes exemplos ilustram outras máscaras de edição:

PICTURE	VALUE	IMPRESSÃO
R\$ZZZZ9,99	2	R\$ 2,00
\$\$\$\$\$9,99	2	\$2,00
R\$****9,99	2	r\$****2,00
+ZZZZ9,99	-2	- 2,00
++++9,99	-2	-2,00
ZZZZ9,99+	-2	2,00-
ZZZZ9,99CR	-2	2,00CR
99/99/99	020304	02/03/04
99B99B99	020304	02 03 04

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

7.5. BLANK WHEN ZERO

Esta cláusula, usada após a máscara de edição da **PICTURE**, envia espaços em branco para a impressora quando a variável numérica a ser impressa tem valor zero, independente do formato da máscara. Pode ser abreviado para **BZ**. Ex.:

```
03 VALOR PIC ZZZZ9,99 BLANK WHEN ZERO.
```

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

8. ORDENAÇÃO DE ARQUIVOS

Os registros estão dispostos nos arquivos na ordem em que foram gravados. Por exemplo, um arquivo de clientes pode ter sido gravado em ordem crescente de código de cliente. Para se fazer um relatório dos clientes em ordem alfabética de nomes, é necessário reordenar o arquivo nesta ordem. Isto é feito em Cobol usando o recurso do **SORT**.

Para usar o **SORT** é necessário definir no programa um arquivo temporário. O **SORT** grava neste arquivo os registros a serem ordenados, efetua a classificação, e devolve nele os registros ordenados. Além deste temporário, define-se também no programa um arquivo de entrada (arquivo original), e um arquivo de saída que receberá os registros ordenados.

A definição do arquivo temporário é feita com uma instrução **SELECT** padrão. Na **FILE SECTION** porém, o parágrafo **FD** deve ser substituído por **SD (Sort Description)**, e na descrição do registro é suficiente definir os campos envolvidos na ordenação.

Existem duas maneiras de codificar a instrução **SORT** na **PROCEDURE DIVISION**: Na primeira (**SORT Intrínseco**), o processo de classificação não altera o conteúdo do registro do arquivo. Na segunda, o pode-se alterar o lay-out dos registros do arquivo de saída, eliminar campos na saída e até eliminar registros na classificação (mantendo somente os que interessarem ao relatório de saída). Esta segunda opção pode ser usada para se melhorar a performance, ao se utilizar registros menores.

8.1. SORT INTRÍNSECO

No **SORT INTRINSECO**, a instrução **SORT** tem o formato:

```
SORT arquivo-sd ASCENDING campo      USING arquivo-entrada  
                                           GIVING arquivo-saida.
```

Este comando executa todas as etapas necessárias à classificação, e o programa Cobol não precisa abrir, ler, gravar ou fechar qualquer dos arquivos envolvidos no processo. Também pode-se usar **DESCENDING** quando se quiser ordem decrescente na ordenação, assim como colocar vários campos como chave de classificação.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

8.2. SORT EXTRÍNSECO

Quando não usamos o **SORT** Intrínseco, o processo é mais complexo e envolve os seguintes passos:

- É necessário definir 2 parágrafos na **PROCEDURE DIVISION**, a **INPUT PROCEDURE** e a **OUTPUT PROCEDURE**, para respectivamente ler o arquivo de entrada e gravar o arquivo de saída.
- Na **INPUT PROCEDURE** abre-se o arquivo de entrada (**OPEN**), faz-se a leitura dos registros (**READ**), e move-se para a **SD** do arquivo **SORT** os campos escolhidos. Em seguida grava-se o registro **SD**. Repete-se o ciclo de leituras da entrada e gravação do **SD** até se atingir o fim do arquivo de entradas, e finalmente fecha-se o arquivo de entrada.
- A gravação do registro **SD** deve ser feito com a instrução **RELEASE**.
- Na **OUTPUT PROCEDURE**, o arquivo de saída deve ser aberto com **OPEN OUTPUT**. Em seguida deve-se ler cada registro classificado no arquivo **SD** com a instrução **RETURN**, mover os campos da **SD** para o registro de saída (**FD**), e gravar o arquivo de saída (**WRITE**). Repetir as leituras da **SD** e gravação da saída até o fim do arquivo de **SORT**, e finalmente fechar o arquivo de saída.
- O formato da instrução **SORT** nesta opção é:

SORT *arquivo-sd* **ASCENDING** *campo*

INPUT PROCEDURE *parágrafo-1*

OUTPUT PROCEDURE *parágrafo-2*.

No exemplo abaixo, classificamos o arquivo Cliente usando o método explícito.

IDENTIFICATION DIVISION.

PROGRAM-ID. LISTACLI.

AUTHOR. TCS.

*

ENVIRONMENT DIVISION.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

CONFIGURATION SECTION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT CLIENTES ASSIGN TO DA-S-CLIENTES.

SELECT ORD-CLIENTES ASSIGN TO DA-S-ORDCLIENTES.

SELECT RELAT ASSIGN TO DA-S-RELAT.

DATA DIVISION.

FILE SECTION.

FD CLIENTES.

