Curso de Progress®

CONTENIDO

1. INTRODUCCION A PROGRESS

Progress es un paquete de software que posee todo lo necesario para el desarrollo de aplicaciones multiusuarias, estando conformado por:

- Base de Datos Relacional
- Diccionario de Datos
- Lenguaje de Programación
- Formateador de Pantallas y reportes
- Editor de Procedimientos

Base de datos relacional :
Es el lugar donde se almacena la información en forma de tablas.

Diccionario de Datos:
Es la herramienta que permite describir el esquema o estructura de la base de datos a cargar o modificar.

Lenguaje de Programación:
El lenguaje de 4ta. Generación Progress es un lenguaje NO procedural, es decir que el programador le tiene que indicar **QUE** es lo que desea hacer, y no **COMO** quiere hacerlo.

Formateador de Pantallas y Reportes:
Permite la generación en forma sencilla (y hasta automática) de pantallas y reportes.

Editor de Procedimientos:
Es la herramienta de la cual dispone el programador para escribir los procedimientos que administran la información.

2. SENTENCIAS DE MANEJO DE DATOS

Progress usa dos tipos de buffers para el control de movimientos de datos desde y hacia la base de datos:

***Buffer de Registro:** Es un área temporaria en memoria de almacenamiento de datos para campos, registros o variables.

. Cuando Progress lee un registro, lo coloca en el buffer de registro.

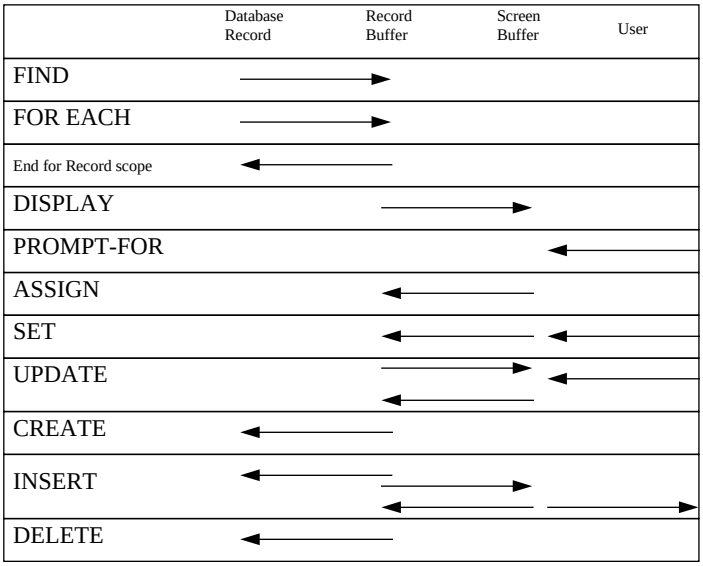
Cuando se escribe un registro a la base, Progress lo vuelca desde el buffer de registro.

***Buffer de Pantalla:** Es un área de pantalla para mostrar o ingresar información.

Las sentencias que mueven los datos de Progress son:

FIND:	Lectura directa de un registro
FOR EACH:	Lectura secuencial de registros.
DISPLAY:	Muestra datos en un dispositivo de salida.
PROMPT -FOR:	Pide el ingreso de datos.
ASSIGN:	Asignar valores
SET:	Pide y asigna datos.
UPDATE:	Muestra, pide y asigna datos.
CREATE:	Crea un registro vacío.
INSERT:	Crea, muestra, pide y asigna datos.
DELETE:	Elimina un registro.

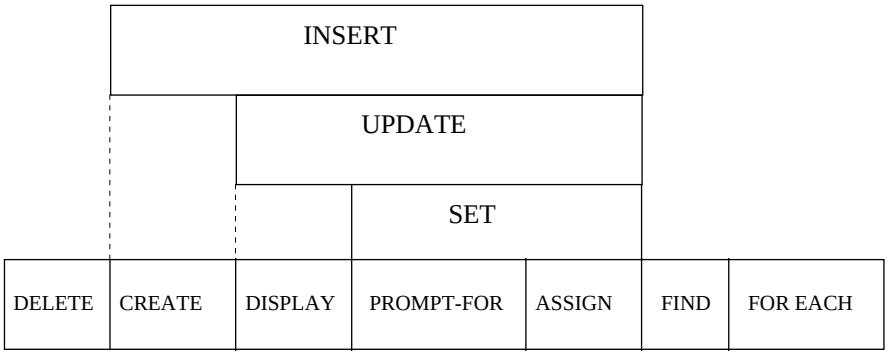
La siguiente tabla muestra como cada una de las sentencias mueven datos:



Para utilizar éste gráfico, busque la sentencia en la que está interesado. El origen de la flecha indica desde donde la sentencia toma los datos. La flecha indica a donde los datos son almacenados.
Por ejemplo: La sentencia **DISPLAY** toma datos del buffer de registro, mueve éste al buffer de pantalla y muestra los mismos en la pantalla.

Algunas sentencias más complejas son combinaciones de otras más sencillas.
Por ejemplo: la sentencia **INSERT** es muy potente y combina varios pasos de procesamientos en una sentencia. Pero en algunas situaciones ésta es menos flexible que utilizar las sentencias **CREATE**, **DISPLAY** y **SET** en forma individual.

La siguiente figura muestra todas las sentencias de manejo primario de datos e ilustra que sentencias reemplazan a otras.



Agregando Registros

Para agregar registros a un archivo de la base de datos se puede utilizar la sentencia **INSERT**, cuyo formato genérico es:

INSERT *nombre-de-archivo* **Frase-de-diseño**.

Sentencia:

INSERT *clientes* **WITH 2 COLUMNS.**

permitirá agregar un registro de clientes, brindando ayuda en cada uno de los campos que solicite, al mismo tiempo que valida el ingreso, mostrando el mensaje de error correspondiente si el dato ingresado fuera incorrecto.

En el archivo de clientes, el código del cliente ha sido definido como clave primaria del archivo. Si se ingresa un código de cliente ya asignado a otro registro, aparecerá un mensaje de error.

El siguiente procedimiento realiza la misma función que el mostrado anteriormente, pero mediante otras sentencias (Ver tabla anterior).

CREATE *clientes*.
DISPLAY *clientes* **WITH 2 COLUMNS.**
SET *clientes*.

CREATE	Crea un registro de clientes vacíos (con los valores iniciales)
DISPLAY	Mueve el registro desde el buffer del registro al buffer de pantalla. (mostrando el contenido del buffer en la pantalla)

SET	
.PROMPT-FOR	Pide los datos al usuario, poniendo el ingreso en el buffer de pantalla.
.ASSIGN	Mueve los datos desde el buffer de pantalla al buffer de registro. (cuando finaliza el proceso, el registro será grabado en la base de datos).

Modificando Registros

Si se necesita modificar datos en un registro, se deberán utilizar tres simples sentencias para realizar ésta tarea:

PROMPT-FOR *clientes.nro-cliente*.	/* Solicita nro-cliente para que el usuario lo ingrese */
FIND *clientes* **USING** *nro-cliente*.	/* Lee el archivo clientes de acuerdo al nro-cliente ingresado */
UPDATE *nombre domicilio localidad*.	/* Muestra el registro apropiado y permite cambiar la información en los campos especificados./*

Borrando Registros

Mediante una simple modificación del procedimiento anterior, se podrán borrar registros.

PROMPT-FOR *clientes.nro-de-cliente.*
FIND *clientes* **USING** *clientes.nro-de-cliente*.
- **DELETE** *clientes.*

La primera y segunda sentencia son las mismas, pero la tercera sentencia deletea el registro en vez de modificarlo.

PROMPT-FOR: Acepta su entrada (nro. de cliente), colocándola en el buffer de pantalla.
FIND: Lee el registro del cliente y coloca éste en el buffer de registro.
DELETE: Limpia el buffer de registro y borra el registro de la base de datos.

Mostrando Datos

El siguiente procedimiento muestra información, de un cliente determinado, por pantalla:

PROMPT-FOR *clientes.nro-de-cliente.*
FIND *clientes* **USING** *clientes.nro-de-cliente*.
DISPLAY *nombre domicilio localidad.*

Procedimiento similar a los anteriores.
Primero se solicita el número del cliente a consultar.
Se lee el archivo correspondiente de acuerdo al dato ingresado.
En lo que varía este procedimiento, es la última sentencia, que en vez de permitir modificar o eliminar el registro accedido, lo muestra por pantalla.

PROMPT-FOR: Acepta su entrada (nro. del cliente), colocándola en el buffer de pantalla.
FIND: Lee el registro del cliente y coloca este en el buffer de registro.
DISPLAY: Mueve los valores de los campos del registro en el buffer de registro al buffer de pantalla, mostrándolos en la terminal.

Leyendo Registros

-Lectura Directa

En todos los procedimientos que hemos realizado hasta ahora, para leer registros de la base de datos, hemos utilizado la sentencia **FIND**.

La sentencia **FIND**, utiliza un índice para localizar un determinado registro en un archivo, moviendo dicho registro al "buffer de registros".

- Lectura Secuencial

Para leer un archivo, un registro tras otro en forma secuencial, se utiliza la sentencia **FOR EACH**.

La sentencia **FOR EACH** es la primera de un conjunto de sentencias. Estas sentencias se ejecutarán cada vez que la sentencia **FOR EACH** lea un registro en forma secuencial. Dicho conjunto de sentencias finalizarán con una sentencia **END**. Es decir:

```
FOR EACH nombre-de-archivo:

        SENTENCIA-1.
        SENTENCIA-2.
        SENTENCIA-3.

END.
```

Con la sentencia **FOR EACH**, **PROGRESS** lee un registro del archivo. Luego ejecuta la SENTENCIA-1. Luego ejecuta la SENTENCIA-2. Luego ejecuta la SENTENCIA-3. Por último, con la sentencia **END**, graba el registro en el archivo de la base de datos, limpia el buffer de registro para permitir que sea leído el próximo registro mediante el **FOR EACH**.

Notar que la sentencia **FOR EACH**, esta terminada en dos puntos. Esto es por que la misma es "sentencia cabecera" de un block (conjunto de sentencias que termina en **END**).

-Distintas Opciones de Lectura:

El Bloque **FOR EACH** y la sentencia **FIND**, son las principales herramientas de lectura. Utilizando opciones en éstas sentencias, podrá especificar los registros que busca leer.

-Leyendo registros condicionalmente: Where

Cuando se necesite leer registros que satisfagan una cierta condición o conjunto de condiciones, se podrá utilizar la opción **WHERE** para indicar dichas condiciones.

En el siguiente ejemplo, la sentencia **FOR EACH** lee y muestra solamente aquellos registros de clientes con un saldo mayor a 200.

```
FOR EACH clientes WHERE saldo > 200:
        DISPLAY nombre saldo.
END.
```

Supóngase que busca mostrar los nombres de clientes que estén radicados en "***AVELLANEDA***":

```
FOR EACH clientes WHERE localidad EQ "AVELLANEDA":
        DISPLAY nombre.
END.
```

"AVELLANEDA" está encerrado entre comillas por que ésta es una constante de tipo caracter.

-Leyendo Registros que coincidan con un valor suministrado por el usuario: USING

La opción **USING** permite a **PROGRESS** buscar un registro que tiene un campo que coincide con el valor ingresado por el usuario.
Por ejemplo:

```
PROMPT-FOR clientes.nro-de-cliente.
FIND clientes USING nro-de-cliente.
DISPLAY clientes WITH 2 COLUMNS.
```

Utilizar **USING**, es decirle a **PROGRESS** que ubique un registro de manera tal que el campo indicado tenga el mismo valor en el buffer de pantalla que en el buffer de registro. Es decir que el ejemplo suministrado es igual que el siguiente:

```
PROMPT-FOR clientes.nro-de-cliente.
FIND clientes WHERE nro-de-cliente = INPUT nro-de-cliente.
DISPLAY clientes WITH 2 COLUMNS.
```

-Buscando el siguiente, el previo, el primer, ó el último registro en un archivo.

PROGRESS podrá leer el siguiente, el previo, el primer ó el último registro de un archivo mediante las opciones:

***NEXT**
***PREV**
***FIRST**
***LAST**

Por ejemplo:

FIND FIRST *clientes.*
Busca el primer registro en el archivo de clientes.

-Comparando valores de campos con una expresión de carácter: BEGINS y MATCHES.

Si necesita buscar una lista de nombres de clientes que comienzan con **"PE"**, deberá utilizar la función **BEGINS** para realizarlo:

```
FOR EACH clientes WHERE nombre BEGINS "PE":
        DISPLAY nombre.
END.
```

Cada registro del archivo de clientes con el nombre que comience con **"PE"** es leído en el buffer de registro en cada iteración del bloque **FOR EACH**, y es mostrado en el buffer de pantalla.

*NOTA: Como es posible que más de un registro de clientes comience con "PE", se deberá escribir el procedimiento con **FOR EACH** y no con **FIND.***

Otra manera de recuperar registros comparando valores de campos es con el uso de la función **MATCHES**.

```
FOR EACH clientes WHERE nombre MATCHES "PE*":
        DISPLAY nombre.
END.
```

El asterisco **"*"** que continúa a **"PE"**, recupera todos los registros que comienzan con **PE**. Aquí, la función **MATCHES**, trabaja de igual manera a la función **BEGINS**.

La diferencia entre **BEGINS** y **MATCHES**, es que **BEGINS** busca por un campo que comience con los caracteres especificados, y si se utiliza un campo que es índice, BEGINS utiliza ese índice para encontrar los registros apropiados, sin buscar en todos los registros del archivo.

MATCHES en cambio accede y busca todos los registros de un archivo, pero puede ubicar caracteres en el medio del campo. Por Ejemplo: **"*SA*"**.

-Identificando un Indice para la selección de Registros: USE-INDEX

Mediante la opción **USE-INDEX** podrá hacer mención a un especificado índice para buscar y clasificar los registros que busca leer de la base de datos:

```
FOR EACH CLIENTES USE-INDEX clave2:
        DISPLAY nombre domicilio localidad.
END.
```

Este procedimiento utiliza el índice llamado 'clave2' para leer registros de clientes en cada iteración del bloque **FOR EACH** y ordena el listado por el contenido de dicha clave.

-Ordenando registros en un especificado orden: BY

Si busca ordenar un archivo por un campo que no pertenezca a una clave, deberá utilizar la opción **BY**.

```
FOR EACH clientes WHERE saldo > 100 BY nombre DESCENDING:
        DISPLAY nombre saldo.
END.
```

En éste procedimiento, **PROGRESS** usa el índice primario para buscar los registros de clientes que satisfagan la condición **WHERE** y ordena los registros de clientes por (**BY**) el nombre en orden alfabético descendiente (**DESCENDING**).

3. BLOQUES

Introducción

Uno de los más importantes logros en lenguajes de computación es el concepto de "programación estructurada". La mayoría de los lenguajes requieren un gran esfuerzo para segmentar un programa en estructuras apropiadas. **PROGRESS** permite programar en forma estructurada de manera natural y sin mucho trabajo.
La unidad básica de estructura en un procedimiento de **PROGRESS**, es un **BLOCK**. Un **BLOCK** es una serie de sentencias que **PROGRESS** trata como una sola unidad. Cada block comienza con una sentencia de cabecera y concluye con una sentencia **END**.

Ejemplo:

```
FOR EACH clientes:
        DISPLAY  clientes:
END.
```

La sentencia que comienza un block se denomina *"cabecera de block"*, y es diferente a otros tipos de sentencias, en dos sentidos:

- Termina en dos puntos (:). Las demás terminan en un punto.
- Puede tener una etiqueta. La etiqueta es un nombre de proceso, y también terminan en dos puntos (:).

La estructura de **BLOCKS** del lenguaje **PROGRESS**, ofrece un ancho de servicios de procesamiento, que están por encima de las capacidades de cada sentencia en particular:

*Looping.
*Lectura de registros.
*Alcances de Frames.
*Alcances de Registros.
*Undo.
*Procesamiento de error.
*Procesamiento de Endkey.

