UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Escuela Nacional de Estudios Profesionales Aragón
INGENIERIA EN COMPUTACION

ASPECTOS GENERALES DEL INFORMIX - 4GL

T E S I S

Que para obtener el Título de
INGENIERO EN COMPUTACION
p r e s e n t a

JESUS HERNANDEZ ALVAREZ

Asesor: ING. MANUEL MARTINEZ ORTIZ

México, D. F. Noviembre 1995

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

IN MEMORY
A MI MADRE MARGARITA
A MI ABUELO JUSTO

A MI ABUELA SOCORRO
A MIS TÍAS: MARÍA, ANITA Y ESTELA
A MI HERMANO RAÚL

AGRADECIMIENTOS.

Deseo expresar mi agradecimiento con mucho cariño y respeto, a la Sra. Rosa Tinajero y al Sr. Felipe Tapia, quienes me ofrecieron su casa, su tiempo, su paciencia, en los momentos más difíciles de mi vida.

Mi agradecimiento también, a los autores de los libros que consulté, ya que todo lo escrito no es de mi única responsabilidad.

Finalmente, mi agradecimiento a mi Abuela Socorro. Por confiar siempre en mi. A mis tías: María, Anita, Estela y Antonia; quienes no me dejaron solo; y sin olvidar a mi hermano Raúl. Asimismo, mi agradecimiento a mis amigos, quienes me estimularon y colaboraron en la realización del presente trabajo.

Nuevamente, en nombre de Mamá Margarita y Papá Justo gracias a todos.

. . . Cuánto gané, Cuánto perdí
Cuánto de niño pedí, Cuánto de
grande logré . . .

INTRODUCCION

Con la apertura del tratado del libre comercio, en el mercado del software aparecieron muchos manejadores de base de datos. Uno de los más solicitados por las empresas públicas y privadas para desarrollar sus aplicaciones o actualizar sus sistemas es el INFORMIX-4GL.

INFORMIX-4GL es un lenguaje de programación diseñado especificamente para desarrolladores de aplicaciones. Esta basado en RDSQL (Rapid Development Systructure Query Language), la extensión de INFORMIX-SQL (Structure Query Language) desarrollado por la International Busines Machines (IBM). INFORMIX-4GL es un lenguaje de cuarta generación que ha sido designado para aplicaciones de base de datos.

¿A qué se debe tal exito del INFORMIX-4GL?. Las razones son muchas; se puede facilmente: crear menús de anillos, colectar entradas de datos desde una forma de pantalla, colectar múltiples renglones de datos utilizando una simple pantalla con desplazamiento de datos. Es compatible con UNIX, MS-DOS y sistema VMS y corre en una gran variedad de máquinas, desde micros hasta mainframes. Además provee estructuras de control (IF, WHILE, FOR) para utilizarlos cuando sea necesario. Esto proporciona a los programadores la completa funcionalidad de los lenguajes de tercera generación.

En este trabajo me propongo dar a conocer en forma breve al lector los elementos más importantes del INFORMIX-4GL. Estos le permitiran: armar un programa INFORMIX-4GL, utilizar las formas de pantalla para capturar arreglos de información, utilizar formas y ventanas; lo que producen una interacción directa y amigable con el usuario, crear una base de datos, construir un reporte para explotar una base de datos, y

la presentación de menús circulares. Espero sea de gran utilidad el ejemplo de aplicación presentado en el capitulo 4.

El capitulo 1 presenta la creación, modificación y reorganización de una base de datos. Asi como las instruciones de manipulación de ésta.

El siguiente capitulo trata las estructuras de control, el esquema general de un programa, el uso de formas, ventanas, menús y arreglos de pantalla.

El capitulo 3 muestra lo fácil que es generar un reporte, su estructura y las secciones que lo componen.

En el último capitulo se emplea las instrucciones de los capitulos anteriores para construir el sistema de información de una biblioteca pública municipal.

INDICE

CAPITULO 1 BASE DE DATOS

1.1 BASE DE DATOS

Una base de datos, es un conjunto de datos (almacenados en archivos) organizados de tal forma que permiten guardar y extraer información por medio de la ejecución de programas especiales. Estos archivos comparten cierta información que es necesaria para poder relacionarlos y accesar los datos contenidos en ellos.

Un manejador de base de datos (DBMS: Data Base Management System) es un programa o conjunto de ellos que permiten crear y mantener bases de datos por medio de ciertas facilidades integradas. El DBMS permite que el acceso a la información se lleve a cabo en forma lógica y no física (accesarla según el significado de la información).

1.2 BASE DE DATOS RELACIONAL

Existe un modelo de base de datos en el cual los datos se almacenan en entidades bidimensionales de renglones y columnas llamadas tablas. Una tabla es un archivo en donde cada registro corresponde a un renglón y cada columna a un campo. Las bases de datos que cumplen este modelo se llaman relacionales.

Otra característica de los manejadores de base de datos relacionales (RDBMS: Relational Data Base Management System) es que la información debe ser accesada y actualizada en base a un grupo lógico de información (por ejemplo todos los renglones que cumplan una condición) y no sólo en base a registro a registro. Además, un RDBMS debe soportar valores nulos (indeterminados o inaplicables).

También, se pueden relacionar tablas a través de columnas comunes (lo que genera cierta redundancia pero que es estrictamente indispensable) para obtener información dispersa en varias de esas tablas.

1.3 LENGUAJE DE CUARTA GENERACION

Un lenguaje de cuarta generación tiene dos características fundamentales:

- Primera, es diseñado para una clase particular de aplicaciones: manejo de base de datos. Por ello puede anticiparse a lo que el usuario quiere hacer con los programas y es menos complejo que los lenguajes de propósito general como Pascal, C , COBOL, etc. Un programa escrito en un lenguaje de cuarta generación tiene muchos menos instrucciones que uno escrito en un lenguaje de propósito general.

- Segunda, incorpora ventajas de lenguajes procedimentales y no procedimentales. Un lenguaje procedimental es áquel en el cual se especifica como hacer algo paso a paso, mientras que un lenguaje no procedimental se especifica el resultado final y el lenguaje proporciona el procedimiento. Como ejemplo considérese el caso de leer información de la pantalla. En un lenguaje procedimental hay que especificar paso a paso en que orden deben leerse los datos, en que posición de la pantalla y verificar si son del tipo adecuado además de ver si se dieron valores válidos. En el no procedimental sólo se usa una instrucción especial y se especifican los valores a leer. El lenguaje se encarga de pedir los datos, permite editarlos y verificar si son del tipo correcto (según se especificó previamente en la forma capturada).