01 REG-CLIENTES.

10 CLIENTES-CODIGO PIC 9(006).

10 CLIENTES-NOME PIC X(050).

10 CLIENTES-DATA-NASC.

20 CLIENTES-DATA-NASC-ANO PIC 9(004).

20 CLIENTES-DATA-NASC-MÊS PIC 9(002).

20 CLIENTES-DATA-NASC-DIA PIC 9(002).

10 CLIENTES-TELEFONE PIC X(030).

10 CLIENTES-ENDERECO PIC X(050).

10 CLIENTES-BAIRRO PIC X(030).

10 CLIENTES-CIDADE PIC X(030).

10 CLIENTES-ESTADO PIC X(002).

10 CLIENTES-CEP PIC 9(008).

10 CLIENTES-E-MAIL PIC X(050).

SD ORD-CLIENTES.

01 SORT-REG-CLIENTES.

10 SORT-CLIENTES-CODIGO PIC 9(006).

10 SORT-CLIENTES-NOME PIC X(050).

10 SORT-CLIENTES-E-MAIL PIC X(050).

FD RELAT.

01 REG-RELAT PIC X(106).

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

WORKING-STORAGE SECTION.

PROCEDURE DIVISION.

INICIO.

SORT ORD-CLIENTES ASCENDING SORT-CLIENTES-NOME

INPUT PROCEDURE SORTIN-CLIENTES THRU EXIT-SORTIN-CLIENTES

OUTPUT PROCEDURE SORTOUT-CLIENTES

THRU EXIT-SORTOUT-CLIENTES

STOP RUN

SORTIN-CLIENTES.

OPEN INPUT CLIENTES.

LOOP-CLIENTES.

READ CLIENTES

AT END GO TO FIM-CLIENTES.

MOVE CLIENTES-CODIGO TO SORT-CLIENTES-CODIGO

MOVE CLIENTES-NOME TO SORT-CLIENTES-NOME

MOVE CLIENTES-E-MAIL TO SORT-CLIENTES-E-MAIL

RELEASE SORT-REG-CLIENTES

GO TO LOOP-CLIENTES.

FIM-CLIENTES.

CLOSE CLIENTES.

EXIT-SORTIN-CLIENTES.

EXIT.

SORTOUT-CLIENTES.

OPEN OUTPUT RELAT.

LOOP-SORTOUT.

RETURN ORD-CLIENTES

AT END GO TO FIM-SORTOUT.

WRITE REG-RELAT FROM SORT-REG-CLIENTES.

GO TO LOOP-SORTOUT.

FIM-SORTOUT.

CLOSE RELAT.

EXIT-SORTOUT-CLIENTES.

EXIT.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

9. ARQUIVOS DE ACESSO ALEATÓRIO(VSAM)

Toda a programação de arquivos visto até aqui tratava de um tipo de arquivo denominado **SEQUENCIAL**. Nestes arquivos os registros estão dispostos na ordem em que foram gravados. Para se ter acesso a um registro específico é necessário ler todos os registros do arquivo a partir do início até atingirmos o registro pretendido.

O mainframe tem, porém um outro tipo de arquivo de acesso aleatório, que só pode ser acessado em discos magnéticos: os arquivos **VSAM**. Nos arquivos **VSAM** escolhe-se um ou mais campos do registro como **CHAVE** de acesso. No programa Cobol, fornecendo-se o valor da chave de um registro, as instruções de leitura/gravação acessam o registro diretamente, independente da sua posição no arquivo.

Há dois tipos de organização interna dos arquivos **VSAM**:

SEQUENCIAL INDEXADA RELATIVA.

Na organização **SEQUENCIAL INDEXADA**, a chave é formada por campos do registro, como foi exposto acima. Na organização **RELATIVA**, a chave é a posição do registro dentro do arquivo. Por exemplo, para ler o centésimo registro do arquivo, fornecemos o valor 100 como chave. Nos arquivos **VSAM RELATIVOS** é necessário um algoritmo para associar um registro específico com sua posição no arquivo. Apesar dos arquivos **VSAM RELATIVOS** serem mais eficientes no processamento, a exigência de se criar este algoritmo restringe muito seu uso.

Os arquivos **VSAM** exigem sintaxe especializada em algumas partes do Cobol que descrevemos em seguida.

9.1. MANIPULAÇÃO ARQUIVOS VSAM

9.1.1. INPUT-OUTPUT SECTION

A cláusula **SELECT** terá a seguinte estrutura para arquivos indexados:

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

.....

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT [OPTIONAL] *nome-arquivo* **ASSIGN TO** *nome-externo*

ORGANIZATION IS *tipo-de-organização*

ACCESS MODE IS *tipo-de acesso*

RECORD KEY IS *key-principal*

ALTERNATE KEY IS *key-alternada* [**WITH DUPLICATES**]

FILE STATUS IS *campo-status*.

A cláusula **OPTIONAL** já foi descrita no **SELECT** para arquivos sequenciais.

Os detalhes do comando, mostrados em letras minúsculas são explicados abaixo:

- **Nome-do-arquivo** – sintaxe idêntica aos arquivos seqüenciais.
- **Nome-externo** - sintaxe idêntica aos arquivos seqüenciais.
- **Tipo-de-organização:**
 - **INDEXED** para arquivos seqüências indexados
 - **RELATIVE** para arquivos relativos
- **Tipo-de-acesso:**
 - **SEQUENTIAL** – O arquivo será lido seqüencialmente
 - **RANDOM** – O arquivo será lido aleatoriamente.
 - **DYNAMIC** – O arquivo será lido e atualizado.
- **key-principal** – Campo do registro escolhido para identificar de maneira inequívoca o registro. O arquivo só pode ter uma **RECORD KEY**.
- **key-alternada** – Qualquer outro campo do registro usado para pesquisa de registros. Pode-se acrescentar a esta cláusula a declaração **WITH DUPLICATES**, que indica que a pesquisa por esta chave pode conduzir a mais de um registro. O arquivo pode ter várias **ALTERNATE KEY**. **ALTERNATE KEY** deve ser usado

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

com cuidado porque aumenta o processamento nas atualizações para manter mais chaves no arquivo.