PROPIEDAD	REPEAT Implícito	REPEAT Explícito	FOR EACH Implícito	FOR EACH Explícito	DO Implícito	DO Explícito	PROCEDIM. Implícito	PROCEDIM. Explícito
LOOPING	SI	WHILE TO/BY	SI	WHILE TO/BY	NO	WHILE TO/BY	NO	NO
LECTURA DE REGISTROS	NO	NO	SI		NO	NO	NO	NO
ALCANCES DE FRAMES	SI	WITH FRAME	SI	WITH FRAME		WITH FRAME	SI	NO
ALCANCE DE REGISTROS	SI	FOR	SI	NO	NO	FOR	SI	NO
UNDO	SI	NO	SI	NO	NO	Transaction on error	SI	NO
PROCESAM. DE ERROR	SI	ON ERROR	SI	ON ERROR	NO	ON ERROR	SI	NO
PROCESAM. DE ENDKEY	SI	ON ENDKEY	NO	ON ENDKEY	NO	ON ENDKEY	SI	NO

Tipos de Bloques

Hay 4 tipos de bloques:

***FOR EACH**
***REPEAT**
***DO**
***Procedimientos**

FOR EACH

Comienza con una sentencia **FOR EACH**. Posee las capacidades de "looping" y lectura. Cada vez que el block **FOR EACH** hace un loop, lee en forma secuencial otro registro del archivo indicado en la sentencia cabecera.

REPEAT

Comienza con una sentencia **REPEAT**. Al igual que el block **FOR EACH**, realiza loops, pero no lee automáticamente registros.

DO

Comienzan con una sentencia **DO**. Utilizados especialmente para agrupar sentencias. No realizan Loops, ni leen registros automáticamente a no ser que se le indique en forma específica.

Procedimientos

Todo procedimiento que se escriba es un Block.

Otras aplicaciones de los Blocks

*Agrupar sentencias para "flow" control: Permiten procesar un grupo de sentencias bajo ciertas circunstancias.

*Identificar un contexto para un bloque: Identifica un archivo determinado para ser utilizado por todo el grupo de sentencias.

*Identificar puntos de un procedimiento donde se busca que PROGRESS realice un servicio de procesamiento.

Agrupando sentencias para Flow Control

Vea el siguiente procesamiento:

```
PROMPT-FOR clientes.nro-de-cliente.
FIND clientes USING nro-de-cliente NO-ERROR.
IF NOT AVAILABLE clientes THEN
        DO:
                MESSAGE "Nuevo registro de clientes creado".
                CREATE clientes.
                ASSIGN nro-cliente.
        END.
        ELSE
                UPDATE clientes WITH 2 COLUMNS.
```

Similar a todo procedimiento **PROGRESS**, el procedimiento completo es un block. Este block también contiene un block **DO**.

El procedimiento comienza poniendo el número de cliente y utiliza el mismo para buscar el registro en el archivo de clientes. En caso de que el cliente no esté disponible, el procedimiento realiza lo siguiente:

* Comunica que se creará un cliente nuevo.
* Crea un registro nuevo.
* Asigna al registro el número de cliente que se ha ingresado en el comienzo del procedimiento.

Un bloque **DO** que solamente agrupa sentencias, maneja esta tarea:

DO:
MESSAGE "Nuevo registros de clientes creado".
CREATE clientes.
ASSIGN clientes.
END.

Identificando el contexto para un bloque

Si busca información acerca de un particular cliente:

- **REPEAT:**
  PROMPT-FOR nro-de-cliente.
  FIND clientes USING nro-de-cliente.
  DISPLAY nombre domicilio localidad.
  END.

Cuando ejecuta este procedimiento, si existe un campo con el nombre nro-de-cliente en otro archivo, aparecerá el siguiente mensaje:

**nro-de-cliente is ambiguous with clientes.nro-de-cliente with arch.nro-de-cliente.

El mensaje indica que el campo cliente está en los archivos clientes y arch. **PROGRESS** no puede determinar a qué campo de los dos archivos se está haciendo referencia, para poder pedir información. Para solucionar este inconveniente se deberá modificar la línea del **REPEAT** de la siguiente manera:

REPEAT FOR *clientes*:
.
.
.

De esta manera en el block **REPEAT**, se especifica (**PARA**) que archivo se va a realizar el pedido de la información.

Servicios de procesamiento de los bloques

Looping

El 'looping' ó iteración es uno de los servicios automáticos de procesamiento que **PROGRESS** provee para los blocks. Específicamente, los block: **REPEAT**, **FOR EACH** iteran implícitamente. Por ejemplo:

/* El siguiente block **REPEAT** itera a través de las sentencias hasta que el usuario presione END-ERROR */

```
REPEAT:
        PROMPT-FOR clientes.nro-de-cliente.
        FIND clientes USING nro-de-cliente.
        DISPLAY clientes WITH 2 COLUMNS.
END.
```

/* El siguiente block FOR EACH lee los registros del archivo de clientes de a uno por vez, procesando las sentencias del block para cada uno de los registros */

```
FOR EACH clientes:
        DISPLAY clientes WITH 2 COLUMNS.
END.
```

Los blocks **REPEAT** y **FOR EACH** iteran automáticamente. Podrá realizar un block **DO** que itere; mediante la opción **WHILE** o la opción **TO** en el encabezado del block **DO**:

/* El siguiente block itera a través de las sentencias variando a la variable i desde 1 hasta 5, sumando 1 en cada iteración */

```
DEFINE VARIABLE i AS INTEGER.
DO i = 1 TO 5:
        FIND NEXT clientes.
        DISPLAY clientes WITH 2 COLUMNS.
END.
```

Lectura de Registros

Solamente un tipo de block, (el block **FOR EACH)**, implícitamente lee los registros cada vez que itera. Los registros son leídos del archivo o de los archivos que se nombre en el encabezado del block **FOR EACH.**

Por ejemplo:

/* El siguiente block FOR EACH lee los registros del archivo de clientes de a uno por vez, procesando las sentencias del block para cada uno de los registros */

```
FOR EACH clientes:
        DISPLAY clientes WITH 2 COLUMNS.
END.
```

Asignación de diseños para bloques

Cada sentencia que muestre datos o requiera datos a ser ingresados, utiliza un diseño (recuadro) determinado. Un procedimiento puede utilizar varios diseños diferentes. El diseño utilizado por cada sentencia en particular está determinado por dos cosas:

* Primero, si la sentencia explicita el nombre de un diseño, la sentencia utiliza éste diseño para mostrar datos.

* Si la sentencia no explicita el nombre de un diseño, utilizará por defecto el diseño para el bloque en el cual la sentencia ocurre.

Diseños por defecto

PROGRESS automáticamente crea un diseño por defecto para blocks **REPEAT**, **FOR EACH** y Procedimientos, el cual se denomina "sin nombre".

Si se nombra un diseño en el encabezado del block, este diseño será el diseño del block en lugar del diseño 'sin nombre'. El diseño es también asignado a un block, si dicho block es el primero en referenciar el diseño.

Por ejemplo:

/* El diseño por defecto para el block de procedimiento es el diseño sin nombre. La primera sentencia **DISPLAY** usa un diseño diferente, denominado aaa. El diseño por block **REPEAT** es denominado bbb. La sentencia **PROMPT-FOR** no utiliza el diseño por defecto para el block, sino que utiliza el diseño aaa. La sentencia **DISPLAY** utiliza el diseño por defecto para el block **REPEAT**. */

```
DISPLAY "Muestra Cliente" WITH FRAME aaa.
REPEAT WITH FRAME bbb.
        PROMPT-FOR clientes.nro-de-cliente WITH FRAME aaa.
        FIND clientes USING nro-de-cliente.
        DISPLAY clientes WITH 2 COLUMNS.
END.
```

En este ejemplo, el diseño por defecto para el block del procedimiento es 'sin nombre'. El diseño por defecto para el block **REPEAT** es el diseño bbb.

Si el diseño por defecto para un block **DO**, no es especificado en el encabezado del block entonces el diseño por defecto para el mismo es el diseño por defecto del block **FOR EACH**, **REPEAT** o procedimiento que lo contenga, o del block **DO** que lo contenga y que explícitamente nombre un diseño.

PROGRESS crea un diseño para un block **DO**, siempre que la sentencia **DO** de cabecera contenga una frase de diseño.
Si esta frase de diseño indica un nombre de diseño mediante la opción **FRAME**, entonces dicho diseño será el diseño por defecto del block **DO**.
Si esta frase de diseño no indica específicamente un nombre de diseño mediante la opción FRAME, **PROGRESS** igualmente generará un diseño para ese block, pero será 'sin nombre'.

Asignaciones de diseños

Un diseño es ASIGNADO al primer block en el cual el diseño es usado.

Por ejemplo:

/* La asignación del diseño aaa es al bloque de procedimiento porque este es el primer bloque en el cual el diseño aaa es usado */

```
DISPLAY "Clientes" WITH FRAME aaa.
REPEAT:
        PROMPT-FOR clientes.cliente WITH FRAME aaa.
        FIND clientes USING cliente.
        DISPLAY clientes WITH 2 COLUMNS.
```

END.

Los blocks **REPEAT**, **FOR EACH** y procedimientos automáticamente tienen diseños asignados (-scoped) a ellos. **PROGRESS** solamente asigna diseños a bloques **DO** si se indica un nuevo diseño con la frase Frame en la sentencia **DO**.

Servicios de Frames

PROGRESS utiliza los siguientes servicios para cada uno de los diseños en el bloque:

**Avance.*
**Borrado*.
**Limpieza*.
**Retención del dato previamente ingresado durante un RETRY de un bloque*.
**Repaint optimization*.

PROGRESS determina cuando proveer estos servicios en base al bloque al cual el diseño es asignado.

Asignación de registros

Cuando una sentencia **FIND** o **FOR EACH** lee un registro desde la base de datos, **PROGRESS** almacena este registro en el buffer de registro. Los datos en ese buffer de registro están disponibles para el procedimiento durante la asignación de ese registro.

Por defecto, la asignación de un registro es al block más pequeño **FOR EACH**, **REPEAT** o **procedimiento** que cubre todas las referencias del registro.

/* La asignación del registro de clientes es al block de procedimiento que es el block más externo en el cual el registro de cliente es referenciado */

```
REPEAT:
        INSERT clientes.
END.
DISPLAY clientes WITH 2 COLUMNS.
```

PROGRESS automáticamente asigna registros para los block **REPEAT**, **FOR EACH** y procedimientos. Podrá explicitar la asignación de registros para los block **REPEAT** y **DO** usando la opción **FOR** en el encabezado del block.

Con la asignación de registros, podrá utilizar nombre de campos, sin especificar el archivo, para referirse a campos que tienen el mismo nombre en otros archivos de la base de datos. Ud. deberá especificar el archivo, si campos de dos o mas archivos son referenciados juntos.

4. DISEÑANDO PANTALLAS

PROGRESS utiliza DISEÑOS para mostrar datos. Si no utiliza opciones adicionales acerca de como requiere que aparezcan los datos, **PROGRESS** utiliza características por defecto para diseñar la pantalla.

Cualquier diseño que sea mostrado en un procedimiento puede utilizar todas las líneas de la "pantalla" excepto las tres líneas inferiores de la misma.
Dos de esas líneas son utilizadas para los mensajes de error y los mensajes producidos por un procedimiento mediante la sentencia **MESSAGE**, mientras que la última línea es usada para mensajes de ayuda y de estado. Algunas terminales tienen 25 líneas (siendo 22 las líneas disponibles para diseños). Este procedimiento muestra cómo **PROGRESS** realiza el diseño:

```
FOR EACH clientes:
    DISPLAY nro-de-cliente domicilio localidad.
END.
```

Hay dos niveles de diseño de un frame:

- Características que afectan al diseño entero. Por ejemplo:
  - que salga recuadrado o no
  - que los títulos vayan sobre o al costado de los datos
- Características que afectan a campos individuales.

Cambiando características del diseño entero

Podrá utilizar la frase-de-diseño para describir las diferentes características de un diseño. Una frase-de-diseño siempre comienza con la palabra **WITH** seguida por una o más de las opciones listadas en la siguiente tabla. En la misma, el texto en la columna **DEFAULT** indica como **PROGRESS** muestra el diseño si no se especifica una característica particular.

WITH	PROPOSITO	DEFAULT
FRAME diseño	Nombra un diseño.	Usa el diseño para un bloque.
TITLE	Muestra el título como parte	No hay título.

		de la línea superior del recuadro del diseño mostrado.
SIDE-LABELS	Muestra los títulos de los campos a la izq. de los datos.	Muestra los títulos de los campos arriba de los datos.
NO-UNDERLINE	No subraya los títulos	Subraya los títulos.
n COLUMNS	Ubica los datos en n columnas con los títulos a la izq. de los mismos.	Ubica los campos en el diseño uno al lado del otro a traves de pantalla, separando en varias líneas como se necesite para acomodar todos los campos.
DOWN	Muestra múltiples instancias	Muestra múltiples instancias de los datos si es el diseño por defecto del bloque mas interno del procedimiento. En caso contrario muestra una simple instancia.
expresión DOWN	Muestra un número específico	Idem al anterior. de registros en un solo diseño.
RETAIN n	Retiene los datos de n iteraciones del block, pertenecientes a la pantalla ó a la pág. anterior.	No retiene los datos de la pantalla ó página anterior.
WIDTH n	Especifica el número de columnas que tendrá el ancho del diseño.	Usa el ancho en base a los campos que está mostrando y el ancho de la terminal que está usando.
ROW expresión	Posiciona el diseño en una línea específica de la pantalla.	Posiciona el diseño en la línea 1.
CENTERED	Centra el diseño en pantalla.	No centra el diseño.
NO-BOX	No muestra el recuadro alrededor del diseño.	Muestra el recuadro alrededor de un diseño.
NO-HIDE	Suprime el borrado automático.	Automáticamente borra el diseño de acuerdo al conjunto de reglas de borrado por defecto.
NO-ATTR-SPACE	No reserva espacios para los atributos de campo tal como subrayado.	Depende del tipo de terminal, a menos que se especifique ATTR-SPACE o NO-ATRR-SPACE en una o varias maneras.

PAGE-TOP	Muestra el diseño cada vez que en la salida comienza una nueva pantalla ó página.	No remuestra el diseño en cada página.
PAGE-BOTTOM	Muestra el frame al final de página cada vez que la página es completada.	No remuestra el diseño al final de cada página.
NO-VALIDATE	No valida por las condiciones especificadas en el Diccionario para los campos comprendidos en el diseño.	Valida todos los campos en el diseño de acuerdo a las especificaciones en el Diccionario.

Nombrando los diseños que busca utilizar

Si no nombra los diseños en un procedimiento, **PROGRESS** decide qué va a mostrar cada diseño. Por ejemplo:

Bloque **Procedimiento**
Bloque **For Each**

```
DISPLAY " Saldo del Cliente".
FOR EACH clientes:
        DISPLAY nombre saldo.
END.
```

PROGRESS automáticamente crea un diseño para cada bloque FOR EACH, bloque REPEAT, bloque de procedimiento. Es por eso que en el ejemplo anterior se crearon dos diseños; uno para el bloque de procedimiento y otro para el bloque FOR EACH. Si necesita que salgan impresos los dos recuadros en uno solo, basta con nombrar el frame con un nombre. Por ejemplo:

```
DISPLAY "Saldo del Cliente" WITH FRAME muestra.
FOR EACH clientes:
        DISPLAY nombre saldo WITH FRAME muestra.
END.
```

En este caso, el bloque de procedimiento y el bloque FOR EACH utilizan el mismo frame para mostrar información.

En algunos casos **PROGRESS** muestra múltiples registros en un solo diseño. En dichos casos el diseño es llamado un multi-frame, ó un frame DOWN. Cuando

PROGRESS muestra en un diseño un solo registro, este es denominado frame INDIVIDUAL.

REGLA: a menos que se indique lo contrario, un frame es INDIVIDUAL excepto cuando éste es el diseño por defecto para el bloque de iteración mas interno y es asignado a dicho bloque, en tal caso es un frame DOWN.