1.4 QUE ES INFORMIX-4GL

Es un manejador de base de datos relacional con un lenguaje propio de programación de cuarta generación el cual contiene muchas características propias:

- Manejo de ventanas.
- Creación de menús.
- Lectura de datos de formas de pantalla.
- Uso de SQL (Structured Query Languaje) para manipular la base de datos.
- Manejo de pantallas de ayuda.
- Creación de reportes.
- Búsquedas de información con condiciones especificadas por el usuario (QBE: Query By Example).
- Detección de teclas de función y de control que oprime el usuario.
- Definición de atributos para la información que se lee y escribe a la pantalla.

1.5 TIPOS DE DATOS Y DEFINICION DE VARIABLES

Los diferentes tipos de datos que se pueden asignar a las columnas de las tablas y a las variables de los programas son:

TIPO	CODIFICACION	DETALLES
Cadena	CHAR (n)	n = longitud en bytes.
Enteros	SMALLINT	Amplitud = 32767.
	INTEGER	Amplitud = 2,147,438,647.
Punto	SMALLFLOAT	Prec. Sencilla, 7 dígitos.
Flotante	FLOAT	Prec. Doble, 14 dígitos.
Decimal	DECIMAL (amp [,dec])	amp = amplitud enteros.
		dec = amplitud fracción.
Secuencia	SERIAL [(N)]	Inicia cuenta en n; ésta se incrementa por cada alta.
Fecha	DATE	Formato: mm{/.-}dd{/.-}aa.
Dinero	MONEY	Maneja el formato tipo decimal. Agrega punto y signo.

1.5.1 DEFINICION DE VARIABLES

Existen dos tipos de variables: las globales y las locales. Las primeras tienen validez en cualquier parte del programa, por lo tanto deben declararse fuera de cualquier función o reporte. La forma de hacerlo es:

GLOBALS

DEFINE *lista_variables tipo* [,...]

END GLOBALS

Donde:

Lista_variables	nombre de una o más variables separadas por coma.
tipo	alguno de los tipos ya visto excepto el serial.

El segundo tipo de variables se caracteriza porque sólo son válidas dentro de la función o reporte dentro del cual se declaran. Estas variables se declaran inmediatamente después de **MAIN, FUNCTION** o **REPORT.**

Es posible declarar una variable del mismo tipo de una columna de una tabla:

DEFINE lista_variable **LIKE** tabla.columna [,...]

Con esto se asegura que el tipo de la variable sea exactamente igual al de la columna.

Las variables pueden definirse como registro. Un registro es la agrupación de varias variables, posiblemente de diferente tipo, bajo un mismo nombre.

La forma de definir un registro es:

DEFINE lista_registros **RECORD**

[lista_variables tipo [,...]]

[lista_variables **LIKE** tabla.columna [,...]]

END RECORD

O DEFINE lista_registros **RECORD LIKE** tabla.*

En el primer caso se definen individualmente los elementos del registro mientras que el segundo se declara un registro que tenga los mismos elementos y con los mismos nombres que la tabla.

Para hacer referencia al registro, se usa la notación *registro.** y para referenciar a un elemento en particular se usa la notación *registro.elemento*. También se puede hacer referencia a un conjunto secuencial de campos al registro mediante la palabra **THRU** o **THROUGH** :

registro.elem1 **THRU** registro.elem2

Las variables pueden definirse como arreglos. Un arreglo es una variable que tiene varias posiciones en cada una de las cuales se puede guardar una variable de cierto tipo. La sintaxis es:

DEFINE lista_variable **ARRAY** [i,j,k] **OF** tipo

Ejemplos:

DEFINE arr0 **ARRAY** [5,7,8] **OF** integer

arr1 **ARRAY** [3,2] **OF LIKE** usuario.nom_u

1.6 CREACION DE LA BASE DE DATOS, TABLAS E INDICES

La creación de la base de datos y sus tablas se hace a través de un programa de INFORMIX-4GL. Con ciertas instrucciones especiales. La estructura de este programa la podemos observar a continuación:

MAIN

CREATE DATABASE nombre

CREATE TABLE nombre

[CREATE INDEX]

END MAIN

CREATE DATABASE. Es la instrucción para crear una base de datos. Con esto INFORMIX-4GL genera un directorio con el nombre asignado y una extensión **.dbs**. Este directorio contendrá los catálogos del sistema los cuales albergan información con respecto a las tablas, columnas, índices y autorizaciones.

Restricciones:

- El nombre de la base de datos no puede ser mayor de 10 carácteres.
- Debe iniciar con una letra.
- No usar palabras reservadas.
- El nombre debe ser único.

CREATE TABLE. Instrucción para crear una tabla en una base de datos. Su formato es:

CREATE TABLE nombre _tabla

(. . .

nombre_columna tipo_dato [,...]

. . .)

Los nombres de las tablas y las columnas no deben exceder de 18 caracteres de longitud. Puede darse el mismo nombre a columnas de diferentes tablas pero se recomienda evitarlo. El tipo de dato de una columna puede ser cualquiera de los ya vistos.

CREATE INDEX

La forma de crear un índice es como se muestra a continuación:

CREATE [UNIQUE] INDEX nombre_índice **ON** nombre_tabla

(nombre_columna [DESC][,...])

Donde:

nombre_índice	es el nombre que tendrá el índice.
nombre_tabla	es la tabla sobre la cual se creará el índice.
nombre_columna	es la columna sobre la cual se creará el índice. Puede especificarse varias columnas en un índice.
DESC	es opcional e indica que el ordenamiento sobre la columna será descendiente.
UNIQUE	es opcional e indica que no se pueden insertar valores duplicados en el índice.

1.7 MODIFICACION A LA BASE DE DATOS

Muchas veces es necesario modificar la estructura de la base de datos para agregar alguna columna, ampliar el tamaño de la otra, etc.

1.7.1 AGREGAR, BORRAR Y CAMBIAR EL TIPO DE UNA COLUMNA.

ALTER TABLE nombre_tabla

{ ADD (nueva_col tipo [**BEFORE** columna][,...])

| DROP (columna [,...])

| MODIFY (columna nuevo_tipo [,...])

}

La cláusula **ADD** sirve para agregar una columna al final de la tabla (o antes de la columna que se especifique con **BEFORE**).