Os campos de chave de um arquivo **VSAM** são partes integrantes da estrutura física do arquivo, e são definidas na criação do mesmo pelo **DBA**.

As seguintes cláusulas são opcionais no **SELECT: ACCESS MODE / ALTERNATE KEY**.

9.1.2. FILE-STATUS

O **FILE STATUS** permite ao usuário monitorar a execução de operações de entrada e saída (**I/O**) requisitadas para os arquivos de um programa.

Após cada operação de **I/O**, o sistema move um valor para a **STATUS KEY** (campo alfanumérico, com 2 caracteres definidos na **WORKING-STORAGE SECTION** e especificado na **ENVIRONMENT DIVISION**, através do **SELECT**) que acusa o sucesso ou o insucesso da operação.

Qualquer valor movido para a **STATUS KEY** diferente de zeros, revela que a execução não foi bem sucedida.

Alguns exemplos de operações de I/O que podem ser testadas o **FILE STATUS**:

- **OPEN - START - WRITE - READ - REWRITE - CLOSE**

Os testes **de FILE-STATUS** devem ser realizado para todas as operações de arquivos, seja ele **SEQUENCIAL** ou **INDEXADO**.

Todos os comandos para arquivos **VSAM** na **PROCEDURE DIVISION** devem ser seguidos de um teste na variável **FILE STATUS** para verificar possíveis erros.

A variável **FILE STATUS** pode retornar um dos valores abaixo:

00- Comando com sucesso.

02- Sucesso, mas existe Alternate Key.

04- Sucesso, Mas o compr do registro não confere com FD.

05- Open Optional com sucesso, mas não existia arquivo anterior.

10- Fim do arquivo.

14- Para arquivos relativos, a key é maior que numero de registros

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

- 21- Registro fora de ordem.
- 22- No write, registro já existe.
- 23- No read, registro não existe.
- 24- Write fora dos limites lógicos do arquivo.
- 30- Parity error.
- 34- Write fora dos limites físicos do arquivo.
- 35- No open, arquivo não existe.
- 37- Arquivo não suporta este tipo de open.
- 38- Open error. Arquivo esta fechado com lock.
- 39- Open error. Atributos do arquivo não batem com FD.
- 41- Arquivo já esta aberto.
- 42- Arquivo já esta fechado.
- 43- Tentativa de atualizar sem ler.
- 44- Violação de limites.
- 46- Read next sem start.
- 47- Read em arquivo open output.
- 48- Write em arquivo open input.
- 49- Delete ou rewrite sem open I-o
- 9X- Erro de run-time (X é o valor binário do erro).

9.1.4. ABERTURA DO ARQUIVO - VSAM

O comando **OPEN** é o mesmo que o dos arquivos seqüenciais.

9.1.5. LEITURA DO ARQUIVO - VSAM

Comando **READ:**

READ arquivo

Este comando lê o registro cuja chave foi carregada na **RECORD KEY**.

Exemplo de um programa com **READ:**

FILE-CONTROL.

```
SELECT ARQ-CLIENTE ASSIGN TO DA-S-CLIENTE  
ORGANIZATION IS INDEXED
```

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

RECORD KEY IS COD-CLIENTE
FILE STATUS IS STATUS-CLIENTE.

.....

WORKING-STORAGE SECTION.

77 WORK-COD PIC
77 STATUS-CLIENTE PIC XX.

.....

PROCEDURE DIVISION.

.....

ACCEPT WORK-COD
MOVE WORK-COD TO COD-CLIENTE.
READ ARQ-CLIENTE
IF STATUS-CLIENTE = "10"
MOVE "CLIENTE NÃO EXISTE " TO MENSAGEM
GO TO FIM-PROGRA.

9.1.6. READ NEXT - VSAM

Comando **READ/NEXT:**

READ arquivo NEXT.

Este comando lê o próximo registro do arquivo na seqüência de chaves. Uma seqüência de comandos **READ NEXT** retorna uma coleção de registros ordenada por uma chave. Esta seqüência de **READ NEXT** deve ser precedida de um comando **START** que especifica a chave usada nas leituras e o valor do início desta chave.

9.1.7. READ PREVIOUS – VSAM

Comando **READ/PREVIOUS:**

READ arquivo PREVIOUS.

Este comando lê o registro anterior do arquivo na seqüência de chaves. Funciona de maneira idêntica ao **READ NEXT**, usando a ordem decrescente da chave.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

9.1.8. START - VSAM

Comando **START**:

START arquivo **KEY** condição chave .

O comando **START** é usado na preparação de uma sequência de **READ NEXT** ou **READ PREVIOUS**. Ele não carrega registros na memória, mas prepara o **VSAM** para as próximas leituras.

A condição especificada para a **KEY** pode ser qualquer uma das condições relacionais vistas na instrução **IF (=, GREATER, NOT LESS etc.)**.

A chave usada no comando deve ser uma das chaves definidas na cláusula **SELECT** para este arquivo (pode ser a **RECORD KEY** ou uma **ALTERNATE KEY**). Esta chave será usada nos comandos **READ** posteriores.