Podrá utilizar la opción **DOWN** para:

a) Convertir un DISEÑO SIMPLE (con nombre) en un frame **DOWN.**

DISPLAY clientes.nro-cli clientes.saldo WITH FRAME a **DOWN**.

b) Para definir la cantidad de registros a mostrarse en un diseño. Por ejemplo:

DISPLAY clientes.nro-cli clientes.saldo WITH 4 **DOWN**.

Describiendo otras características para diseños

Volvamos a ejecutar el siguiente procedimiento:

```
DISPLAY "Saldo del cliente".
FOR EACH clientes:
        DISPLAY nombre saldo.
END.
```

Realizaremos simples cambios al programa:

```
DISPLAY "Saldo del Cliente" WITH NO-BOX COLUMN 25.
FOR EACH clientes:
        DISPLAY nombre saldo WITH NO-BOX COLUMN 17.
END.
```

La frase-de-diseño en la primera sentencia DISPLAY (**WITH NO-BOX COLUMN 25**) le permite a **PROGRESS** mostrar el diseño conteniendo las palabras "Saldo del Cliente" comenzando en la columna 25 de la pantalla. La opción NO-BOX le permite a **PROGRESS** no utilizar el recuadro que normalmente se utiliza cuando se muestran diseños, la frase-de-diseño en el segundo DISPLAY es similar.

Si modificamos el primer DISPLAY del programa de la siguiente manera:

DISPLAY *"Saldo del Cliente"* **SKIP** (2) *WITH NO-BOX COLUMN 25*.

Luego de "Saldo del Cliente", saltará dos líneas antes de imprimir el encabezado de los nombres y saldos de clientes.

```
DISPLAY "Saldo del Cliente"
                WITH TITLE "Reporte de Saldos" CENTERED.
FOR EACH clientes:
        DISPLAY nombre-saldo
                WITH NO-BOX 10 DOWN CENTERED RETAIN 2.
END.
```

En este ejemplo, la primer sentencia DISPLAY usa la frase-de-diseño **WITH TITLE "Reporte de Saldos" CENTERED**.

*La opción **CENTERED** permite a **PROGRESS** centrar el frame en la pantalla.

*La opción **TITLE** nombra un título para ser utilizado con la sentencia DISPLAY. Siempre ubica el título en el centro de la línea de arriba en el recuadro que está alrededor del frame recién mostrado.

Cambiando características de campos y variables

Cuando se muestra un campo o variable, ya sea para mostrarlo o solicitar un ingreso, **PROGRESS** aplica características especiales para ese campo o variable.

Se puede utilizar la frase **FORMAT** para reemplazar estas características asumidas por Progress. La siguiente tabla muestra las opciones a utilizar la frase **FORMAT**:

FRASE FORMAT	**PROPOSITO**	**DEFAULT**
FORMAT formato	Definir el formato en el cual los valores serán mostrados ó ingresados.	Un campo utiliza el formato en el Diccionario, una variable usa el formato de la definición de la variable, y una expresión usa el formato por defecto para el tipo de datos de la expresión.
LABEL título	Especifica el título para un campo, variable o expresión	Utiliza el título del diccionario de o de la variable.

NO-LABEL	No utiliza el título.	Utiliza el título por defecto.
AT n	Comienza el diseño del campo en la columna n.	Comienza a mostrarlo en la siguiente columna disponible.
TO n	Posiciona el diseño del campo para que n sea la última posi- del mismo.	La columna en el cual el diseño finaliza es dependiente de la co- lumna en el cual el display comienza y del formato del diseño.
ATTR-SPACE	Reserva espacios para atributos de campos tal como subrayado.	Depende del tipo de terminal a amenos que especifique ATTR-SPACE o NO-ATTR-SPACE de una o varias maneras.
NO-ATTR-SPACE	No reserva espacios para atribu os tal como subrayado.	Idem.
HELP mens-ayuda	Define el mensaje de ayuda a ser mostrado cuando el usuario esta ingresando un dato en ese campo.	PROGRESS mestra la frase de ayuda definida en el diccionario, si existe.

VALIDATE	Valida el valor del campo con el criterio de validación especifi- cado y muestra el mensaje si La validación falla.	No realiza una validación salvo que la validación haya sido espe- cificada en el Diccionario.
DEBLANK	Elimina los espacios de adelante en el ingreso de un campo tipo caracter.	No saca los caracteres en blanco de adelante.
BLANK	Permite a PROGRESS mostrar blancos para el diseño del campo en lugar del valor real.	Muestra el valor del campo.
AUTO-RETURN	Cuando completa el campo con datos, PROGRESS automática- mente pasa al siguiente campo de entrada.	No pasa automáticamente a otro campo. Deberá utilizar TAB o RETURN.
AS tipo-de-dato	Crea un campo o una variable con el tipo de datos especificado.	No crea un nuevo campo o variable

LIKE campo	Crea un campo y una variable con la misma definición de un campo especificado.	No crea un nuevo campo y variable.

Descubriendo el formato para mostrar un campo o una variable

Si no se utilizan sentencias de formateo en un procedimiento, **PROGRESS** utiliza el formateo por defecto para mostrar campos y variables. Si busca cambiar el formateo con que **PROGRESS** muestra los campos, encontramos dos maneras:

1) Cambiar el formateo en el Diccionario; pero usted no buscará cambiar el formateo del display cada vez que desea mostrar el campo de manera diferente.

2) Usando la opción **FORMAT** con la sentencia **DISPLAY**.

FOR EACH clientes:
- *DISPLAY nombre* **FORMAT "X (10)"**.
- *UPDATE saldo* **FORMAT ">>9.99"**.

END.

De esta manera hay solamente 10 espacios para el nombre del cliente y solo 5 espacios para el saldo.

Cuando se cambia el formateo de un campo, cambia la manera de como el campo es mostrado, no como los datos están actualmente almacenados en la base de datos.

Cuando se muestran variables, **PROGRESS** utiliza las mismas reglas de formateo como en los campos.

```
DEFINE VARIABLE prog     AS CHARACTER FORMAT "X(3)".
DEFINE VARIABLE otro     AS LOGICAL FORMAT "Si/No".
```

Si no utiliza la opción FORMAT en la sentencia DEFINE VARIABLE; podrá utilizar ésta con cualquier sentencia de manejo de datos que muestren datos.

UPDATE otro **FORMAT "Si/No".**
SET prog **FORMAT "X(3)".**

Describiendo el título para un campo o variable

Si no incluye sentencias acerca de TITULOS en un procedimiento, **PROGRESS** utiliza los títulos por defecto para los campos y variables.
Si desea cambiar la etiqueta que **PROGRESS** utiliza cuando muestra un campo hay dos maneras:

1) Modificando la etiqueta en el Diccionario.

2) Usando la opción LABEL con las sentencias **DISPLAY, PROMPT-FOR, SET** y **UPDATE**.

DEFINE VARIABLE preg AS CHARACTER **LABEL** *"Pregunta".*
ó
DISPLAY preg **LABEL** *"Pregunta".*

Características NO-DISPLAY

Hay varias opciones que no hacen a las características de display. Estas opciones son:

HELP tira-de caracteres
VALIDATE
DEBLANK
BLANK
AUTO-RETURN
AS tipo-de-datos
LIKE campo

Describiendo todo un diseño de salida y propiedades de procesamiento

La sentencia FORM.

Hay veces que se busca describir características de un diseño pero no se quiere incluir esta descripción en la frase **FRAME** de la sentencia de manejo de datos, puesto que las

características del diseño son muy complejas. En esos casos podrá utilizar la sentencia **FORM** para describir la forma y propiedades de procesamiento de un cierto diseño. Por ejemplo:

*Describir encabezados de diseños.

*Darles un orden a los campos en los diseños; cuando busca procesar los mismos en otro orden.

*Crear un diseño que no necesariamente coincide con el orden en que se van a ejecutar las sentencias del procedimiento.

Tener en cuenta que las sentencias **FORM** describen un diseño. Pero no pone dicho diseño a la vista. Para eso, deberá utilizar la sentencia **VIEW**.

Describiendo encabezados de Frames

Podrá utilizar la sentencia FORM para describir un encabezado que aparece en la parte superior del diseño. Por ejemplo:

```
FORM HEADER "Reporte de Clientes" WITH CENTERED.
VIEW.
FOR EACH clientes WITH CENTERED:
        DISPLAY nombre saldo.
END.
```

FORM describe el encabezado del diseño. La sentencia **VIEW** muestra el diseño por pantalla.

Cambiando el mensaje del sistema PROGRESS

Mediante la sentencia STATUS podrá especificar el mensaje que aparece en la línea inferior de la pantalla. Hay tres maneras que puede utilizar esta sentencia:

1) STATUS DEFAULT mensaje o expresión.

Especifica el mensaje de estado a ser mostrado cuando se esta ejecutando un procedimiento pero no esta ingresando ningún dato.

2) STATUS INPUT OFF.

Indica a PROGRESS no mostrar cualquier mensaje mientras está ingresando datos en un campo de entrada.

3) STATUS INPUT expresión.

Especifica el mensaje de estado a ser mostrado cuando está ingresando datos en un campo de entrada. El mensaje de estado por defecto es:

"enter data a press F4 to end".

Para campos específicos, este mensaje es reemplazado por cualquier mensaje definido para el campo en el Diccionario o cualquier mensaje de ayuda que se haya especificado con la opción **HELP**.

Determinando el Diseño

Cuando está usando un diseño muy complejo o cuando busca utilizar el mismo diseño varias veces en un solo procedimiento, podrá utilizar la sentencia **FORM** para describir el diseño.

De esta manera, necesita describir el diseño una sola vez, y hacer mención del mismo en cada sentencia de manejo de datos. Los campos son mostrados en el orden descripto en la sentencia **FORM**, y modificados en el orden listado en la sentencia **UPDATE**. Esto significa, que el orden en el cual los campos son descriptos en la sentencia **FORM** afectan solamente la manera en que estos campos son posicionados en el frame y no el orden en el cual ellos son procesados.
Presione **TAB** para moverse a través de los campos.

Usando el área de mensajes

El área de mensajes consiste en tres líneas:

*Las dos primeras líneas son para mensajes específicos de procedimiento.
*La tercer línea (la última línea en pantalla) es para mensajes del sistema y mensajes de ayuda.

Mostrando mensajes específicos del procedimiento

Deberá utilizar la sentencia **MESSAGE** para mostrar mensajes en el área de mensajes. Por Ejemplo:

```
.
.
MESSAGE " Cliente" nro-de-cliente "ha sido eliminado".
.
.

.
.
MESSAGE "Desea eliminar otro cliente". SET preg.
.
.
```

El primer procedimiento muestra un mensaje indicando el nro. del cliente que haya sido dado de baja; luego un segundo mensaje si desea eliminar otro cliente.

5. VARIABLES, EXPRESIONES, FUNCIONES

Su aplicación requerirá procedimientos para trabajar con:

- Datos temporarios que no están almacenados en un campo de la base de datos.
- Datos que son producidos como resultado del uso de un campo o como resultado de la combinación de varios campos de la base de datos.

Usando datos temporarios - variables

Cuando se escribe un procedimiento, se necesitan lugares donde almacenar temporariamente datos. Un campo temporario donde almacenar datos se lo denomina **VARIABLE.**

Las variables en **PROGRESS** deben ser definidas antes de ser utilizadas. La definición de las mismas se realiza madiante la sentencia **DEFINE VARIABLE**.
La sentencia **DEFINE VARIABLE** tiene el siguiente formato:

DEFINE NEW SHARED VARIABLE *nombre-de-variable*
NEW GLOBAL SHARED

AS *tipo-de-variable*
FORMAT *formato*
LIKE *nombre-de-campo*
LABEL *etiqueta*
DECIMALS *nro-decimales*
INITIAL *valor-inicial*
EXTENT *nro-items*
NO-UNDO

Las frases opcionales **NEW SHARED** y **NEW GLOBAL SHARED** son para las ocaciones en que las variables son compartidas por más de un procedimiento, tema que se verá más adelante.

La palabra **AS**, permite indicar el tipo de variable (entera, decimal, alfanumérica, fecha, lógica).

La palabra **LIKE** permite definir la variable al igual que un campo ya definido en el diccionario.

Como se ve, el resto de las posibilidades permite dar a una variable todos los atributos que posee un campo definido en un diccionario de datos.

Definiendo el tipo de datos para una variable

Se puede definir una variable con los mismos tipos de datos y formatos que utilizan para definir un campo. Por ejemplo: en el siguiente caso la variable 'respu' es definido del tipo **LOGICAL**.

DEFINE VARIABLE *respu* **AS LOGICAL FORMAT "S/N".**

Si utiliza la opción **AS** (como) y no utiliza la opción **FORMAT**, las variables usan el formato por defecto para cada tipo de datos:

Tipo de datos	**Formatos de display por defecto**
Caracter	x(8)
Fecha	99/99/99
Decimal	->>>,>>9.99
Entero	->>>,>>9
Lógico	yes/no

Definiendo una variable 'LIKE' a un campo

Se puede definir una variable con las mismas características que un campo de la base de datos; es decir con el mismo tipo de datos, formato o etiqueta. En el siguiente ejemplo se define una variable 'nomb' con las mismas características que el campo 'nombre' definido en el archivo de clientes.

DEFINE VARIABLE *nomb* **LIKE** *clientes.nombre.*

Usando expresiones

Una expresión es una constante, nombre de campo, nombre de variable ó cualquier combinación de estos que resulten en un valor.

Usando constantes en expresiones

Algunas expresiones pueden utilizar constantes para procesar información. Las constantes en una expresión pueden ser datos numéricos, caracteres, Fechas, o valores lógicos (verdadero/falso).

En el siguiente ejemplo se utilizan **CONSTANTES NUMERICAS:**

```
máximo-cre = máximo-cre + 1000.
mínimo-cre = mínimo-cre - 150.          constantes numéricas.
```

Las **CONSTANTES** tipo **CARACTER**, las cuales están compuestas por datos no-numéricos o alfanuméricos, deben estar encerradas entre comillas, cuando se las utiliza en un procedimiento.

```
ciudad = "Avellaneda".
domicilio = "Mitre 2012"                Constantes tipo caracter.
```

Las **CONSTANTES** tipo **FECHA** están representados en el formato "mes/día/año".

```
...día 11/06/74                         Constante tipo Fecha.
```

Las **CONSTANTES LOGICAS** son representadas mediante un "yes" -- Verdadero, o un "no" -- falso.

Usando operadores en expresiones

Muchas aplicaciones requieren que se combinen valores constantes, campos o variables para obtener resultados aritméticos.

Los operadores son símbolos que se usan para realizar cálculos numéricos, cálculos de fechas, manipulaciones de tiras de caracteres, o comparación de datos. La siguiente tabla muestra los operadores que se pueden utilizar en **PROGRESS** y las operaciones que se realizan.

Operadores PROGRESS

- Unario Negativo	Revierte el signo de una expresión numérica.
+ Unario Positivo	Retorna el valor positivo o negativo de una expresión numérica.
/ (División)	Divide una expresión numérica por otra expresión aritmética, produciendo un resultado decimal.
* (Multiplicación)	Multiplica dos expresiones numéricas.

- Substracción	Substrae una expresión numérica de otra.
- Substracción de Fecha	a) Substrae un número de días de una fecha, produciendo como resultado una nueva fecha. b) Substrae una fecha de otra, produciendo un resultado entero que representa el número de días entre las dos fechas.
+ Suma de días	Suma un número de días a una fecha, produciendo una fecha como resultado.
+ Concatenación	Produce un valor caracter por unión o concatenación de dos tipos de caracteres o expresiones.
+ Adición	Suma dos expresiones numéricas.
< o LT	Retorna un valor verdadero, si la primera de dos expresiones es menor que o igual a la segunda expresión.
= o LE	Retorna un valor verdadero, si la primera de dos expresiones es menor que o igual a la segunda expresión.
> o GT	Retorna un valor verdadero si la primera de dos expresiones es mayor que la segunda expresión.
> = o GE	Retorna un valor verdadero si la primera de dos expresiones es mayor que o igual a la segunda expresión.
= o EQ	Retorna un valor verdadero si dos expresiones son iguales.
< > o NE	Compara dos expresiones y retorna un valor verdadero si ellas no son iguales.
NOT	Revierte un valor verdadero o falso de una expresión.
AND	Retorna un valor TRUE (Verdadero) si cada una de Dos expresiones lógicas es verdadero.
OR	Retorna un valor Verdadero si alguna de dos lógicas es verdadero.