La cláusula **DROP** sirve para borrar una columna. Si la tabla en cuestión ya tenía información, ya no se podrá recuperar información de esta columna.

La cláusula **MODIFY** es para cambiar el tipo de una columna. El nuevo tipo debe ser compatible con el actual para que pueda convertirse la información.

Ejemplo:

```
ALTER TABLE credencial
        ADD (fe_ven date BEFORE c_nom_u)
ALTER TABLE credencial
        DROP (fe_ven)
```

ALTER TABLE credencial

MODIFY (c_nom_u CHAR (50))

1.7.2 RENOMBRAR TABLAS Y COLUMNAS

Para renombrar una tabla tenemos la siguiente instrucción:

RENAME TABLE nombre_tabla **TO** nuevo nombre

La forma de cambiar el nombre a una columna es:

RENAME COLUMN tabla.columna **TO** nuevo_nombre

1.7.3 BORRAR TABLAS, ÍNDICES Y BASE DE DATOS

DROP TABLE. Es la instrucción que permite borrar una tabla (incluyendo toda su información). La sintaxis es:

DROP TABLE tabla

Ejemplo: DROP TABLE préstamo

Una vez borrada una tabla la única forma de recuperar la información es crear la tabla de nuevo y recuperar el respaldo.

DROP INDEX. Es la instrucción para borrar un índice. La sintaxis es:

DROP INDEX nombre_índice

Ejemplo: DROP INDEX i_nadq

Un índice se puede borrar y crear tantas veces sea conveniente la información de la tabla no se afecta.

DROP DATABASE. Es la instrucción para borrar una base de datos (incluyendo sus tablas, índices e información). La sintaxis es:

DROP DATABASE base de datos

Ejemplo: DROP DATABASE bibmun

1.8 INSTRUCCIONES DE MANIPULACIÓN DE LA BASE DE DATOS

A continuación se describe como accesar la información de la base de datos: insertar, consultar, modificar y borrar.

1.8.1 INSERCIÓN DE INFORMACIÓN.

La inserción a la base de datos se hace mediante la instrucción INSERT:

INSERT INTO tabla [(lista_columnas)]

VALUES (lista_valores)

Donde:

lista_columnas es opcional e indica a que columnas se les va insertar información. Si se omite esta parte se asume que se agregará información a todas las columnas de la tabla.

lista_valores es la lista que contiene los datos que tomarán las columnas. El número de valores aquí especificado debe concordar con el número en lista_columnas o, en caso de haberse omitido esta, con el número de columnas en la tabla, lista_valores puede ser un registro.

1.8.2 SELECCIÓN DE INFORMACIÓN.

Hay dos formas para seleccionar información de la base de datos: seleccionar sólo un renglón o seleccionar más de un renglón.

a) Obtención de un solo renglón de información

La instrucción **SELECT** permite recuperar información de una base de datos. Así mismo es la base para todas las consultas en INFORMIX-4GL, y el resultado de un SELECT puede ser usado en reportes, desplegados por pantalla y otras opciones más. La sintaxis es:

SELECT [UNIQUE] {lista_columnas|*} **[INTO** lista_variables **]**

FROM lista_tablas

[WHERE condición **]**

[GROUP BY columna **]**

[ORDER BY columna **[DESC] [** ,... **]]**

[INTO TEMP tabla_temporal **]**

En donde:

UNIQUE es opcional e indica que se deben eliminar renglones cuyos valores de la lista_columnas estén duplicados.

lista_columnas son las columnas de la(s) tabla(s) que se seleccionan.

* indica que se seleccionan todas las columnas de la(s) tabla(s) especificadas.

INTO es opcional y especifica las variables en donde se guardan los datos

leídos. El número de variables debe corresponder al número de la lista_columnas.

FROM indica la(s) tabla(s) de la(s) cual(es) se toma la información.

GROUP BY es para producir un sólo renglón de resultados para cada grupo de renglones que tengan los mismos valores para cada columna listada.

ORDER BY es opcional y sirve para especificar si se quiere que la información leída sea ordenada por alguna columna. Si se pone DESC el ordenamiento será inverso.

INTO TEMP indica que la información resultante de esta selección se guardará en la tabla tabla_temporal (que se creara en ese momento) y tendrá tantas columnas como lista_columnas.

WHERE es opcional e indica cuales registros se quieren seleccionar. Si se omite se seleccionan todos. Las condiciones válidas son:

expre1 [NOT] BETWEEN expre2 AND expre3. Es para ver si una variables o constante esta dentro de un rango de valores.

expre [NOT] IN (lista_valores) ver si el valor de una variable o una constante están en la lista_valores.

columna IS [NOT] NULL ver si el valor de una columna es nulo.

expre operador_relacional expre. Son las condiciones comunes de comparación de un elemento a otro: (= , < > , < , > , <= , => , AND, OR).

columna [NOT] MATCHES "expresión" . Es para ver si el valor de una columna de tipo CHAR concuerda con la expresión.

Existe una variación de SELECT que consiste en especificar una función predefinida (llamada función agregada) en vez de lista_columnas. Las funciones válidas son:

COUNT (*) dice cuantos renglones cumplen la condición.

AVG (columna) obtiene el promedio de la columna.

SUM (columna) obtiene la suma de todos los valores de la columna.

MAX (columna) obtiene el valor máximo de la columna.

MIN (columna) obtiene el valor mínimo de la columna.

b) Obtención de información a partir de una serie de renglones.

La forma de poder procesar una serie de renglones es a través de un CURSOR. El CURSOR es una apuntador a un renglón en un conjunto de renglones. Su formato es:

DECLARE nombre_cursor **CURSOR FOR**

SELECT instrucciones

nombre_cursor es un nombre cualquiera que se le asigna al cursor y debe ser único.

Si se desea seleccionar todos los renglones de una tabla, se definirá el cursor como:

DECLARE nombre_cursor CURSOR FOR SELECT * FROM tabla

Por el contrario, si se desea seleccionar varios renglones, entonces:

```
DECLARE nombre_cursor CURSOR FOR
    SELECT * FROM tabla
    WHERE nombre_columna = "algún valor"
```

Una vez leída y guardada la información en un cursor hay que procesarla para obtener dicha información que se accesara un renglón a la vez:

```
FOREACH nombre_cursor [ INTO lista_variables ]
    instrucciones
    [ CONTINUE FOREACH ]
    [ EXIT FOREACH ]
END FOREACH
```

FOREACH. Es una estructura cíclica que va a procesar la información del cursor un renglón a la vez. La primera vez asigna a lista_variables los valores del primer renglón y se ejecutan todas las instrucciones que haya hasta END FOREACH y luego se vuelve al principio y se asigna el segundo renglón, etc. El ciclo termina cuando se hayan agotado los renglones que hay en el cursor o cuando se ejecuta EXIT FOREACH.