Exemplo:

```
.....  
FILE-CONTROL.  
    SELECT ARQ-CLIENTE ASSIGN TO DA-S-CLIENTE  
    ORGANIZATION IS INDEXED  
    RECORD KEY IS COD-CLIENTE  
    ALTERNATE KEY IS NOME-CLIENTE  
    FILE STATUS IS STATUS-CLIENTE.  
.....  
WORKING-STORAGE SECTION.  
    77  STATUS-CLIENTE          PIC XX.  
.....  
PROCEDURE DIVISION.  
.....  
    MOVE "JOSE" TO NOME-CLIENTE.  
    START ARQ-CLIENTE KEY NOT LESS NOME-CLIENTE  
    IF STATUS-CLIENTE = "23"  
        MOVE 'NÃO EXISTEM CLIENTES PARA ESTA CONDICAÇÃO.'  
        GO TO FIM-PROGRA.
```

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

LOOP-LEITURA.

```
READ ARQ-CLIENTE NEXT
IF STATUS-CLIENTE = "10"
    GO TO FIM-PROGRA.
MOVE REG-CLIENTE TO REG-SAIDA
GO TO LOOP-LEITURA.
```

9.1.9. WRITE - VSAM

Comando **WRITE**:

WRITE record-name.

Este comando cria um registro novo no arquivo, com o conteúdo de record-name. Record-name é a variável de nível **01** definida na **FD** do arquivo a atualizar. O **VSAM** examina o campo **RECORD KEY** dentro do record-name, e coloca o registro criado dentro do arquivo na posição correta para manter a ordem crescente da **RECORD KEY**. Se o valor da **RECORD KEY** do registro a gravar já existir no arquivo, o registro não será incluído e o status 22 será retornado.

9.1.10. REWRITE - VSAM

Comando **REWRITE**:

REWRITE record-name.

Este comando regrava o registro de chave igual a **RECORD KEY** no arquivo, com o conteúdo de record-name. O **VSAM** examina o campo **RECORD KEY** dentro do record-name, pesquisa o registro contendo esta **RECORD KEY** e o regrava com o conteúdo de record-name.

O comando **REWRITE** só pode ser usado depois de um comando **READ** que leia o registro a atualizar para a memória (portanto usando a **RECORD KEY** deste registro).

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

9.1.11. DELETE - VSAM

DELETE arquivo **RECORD**.

Este comando apaga o registro cuja chave coincide com o valor da **RECORD KEY** existente na **FD**.

9.1.12. FECHAMENTO DE ARQUIVOS - VSAM

O comando para fechar o **VSAM** é idêntico ao dos arquivos seqüenciais.

CLOSE nome-do-arquivo.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

10. COMUNICAÇÃO ENTRE PROGRAMAS.

A instrução **CALL** é usada em Cobol para executar um outro programa que já esteja compilado dentro das bibliotecas do Mainframe, e quando este outro programa terminar a execução o nosso programa retoma o controle na instrução seguinte ao **CALL**. Por exemplo, se o nosso programa processa saídas de material, ele pode em cada saída executar um programa **CALCSENH** que confere a exatidão da senha do requisitante.

O processo de invocar outro programa envolve também a passagem de dados entre os dois programas, que está explicado no exemplo abaixo.

Programa SAIDAS-MAT.

IDENTIFICATION DIVISION.

PROGRAM-ID. SAIDAMAT.

*

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT SAIDA ASSIGN TO DA-S-SAIDA.

SELECT MATERIAIS ASSIGN TO DA-I-MATERIAIS

ORGANIZATION IS INDEXED

RECORD KEY IS COD-MATER

FILE STATUS IS STATUS-MATER.

*

DATA DIVISION.

FILE SECTION.

FD SAIDA.

01 REG-SAIDA.

03 COD-CLIENTE PIC 9(6).

03 DIGITO-CLIENTE PIC 9.

03 COD-PROD PIC X(6).

03 QUANT-PROD PIC 9(6).

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

FD MATERIAIS.

01 REG-MATERIAIS.

03 COD-MATER PIC X(6).

03 NOME-PROD PIC X(30).

03 ESTOQUE PIC 9(8).

*

WORKING-STORAGE SECTION.

77 STATUS-MATER PIC XX.

77 SUBROTINA PIC X(09) VALUE "CALCSENHA".

01 AREA-SENHA.

05 WCOD-CLIENTE PIC 9(6).

05 WDIGITO-CLIENTE PIC 9.

05 WRETORNO PIC 9.

*

PROCEDURE DIVISION.

INICIO.

OPEN INPUT SAIDA.

OPEN I-O MATERIAIS.

IF STATUS-MATER NOT = "00"

DISPLAY "ERRO " STATUS-MATER " NO OPEN DO MATERIAL."

STOP RUN.

LEITURAS.

READ SAIDA

AT END GO TO FINAL.

MOVE COD-CLIENTE TO WCOD-CLIENTE

MOVE DIGITO-CLIENTE TO WDIGITO-CLIENTE.

CALL "CALCSENHA" USING AREA-SENHA.

IF WRETORNO NOT = ZERO

DISPLAY WCOD-CLIENTE " " WDIGITO-CLIENTE " SENHA INVALIDA."

GO TO LEITURAS.

MOVE COD-PROD TO COD-MATER

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

READ MATERIAIS

IF STATUS-MATER = "23"

DISPLAY WCOD-CLIENTE " " COD-MATER " MATERIAL NÃO EXISTE."