Usando operadores para calculos numéricos y comparaciones de datos

En el siguiente ejemplo se muestra como se realiza una multiplicación, mediante el operador (*), para determinar el importe total de un producto al multiplicar su precio-unitario por la cantidad adquirida:

imp-total = precio-unit * cantidad

"*" Operador de Multiplicación

El resultado de la operación es almacenado en la variable imp-total. Note que a ambos lados del operador de multiplicación, se deberá dejar un espacio.
Si queremos comparar si un importe determinado es mayor a 1000 se deberá proceder de la siguiente manera:

...importe-x **GT** 1000 o
...importe-x > 1000

"**GT**" y ">" Operadores de Comparación.-

Si importe-x es mayor a 1000, el resultado es verdadero, en caso contrario el resultado es falso.

Realizando cálculos con fechas

Entre las operaciones que se pueden realizar con fechas encontramos:

a) Suma de Fechas: Suma un número de días a una fecha, produciendo como resultado una fecha.

fecha + *días* Por ejemplo: 10/15/85 + 10; a la fecha 15/10/1985 se le suman 10 días dando como resultado la fecha 25/10/1985.

fecha: Una expresión (una constante, nombre de campo, nombre de variable, o cualquier combinación de éstas) que su valor sea una fecha.

días: Una expresión (una constante, nombre de campo, nombre de variable, o cualquier combinación de éstas) que su valor sea el número de días que busca sumar a la fecha.

b) Substracción de fechas: Resta un número de días de la fecha, produciendo como resultado una fecha, o resta una fecha a otra, produciendo como resultado un número entero representando el número de días entre dos fechas.

* Fecha - días
* Fecha - fecha

Ejemplos: 06/11/74 - 10 = 06/01/74

06/11/74 - 06/01/74 = 10

Usando funciones en expresiones

Las funciones son "herramientas" que provee PROGRESS para facilitar la realización de ciertas tareas. Por ejemplo funciones para convertir datos de un tipo a otro, ó realizar cálculos complejos.

Hay nueve categorías de funciones **PROGRESS**:

* Funciones **AGGREGATE** que evaluan grupos de valores.

* Funciones **ARITMETICAS** para realizar operaciones aritméticas con valores numéricos.

* Funciones **CARACTER** para manipular tiras de caracteres ó expresiones.

* Funciones que proveen información para su aplicación sobre FECHA, DIA, MES Y AÑO.

* Funciones de **CONVERSION DE DATOS**, que permiten cambiar datos de un tipo a otro.

* Funciones de **DECISION**, que evaluan una de dos expresiones, dependiendo de un valor de una especificada condición.

* Funciones de **ESTADOS DE REGISTROS**, que determinan el estado del registro, tal como la disponibilidad para procesamiento.

* Funciones de **ESTADOS DE PANTALLA**, que retornan información acerca de la pantalla.

* Funciones de **ESTADO DEL SISTEMA**, que ofrecen información acerca del sistema operativo, terminal y teclado.

Function Type	PROGRESS Functions		
Aggregate value	ACCUM MAXIMUM MINIMUM		
Arithmetic	EXP LOG MODULO	RANDOM ROUND SQRT	TRUNCATE
Character	BEGINS CAPS ENTRY FILL	INDEX LC LENGH LOOKUP	MATCHES SUBSTRING
Date	DATE DAY MONTH	TODAY WEEKDAY YEAR	
Data Conversion	ASC CHR DECIMAL	INTEGER STRING	
Decision	IF-THEN-ELSE		
Record Status	AMBIGUOUS AVAILABLE CAN-FIND FIRST	FIRST OF LAST LAST-OF LOCKED	NEW RECEID
ScreeN Status	ENTERED FRAME-FIELD FRAME-FILE	FRAME-VALUE GO-PENDING INPUT	MESSAGE-LINES NOT ENTERED SCREEN-LINES
System status	CAN-DO DBNAME KBLABEL KEYCODE KEYFUNCTION KEYLABEL	LASTKEY LINE-COUNTER OPSYS PAGE-NUMBER PAGE-SIZE	RETRY SEARCH TERMINAL TIME USERID

Precedencia de funciones y operadores

El orden en el cual **PROGRESS** evalua una expresión depende de la prioridad de los operadores. Considere la siguiente expresión aritmética:

3 * 5 + 2

El resultado de esta expresión es diferente, dependiendo del orden en el cual procese las operadores de multiplicación y suma.

3 * (5 + 2) = 21, pero (3 * 5) + 2 = 17

La siguiente tabla muestra el orden de prioridad de las fuciones y operadores **PROGRESS**.

Nombre del operador	**Precedencia**
- Unario Negativo	7 (más alto)
+ Unario Positivo	7
Módulo	6
/ División	6
* Multiplicación	6
- Substracción de Fechas	5
- Substracción	5
+ Suma de Fechas	5
+ Suma	5
Comparaciones	4
LT o <	4
LE o = <	4
GT o >	4
GE o > =	4
EQ o =	4
NE	4
BEGINS	4
NOT	3
AND	2
OR	1 (más bajo)

Si la expresión contiene dos operadores de igual prioridad, **PROGRESS** evalua la expresión de izquierda a derecha. Si los operadores no son de igual prioridad, **PROGRESS** evalua el operador de alta prioridad primero. Para cambiar el orden de evaluación de una expresión deberá agrupar las expresiones y operadores con paréntesis.

Usando tablas

Las **TABLAS** contienen múltiples elementos.

Por ejemplo: una tabla puede ser un campo conteniendo los saldos mensuales durante un año. Este campo contendrá 12 elementos, uno para cada mes del año. Para definir una tabla solo hay que indicar su extensión mediante **EXTENT,** ya sea una variable o un campo. La extensión de la tabla es el número de elementos contenidos. Los campos y variables pueden ser tablas.

Supongamos la variable SALDO, que esta compuesta por 12 elementos. Cada elemento corresponderá al saldo de un mes. Por lo tanto, el SALDO [1] - saldo Enero, SALDO [2] - saldo Febrero, etc.

En los procedimientos, se podrá:

* Referirse a todos los elementos de la tabla en un momento. Simplemente nombrando la variable o el campo. *Pro ejemplo*:

DISPLAY saldo. ---- mostrará los 12 saldos.

* Referirse a un elemento específico de la tabla, subindicando con corchetes. *Por ejemplo:*

```
DISPLAY saldo [1] label "Saldo Enero"
        saldo [2] label "Saldo Febrero"
        saldo [3] label "Saldo Marzo".
```

* Referirse a un rango de elementos de la tabla, subindicando el número de elemento de comienzo y a cuántos elementos se quiere hacer referencia. *Por ejemplo*:

```
DISPLAY saldo [1 for 3] label "saldos 1er trimestre".
```

3- Indica 3 elementos.
1- Indica a partir del elemento 1.

6. PROCEDIMIENTOS, ARCHIVOS DE INCLUSION

Compilando y precompilando procedimientos

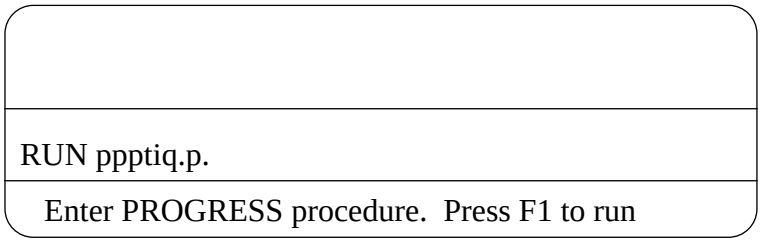
Los procedimientos pueden ser ejecutados directamente desde el editor PROGRESS ó usando la sentencia RUN.

Compilando procedimientos

El procedimiento que usted crea con el editor es denominado PROCEDIMIENTO FUENTE. Antes de que **PROGRESS** pueda utilizar este procedimiento fuente, éste debe crear una VERSION COMPILADA u OBJETO DEL PROCEDIMIENTO. Este procedimiento de compilación traslada el procedimiento fuente a un formato interno que puede ser eficientemente procesado por **PROGRESS**.

Compilando procedimientos con la sentencia RUN.

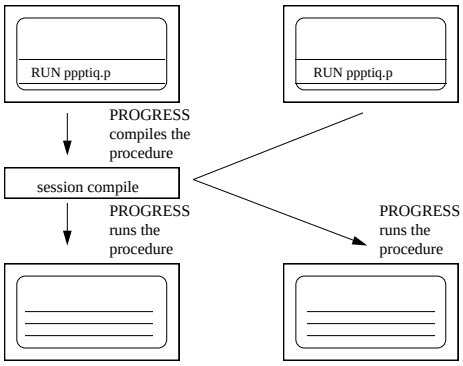
Supongase que busca ejecutar el procedimiento *"ppptiq.p"*. Podrá realizar esto presionando GO (F1) después que el procedimiento fuente se encuentre en el editor, ó usando la sentencia RUN.



Después que **PROGRESS** procesa la sentencia **RUN**, este crea una versión compilada del procedimiento *"ppptiq.p"* para entonces ejecutar la misma. A la versión del procedimiento objeto obtenida de esta manera la llamamos SESION COMPILADA porque esta es utilizada durante una sesión **PROGRESS**. Una sesión **PROGRESS** comienza cuando usted comienza **PROGRESS** mediante el comando pro y finaliza con la sentencia QUIT.

Si ejecuta nuevamente el procedimiento *"ppptiq.p"* sin haberle realizado modificaciones y sin salir de **PROGRESS**, éste utilizará la versión compilada anteriormente en la corriente sesión, ahorrando el tiempo de recompilación.

En cambio, si sale de **PROGRESS** después de ejecutar el procedimiento *"ppptiq.p"*, cuando convoque nuevamente a **PROGRESS**, para ejecutar nuevamente el procedimiento *"ppptiq.p"*, **PROGRESS** en esta segunda vez deberá crear otra vez una versión compilada del procedimiento para la sesión nueva.

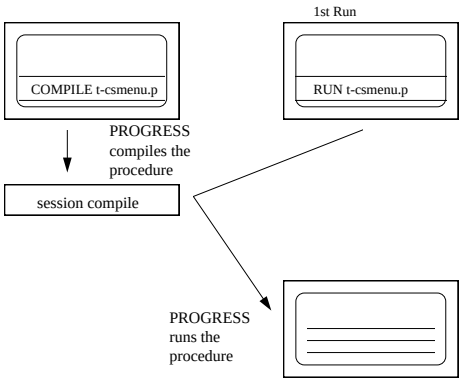


Compilando procedimientos explícitamente

La sentencia COMPILE permite realizar una compilación del procedimiento.

```
COMPILE t-csmenu.p.
```

Como con la sentencia RUN, esta sentencia también crea, desde una versión fuente del procedimiento "t-csmenu.p", una versión compilada del mismo. Una vez que el procedimiento es compilado, la sentencia RUN simplemente ejecuta la versión compilada de la sesión, sin tener que compilar el programa fuente.



Precompilando procedimientos para rápida ejecución

Cuando se utiliza la opción **SAVE** con la sentencia COMPILE, **PROGRESS** produce una VERSION PRECOMPILADA u OBJETO DEL PROCEDIMIENTO, generando un archivo objeto cuyo nombre del procedimiento más un ".r" al final del nombre.

Si el nombre del procedimiento tiene una extensión ".p", tal como "menu.p", la sentencia **COMPILE** reemplaza la extensión ".p" del aarchivo con una extensión ".r".

COMPILE *menu.p* **SAVE** ----> *menu.r*

Cada vez que se ha precompilado un procedimiento y creado un archivo ".r", esta versión precompilada del procedimiento podrá ser ejecutada usando la sentencia **RUN**.

Si realiza cambios a su procesimiento fuente, no mostrará dichos cambios a menos que:

* Compile y almacene nuevamente el mismo.

(COMPILE nombre-procedimiento SAVE.).

o

* Elimine el archivo ".r". En este caso, el procedimiento fuente podrá ser utilizado en una compilación de una sesión.

o

* Ejecute el procedimiento desde el editor.

La precompilación le permitirá:

* Ejecutar los procedimientos más rápidamente.
* Conservar recursos del sistema.

Subprocedimientos

Algunos procedimientos pueden ser seleccionados en un conjunto de procedimientos más pequeños, los cuales a su vez pueden ser ejecutados desde un procedimiento principal. Por ejemplo ese proceso principal puede mostrar un menú, el cual brinde varias opciones. De acuerdo a la selección del operador, ese procedimiento principal

puede llamar a otro procedimiento para realizar una tarea más específica. Cada procedimiento llamado desde otro, se denomina **SUBPROCEDIMIENTO**.

Algunas veces, diferentes procedimientos en una aplicación necesitan utilizar el mismo conjunto de sentencias para realizar un determinado trabajo. Por ejemplo, si varios procedimientos necesitan utilizar un mismo diseño de pantalla. Si definimos dicha pantalla mediante la sentencia FORM, necesitaremos repetir dicha sentencia en cada uno de los procedimientos. Eso trae aparejado que si se necesita cambiar el diseño de pantalla, deberá corregir todos los procedimientos que utilizan dicho diseño. La mejor solución para este caso, es colocar la sentencia FORM del diseño de pantalla en un archivo separado, de manera tal de poder incluirlo en los procedimientos necesarios. Esos archivos que se incluyen en procedimientos se denominan "ARCHIVOS DE INCLUSION".

Los SUBPROCEDIMIENTOS y los ARCHIVOS DE INCLUSION le permiten a usted:

* Minimizar el tiempo de desarrollo de una aplicación: No tendrá que retipear una secuencia de sentencias cada vez que necesite estas sentencias en un procedimiento.

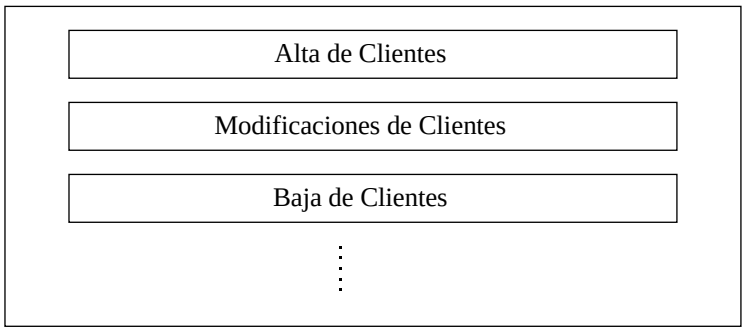
* Lograr un funcionamiento más consistente: Las acciones realizadas por un conjunto de sentencias funcionarán de una manera predecible cuando son llamados por ó incluídos en otros procesos, puesto que ya estarán funcionando en uno.

* Mantenimiento fácil de la aplicación: Si busca cambiar o corregir un error en un subprocedimiento o en un archivo 'include', podrá cambiar el mismo, sin necesidad de modificar todos los procedimientos.

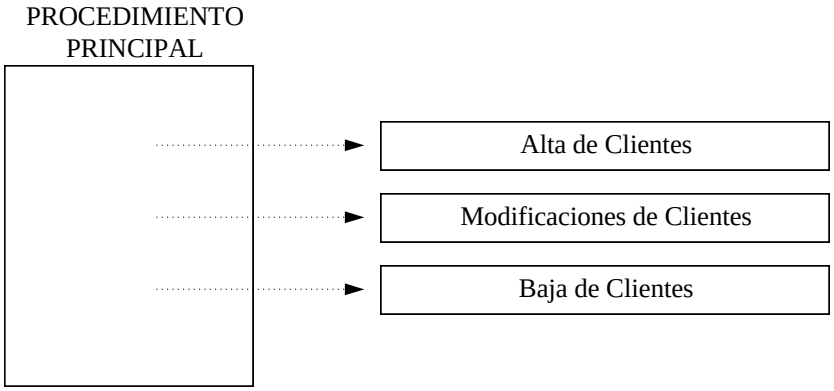
Llamando procedimientos desde un procedimiento principal: Subprocedimientos

Supóngase tener en su aplicación un único procedimiento que realiza diferentes tareas relacionadas a los clientes.