Donde:

INTO es opcional sirve para especificar en donde se guardarán los datos recién leídos. Es válido usar la notación **registro.***. Esta cláusula sólo debe usarse si no se usó INTO en la instrucción SELECT del cursor.

CONTINUE FOREACH indica que se procese el siguiente renglón sin ejecutar las instrucciones que pudieran seguir en el cuerpo del FOREACH. Usualmente CONTINUE FOREACH se usa en conjunción con algún condicional (como IF).

EXIT FOREACH termina la ejecución del ciclo FOREACH. Es útil cuando ya no se desea seguir procesando los renglones. Generalmente se usa en conjunción con algún condicional (como IF).

Ejemplo:

```
# Se selecciona toda la tabla credencial
DECLARE cur0 CURSOR FOR
        SELECT * FROM credencial
# Se procesa el cursor 0
FOREACH cur0 INTO reg_cred.*
        DISPLAY BY NAME reg_cred.*
        PROMPT " Quiere ver el siguiente (s/n) ? " FOR CHAR s_n
        IF s_n NOT MATCHES "[sS]"
                THEN EXIT FOREACH
        END IF
END FOREACH
```

1.8.3 ACTUALIZACIÓN DE DATOS EN UNA TABLA

Se pueden cambiar los datos de una tabla usando la siguiente instrucción:

UPDATE tabla **SET** { columna = expresión [,...]

| { (lista_columnas) | [tabla.] * = { (lista_expr) | registro.* }

}

[**WHERE** condición]

Donde:

tabla nombre de la tabla que contiene la(s) columna(s).

WHERE incluye una condición que indica que renglones actualizar.

Si se desea cambiar todas las columnas de uno o varios renglones de una tabla, se usa el siguiente formato del UPDATE.

UPDATE tabla SET tabla.* = registro.*

[WHERE condición]

1.8.4 BORRADO DE INFORMACIÓN

La instrucción DELETE permite borrar uno o más renglones dentro de una tabla. Su formato es el siguiente:

DELETE FROM tabla

[**WHERE** condición]

1.8.5 CONSULTA DE INFORMACION

Existe una manera de seleccionar información en el cual el usuario le aparece una pantalla en donde el selecciona las condiciones a cumplir en los campos que desee. Esto es posible a través de la instrucción CONSTRUCT:

CONSTRUCT { **BY NAME** variable_char **ON** lista_columnas_tabla

| variable_char **ON** lista_columnas_tabla

FROM { lista_campos_forma | registro_pantalla.* } [,...]

} [**ATTRIBUTE** (lista_atribs)]

En donde:

variable_char es una variable de tipo carácter de longitud suficiente para contener varias condiciones. Es esta variable donde se almacenan todas las condiciones que el usuario especifique.

lista_columnas_tabla es la lista de columnas de una tabla sobre las cuales el usuario podrá dar condiciones.

lista_campos_forma es la lista de campos en la forma de donde se leerán los datos. Debe haber el mismo número de campos que de lista_columnas_tabla.

registro_pantalla es un registro de pantalla que se declara en la sección INSTRUCTIONS de la forma en cuestión.

ATTRIBUTE es para especificar los atributos de vídeo con los que aparecen los campos en la pantalla al capturarlos.

BY NAME se usa cuando los campos en la forma se llaman igual que las columnas de la tabla.

Para utilizar el CONSTRUCT son necesarios varios pasos:

1. Formar una cadena de carácteres que contengan una instrucción SELECT. A dicha cadena hay que concatenarle las condiciones que el usuario especifico al ejecutarse el CONSTRUCT:

LET cad0 = " SELECT * FROM tabla WHERE "[1] **, variable_char**

2. Preparar esta cadena para que pueda ser ejecutada:

PREPARE listo FROM cad0

3. Declarar un cursor y sustituir la instrucción SELECT por la cadena recién preparada:

DECLARE cur1 CURSOR FOR listo

Donde:

cad0 es una variable tipo CHAR al menos 30 carácteres más grande que variable_char.

listo no es una variable, es sólo un almacenamiento temporal.

[1] dejar espacio en blanco antes y después de cerrar las comillas cuando se construye una instrucción SELECT de esta manera.

1.8.6 RECORRIDO ALEATORIO DE CURSORES

En una sección anterior se describió el recorrido secuencial de un cursor. Esto a través de la instrucción FOREACH. Sin embargo, por medio de esta instrucción no es posible regresar a un renglón del cursor por el que ya se había pasado ni es posible moverse de manera aleatoria entre los renglones. La instrucción alternativa para poder realizarlo es FETCH.

La instrucción FETCH no es una estructura cíclica como el FOREACH, su único objetivo es mover el apuntador del cursor al renglón que se le pida. Esto significa que antes de poder utilizar esta instrucción tenemos que abrir el cursor y además controlar el flujo del programa si es que queremos ciclar. Una vez que dejamos de utilizar el cursor hay que cerrarlo.

Para abrir el cursor :

OPEN nombre_cursor

Para cerrarlo contamos con la instrucción :

CLOSE nombre_cursor

La sintaxis de la instrucción **FETCH** es la siguiente:

FETCH [NEXT { PREVIOUS | PRIOR } | CURRENT | FIRST | LAST

| ABSOLUTE n | **RELATIVE** m] nombre_cursor [**INTO** lista_variables]

Donde:

NEXT opción por default e indica que se moverá al siguiente renglón.

PREVIOUS indica que se moverá al renglón anterior.

PRIOR	es un sinónimo de PREVIOUS.
CURRENT	se queda posicionado en el renglón actual pero relee los datos que hay allí.
FIRST	es para moverse al primer renglón del cursor.
LAST	es para moverse al último renglón del cursor.
ABSOLUTE n	es para ir al n_ésimo renglón del cursor.
RELATIVE m	es para hacer un movimiento relativo al renglón actual e indica cuantos renglones moverse a partir del actual.
nombre_cursor	es para indicarle sobre cual cursor se hará el movimiento.
INTO lista_variables	es para indicar en donde guardar los datos del renglón leído.