GO TO LEITURAS.

IF QUANT-PROD > ESTOQUE

**DISPLAY WCOD-CLIENTE " " COD-MATER " " QUANT-PROD
 " SEM ESTOQUE."**

GO TO LEITURAS.

SUBTRACT QUANT-PROD FROM ESTOQUE.

REWRITE REG-MATERIAIS.

IF STATUS-MATER NOT = "00"

DISPLAY "ERRO " STATUS-MATER " NO REWRITE."

GO TO LEITURAS.

FINAL.

CLOSE SAIDA MATERAIS.

STOP RUN.

Programa CALCSENH.

IDENTIFICATION DIVISION.

PROGRAM-ID. CALCSENH.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

77 WTRAB PIC 9(6).

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

LINKAGE SECTION.

01 LINK-SENHA.

05 LCOD-CLIENTE PIC 9(6).

05 LDIGITO-CLIENTE PIC 9.

05 LRETORNO PIC 9.

PROCEDURE DIVISION USING LINK-SENHA.

INICIO.

MOVE 0 TO LRETORNO.

DIVIDE 9 INTO LCOD-CLIENTE GIVING WTRAB REMAINDER LRETORNO.

GOBACK.

Explicações sobre os programas acima:

- O programa **SAIDAMAT** invoca o programa **CALCSENH** com a instrução:

CALL "CALCSENH" USING AREA-SENHA.

- A cláusula **USING AREA-SENHA** da instrução acima indica que toda a informação da área **AREA-SENHA** deve ser passada para o programa **CALCSENH**.

- O programa **CALCSENH** deve ter na **DATA DIVISION** a **LINKAGE SECTION**. Nesta **SECTION** devem ser descritos os dados recebidos pelo programa, ou seja, os campos da **LINKAGE SECTION** do **CALCSENH** representam os campos **AREA-SENHA** passados pelo **CALL** do **SAIDAMAT**.

- A **LINKAGE SECTION** funciona como uma "janela" através da qual o programa **CALCSENH** vê a área **AREA-SENHA**. As alterações feitas nos campos da **LINKAGE SECTION** são na realidade executados sobre a **AREA-SENHA**.

- Na **PROCEDURE DIVISION** do programa **CALC-SENHA** precisa ter a cláusula **USING** para especificar que os dados recebidos do programa chamador estão em **LINK-SENHA**.

- O programa chamado (**CALCSENH**) não pode terminar com a instrução **STOP RUN**, que somente deve ser usada no termino geral do **JOB**. A instrução **GOBACK** deve ser usada para parar o programa chamado e continuar a execução do programa chamador.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Observações:

1 - Pode-se enviar mais de um nível 01 de grupos ou variável pela parâmetro **USING**, porém, é necessário que a Linkage do programa chamado deve estar na mesma sequência;

2 - O comando '**CALL "CALCSENHA" USING AREA-SENHA**' é considerado uma Chamada Estática, uma vez que, o nome do programa já está mencionado no próprio comando, porém, o comando '**CALL "SUBROTINA" USING AREA-SENHA**' é considerada uma Chamada Dinâmica, uma vez que, é invocado uma área do nível **77** da Work-Storage, onde o valor do campo "**SUBROTINA**" poderá ser mudado para outros tipos de chamadas no processamento do programa chamador no caso o Programa **SAIDAS-MAT**.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

11. COMANDOS DIVERSOS

11.1. EVALUATE

A instrução **EVALUATE** serve para substituir cadeias de instruções **IF ELSE** e tornar o programa mais legível. Ela usa a opção **EVALUATE** da sintaxe para avaliar uma expressão ou o valor de uma variável, e um conjunto de opções **WHEN** para definir condições e comandos a executar baseando-se na avaliação acima.

A sintaxe desta instrução é:

EVALUATE variável-ou-expressao **ALSO** variável-ou-expressão ...

WHEN condição

Comandos

WHEN condição

Comandos

.....

WHEN ANY

Comandos

WHEN OTHER

comandos.

Nesta instrução, variavel-ou-expressão pode ter os seguintes valores:

- Uma variável. Neste caso as opções **WHEN** devem testar valores ou faixas de valores (usando o conector **THRU**) para esta variável. Exemplo:

EVALUATE NOTA

WHEN 1 THRU 5

DISPLAY "REPROVADO"

WHEN 6 TO 9

DISPLAY "APROVADO"

WHEN 10

DISPLAY "APROVADO COM LOUVOR".

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

- Uma expressão. Neste caso as opções **WHEN** devem testar somente as constantes lógicas **TRUE** e **FALSE**. Exemplo:

```
EVALUATE NOTA > 5
  WHEN TRUE
    DISPLAY "APROVADO"
  WHEN FALSE
    DISPLAY "REPROVADO".
```

- As constantes lógicas **TRUE** ou **FALSE**. Neste caso as opções **WHEN** podem testar qualquer condição, mesmo que não tenham relação entre si. Se as
- condições testadas forem verdadeiras, os comandos associados no **WHEN** serão executados. Exemplo:

```
EVALUATE TRUE
  WHEN NOTA > 5
    DISPLAY "APROVADO"
  WHEN MES = 1
    DISPLAY "JANEIRO".
```

11.2. REDEFINES

A cláusula **REDEFINES** é usada quando alguma informação da **DATA DIVISION** precisa ser definida em dois ou mais formatos diferentes. Por exemplo, em um campo de data podemos o formato dia/mês/ano ou ano/mês/dia:

01 DATA-DIRETA.

```
03 DIA-D    PIC 99.
```

```
03 MÊS-D    PIC 99.
```

```
03 ANO-D    PIC 99.
```

01 DATA-INVERSA REDEFINES DATA-DIRETA.

```
03 ANO-I    PIC 99.
```

```
03 MÊS-I    PIC 99.
```

```
03 DIA-I    PIC 99.
```

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Os campos redefinidos ocupam o mesmo lugar na memória. No exemplo acima, o valor colocado no campo **ANO-I** aparece também em **DIA-D**. Os campos da área redefinida não guardam nenhuma relação com o formato e posição dos campos da primeira definição da área, e a única condição é que o total de bytes das duas áreas seja igual. Outro ex.:

01 VALORES-PEQUENOS.

03 VALOR-1 PIC 9(6).

03 FILLER PIC X(4).

01 VALORES-GRANDES REDEFINES VALORES-PEQUENOS.

03 VALOR-2 PIC 9(8).

03 FILLER PIC XX.

11.3. INSPECT

A instrução **INSPECT** varre um campo alfa-numérico contando caracteres ou fazendo substituições de certos caracteres por outros. A sintaxe para fazer substituição é:

INSPECT campo **REPLACING** condicao-1 var-1 **BY** var-2.

ou

INSPECT campo **REPLACING CHARACTERS BY** var-2.

Condição-1 pode ser uma das seguintes palavras:

ALL - Todos os caracteres iguais a var-1 serão substituídos por var-2.

LEADING - Todos os caracteres iguais a var-1 que estiverem no início do campo serão substituídos por var-2. O primeiro caractere diferente de var-1 interrompe a substituição.

FIRST - O primeiro caractere igual a var-1 será substituído por var-2. Esta condição provoca somente uma substituição.

O uso da opção **CHARACTERS** faz com que todos os caracteres do campo sejam substituídos por var-2.

Os formatos acima podem ser ampliados pela cláusula **AFTER/BEFORE** como segue:

INSPECT campo **REPLACING** condicao-1 var-1 **BY** var-2 condição-2 var-3.

ou

INSPECT campo **REPLACING CHARACTERS BY** var-2 condição-2 var-3.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Aqui, condição-2 pode ser:

AFTER – A substituição começa depois do carácter var-3.

BEFORE – A substituição é efetuada nos caracteres antes de var-3.

A sintaxe para a instrução **INSPECT** contar caracteres é:

INSPECT campo **TALLYING** var-1 condicao var-2.

ou

INSPECT campo **TALLYING CHARACTERS** var-2.

Esta instrução conta caracteres dependendo da condição especificada. Condição pode ser qualquer uma das citadas na opção de substituição vista acima, excluindo a opção **FIRST**.

O resultado da contagem aparece na variável chamada var-1.

11.4. STRING.

A instrução **STRING** concatena o conteúdo de vários campos em um campo receptor. Seu formato é:

STRING campo-1 **DELIMITED BY** delimitador
Campo-2 **DELIMITED BY** delimitador ...
INTO campo-n **WITH POINTER** campo-p

Neste exemplo, os conteúdos de campo-1, campo-2 etc. são reunidos e movidos para campo-n. Por exemplo:

Antes do **STRING**:

Campo-1	campo-2	campo-n
JOSE	SILVA	

Após o **STRING**:

Campo-1	campo-2	campo-n
JOSE	SILVA	JOSESILVA

O parâmetro **DELIMITER** é opcional. Sua função é interromper a movimentação de dados para o campo receptor. Exemplo:

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

STRING VAR-1 DELIMITED BY "S"
VAR-2 DELIMITED BY "V"
INTO RESP.

Antes do **STRING**:

VAR-1	VAR-2	RESP
JOSE	SILVA	

Após o **STRING**:

VAR-1	VAR-2	RESP
JOSE	SILVA	JOSIL

A função de **WITH POINTER**, também opcional, é indicar a posição do campo receptor que começa a receber os dados. Assim no exemplo:

STRING VAR-1 DELIMITED BY "S"
VAR-2 DELIMITED BY "V"
INTO RESP
WITH POINTER DESLOC.

Antes do **STRING**:

VAR-1	VAR-2	DESLOC	RESP
JOSE	SILVA	3	XXXXXXXX

Após o **STRING**:

VAR-1	VAR-2	DESLOC	RESP
JOSE	SILVA	3	XXJOSIL

11.5. UNSTRING

A função do comando **UNSTRING** é inversa do comando **STRING**, isto é, ele quebra o conteúdo de um campo emissor em varias partes movendo os resultados para vários campos. O formato deste comando é:

UNSTRING campo

[DELIMITED BY [ALL] delim [OR [ALL] delim....