MENU DE CLIENTES



Podrá también escribir por separado, pequeños procedimientos que correspondan a cada una de estas tareas.

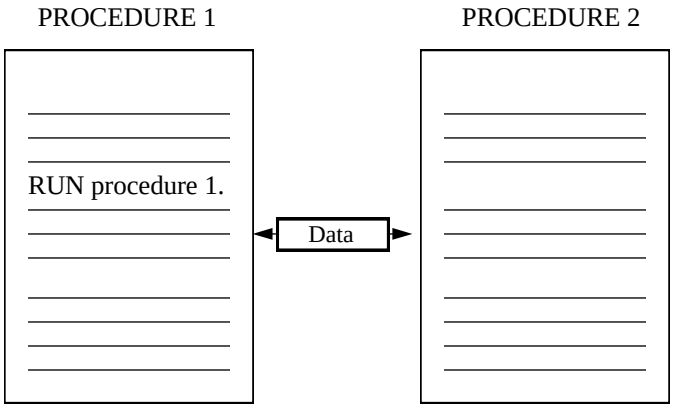


Estos procedimientos individuales, o subprocedimientos, pueden ser llamados o ejecutados desde el procedimiento ppal. Para ejecutar un procedimiento, utiliza la sentencia RUN seguido por el nombre del subprocedimiento a ejecutar.

RUN *bajacli.p.*

Pasando datos a subprocedimientos

Cuando se ejecuta uno ó más procedimientos desde el procedimiento ppal, se puede necesitar compatir información entre el programa ppal y aquellos procedimientos.



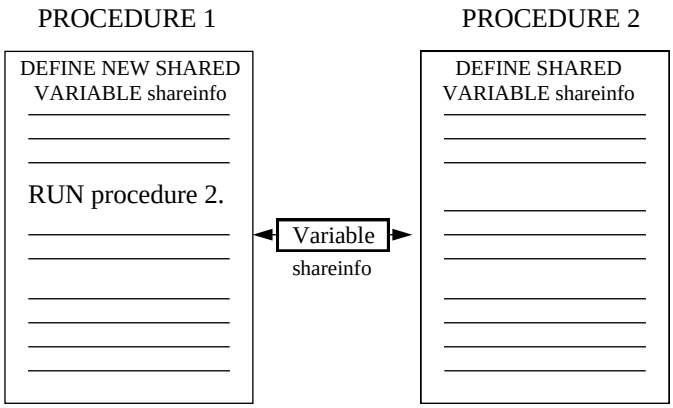
Hay dos maneras de compartir información entre procedimientos:

* Usando una variable compartida.

* Pasando los datos como argumentos.

Usando variables compartidas para pasar datos

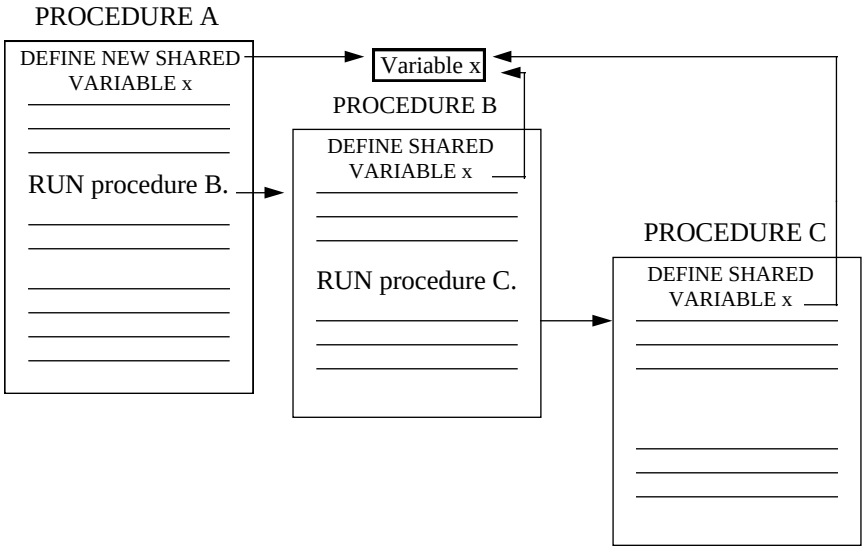
Si busca utilizar variables para pasar información de un procedimiento a otro, las variables deberán ser definidas como compartidas (shared).



El procedimiento-1 ejecuta el procedimiento-2, dando al mismo acceso a compartir la variable 'shareinfo'. Para que esto trabaje, el procedimiento-1 deberá crear la variable usando la sentencia **DEFINE NEW SHARED VARIABLE**, y el procedimiento-2 deberá hacer referencia a la variable, pero usando la sentencia **DEFINE SHARED VARIABLE**.

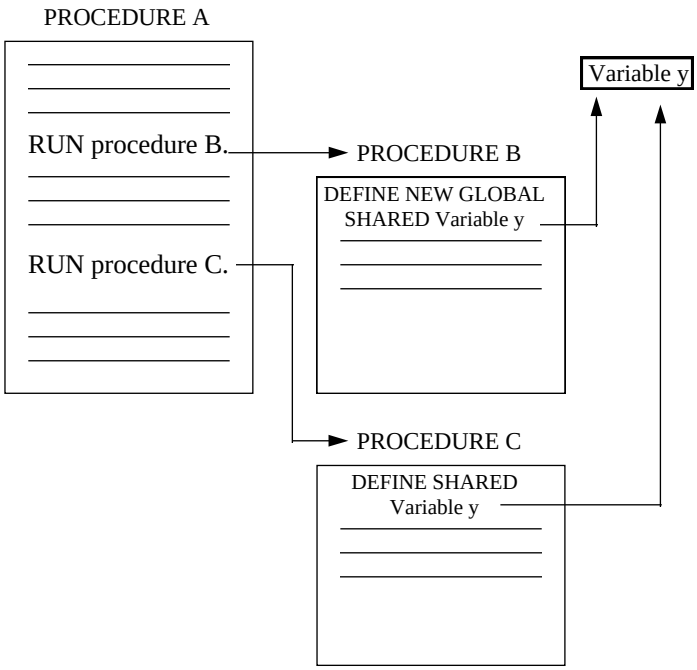
Usando variables compartidas globales para pasar datos

Cuando un procedimiento define una variable como **NEW SHARED**, **PROGRESS** crea esa variable cuando dicho procedimiento comienza a ejecutarse. Otro procedimiento ejecutado directamente o indirectamente desde el procedimiento ppal, tiene acceso a la variable si esta ha sido definida como **SHARED**.



El procedimiento C puede acceder a la variable x esté o no definida en el procedimiento B.

Algunas veces se necesita disponer de compartir información entre procedimientos que no son ejecutados desde un procedimiento en común. O tal vez, se quiere almacenar información en una variable compartida, de manera tal que retenga este valor aún después de que el procedimiento que la definió haya finalizado. **PROGRESS** provee un especial tipo de variable que satisface a ambos de esos requerimientos; una variable definida como **NEW GLOBAL SHARED**.

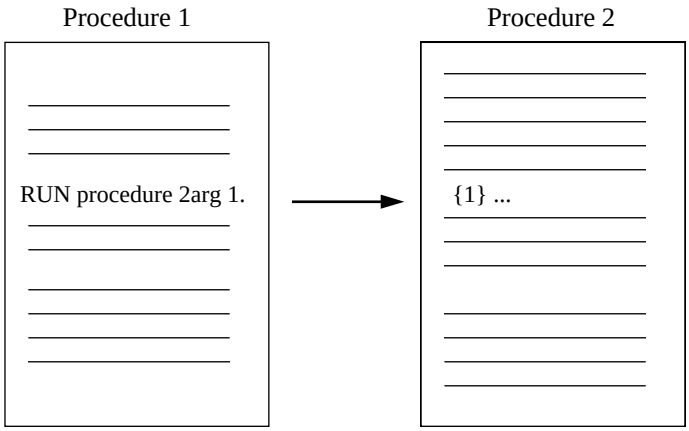


Un ejemplo de la sentencia es la siguiente:

DEFINE NEW GLOBAL SHARED VARIABLE *nombre* **AS CHARACTER FORMAT "X(20)"**.

Usando argumentos para pasar datos

Además de compartir variables, para poder pasar información de un procedimiento a otro, se pueden utilizar **ARGUMENTOS**.
Un **ARGUMENTO** es un conjunto de datos que un procedimiento llamador pasa a un procedimiento llamado. **PROGRESS** utiliza esos datos cuando compila las sentencias en el programa llamado.



En el ejemplo de arriba, el procedimiento-1 ejecuta el procedimiento-2 pasando Arg 1 al mismo.
El procedimiento-2 utiliza el argumento pero nombrando el mismo entre llaves. El símbolo {1} se refiere al primer argumento, {2} al segundo, y así con todos.

En el siguiente ejemplo, el procedimiento "lista.p" llama al procedimiento "lista1.p" pasando las constantes tipo caracter, "clientes" y "nombre" como argumentos.

```
        /* lista.p */

RUN lista1.p "clientes" "nombre".
```

El procedimiento "lista1.p" es llamado por el procedimiento "lista.p":

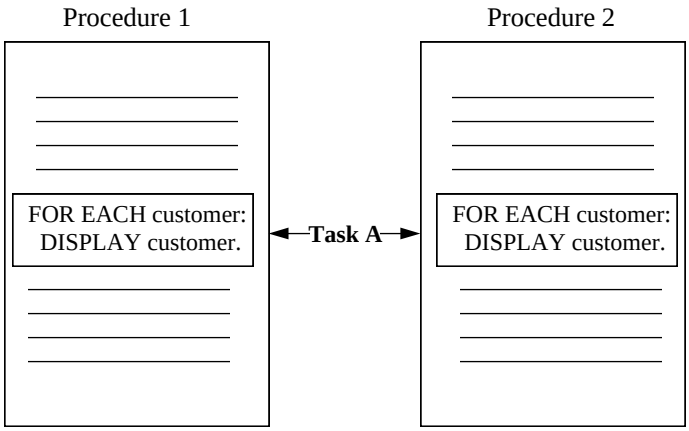
```
        /* lista1.p */

FOR EACH {1}:
        DISPLAY {2}.
END.
```

El procedimiento lista1.p substituye el string "clientes" como primer argumento. Este argumento es referenciado por el 1 encerrado entre llaves {1}. El string "nombre" es substituído como segundo argumento. Una vez que esas substituciones son realizadas, **PROGRESS** interpreta el procedimiento lista1.p como sigue.

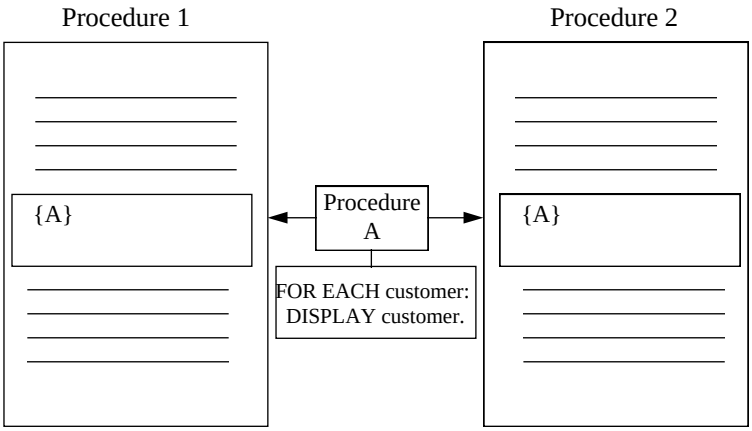
FOR EACH clientes:
DISPLAY nombre.
END.

Incluyendo sentencias en múltiples procedimientos: Archivos Include

Supongase que hay dos procedimientos en su aplicación. El procedimiento 1 y el procedimiento 2, que utilizan la misma secuencia para realizar una cierta tarea.



Para utilizar una o más sentencias en uno ó más procedimientos, deberá colocar estas sentencias en un archivo de inclusión. Cuando Ud. necesite estas sentencias en un procedimiento que está escribiendo, simplemente suministre el nombre del archivo que busca incluir entre llaves. Las llaves le permiten a **PROGRESS**, en el momento de compilación, leer el archivo de inclusión nombrado, colocando las sentencias del mismo en ese procedimiento.



Pasando argumentos a un archivo de inclusión

Como a los subprocedimientos, se pueden pasar argumentos a archivos de inclusión. El procedimiento "muestra.p" muestra un mensaje y luego incluye el archivo "muestra1.i", pasando las palabras clientes y nombre como argumentos al procedimiento.

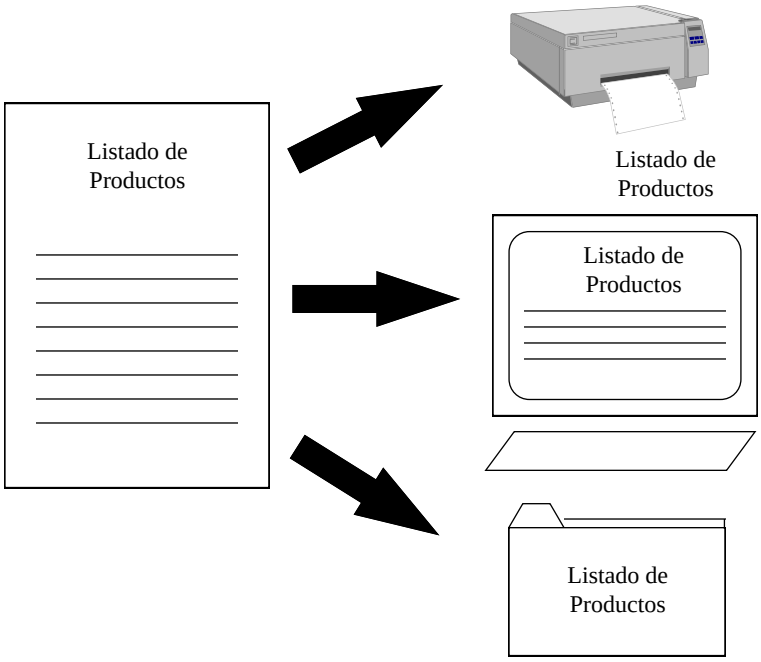
```
        /* muestra.p */

DISPLAY "LISTADO DE CLIENTES" WITH FRAME f 1.
{muestra1.i clientes, nombre}
```

```
        /* muestra1.i */

FOR EACH {1}.
        DISPLAY {2}.
END.
```

En el archivo muestra1.i; clientes sustituye a {1} y nombre substituye a {2}, y muestra1.i es colocado así dentro del procedimiento muestra.p cuando el mismo es ejecutado.

7. REPORTES

Uno de los principales beneficios de un sistema administrador de base de datos es proveer información que ayude a los usuarios a organizarse más eficientemente. Un reporte, puede ser impreso en papel, mostrado en la pantalla ó almacenado en otro medio.



Creando un reporte simple

Podrá ser un reporte en su terminal utilizando la sentencia DISPLAY. En el siguiente ejemplo se muestra el nombre, domicilio, encotel, teléfono y saldo de todos los clientes que tengan saldos mayores a 200.

```
FOR EACH clientes WHERE saldo > 200:
  DISPLAY nombre domicilio encotel teléfono saldo.
END.
```

En este procedimiento no hay sentencias de diseño, PROGRESS utiliza formatos de display por defecto. Este formato por defecto incluye, por ejemplo, etiquetas para cada campo y el recuadro alrededor del display.

Generando reportes categorizados y ordenados

Un reporte que muestra totales para subgrupos en un archivo es llamado reporte con corte de control. En el siguiente ejemplo se realiza un reporte categorizado ó con corte de control por encotel, totalizando el saldo.

FOR EACH clientes WHERE saldo > 200 **BREAK BY** encotel:

.
.
.

Aquí **PROGRESS** agrupa los registros de los clientes, usando la opción **BREAK BY** por encotel. Luego se necesita obtener un subtotal de los saldos para cada encotel. Se necesita modificar la sentencia DISPLAY.

DISPLAY nombre domicilio teléfono saldo (**TOTAL BY** encotel) WITH NO-BOX.