Para poder usar el FETCH en un cursor es necesario declarar este cursor como SCROLL :

DECLARE nombre **SCROLL CURSOR FOR**

SELECT instrucciones

CAPITULO 2. ESTRUCTURAS BASICAS DE PROGRAMACION

2.1 ESQUEMA GENERAL DE UN PROGRAMA

La estructura genérica de un programa INFORMIX-4GL puede definirse en base al siguiente esquema:

```
[ DATABASE base de datos ]
[ GLOBALS
        DEFINE variable tipo [,...]
        END GLOBALS
]
[ DEFINE variable tipo [,...] ]
MAIN
        [ DEFINE variable tipo [,...] ]
instrucciones
END MAIN
[ FUNCTION ]
[ REPORT ]
```

ESTRUCTURA DE UNA FUNCION

FUNCTION nombre ([parámetros])

[**DEFINE** parámetros tipo, variable tipo]

[**RETURN** [lista valores]]

END FUNCTION

ESTRUCTURA DE UN REPORTE

REPORT nombre ([parámetros])

[**DEFINE** parámetros tipo, variable tipo]

cuerpo del reporte

END REPORT

A continuación una breve descripción de cada una de las estructuras:

DATABASE. Para seleccionar la base de datos en un programa. Esta instrucción puede ser usada en cualquier parte del programa, excepto si utilizamos la palabra **LIKE**, en este caso debe anteceder a **MAIN**.

GLOBALS. En la programación con INFORMIX-4GL, pueden ser utilizadas variables locales, globales y parámetros. Una variable global puede ser compartida por todos los módulos de programación (**MAIN, FUNCTION, REPORT**) y para declararla utilizamos esta sección. Si existen variables globales y locales con el mismo nombre, la local toma procedencia dentro del bloque en la cual ésta es definida. Es decir, si asignamos un valor de una variable local en su función, el valor de la variable global permanece sin cambio.

MAIN. Abre y cierra la ejecución de un programa INFORMIX-4GL, es decir, es el módulo principal.

FUNCTION. Función en INFORMIX-4GL es una serie de instrucciones abarcadas por una etiqueta **FUNCTIÓN** y una **END FUNCTION.** Sirve para estructurar nuestro programa en una serie de bloques que contengan instrucciones para un fin determinado. Una función puede recibir parámetros. Utilizar variables locales y globales, y regresar resultados al bloque que la llamó.

REPORT. La estructura nos permite diseñar un reporte al estilo de ACE[2]. Generalmente los datos son proporcionados por una función que contenga un ciclo generador.

2.1.1 FUNCIONES

Una función es un conjunto de instrucciones agrupadas bajo un solo nombre. El propósito de ello es obtener un programa más modular y evitar la repetición de código. Una función debe ser llamada para poder ejecutarse y puede devolver resultados a quién la llamó.

El formato general de una función es:

```
FUNCTION nombre ( [ parámetros ] )
        [ DEFINE parámetros tipo,
                variable tipo ]
END FUNCTION
```

[2] es un manejador de reportes el cual proporciona un lenguaje expresamente orientado para la fabricación de reportes impresos.

Donde :

nombre es el nombre de la función.

parámetros es opcional y es una lista de valores que se transfieren a la función para ser utilizados. En caso de haber parámetros estos deben declararse como si fueran variables.

RETURN es opcional y provoca que termine la ejecución de la función y se devuelve el control a donde se hizo la llamada. Si se especifica lista_valores estos deben asignarse a algún lado en la parte que hace el llamado. Si RETURN se omite, la ejecución de la función termina al llegar a END FUNCTION.

Las variables que se declaran dentro de la función serán locales a ella.

Hay varias formas de llamar a una función :

1. **CALL** función ([lista_parámetros]). La función llamada no puede devolver ningún valor. Solo ejecuta ciertas instrucciones.

2. **CALL** función ([lista_parámetros]) **RETURNING a**. En este caso la función debe incluir la instrucción **RETURN** valor. Este valor será asignado a la variable **a**.

3. **CALL** función ([lista_parámetros]) **RETURNING a, b, c**. Se hace un llamado a la función la cual tiene que devolver tres valores (mediante la instrucción **RETURN**) que se asignarán respectivamente a **a, b, c**.

2.2 ESTRUCTURAS DE CONTROL DE FLUJO

Para controlar el flujo de ejecución de un programa tenemos las siguientes estructuras de control :

2.2.1 IF

Es una estructura condicional que sirve para hacer bifurcaciones, tomar un camino si se cumple una condición y tomar otro camino si no.

IF expresión

THEN

instrucciones

ELSE

instrucciones

END IF

Donde :

expresión puede ser cualquiera expresión que dé como resultado un valor de falsedad o verdad. Generalmente son comparaciones entre variables y constantes.

2.2.2 CASE

Es una estructura que nos permite comparar una variable contra una serie de valores y tomar ciertas acciones en cada caso:

```
CASE [ ( variable ) ]
        WHEN expre1
        instrucciones1
        [ EXIT CASE ]
        WHEN expre2
                instrucciones2
                [ EXIT CASE ]
        ...
        ...
        [ OTHERWISE
                instruccionesN
                [ EXIT CASE ]
        ]
END CASE
```

El CASE equivale a una serie de IF's anidados :

```
        IF expre1

                THEN instrucciones1

                ELSE IF expre2

                        THEN instrucciones2

                        ELSE . . .

                                [ ELSE instruccionesN ]

        END IF
```

Ejemplo:

Primera forma del uso de CASE

```
PROMPT "Escoja una opción" FOR CHAR contesta
CASE
    WHEN contesta = "1"
        CALL opcion1 ( ) # llama a la función opcion1
    WHEN contesta = "2"
        CALL opcion2 ( ) # llama a la función opcion2
    OTHERWISE
    DISPLAY " opción inválida " AT 10,20
END CASE
```

Segunda forma del uso de CASE

```
PROMPT "Escoja una opción" FOR CHAR contesta
CASE (contesta)
    WHEN "1"
        call opcion1 ( ) # llama a la función opcion1
    WHEN "2"
        call opcion2 ( ) # llama a la función opcion1
    OTHERWISE
    DISPLAY " opción inválida " AT 10,20
END CASE
```

2.2.3 WHILE

Es una estructura que permite ejecutar repetidas veces las instrucciones que se indiquen dependiendo del valor de la condición. Las instrucciones se ejecutan mientras la condición se cumpla:

WHILE condición

instrucciones

[EXIT WHILE]

[CONTINUE WHILE]

END WHILE

Donde:

EXIT WHILE es opcional e indica que se abandonará la estructura WHILE sin importar si quedan varios ciclos sin ejecutar. Generalmente se usa en conjunción con IF.