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

```
      INTO var-1
      [DELIMITER IN var-d]
      [COUNT IN count]
      Var-2
      .....
      [WITH POINTER pointer]
      [TALLYING IN var-t]
```

Usando esta sintaxe, os dados são extraídos da variável campo e colocados em var-1, var-2, var-3 ... e assim sucessivamente. Veja o exemplo:

```
77 VAR-1      PIC XX.
77 VAR-2      PIC X(4).
77 CAMPO      PIC X(6) VALUE "123456".
```

PROCEDURE DIVISION.

```
      UNSTRING CAMPO INTO VAR-1 VAR-2.
```

Apos o comando:

CAMPO	VAR-1	VAR-2
123456	12	3456

A clausula **DELIMITED** no comando permite quebrar o campo emissor pelos delimitadores. Sem a clausula o comando usará os comprimentos dos campos receptores no processo. Veja o exemplo:

```
77 VAR-1      PIC XX.
77 VAR-2      PIC X(4).
77 CAMPO      PIC X(6) VALUE "JOAO SILVA".
```

PROCEDURE DIVISION.

```
      UNSTRING CAMPO DELIMITED BY SPACES
      INTO VAR-1 VAR-2.
```

Após o comando:

CAMPO	VAR-1	VAR-2
JOAO SILVA	JOAO	SILVA

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Se no campo sendo quebrado houver a ocorrência de um delimitador repetido, como por exemplo duas palavras separadas por dois ou mais espaços, a cláusula ALL faz com que o conjunto de delimitadores seja considerado um só delimitador. Sem a cláusula ALL, para cada delimitador será usado um campo receptor em branco.

DELIMITER IN define uma variável para receber o delimitador encontrado no campo emissor.

COUNT IN define uma variável numérica para receber a quantidade de caracteres colocada no campo receptor.

WITH POINTER define uma variável numérica que conterá a posição de início de movimentação dos dados no campo emissor.

TALLYING IN define um campo numérico para conter a quantidade de campos receptores realmente usados na movimentação. Esta variável deve ser inicializada antes da instrução **UNSTRING**.

11.6. TABELAS INDEXADAS

Em Cobol pode-se associar índices especiais aos elementos de uma tabela. Estes índices são mais eficientes que o processo tradicional de subscrição, onde se usam variáveis da **WORKING-STORAGE** como índice, além de permitir recursos avançados como pesquisa binária para localização de elementos da tabela.

O Cobol aloca diretamente os registradores da **CPU** para os índices especiais. Em consequência o programa não deve aloca-los como uma variável da **WORKING-STORAGE**. A definição destes índices é feita somente na especificação da tabela que agora terá a seguinte sintaxe:

Nível variável **OCCURS** inteiro **TIMES**
 INDEXED BY nome-do-índice
 ASCENDING KEY campo-chave.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

Exemplo:

02 VENDAS OCCURS 12 TIMES

INDEXED BY IND-VENDAS

ASCENDING KEY MES-VENDA.

03 MES-VENDA PIC 99.

03 VALOR-VENDA PIC 9(6)V99.

Neste exemplo, **IND-VENDAS** é o índice alocado para a tabela. Este índice não pode ser definido na **WORKING-STORAGE**.

A cláusula **ASCENDING KEY** é opcional e serve para declarar que a tabela é ordenada pela variável **MÊS-VENDA**. Também podemos declarar tabelas ordenadas em ordem decrescente usando **DESCENDING KEY** nesta cláusula.

11.6.1. SET

Como o índice definido pela opção **INDEXED BY** nas tabelas indexadas não é uma variável padrão do Cobol, não se pode usar as instruções (**MOVE, ADD, SUBTRACT**) do Cobol para alterar seu valor. Usa-se a instrução **SET** com as seguintes sintaxes:

- Para mover um valor para o index:
SET index TO variável-ou-literal
- Para copiar o valor do index para uma variável:
SET variável **TO** index
- Para somar um valor ao index:
SET index UP BY valor
- Para subtrair um valor do index:
SET index DOWN BY valor.

11.6.2. SEARCH

A instrução **SEARCH** é usada para pesquisas em tabelas indexadas. Existem 2 opções de pesquisa com **SEARCH**: pesquisa seqüencial e pesquisa binária.

11.6.2.1. PESQUISA SEQUENCIAL

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

A sintaxe para pesquisa seqüencial é:

```
SEARCH tabela
      VARYING índice
      AT END comandos
      WHEN condição
          Comandos.
```

Nesta instrução, tabela é o item que contem a clausula **INDEXED BY**.

A clausula **VARYING** é opcional e define uma variavel para mostrar o índice da tabela. O Cobol sempre usa o indexador definido pelo **INDEXED BY** para as pesquisas, mas se usarmos a clausula **VARYING** o Cobol varia também a variável índice da clausula, que portanto terá o mesmo valor do indexador da tabela.

A clausula **AT END** especifica os comandos que devem ser executados quando a pesquisa na tabela não teve sucesso.

A clausula **WHEN** define realmente as condições da pesquisa, e os comandos a serem executados quando a pesquisa tiver sucesso. No caso de sucesso o indexador e o índice da clausula **VARYING** estarão posicionados no elemento da tabela que satisfaz as condições da clausula **WHEN**.

Uma medida importante nas pesquisas seqüenciais é inicializar o indexador da tabela, geralmente com o valor 1 que representa o primeiro item da tabela, porque a instrução **SEARCH** inicia a pesquisa usando o valor encontrado no momento no indexador.

Exemplo de comando **SEARCH** usando a tabela de vendas do exemplo de tabelas indexadas:

```
SEARCH VENDAS
      VARYING VARIABEL-DE-TRABALHO
      AT END
          DISPLAY "ESTE MÊS NÃO TEVE VENDAS"
      WHEN MÊS-VENDA (IND-VENDAS) = MÊS-DA-PESQUISA
          MOVE VALOR-VENDA (IND-VENDAS) TO .....
```

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

11.6.2.2. PESQUISA BINÁRIA

As diferenças entre pesquisa seqüencial e binária são:

- Na pesquisa binária a tabela deve ser definida com a clausula **KEY**, que definirá a chave que será usada na pesquisa.
- A clausula **WHEN** do **SEARCH** deverá testar necessariamente o campo definido na clausula **KEY** da tabela.
- A opção **ALL** da sintaxe do **SEARCH** é a opção que indica pesquisa binária.
- O comando **SEARCH** não tem a opção **VARYING**.