El cálculo del grupo es realizado mediante la opción **TOTAL BY** encotel. **TOTAL** es una opción de frase-agregado. Una fase-agregado identifica uno ó más valores a ser calculados en base a un cambio realizado en el corte de grupo. Esto es, cada vez que el corte de grupo (encotel) cambia; la opción **TOTAL** calcula el total del campo saldo para todos los clientes dentro del último grupo. Este además provee un total general de todos los subtotales calculados.

El siguiente procedimiento ordena los clientes en orden del nombre. Pero es posible ordenar los registros en varios niveles. Esto se logra por ejemplo de la siguiente manera:

FOR EACH clientes WHERE saldo >200 **BREAK BY** encotel **BY** nombre **DESCENDING:**
DISPLAY nombre domicilio encotel teléfono saldo (TOTAL BY encotel).
END.

En el procedimiento anterior además de obtener subtotales por encotel, lista los registros en el orden del campo nombre en forma descendiente.

Usando datos de múltiples archivos en un reporte

En muchos reportes, podrá necesitar información de varios archivos en su base de datos. Por ejemplo, en el siguiente procedimiento combina el archivo de clientes con un archivo de órdenes:

```
FOR EACH clientes:
   DISPLAY cliente nombre domicilio encotel WITH FRAME a.
   FOR EACH ordenes OF clientes
         DISPLAY ordenes WITH FRAME b.
   END.
END.
```

Mostrando el resultado de un cálculo

Muchas veces en un reporte es necesario obtener el resultado de un cálculo determinado. Por ejemplo, si necesitamos multiplicar el saldo de cada cliente por un factor determinado, basta con realizar:

DISPLAY nombre saldo saldo * 0.15 **LABEL** "Nuevo Saldo".

En la sentencia DISPLAY, cuando se realiza un cálculo en el que intervienen uno ó más campos, **PROGRESS** reserva espacio en el diseño, para almacenar el resultado del cálculo con las características que normalmente aplica a cualquier campo ó variable recién mostrada. Estas características incluyen una etiqueta y un formato de display.

Enviando reportes a diferentes dispositivos

PROGRESS permite enviar reportes a otros destinos de salidas que no sea ni la terminal, ni la impresora; tal como un archivo.

Se debe utilizar la sentencia **OUTPUT TO** para enviar datos a un destino que no sea su pantalla. Dispositivos de salida pueden incluir una impresora, archivo ASCII, ó dispositivo DOS o UNIX.

Por ejemplo:

OUTPUT TO PRINTER. (destino: la impresora).
OUTPUT TO *archi.* (destino: archivo ASCII archi).

Diferencias entre la sentencia Display y Put.

Mediante la utilización de la sentencia **DISPLAY**, al compilarse un procedimiento, se realiza un diseño en el cual se reservan lugares para todas las variables y/o campos a ser mostrados mediante esta sentencia, no teniendose en cuenta si van a ser ó no utilizadas.

En cambio, mediante la sentencia **PUT** sólo se mostrarán las variables y/o campos utilizados con esta sentencia en el momento en que se haga efectiva su ejecución.

La limitación de la sentencia **PUT** es la de no definir por defecto un diseño determinado; siendo la misma más orientada para la utilización de diseños en archivos y en impresora.

Enviando secuencias de control a la impresora

Cuando se envía una salida a la impresora, podrá modificar la manera en que la misma genera esta salida. Muchas impresoras tienen el conjunto de "secuencias de escape" que se podrán utilizar para especificar diferentes características de impresión. Por ejemplo: cambiar el número de caracteres a imprimir por pulgada.
Cuando se escribe un procedimiento; que envía la salida a una impresora, se deberán incluir secuencias de control en el mismo.
Muchas secuencias de control incluyen caracteres especiales que son representados en el procedimiento por su equivalente octal. Para distinguir estos códigos en octal, deberá preceder los 3 dígitos octales por un caracter de escape. **PROGRESS** entonces convierte el nro. octal en un simple caracter. Si está utilizando DOS, el caracter de escape es un tilde (~). Si está utilizando UNIX, el caracter de escape es un tilde ó una barra(/).
Por ejemplo: si en un procedimiento se necesita enviar una secuencia para imprimir en modo comprimido, se realiza de la siguiente manera:

DEFINE VARIABLE comprime AS CHARACTER INICIAL **"~ 033[3W"**.
.
. .
PUT CONTROL comprime.

Utilizando encabezados y pies de página en un reporte

Se asume que cuando se imprime un determinado reporte, busca imprimir un encabezado al comienzo de cada página del mismo. Este encabezado, por ejemplo; podrá contener la fecha y el número de página por defecto.

OUTPUT TO archi.

FOR EACH clientes:
FORM HEADER "Listado de Clientes"

```
  AT 25 "Página"
  AT 70 PAGE-NUMBER FORMAT ">9"
  WITH PAGE-TOP FRAME a.
VIEW FRAME a.
DISPLAY clientes.cli-nro clientes.nombre.
END.
```

Hay varias partes del procedimiento que causan que encabezados aparezcan varias veces en el archivo.

* La opción **PAGE-SIZE** en la secuencia OUTPUT TO determina que podrá haber múltiples páginas en el archivo de salida.

* La sentencia **FORM** define un diseño al tope de la página (**PAGE-TOP**). La opción **PAGE-TOP** permite que **PROGRESS** muestre el diseño al tope de cada página.

* Todos los diseños tienen un encabezado y un cuerpo. El encabezado es la parte superior del mismo. La opción **HEADER** en la sentencia **FORM** permite a **PROGRESS** reevaluar las expresiones en la sentencia al comienzo de cada página.

* La única manera de activar un diseño **PAGE-TOP** es mostrando éste. La secuencia **VIEW** permite que el diseño **PAGE-TOP** sea mostrado al comienzo de cada página.

Utilizando encabezados y pies de página

Supóngase que desea imprimir un reporte que incluya encabezados y pies de página en el reporte de clientes.
Al principio de la página necesitan imprimir:

"11/06/74 Listado de Clientes Página 1"

PROGRESS reevalúa el encabezado del diseño al comienzo de cada nueva página.

```
OUTPUT TO archi, PAGE-SIZE 10.
FOR EACH clientes:
  FORM HEADER TODAY
  "Listado de Clientes" AT 30
  "Página" AT 70 PAGE-NUMBER
  FORMAT "Z9" SKIP (1)
  WITH FRAME hdr PAGE-TOP NO-BOX NO-ATTR-SPACE.
```

La sentencia OUTPUT TO archi designa un archivo ASCII como destino de salida (puede cambiar este destino a PRINTER si así lo desea).

La opción PAGE-SIZE indica a PROGRESS que cada página del reporte pueda contener hasta 10 líneas.

La sentencia FORM describe las características del diseño a ser impreso al comienzo de cada página, usando las siguientes opciones:

* **HEADER:** Permite que **PROGRESS** ubique en la sección los items nombrados en la sentencia **FORM**.

* La función **TODAY** retorna la fecha corriente del sistema.

* La constante caracter "Listado de Clientes" comienza en la columna 30. **(AT 30)**.

* La constante caracter "Página" comienza en la columna 70 (AT 70), seguida por el número de página corriente retornada por la función **PAGE-NUMBER**.

* La opción **FORMAT** provee un formato de dos dígitos para el nro de página ("z9").

* La opción **FRAME** hdr nombra el diseño hdr.

* La opción **PAGE-TOP** permite a **PROGRESS** mostrar el diseño al comienzo de cada nueva página.

* La opción **NO-BOX** permite a **PROGRESS** no mostrar un recuadro alrededor del diseño.

* La opción **NO-ATTR-SPACE** indica que no se necesitan espacios en el diseño para atributos de video.

Supongase que busca imprimir al final de cada página:

"Continúa en la siguiente página".

Deberá utilizar la sentencia **FORM** con la opción HEADER para definir este pie de página.

```
FORM HEADER "Continúa en la siguiente página" WITH FRAME
        ftr PAGE-BOTTOM NO-BOX NO-ATTR-SPACE.
```

* La opción **HEADER** permite a **PROGRESS** ubicar en la sección de encabezado (no confundir con el tope de página) los items nombrados en la sentencia FORM.

* La opción PAGE-BOTTOM permite a PROGRESS mostrar el diseño al pie de cada página.

Si luego en el procedimiento utilizamos la opción VIEW FRAME ftr activará el frame ftr. Esto significa que el diseño ftr estará disponible para imprimirse al final de cada página.

Trabajando con Entrada/Salida en los Procedimientos

Introducción

Cuando un procedimiento de **PROGRESS** solicita el ingreso de un dato por pantalla, está utilizando un **CANAL DE ENTRADA**. En forma similar, cuando un procedimiento muestra datos por pantalla, está utilizando un **CANAL DE SALIDA**.

Cada procedimiento automáticamente tiene un canal de entrada y uno de salida. Por defecto, **PROGRESS** asigna ambos canales a la terminal.

Cambiando el destino de salida de un procedimiento

Mediante la sentencia **OUTPUT TO** se puede cambiar el destino del canal de salida por defecto (terminal), por otros como por ejemplo, la impresora o un archivo:

Por ejemplo:

OUTPUT TO PRINTER.

A partir de este comando se redirecciona el canal de salida a la impresora. Lo que una sentencia DISPLAY mostraba por pantalla, ahora lo muestra por impresora.

OUTPUT TO *nombre-archivo* **PAGED**.

A partir de este comando se redirecciona el canal de salida a un archivo en disco. La opción PAGED indica que la salida debe tener cortes por página (se asumen 56 líneas por página). Si la salida es la impresora, esta opción es asumida. Si no se coloca PAGED para la salida en un archivo, los datos en el mismo se verán en forma contínua.

Utilizando múltiples destinos de salida

Supongamos que un procedimiento, en una parte necesita enviar datos a la terminal, pero en otra requiere imprimir o guardarlos en un archivo. Para ello, se puede utilizar la siguiente sentencia:

OUTPUT TO VALUE (nombre-de-variable)

mediante la cual se puede redireccionar el **CANAL DE SALIDA** de acuerdo al contenido de la variable cuyo nombre se coloca entre paréntesis.

Por ejemplo:

```
DEFINE VARIABLE archisal AS CHARACTER FORMAT "X(8)"
                LABEL "Nombre Archivo de Salida:".
REPEAT:
SET archisal WITH SIDE-LABELS.
OUTPUT TO VALUE (archisal).
FOR EACH clientes:
  DISPLAY nro-de-cliente nombre.
END.
END.
```

Cambiando el origen del ingreso a un procedimiento

Mediante la sentencia INPUT FROM se puede cambiar el origen del CANAL DE ENTRADA por defecto (terminal), por otros como, por ejemplo, un archivo.

Supongamos que queremos incorporar datos al archivo de clientes, pero en lugar de ingresarlos por pantalla, necesitamos leerlos de un archivo secuencial que ya tenemos. El procedimiento sería:

INPUT FROM nombre-archivo.
REPEAT:
  CREATE clientes.
  SET nro-de-cliente nombre domicilio.
END.

Utilizando múltiples orígenes de ingreso

Supongamos que en un procedimiento, en una parte se necesita ingresar datos desde la terminal y en otra parte se requiere tomar datos desde un archivo.

En dichos casos, como se dijo con:

INPUT FROM nombre-de-archivo.

se redirecciona el CANAL DE ENTRADA por defecto a un archivo. Para volver a tener el canal de entrada en la terminal se debe utilizar:

INPUT FROM TERMINAL.

Preparando archivos de entrada.

Como se utiliza un archivo de datos como entrada, PROGRESS necesita que responda a las siguientes características:

- Uno ó más espacios deben separar cada valor de campo.
- Los campos alfanuméricos (tipo caracter) que contengan blancos incluídos deben estar entre comillas.
- Cualquier comilla en los datos debe ser representada comillas dobles ("").

La pregunta es: *cómo conseguimos archivos con este formato ?*

PROGRESS contiene un programa utilitario llamado QUOTER que realiza lo siguiente:

- Coloca comillas al principio y al fin de cada registro del archivo.
- Reemplaza cualquier comilla existente (") por dos comillas ("").

Supongamos que tenemos un archivo externo llamado ARTI (Por ejemplo de un sistema COBOL) cuyo formato es:

columnas 1-6 nro-de-artículo
columnas 7-30 artículo
columnas 31-35 existencias

Para leer este archivo y colocar esos datos dentro de un archivo ARTIC de la Base de Datos habría que realizar el siguiente procedimiento:

```
DEFINE VARIABLE línea AS CHARACTER FORMAT "X(35)".
UNIX quoter ARTI ARTI.Q.
INPUT FROM ARTI.Q NO-ECHO.
REPEAT:
  CREATE ARTIC.
  SET línea.
  nro-artic = INTEGER (SUBSTR(línea,1,6)).
  desc-artic = SUBSTR (línea,7,24).
  existencia = INTEGER (SUBSTR(línea,31,5)).
END.
INPUT CLOSE.
```

En este procedimiento **quoter** lee cada línea del archivo ARTI, le agrega las comillas tal cual se describió anteriormente, y graba el resultado en el archivo ARTI.Q. Luego cada vez que la sentencia SET es ejecutada, en la variable "línea" se colocan todos los caracteres de un registro del archivo ARTI.Q. Como toda la línea está entre comillas, **PROGRESS** trata toda la línea como un solo campo. Las tres sentencias de asignación utilizan las funciones SUBSTR e INTEGER para extraer porciones de ese campo largo y colocarlos dentro de la Base de Datos.

Opciones del programa quoter

Hay dos opciones que se pueden utilizar con **QUOTER**:

-d Se puede utilizar esta opción para identificar el caracter de separación entre campos. Por ejemplo:

```
quoter -d,
```

En este caso se indica que los campos en el registro del archivo de entrada estarán separados entre si por el caracter coma (,).

-c Se puede utilizar esta opción para nombrar una lista de los números de columnas identificando aquellas que debieran ser consideradas como campos. Por ejemplo:

quoter -c 1-4,5,7-9

En este ejemplo, quoter interpreta las columnas 1-4 como un campo, la columna 5 como un campo, y las columnas 7-9 como otro. Si el registro de entrada fuera:

abcdefghi----------> "abcd" "e" "ghi"

Definiendo canales de entrada y salida adicionales

Hasta ahora hemos visto procedimientos que tienen solamente un canal de entrada y uno de salida.

Algunas veces es necesario tomar datos desde más de un canal de entrada simultáneamente, y al mismo tiempo enviar datos a más de un canal de salida en forma simultánea.

En estos casos se deben crear canales de entrada y salida adicionales para un procedimiento, mediante la sentencia **DEFINE STREAM**.

Por ejemplo, supongamos que se necesite un procedimiento que pueda tomar datos tanto desde una terminal como desde un archivo de datos, como así también enviar datos tanto a la terminal como a otro archivo de datos.

```
DEFINE VARIABLE x AS CHARACTER.
DEFINE VARIABLE y AS CHARACTER.
DEFINE STREAM in.
DEFINE STREAM out.
        .
        .
        .
        .
INPUT STREAM in FROM infile.dat.
OUTPUT STREAM out TO outfile.dat.
DISPLAY "Esto va a la pantalla" WITH FRAME a.
DISPLAY STREAM out "Esto va a un archivo" WITH FRAME b.
```

SET x LABEL "Esto se solicita por pantalla" WITH FRAME c.
SET STREAM in y WITH FRAME d.

Este procedimiento usa 4 canales:

1- Canal de Entrada por defecto (terminal).
2- Canal de Entrada adicional llamado in.
3- Canal de Salida por defecto (terminal).
4- Canal de Salida adicional llamado out.

La primera sentencia DISPLAY no nombra un canal en particular, por lo tanto utiliza el canal de salida por defecto (terminal).

La segunda sentencia DISPLAY explícitamente nombra un canal de salida out y envía los datos a ese canal (un archivo de datos).

La primera sentencia SET no nombra un canal en particular, por lo tanto utiliza el canal de entrada por defecto (terminal).

La segunda sentencia SET explícitamente nombra un canal de entrada in y por lo tanto los datos desde ese canal (un archivo de datos).