CONTINUE WHILE es opcional y permite continuar con el siguiente ciclo de WHILE (si la condición se sigue cumpliendo) sin importar que haya más instrucciones que ejecutar después de CONTINUE WHILE y antes de END WHILE. Se usa en conjunción con IF.

2.2.4 FOR

Es una estructura que permite ejecutar las instrucciones que se especifiquen un cierto número de veces:

FOR variable_entera = expent1 **TO** expent2 **[STEP** entero **]**

instrucciones

[CONTINUE FOR]

[EXIT FOR]

END FOR

En donde:

expent1 y expent2 son expresiones de números enteros.

STEP es opcional e indica los incrementos que se harán a expent1 cada vez que se cicle. Si se omite, el valor de default es 1. STEP puede ser negativo en cuyo caso expent2 debe ser menor que expent1.

CONTINUE FOR es opcional e indica continuar con el siguiente ciclo. Se usa en conjunción con IF.

EXIT FOR Sirve para terminar la estructura FOR sin importar que falten varios ciclos por ejecutar. Se usa en conjunción con IF.

Este ciclo se ejecuta expent2 - expent1 +1 veces (si la clásula STEP se omite).

El ciclo trabaja de la siguiente forma:

- A variable_entera se le asigna expent1 y se ejecuta el ciclo.

- Al llegar a END FOR se vuelve al inicio y variable_entera se incrementa según lo indica STEP. Si el valor resultante es menor o igual que expent2 entonces se ejecuta el ciclo, si no termina.

- Si expent2 es menor que expent1 y STEP es positivo no se ejecuta el ciclo ni una vez.

Ejemplo:

```
LET positivos = 0
FOR y = 1 TO 10
    IF elemento [ y ] <= 0
        THEN CONTINUE FOR
    END IF
Display elemento [ y ]
    LET positivos = positivos + 1
END FOR
```

2.3 FORMAS

Una forma es una pantalla que se utiliza para leer y desplegar información. En ella se define la información con la que se desea trabajar así como ciertos atributos y condiciones que debe cumplir. El formato general que debe tener una forma es :

```
DATABASE base de datos
SCREEN
{
        texto1 [etiq1 ]
        texto2 [etiq2 ]
}
[ END ]
[ TABLES
        tabla1
        [ tabla2 ... ] ]
[ END ]
ATTRIBUTES
        etiqueta = [ tabla. ] columna [ , lista_atributos ] ;
[ END ]
[ INSTRUCTIONS
        DELIMITERS "ab"
        SCREEN RECORD registro [ [ tamaño ] ] ( lista_campos)
[ END ]
]
```

Donde :

DATABASE indica con cual base de datos se está trabajando.

SCREEN es la sección en donde se dibuja la forma tal y como se quiere que aparezca en la pantalla. Se delimita por los caracteres { y }.

texto1, texto2, etc. son textos que indican que información se debe dar a continuación.

etiq1, etiq2, etc. son etiquetas que no aparecerán en la pantalla y que sirven para que en la sección de ATTRIBUTES se especifique atributos y condiciones que deben cumplir los datos que se ingresen.

TABLES es una sección en la que se especifican las tablas de las cuales se toman columnas para desplegar o leer información.

ATTRIBUTES es una sección en la que se especifican las etiquetas declaradas en la sección SCREEN y los atributos que deben cumplir los datos asociados a ellas. Los atributos que se pueden asociar a los campos de la pantalla son:

COMMENTS. Es para poner un comentario que será desplegado al momento de estar capturando. El formato es COMMENTS = " texto".

DEFAULT. Sirve para especificar un valor por default para este campo. El valor indicado es asignado automáticamente oprimiendo retum o puede cambiarse. El formato es DEFAULT = valor.

INCLUDE. Es para indicar que rango o lista de valores puede ser introducido en el campo. El formato es INCLUDE = (lista_valores). Los valores de la lista_valores pueden separarse por comas o utilizar la palabra TO para indicar un rango; puede hacerse una combinación de ambos.

PICTURE. Sirve para especificar un formato al campo. Los carácteres válidos son:

A letra

dígito

X carácter alfanumérico

El formato es PICTURE = " formato ".

FORMAT. Es para dar un formato a variable tipo: decimal, smallfloat, float y date. El carácter válido para las cantidades numéricas es solo " # " y para fechas son:

dd	día numérico
ddd	día con letras
mm	mes numérico
mmm	mes con letra
yy	año con 2 dígitos
yyyy	año con 4 dígitos

El formato es FORMAT = " formato "

REQUIRED. Especifica que debe darse un valor a este campo no puede quedarse vacío.

AUTONEXT. Cuando el campo se llena se hace el pase automático al siguiente campo.

NOENTRY. Indica que en este campo no se puede capturar información, sólo desplegar. Este campo es brincado al momento de la captura.

VERIFY. Se pide al usuario que capture dos veces este campo

antes de pasar a otro.

UPSHIFT. Todos los carácteres en minúsculas que se digiten se convierten a mayúsculas.

DOWNSHIFT. Como en el anterior, pero las letras son pasadas a minúsculas.

REVERSE. La amplitud del campo es puesta en vídeo inverso cuando se captura.

En la sección INSTRUCTIONS se declaran los carácteres que desean usarse para delimitar los campos y también se declaran los registros de pantalla.

DELIMITERS especifica los nuevos delimitadores. Por default son "[ ]".

SCREEN RECORD es en donde se declaran registros de pantalla. Un **registro de pantalla** es un grupo de varios campos de la forma bajo un mismo nombre para que puedan manejarse en conjunto facilitando la programación. **Registro** es el nombre que tendrá el registro de pantalla. **Tamaño** es opcional e indica que se declara un arreglo de pantalla. **Lista_campos** es la lista de los campos de la forma que formarán parte del registro.

Ejemplo :

SCREEN RECORD registro (campo1 THRU campo2)

SCREEN RECORD registro (campo1, campo [,...])