A sintaxe do **SEARCH** em pesquisa binária é:

SEARCH ALL tabela

AT END comandos

WHEN condição

Comandos.

O exemplo do **SEARCH** para pesquisa binária é:

SEARCH ALL VENDAS

AT END

DISPLAY "ESTE MÊS NÃO TEVE VENDAS"

WHEN MÊS-VENDA (IND-VENDAS) = MÊS-DA-PESQUISA

MOVE VALOR-VENDA (IND-VENDAS) TO

11.7. ALTER

A instrução **ALTER** serve para mudar o direcionamento de uma instrução **GO TO**. A instrução a ser alterada deve ser a única dentro de um parágrafo, e a alteração ocorre dinamicamente na memória quando o programa estiver sendo executado. Muitas empresas não recomendam o uso desta instrução porque ela força o programa a executar um desvio que não esta escrito no fonte original. Por exemplo no exemplo abaixo:

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

PARAGR-1.

GO TO PARAGR-2.

.....

ALTER PARAGR-1 TO PROCEED TO PARAGR-3

.....

PARAGR-2.

.....

PARAGR-3.

Neste exemplo, se o programa passar alguma vez pela instrução **ALTER**, a instrução **GO TO** que aparece no **PARAGR-1** funcionará como se estivesse sido escrito:

GO TO PARAGR-3.

11.8. GO TO DEPENDING ON

O comando **GO TO** pode apontar para varios paragrafos quando usado com a clausula **DEPENDING ON**. Para isto é necessario ter um campo para servir como indexador, e o desvio será feito para o paragrafo cuja sequencia for igual ao valor do indexador. Por exemplo:

GO TO PARAGR-1 PARAGR-2 PARAGR-3 DEPENDING ON CONTADOR.

O desvio será feito para **PARAGR-1, PARAGR-2 PARAGR-3** se o valor de **CONTADOR** for respectivamente 1, 2 ou 3. Qualquer outro valor de **CONTADOR** não provoca desvio do **GO TO**.

11.9. NOMES CONDICIONAIS

Nomes condicionais (**CONDITION NAMES**) são recursos para atribuir nomes alternativos para variáveis quando elas recebem determinados valores. Para esta atribuição, é usado o nível **88** da **DATA DIVISION**, como no exemplo:

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

77 ESTADO-CIVIL	PIC 9.
88 SOLTEIRO	VALUE 1.
88 CASADO	VALUE 2.
88 DESQUITADO	VALUE 3.

Na **PROCEDURE DIVISION**, os testes condicionais da variável **ESTADO-CIVIL** podem ser feitos agora de duas maneiras, usando o nome original da variável ou o nome alternativo que implicitamente representa um valor da variável. Ex.:

IF ESTADO-CIVIL = 1

DISPLAY 'ENTRADA PROIBIDA. ESTE CLUBE E SOMENTE PÁRA CASADOS.'

ou

IF SOLTEIRO

DISPLAY 'ENTRADA PROIBIDA. ESTE CLUBE E SOMENTE PÁRA CASADOS.'

Os nomes condicionais só podem ser usados em variáveis de nível elementar (que tenham a cláusula **PIC**).

11.10. COPY

O comando copy é usado para copiar trechos de programa fonte que estão armazenados em bibliotecas (**BOOKS**) para dentro dos programas Cobol. Estes trechos de programa podem conter descrições de registros de arquivos, rotinas padronizadas que devem ser repetidas em muitos programas etc. No local conveniente do programa onde deverá ser copiado o **BOOK** basta codificar:

COPY nome-do-book.

Manual de Referência: COBOL - Programação

Curso de Cobol - Apresentação

12. DIAGRAMA DE BLOCO

12.1. O QUE É UM DIAGRAMA DE BLOCO?

O diagrama de blocos é uma forma padronizada e eficaz para representar os passos lógicos de um determinado processamento.

Com o diagrama podemos definir uma seqüência de símbolos, com significado bem definido, portanto, sua principal função é a de facilitar a visualização dos passos de um processamento.

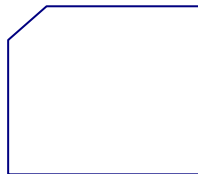
12.2. SIMBOLOGIA

Existem varios tipos de simbolos utilizados na confecção de um diagrama de blocos, porem, abaixo iremos demosntras os mais utilizados:

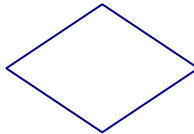
Ação



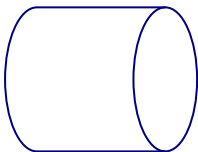
Cartão Perfurado



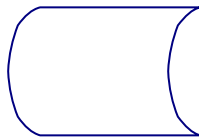
Decisão



Arquivo VSAM



Arquivo Sequencial



Fita Magnética



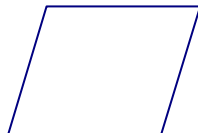
Conector



Banco de Dados



Manuseio de arquivos



Relatórios



Início / Fim de processo