Canales compartidos

Dos o más procedimientos pueden compartir los mismos canales de entrada y salida. La compartición de canales es muy similar a la compartición de variables.

- Se utiliza **DEFINE STREAM** para definir un canal que sólo será accesible a ese procedimiento.

- Se define un canal como **NEW SHARED** en el procedimiento que crea el canal a compartir y como **SHARED** en todos los procedimientos que usen ese canal.

- Se define un canal como **NEW GLOBAL** cuando se quiere que el canal permanezca accesible aún después de que el procedimiento conteniendo la sentencia **DEFINE NEW GLOBAL SHARED STREAM** finalice.

Utilización de canales

Cuando un canal por defecto es cerrado (automáticamente o explícitamente), el mismo es adjudicado automáticamente al respectivo canal de entrada o de salida del procedimiento llamador. Si no hay procedimiento llamador, el canal correspondiente es asignado a la terminal.

Cuando se cierra un canal específico, no se lo puede volver a utilizar hasta que el mismo sea reabierto.

Cuando se cierra un canal de entrada asociado a un archivo, y el mismo se reabra para el mismo archivo, el ingreso recomenzará desde el principio del archivo.

Procesos como canales de entrada y salida

En sistemas UNIX, se pueden utilizar las sentencias I**NPUT THROUGH** para entubar datos a **PROGRESS** desde otro proceso. Similarmente la sentencia **OUTPUT THROUGH** entuba datos desde **PROGRESS** y regresar datos desde **PROGRESS** a otros procesos UNIX.

También se puede utilizar la sentencia **INPUT-OUTPUT THROUGH** para entubar la salida de un proceso UNIX hacia un procedimiento **PROGRESS** y regresar datos desde **PROGRESS** hacia ese mismo proceso.

Sentencia IF...THEN...ELSE...

Una sentencia de fundamental importancia para poder desarrollar con eficacia la lógica de los procedimientos es la sentencia **IF**, la cual realiza la ejecución de una sentencia o conjunto de sentencias en forma condicional. Su formato es:

IF expresión THEN block-1 ELSE block-2

donde expresión es una constante, nombre de campo, nombre de variable, o cualquier combinación de estos, cuyo valor sea "verdadero" o "falso".

Si el valor de la expresión es "verdadero" se ejecutará block-1; si en cambio es "falso" se ejecutará block-2.

block-1 y block-2 pueden ser una sola sentencia, o un block DO, o un block REPEAT o un block FOR EACH.

La palabra ELSE describe el procesamiento a realizarse si la expresión tiene el valor FALSO.

8. APLICACIONES MULTIUSUARIAS

Escribiendo aplicaciones multiusuarias

Si se está escribiendo una aplicación que debe ser utilizada por varios usuarios en forma simultánea, hay algunos usos y características adicionales que deberá tener en cuenta en el ciclo de desarrollo de la aplicación. Los conceptos explicados en este capítulo son orientados a aplicaciones que se ejecutan en configuraciones multiusuarias (UNIX). Si está desarrollando una aplicación en el sistema operativo DOS, podrá utilizar las sentencias para configuraciones en multiusuario. PROGRESS simplemente ignorará estas sentencias cuando ejecute esta aplicación en modo monousuario.

Utilizando aplicaciones en una configuración multiusuaria. Una situación común en multiusuario.

Supongase que hay dos departamentos en una empresa, uno de compras y otro de ventas que accedan al mismo registro (artículo) en el mismo momento, sumándole stock uno y restándole stock el otro. Por ejemplo: supongamos que tenemos 67 artículos código 121, y

Ventas	Compras
1) Vende 20 del artículo 121	3) Compra 10 del artículo 121.
2) Lee el registro del archivo	4) Lee el registro del archivo
5) Resta 20 del campo stock.	6) Suma 10 al campo stock.

El stock del artículo finaliza siendo 77 cuando debería haber sido 57 (67-20+10=57) que es lo que esperábamos. Naturalmente este tipo de situación ocurre en todo momento en una configuración multiusuaria y por esta razon **PROGRESS** suministra servicios para asegurar que los datos en su base de datos realicen lo que uno desea.

Compartiendo registros de la Base de Datos

La situación descripta anteriormente ocurre cuando varios usuarios acceden al mismo registro al mismo tiempo. La solución a este problema de compartir registros es estar seguro de:

* Que múltiples usuarios puedan leer, ó mirar, el mismo registro al mismo tiempo.

* Justo un usuario en un momento pueda modificar, ó realizar cambios a un registro.

PROGRESS utiliza *'lockeo'* de registros para responder a estas reglas.

Usando locks para solucionar conflictos de registros

En el caso anterior podrá utilizar locks para administrar usos concurrentes del mismo registro de la base de datos.

VENTAS	**COMPRAS**
1) Vende 20 del artículo 121.	3) Compra 10 del artículo 121.
2) Lee el registro del archivo, colocando un EXCLUSIVE-LOCK en el registro. COMPRAS no puede leer	4) Lee el registro del archivo. Como ventas tiene un EXCLUSIVE-LOCK. COMPRAS no puede leer éste y debe esperar hasta que se desocupe.
5) Resta 20 del campo stock.	7) Una vez que VENTAS lo libere, lee el registro.
6) Graba el registro en la base de datos y libera el registro.	8) Suma 10 al campo stock.

En el paso 2) el departamento de VENTAS lee el registro del archivo, colocando un EXCLUSIVE-LOCK en el registro. Esto significa que otro usuario no podrá utilizar el registro hasta que VENTAS no finalice con éste. En este registro, el valor del campo stock es de 67.

En el paso 4), el departamento de COMPRAS, intenta leer el registro del archivo, pero no puede por que el departamento de VENTAS tiene un EXCLUSIVE-LOCK en el registro.

En el paso 5), el departamento de VENTAS le resta 20 al campo stock, y retorna el registro a la base de datos. Una vez que el registro ha retornado a la base de datos, el EXCLUSIVE-LOCK es liberado y el departamento de COMPRAS puede sin problemas leer el registro de la base de datos. El valor del campo stock en este momento es de 47.

En el paso 6), el departamento de COMPRAS le suma 10 al campo stock y retorna el registro a la base de datos. Por lo tanto, el valor final del campo stock es de 57, exactamente lo que se esperaba.

Como PROGRESS aplica locks

Si ud. no aplica locks en sus procedimientos, **PROGRESS** aplica lockeos por defecto. En particular:

* Cuando un registro es leído, **PROGRESS** coloca un **SHARED-LOCK** en el registro. Esto significa que otros usuarios pueden leer el registro, pero NO pueden modificarlo hasta que el **SHARED-LOCK** es liberado. Si intenta leer un registro con **SHARED-LOCK**, cuando el mismo ya tiene un **EXCLUSIVE-LOCK** recibirá un mensaje de que el registro está en uso y que debe esperar a que sea liberado.

* Cuando un registro es modificado, PROGRESS coloca un EXCLUSIVE-LOCK en el registro. Esto significa que otro usuario no puede leer ni modificar el registro hasta que el EXCLUSIVE-LOCK sea liberado. Si intenta leer un registro con EXCLUSIVE-LOCK cuando otro usuario tiene un registro con SHARED- LOCKó EXCLUSIVE-LOCK, recibirá un mensaje de que el registro está en uso y debe esperar.

Ejemplo:

```
REPEAT:
  PROMPT-FOR clientes.cliente.
  FIND clientes USING cliente.
  UPDATE nombre saldo.
```

Cuando la sentencia FIND lee el registro de clientes, **PROGRESS** coloca un **SHARED-LOCK** en el registro. Cuando la sentencia **UPDATE** le permite modificar el registro, **PROGRESS** cambia el **SHARED-LOCK** a un **EXCLUSIVE-LOCK**.

Hasta cuando PROGRESS retiene un Lock ?

Los lockeos de registros y transacciones están directamente relacionados. Los siguientes son bloques de transacciones.

* Cualquier bloque que utiliza la palabra reservada TRANSACTION en la sentencia del bloque (DO, FOR EACH, o REPEAT).

* Un bloque de procedimiento y cualquier iteración de un bloque DO ON ERROR, FOR EACH o REPEAT que directamente modifica la base de datos o directamente lee registros con **EXCLUSIVE-LOCK**.

Para manejar la relación entre el lockeo de registros y transacciones mire la versión modificada del último procedimiento:

```
DEFINE VAR respuesta AS LOGICAL.
REPEAT:
  PROMPT-FOR clientes.cliente.
  FIND clientes USING cliente.
  UPDATE nombre saldo.
  SET respuesta LABEL “Modifica”.
  IF NOT respuesta THEN UNDO, NEXT.
END.
```

Imagine que pasa lo siguiente:

1) Al comienzo de la primera iteración del bloque REPEAT, **PROGRESS** comienza una transacción.

2) Usted suministra el número del cliente 1 en la sentencia PROMPT-FOR.

3) La sentencia FIND lee el registro 1 de la base de datos del archivo de clientes y **PROGRESS** ubica un SHARED-LOCK en el registro.

4) La sentencia UPDATE le permite realizar cambios al registro y PROGRESS ubica un **EXCLUSIVE-LOCK** en el registro.

5) Al final de la sentencia UPDATE, el EXCLUSIVE-LOCK es liberado.

6) Otro usuario ejecuta el mismo procedimiento buscando el cliente nro 1 y modifica información para dicho cliente (Este usuario lee el registro con los nuevos datos que se ingresó en el paso 4).

7) Si responde que no "modifica"; los cambios realizados son desechos.

8) El otro usuario responde que si en la pregunta "modifica" y el registro es grabado en la base de datos, incluyendo cambios que usted no busca grabar.

Como éste es un problema y hay otros semejantes a él, **PROGRESS** utiliza algunos estandards para determinar hasta dónde permanece el lockeo. La siguiente tabla muestra los estandards **PROGRESS** que utilizan para determinar cuándo se libera el lockeo de registros.

Tipo de lockeo	Adquirido durante una transacc. (1)	Adquirido fuera de de una transacc. (2)	Adquirido fuera pero retenido en una transacc. (3)
SHARED	Retenido hasta después de la finalización de la transacc. y la liberación del registro.	Retenido hasta que el registro es liberado.	(4)
EXCLUSIVE	Retenido hasta el fin de la transacción. Entonces se convierte a SHARE (5), si la asignación del registro es más larga que la transacc. y el registro permanece activo en cualquier buffer.	N/A	N/A

(1) Un lock es adquirido durante una transacción si el registro es leído o releído después de comenzar y antes de la finalización de una transacción.

(2) Un lock es adquirido fuera de una transacción si el registro es leído cuando la transacción no es activa y no ha sido liberado cuando la transacción comienza.

3) Un lock es adquirido fuera de una transacción y retenido en la misma, si el registro es leído cuando la transacción no es activa y no ha sido liberado cuando la transacción comienza.

4) La liberación del registro del buffer ocurre al final de la asignación, cuando la sentencia **RELEASE** es ejecutada, ó cuando un registro es reemplazado en el buffer por una sentencia CREATE, FIND, o FOR EACH.

5) Esto es así aún si el registro fue leído con un NO-LOCK anterior a la transacción y es releído con EXCLUSIVE-LOCK durante una transacción.

Cómo estas reglas de la tabla afectan el procedimiento dado ?

1) Al comienzo de la primera iteración del bloque REPEAT, **PROGRESS** comienza una transacción.

2) Usted suministra el número de cliente 1 en la sentencia PROMPT-FOR.

3) La sentencia FIND lee el registro de la base de datos y **PROGRESS** coloca un **SHARED-LOCK** en el registro. De acuerdo a estas reglas, este **SHARED-LOCK** podrá ser retenido hasta el final de la transacción o cuando el registro es liberado, posteriormente.

4) La sentencia UPDATE permite realizar cambios en la base de datos y PROGRESS modifica el SHARED-LOCK a un EXCLUSIVE-LOCK. De acuerdo a estas reglas, este proceso es retenido hasta el final de la transacción, la cual es el final de la iteración del bloque REPEAT.

6) Otro usuario ejecuta el mismo procedimiento, e intenta buscar el cliente nro 1. Sin embargo, por que el registro esta con EXCLUSIVE-LOCK, la sentencia FIND espera hasta que el registro este liberado.

7) Usted responde un no en ‘modifica’, **PROGRESS** deshace los cambios que ha realizado en el registro, busca el final de la primera iteración del bloque REPEAT y la transacción finaliza, liberando el EXCLUSIVE-LOCK en el registro.

8) El otro usuario esta disponible a buscar el registro y modificarlo.

Resolviendo conflictos de lockeo

Podrá utilizar diferentes opciones de lockeo para describir la manera en que busca el lockeo de registros:

* *EXCLUSIVE-LOCK*
* *SHARED-LOCK*
* *NO-LOCK*

NO-LOCK implica acceder al registro sin colocar ningún tipo de lock en el mismo.

Leyendo un registro con **NO-LOCK** significa que puede leer el registro independientemente de cualquier locks que los usuarios puedan tener en el registro, y otros usuarios no serán prevenidos en posteriores accesos con **SHARED-LOCK** o **EXCLUSIVE-LOCK**, sobre un registro leído como **NO-LOCK** puede no estar en un estado consistente ya que la transacción de otro usuario en proceso puede estar modificando este registro y registros relacionados.

Sin embargo, esto es aceptable en muchos casos, y puede reducir usualmente la retención del registro ya que los registros leídos con **NO-LOCK** por un usuario podrán ser modificados por otro usuario.

9. TRANSACCIONES Y MANEJOS DE ERROR

Imaginemos esta situación: se esta ingresando un nuevo registro de clientes en la base de datos. Ya se han ingresado 98 registros y se está ingresando el número 99, y el equipo se detiene. Los primeros 98 registros que se acaban de ingresar, se pierden ? NO, **PROGRESS** realiza lo siguiente:

1) Los primeros 98 registros estan en la base de datos.
2) El registro parcial 99 es descartado.

Esos dos pasos aseguran la integridad de la base de datos. Valida que los datos que ha ingresado sean grabados y los datos parciales sean descartados.

Esto es sólo un ejemplo. Supongamos que un procedimiento ha estado actualizando múltiples archivos. Lo que se quiere es que haya seguridad en que todos los cambios se hayan hecho en todos los archivos y todos los cambios no completados sean desechados.

La falla del sistema en un tipo de error. Hay otros tipos de errores que pueden ocurrir mientras se está ejecutando un procedimiento, y lo más importante es conservar la integridad de los datos. Para automatizar este procesamiento **PROGRESS** utiliza **TRANSACCIONES**.

Transacciones definidas

Una **TRANSACCION** es un conjunto de cambios a la base de datos, la cual deberá realizarse en forma completa o deber dejar la base de datos sin modificaciones.

En el siguiente ejemplo, se están agregando registros de clientes. Cada regiostro que se agrega es una **TRANSACCION**.

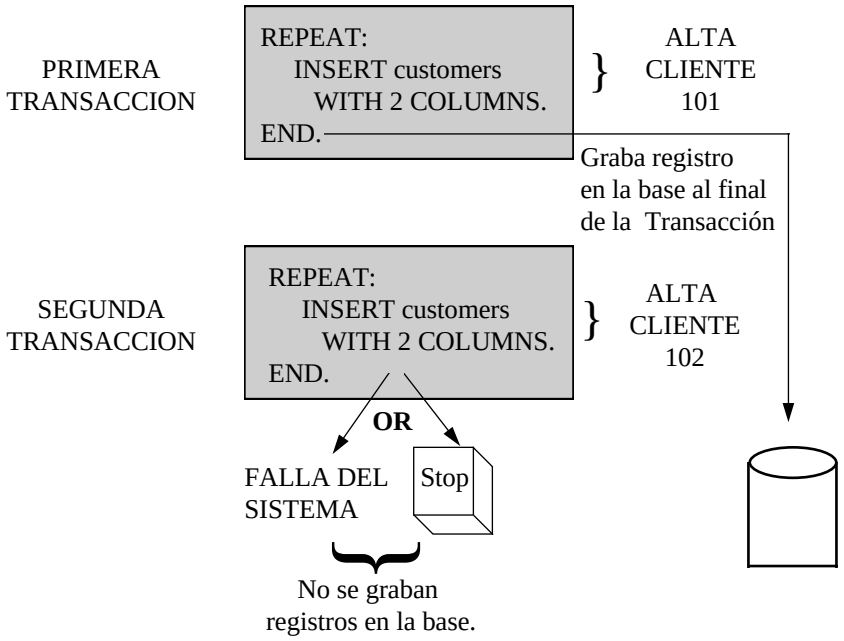
```
REPEAT:
  INSERT clientes WITH 2 COLUMNS.
END.
```

Cada iteración de este block REPEAT es una **TRANSACCION**.