SCREEN RECORD registro (tabla.*)

2.3.1 USO DE LA FORMA

Para habilitar una forma está debe abrirse con la siguiente instrucción :

OPEN FORM *nombre* **FROM** "*forma*"

Donde ***nombre*** es el nombre que tendrá la forma dentro del programa; ***forma*** es el nombre con el que se creó la forma.

Y después desplegar la forma. La manera de hacerlo es:

DISPLAY FORM *nombre*

nombre es el nombre que se le asignó a la forma al momento de abrirla. A partir de aquí ya se puede leer y escribir información en la pantalla.

Una vez que se termine de utilizar una forma, ésta debe cerrarse:

CLOSE FORM *nombre*

2.3.2 DESPLEGAR INFORMACION EN LA FORMA

Para desplegar información en una forma se usa la siguiente instrucción :

DISPLAY { **BY NAME** lista_variables

| lista_variables **TO** { lista_campos_forma | registro_pantalla.* } [,..]

} [**ATTRIBUTE** (lista_atrib)]

El primer formato DISPLAY BY NAME. Puede ser utilizado cuando los nombres de las variables tienen el mismo nombre que los campos de la forma. Cuando los nombres de las variables sean distintos a los de los campos se puede usar el segundo formato (TO) en donde se despliegan ciertas variables en los campos que se especifique o en un registro de pantalla.

La cláusula ATTRIBUTE es para desplegar la información con ciertos atributos de vídeo.

Ejemplo :

```
DISPLAY BY NAME registro.* ATTRIBUTE (reverse)

DISPLAY BY NAME nom_u, dom_u

DISPLAY nom_u, dom_u TO r_nom_u, r_dom_u

DISPLAY num_u, dom_u, nom_u TO reg_pant.*
```

2.3.3 INFORMACION DE LA FORMA

Para leer información de una forma se utiliza la instrucción INPUT :

```
INPUT [ BY NAME lista_variables [ WITHOUT DEFAULTS ]
      | lista_variables [ WITHOUT DEFAULTS ]
            FROM [ lista_campos_forma | registro_pant.* ]
      ] [ ATTRIBUTE (lista_atrib) ] [ HELP n ]
       [ { BEFORE FIELD lista_campos_forma
       | AFTER { FIELD lista_campos_forma | INPUT }
       | ON KEY ( tecla )
         }
      instrucciones
      [ NEXT FIELD nombre_campo_forma ]
      [ EXIT INPUT ]
      . . .
      END INPUT
      ]
```

La primer forma es INPUT BY NAME. Es usada cuando los nombres de las variables son los mismos que los nombres de los campos en la forma de donde se van a leer.

Es importante tener en cuenta que la instrucción INPUT lee la información de la pantalla, pero no pasa a las columnas de la base de datos. Para ello debe usarse la instrucción INSERT.

La cláusula WITHOUT DEFAULTS sirve para que antes de leer la información se despliegue en los campos respectivos los valores que tienen almacenadas las variables.

La segunda forma de INPUT se usa cuando los nombres de las variables no son iguales a los nombres de los campos de la forma. Se especifica que variables leer y de cuales campos. Si se especifica un registro de pantalla se leerán los datos de los campos contenidos en ese registro de pantalla.

ATTRIBUTE es para especificar ciertos atributos de vídeo para los campos que se leen.

HELP n se utiliza para especificar mensajes de ayuda mientras se está en la captura de los datos.

BEFORE FIELD se usa para ejecutar alguna acción cuando el cursor entra a un campo.

AFTER FIELD para ejecutar una acción cuando el cursor sale del campo.

AFTER INPUT son acciones que se ejecutan cuando el usuario desea terminar INPUT.

ON KEY permite especificar acciones que se ejecutan al oprimir cierta tecla en cualquier momento dentro del INPUT.

NEXT FIELD para controlar el movimiento del cursor durante el INPUT.

EXIT INPUT provoca que se abandone INPUT ignorando las restantes instrucciones del bloque de INPUT.

END INPUT instrucción usada sólo si se utiliza alguna de las siguientes cláusulas : BEFORE FIELD, AFTER FIELD, AFTER INPUT, ON KEY.

2.3.4 LIMPIEZA DE UNA FORMA

La instrucción **CLEAR** limpia uno o varios campos de una forma. Las opciones son :

CLEAR SCREEN borra toda la pantalla.

CLEAR FORM limpia todos los campos de la forma desplegada.

CLEAR campo [,...] limpia los campos especificados.

2.4 VENTANAS

INFORMIX-4GL permite un manejo de ventanas de una manera muy sencilla. Basta especificar un nombre y las coordenadas para ella. Las principales aplicaciones de las ventanas son :

- si al capturar un artículo y deseamos su código, se abre una ventana para consultarlo.

- si una matrícula no existe, podríamos registrarla en la misma rutina abriendo una ventana.

- podríamos abrir una ventana central con un menú y alrededor ventanas con las diferentes aplicaciones.

Para abrir una ventana se usa la instrucción OPEN WINDOW :

OPEN WINDOW nombre **AT** ren, col

WITH { n **ROWS**, m **COLUMNS** | **FORM** " forma " }

[**ATTRIBUTE** (lista_atribs)]

Donde :

nombre es el nombre que se asigna a la ventana.

ren, col son las coordenadas en la pantalla de la esquina superior izquierda de la ventana.

WITH es para especificar el tamaño de la venta o la forma que se desplegara en ella.

n número de renglones que tendrá la ventana.

m número de columnas que tendrá la ventana.

FORM "forma" especifica que la forma automáticamente se desplegara dentro de la ventana y no es necesario cerrarla con CLOSE FORM.

ATTRIBUTE indica los atributos de vídeo que se usaran para desplegar la ventana.

lista_atribs es una lista de un color y más de otros atributos:

COLOR	ATRIBUTOS
white	border
yellow	reverse
magenta	prompt line n
red	message line n
cyan	form line n
green	comment line n
blue	
black	

Es posible abrir más de una ventana a la vez y pueden sobreponerse. La forma de activar una ventana diferente a la actual es con la instrucción :

CURRENT WINDOW IS { SCREEN | nombre_ventana **}**

Una vez que se activa una ventana ésta pasa a ser la ventana actual. Si esta ventana estaba oculta por otra antes de ser activada, pasa al frente y es visible completamente.

Si se especifica SCREEN en vez del nombre de la ventana, se activa toda el área de la pantalla que no pertenece a ninguna ventana.