La transacción se deshace o se vuelve atrás así:

- El sistema se detiene.

- El usuario presiona STOP. (Ctrl-Break en DOS, usualmente Ctrl-C en UNIX)



Procesando todo o nada

Supongamos que a un cliente se han facturado 10 artículos, los cuales han sido automáticamente descontados del stock. Pero el cliente en realidad había solicitado 6 artículos. Lo que se debe realizar es:

1) Localizar el item de la factura correspondiente, y cambiar la cantidad facturada.
2) Cambiar las existencias en el archivo de stock para el artículo indicado.

Qué pasa, si uno ha cambiado la cantidad en el item de la factura, y el equipo se para ? Lo que se pretende es que los dos archivos regresen a su estado original, es decir que los cambios se realicen en los dos archivos o en ninguno.

El procedimiento que haría este trabajo, sería el siguiente:

```
DEFINE VARIABLE cant-ant LIKE cantidad.
  REPEAT WITH 2 COLUMNS:
```

```
        PROMPT-FOR itemfac.factnum itemfac.numlin.
        FIND itemfac USING factnum itemfac.numlin.
        FIND articulos OF itemfac.
        DISPLAY cantidad itemfact.nro-art des-art pre-art
        pre-art * cantidad Label "Importe".
        cant-ant = cantidad.
        DISPLAY pre-art * cantidad label "N.Importe".
        PAUSE.
        stk-art = stk-art + cant-ant - cantidad.
        PAUSE.
END.
```

PROGRESS comienza una transacción al inicio de cada iteración del block REPEAT, y termina dicha transacción al final del block REPEAT.

Este procedimiento primeramente solicita NUMERO DE FACTURA y NUMERO DE ITEM de la factura a cambiar. A continuación mediante dichos valores accede al archivo de ITEMS DE FACTURAS, y mediante él al archivo de ARTICULOS. Muestra la cantidad facturada, el código, la descripción y el precio del artículo facturado, con un importe que se obtiene de multiplicar cantidad por precio. Como paso siguiente se guarda en la variable "cant-ant" la cantidad ya facturada y se solicita el ingreso de la nueva cantidad, mostrando el nuevo importe a facturar. Así llegamos a la primer pausa, en la cual ya hemos actualizado la cantidad facturada. Por último se actualiza el stock del archivo de ARTICULOS, llegando a la segunda pausa.

Si se ejecuta enteramente este procedimiento, ver como se actualizan ambos archivos. Pero qué sucede si durante el procedimiento en la primera pausa (donde ya se había actualizado el item de la factura pero no el stock del artículo) se oprime STOP ?. Bueno, si se realiza ésta prueba, se corroborará que el cambio que se realizó en la iteración que TERMINO CON NORMALIDAD está debidamente hecho en AMBOS ARCHIVOS. En cambio, la modificación realizada en la iteración INTERRUMPIDA, no está en NINGUN ARCHIVO.

Recuerde que en cada iteración del block REPEAT comienza una **TRANSACCION**. Una vez que la iteración ha sido completada correctamente, **PROGRESS** termina la transacción y actualiza los datos cambiados en la base de datos. En la próxima iteración **PROGRESS** comienza otra **TRANSACCION**.

Cuando se presiona STOP durante la iteración, **PROGRESS** deshace todos los cambios realizados desde el comienzo de la misma.

Ahora, cómo sabe **PROGRESS** dónde comenzar la transacción y cuánto deshacer o volver hacia atrás ?

Comienzo y fin de las transacciones

Para un determinado procedimiento, la TRANSACCION es el o los Blocks REPEAT, FOR EACH o de procedimiento más externos que actualicen directamente la base de datos. En otras palabras, si todavía no hay una transacción activa, **PROGRESS** comienza una transacción al comienzo de cada iteración de:

- Blocks FOR EACH que directamente actualizan la base de datos.

- Blocks REPEAT que directamente actualizan la base de datos.

- Blocks de PROCEDIMIENTO que directamente actualizan la base de datos.

Los Blocks DO no tienen la propiedad de transacción en forma automática.

Actualizar directamente la base de datos significa que el block contenga por lo menos una sentencia que cambie la base de datos, como por ejemplo CREATE, DELETE, UPDATE, etc.

Una vez que la transacción comenzó, todos los cambios a la base de datos serán partes de esa transacción, hasta que la misma termine. Cada usuario de la base de datos, puede tener solamente una transacción activa por vez.

En el siguiente ejemplo:

```
REPEAT:
  INSERT clientes WITH 2 COLUMNS.
END.
```

existen dos blocks. Uno, el de procedimientos, y el otro el Block REPEAT. En el primero no existen sentencias que directamente actualicen la base de datos, en cambio en el segundo se encuentra la sentencia INSERT que sí lo hace. Por lo tanto PROGRESS comenzará una transacción al comienzo de cada iteración del block REPEAT, y si ocurre algún error antes de la sentencia END, volverá hacia atrás el trabajo realizado en esa transacción.

Especificando cuánto deshacer

Como hemos visto ante la caída del sistema o cuando se presiona STOP, **PROGRESS** deshace la transacción vigente. Sin embargo, supongamos que se quiera deshacer toda la transacción pero en determinadas circunstancias que se puedan controlar por programación, o que tal vez se quiera una porción más chica del trabajo y no todo desde el principio de la transacción.
Veamos este ejemplo, en el cual se controla que una factura no pase $ 500.-

```
DEFINE VARIABLE total-fact LIKE articulos.pre-art.

block-1:
REPEAT:
        INSERT facturas.
        total-fact = 0.
        block-2:
        REPEAT:
                CREATE itemfac.
                itemfac.itemfac.numfact = facturas.numfact.
                DISPLAY itemfac.numfact.
                UPDATE numlin itemfact.nro-art cantidad pre-art.
                total-fact = total-fact + (cantidad * pre-art).
                IF total-fact > 500 THEN DO:
                        MESSAGE "La orden ha excedido $ 500".
                        MESSAGE "No se permite el ingreso de mas líneas".
                        UNDO block-2.
                END.
        END.
END.
```

En este ejemplo, el block más externo, sigue siendo el de la transacción, sin embargo cuando la factura excede el total de $ 500.-, lo que deshace y vuelve hacia atrás es el block-2, es decir el block más interno.

Lo que sucede en este caso, es que el block REPEAT más interno comienza una **SUBTRANSACCION**.

Una **SUBTRANSACCION** comienza cuando:

- Una TRANSACCION está aún activa.

y

- **PROGRESS** encuentra un block FOR EACH, REPEAT, o DO ON ERROR.
  (El DO ON ERROR es explicado más adelante).

La sentencia UNDO deshace el "block-2". La sentencia UNDO puede deshacer solamente aquellos blocks que son TRANSACCIONES o SUBTRANSACCIONES. En el ejemplo anterior "block-2" es una SUBTRANSACCION.

Usando la sentencia UNDO

La siguiente es la sintaxis de la sentencia **UNDO**:

UNDO [etiqueta] ,LEAVE [etiqueta]
,NEXT [etiqueta]
,RETRY [etiqueta]
,RETURN [etiqueta]

La sentencia UNDO puede utilizarse con las opciones *LEAVE, NEXT, RETRY* o *RETURN*.

Controlando donde comienzan y finalizan transacciones

PROGRESS comienza una transacción para cada iteración de tres tipos de bloques:

* Bloques FOR EACH que directamente modifican la base de datos
* Bloques REPEAT que directamente modifican la base de datos
* Bloques de procedimientos que directamente modifican la base de datos

Una transacción finaliza al final del bloque de la transacción o cuando una transacción se deshace por alguna razón.

Algunas veces se busca que una transacción sea más grande o más pequeña dependiendo de la cantidad de trabajo que se busca deshacer en un evento de error.

Puede explicitarse a **PROGRESS** a comenzar una transacción usando la opción TRANSACTION con el encabezado del bloque DO, FOR EACH, o REPEAT.

*** DO TRANSACTION**
*** FOR EACH TRANSACTION**
*** REPEAT TRANSACTION**

Para darse al bloque DO la propiedad de **TRANSACTION,** utilizando la frase **ON ERROR** (**DO ON ERROR**), si contiene sentencias que modifican la base de datos.

Realizando transacciones largas

En el siguiente procedimiento, el bloque REPEAT más externo es un bloque de transacción. Significa que cuando una transacción se deshace, PROGRESS deshace cualquier trabajo en la corriente iteración del bloque REPEAT externo.
Solamente la corriente factura se deshace, mientras todas las otras facturas son almacenadas en la base de datos.

```
REPEAT:
INSERT facturas WITH 2 COLUMNS.
FIND clientes OF facturas.
REPEAT:
  CREATE itemfac.
  itemfac.factunum = facturas.factunum.
  DISPLAY FACTURAS.FACTUNUM.
  UPDATE numlin nro-art pre-art cantidad descuento.
END.
END.
```

Supongase que busca, en el momento que el sistema cae, deshacer todas las facturas ingresadas desde el comienzo del procedimiento. Esto es, si busca realizar un bloque de transacciones no solo para una iteración del bloque, sino para todas las transacciones realizadas. Para realizar esto deberá utilizar el bloque DO con la opción TRANSACTION.

```
DO TRANSACTION:
REPEAT:
INSERT facturas WITH 2 COLUMNS.
FIND clientes OF facturas,
REPEAT:
  CREATE itemfac.
  itemfac.factunum = facturas.factunum.
  DISPLAY facturas.factunum.
  UPDATE numlin nro-art pre-art cantidad descuento.
END.
END.
```

Realizando transacciones pequeñas

En el siguiente ejemplo, se realizan bloques de transacciones más pequeñas, de manera tal que si el sistema falla, o se presiona la tecla STOP, se deshagan los items ingresados. Para ello, se deberá utilizar un bloque **DO** con la opción **TRANSACTION** para permitir a **PROGRESS** comenzar con las transacciones.

REPEAT:

DO TRANSACTION:
INSERT facturas WITH 2 COLUMNS.
FIND clientes OF facturas.
END.

REPEAT TRANSACTION:
CREATE itemfac.
itemfac.factunum = facturas.factunum.
DISPLAY FACTURAS.FACTUNUM.
UPDATE numlin nro-art pre-art cantidad descuento.
END.

END.

En este ejemplo, **PROGRESS** comienza una transacción para cada factura y también para cada item que ingrese. Hay dos bloques que directamente modifican la base de datos.

* El bloque **DO TRANSACTION** conteniendo la sentencia INSERT. La sentencia **DO TRANSACTION** y su correspondiente sentencia **END** realizan la inserción del registro de facturas.

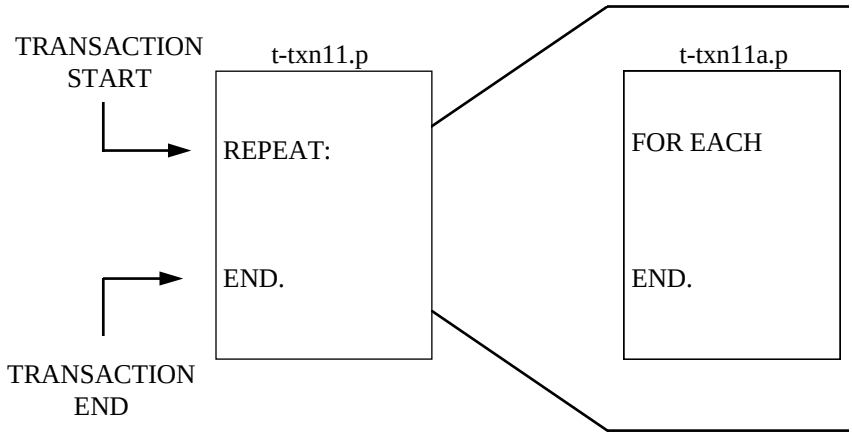
* El bloque interno REPEAT contiene modificaciones directas a la base de datos. (UPDATE).

Si se presiona STOP mientras se está ingresando la segunda factura, PROGRESS solamente deshace la factura que se está ingresando, quedando ya grabada la primer factura ingresada.

Usando transacciones con subprocedimientos

Si comienza una transacción en el procedimiento principal, esta transacción permanece activa en el programa principal que se está ejecutando y en los procedimientos que haya llamado.

El bloque REPEAT t-txn11.p es un bloque de transacción para este procedimiento. Esta transacción comienza al principio de cada iteración del bloque REPEAT y finaliza al final de cada iteración. Esto significa que cuando un procedimiento t-txn11.p llama al procedimiento t-txn11a.p, la transacción permanece activa.
Por lo tanto el trabajo realizado en la subrutina t-txn11a.p es la parte de la transacción comenzada por el procedimiento ppal t-txn11.p



Manejando otros errores

PROGRESS maneja errores tales como la tecla STOP o una falla del sistema en donde una transacción se deshace, no realizando cualquier modificación a la base de datos durante la transacción. Hay 4 tipos de errores que pueden ocurrir:

* Por un error generado en un procedimiento. Por ejemplo, la búsqueda de un registro falla o el procedimiento intenta crear una entrada duplicada en el índice único.

* El usuario presiona una tecla que permite a **PROGRESS** realizar un procesamiento de error.

* El usuario presiona una tecla que permite a **PROGRESS** realizar un procesamiento de '*endkey*'.

* El usuario presiona la tecla *END-ERROR* (F4).

Como maneja PROGRESS errores de procedimientos

Hay dos maneras muy comunes como un procedimiento puede generar un error:

1- Un procedimiento intenta crear una entrada duplicada en un índice único.

2- Un FIND intenta leer un registro que no existe.

En estos casos, **PROGRESS**:

1- Mira las propiedades de error para el bloque en cuestión.

2- Deshace y reintenta el bloque.

Los bloques que tienen automáticamente propiedades de error son:

FOR EACH, REPEAT y bloques de procedimientos. Por "tener propiedades de error" significa que cada uno de los bloques; implícitamente tiene una frase ON ERROR UNDO, RETRY asociado a éste. Por ejemplo:

FOR EACH *clientes*:

es lo mismo tener:

FOR EACH *clientes* **ON ERROR UNDO, RETRY**:

Como maneja usted errores de procedimientos

Usted podrá modificar el manejo de las propiedades de error asignadas por defecto por PROGRESS; usando diferentes frases ON ERROR en las sentencias de bloque DO, FOR EACH o REPEAT.

Comentarios sobre la tecla error

Cuando se está ejecutando un procedimiento, no hay una tecla de error predefinida en el teclado. Sin embargo, podrá definir cualquier tecla de control (CTRL) o una tecla especial como la tecla ERROR. Entonces, cuando el usuario presiona dicha tecla cuando se pide un ingreso, el procedimiento realiza el procesamiento de error. Esto es:

1) Mira para el bloque en cuestión las propiedades de "endkey".

2) Deshace y sale del bloque.

Reglas acerca del UNDO

Cuando utiliza la frase ON ERROR u ON ENDKEY con la opción UNDO, podrá:

* Nombrar el corriente bloque o cualquier bloque con el UNDO.
* Salir (LEAVE) o continuar con el siguiente (NEXT) bloque o cualquier bloque.
* Reintentar (RETRY) solamente el bloque que ha deshecho.

Cualquier bloque que nombra con la opción UNDO debe estar en el corriente bloque o un bloque que contiene el corriente bloque.

Como las transacciones afectan variables

Cualquier cambio hecho en variables en una transacción o subtransacción no es tomada en cuenta cuando una transacción o subtransacción se deshace.

Sin embargo mediante la opción **NO-UNDO** en la definición de la variable o de un arreglo se logra que el servicio del '**UNDO**' no sea tenido en cuenta.