Para poner en blanco una ventana usamos la instrucción :

CLEAR WINDOW { nombre_ventana | **SCREEN }**

La ventana que se especifique no requiere de ser la actual. Si se utiliza SCREEN en vez del nombre de una ventana, se limpia toda la pantalla excepto el área ocupada por otras ventanas.

La instrucción para cerrar ventanas es :

CLOSE WINDOW nombre_ventana

Si se cierra la ventana actual, la siguiente ventana en la pila pasa a ser la ventana actual. Si se cierra una ventana que no es la actual, simplemente desaparece de la pila.

2.5 MENUS

INFORMIX-4GL permite facilidades para el uso de menús. Estos son horizontales e incluyen una línea de ayuda para cada opción.

Un menú se define a través del siguiente formato :

```
MENU " nombre_menu "
    COMMAND [ KEY ( lista_teclas ) ] " opcion1 " [ " mensaje_ayuda " ]
        [ HELP numero_ayuda1 ]
        instrucciones
        [ CONTINUE MENU ]
        [ NEXT OPTION " opción " ]
        [ EXIT MENU ]
    COMMAND [ KEY (lista_teclas ) ] " opcion2 "
END MENU
```

En donde :

nombre_menu	es el nombre que se da al menú y que aparecerá antes de todas las opciones.
COMMAND	indica el inicio de una opción.
mensaje_ayuda	es un mensaje que aparece cada vez que la opción en cuestión esté señalada en el menú.
HELP	es opcional y sirve para especificar un texto de ayuda. Esta ayuda aparece al teclear *control-w* o alguna otra combinación que se defina para ello.
CONTINUE MENU	es opcional y hace que el menú se redespliegue en la pantalla para que el usuario escoja una nueva opción.
NEXT OPTION	es opcional y es para que se sugiera ejecutar a continuación la opción indicada.
EXIT MENU	es opcional y hace que se termine de ejecutar el menú.
lista_teclas	es una o más letras o teclas de función o de control separadas por coma.

Para poder usar la ayuda en línea de los menús hay que crear un archivo que contenga los mensajes de ayuda. El archivo deberá de tener la siguiente estructura :

```
.1
texto de ayuda asociado a HELP1
.2
texto de ayuda asociado a HELP2
. . .
. . .
```

Una vez creado este archivo generar el archivo ejecutable para que INFORMIX-4GL pueda entenderlo mediante el comando:

mkmessage ayuda *ayuda.ex*

Hecho lo anterior, el siguiente paso es activarlo y definir bajo que secuencia de teclas será invocado. Esto mediante:

OPTIONS HELP FILE " *ayuda.ex* "

HELP KEY control-w

Donde :

HELP FILE indica el archivo compilado que contiene los mensajes de ayuda.

HELP KEY es la combinación de teclas que nos permitirá accesar la ayuda.

2.6 ARREGLOS DE PANTALLA

Un arreglo de pantalla permite capturar varios registros en una sola pantalla. Para ello hay que definir un arreglo en la forma y una variable tipo arreglo en el programa. En esta variable es en donde se almacenara la información y en el arreglo de pantalla se captura y despliega.

A continuación se presenta una forma en la que se define un arreglo de pantalla :

```
Database company
Screen
{
  No_stock   cantidad   precio unitario   precio total
  [ p ]      [ q ]      [     u     ]     [     t     ]
  [ p ]      [ q ]      [     u     ]     [     t     ]
  [ p ]      [ q ]      [     u     ]     [     t     ]
  [ p ]      [ q ]      [     u     ]     [     t     ]
  [ p ]      [ q ]      [     u     ]     [     t     ]
}

Tables
Control
end
Attribute
p = control.No_stock
q = control.cantidad
u = control.precio_unit
t = control.precio_tot
end
Instructions
screen record sc_datos [ 5 ] ( control.* )
end
```

Como puede observarse en la sección **screen** se repite la línea de datos que se quiere capturar tantas veces como el tamaño del arreglo. En la sección **instructions** se define un registro de pantalla múltiple :

SCREEN RECORD registro [longitud] (lista campos)

Ahora debe definirse una variable que pueda soportar la información que contenga un arreglo de pantalla :

DEFINE arreglo **ARRAY** [longitud] **OF RECORD**

Elem1 tipo[,..]

END RECORD

Donde :

longitud debe ser igual o mayor que el tamaño del arreglo de pantalla.

La forma de referenciar a un elemento del arreglo es :

arreglo [posicion].elemento

mientras que para referenciar un registro completo se usa la siguiente notación :

arreglo [posicion].*

2.6.1 CAPTURA DE DATOS EN UN ARREGLO

Una vez definidos los elementos del arreglo de pantalla podemos usarlo para capturar información en la pantalla.

La forma de realizarlo es mediante la siguiente estructura :

```
INPUT ARRAY arreglo [ WITHOUT DEFAULTS ]
        FROM arreglo_pantalla.* [ HELP n ] [ ATTRIBUTE ( lista_atribs ) ]
        [ { BEFORE { INSERT | FIELD lista_campos | DELETE }
         | AFTER { INSERT | FIELD lista_campos | DELETE | INPUT }
         | ON KEY ( lista_teclas )
         }
        instrucciones
        [ NEXT FIELD nombre_campo ]
        [ EXIT INPUT ]
        . . .
        END INPUT
        ]
```

En donde :

arreglo es el receptor de la información de la pantalla.

arreglo_pantalla es el nombre del arreglo de pantalla de la forma.

WITHOUT DEFAULTS es opcional y tiene la misma función que en la instrucción INPUT.

HELP n	es para especificar que se despliegue el texto de ayuda asociado al número n cuando se oprima la tecla de ayuda.
lista_atribs	es la lista de atributos de vídeo.
ATTRIBUTE	es para indicar los atributos de vídeo con las cuales se despliegan los campos que se capturan.
AFTER FIELD *BEFORE FIELD* *ON KEY* *AFTER INPUT* *NEXT FIELD* *EXIT INPUT* *END INPUT*	son opcionales y tienen la misma función que en la instrucción INPUT.
BEFORE INSERT	es opcional y son instrucciones que se ejecutan cuando el usuario presiona la tecla de insertar un renglón.
BEFORE DELETE	es opcional y son instrucciones que se ejecutan cuando el usuario presiona la tecla de borrar un renglón.
AFTER DELETE	permite ejecutar instrucciones cuando el usuario presiona la tecla de borrar un renglón y el renglón se haya borrado.
AFTER INSERT	permite ejecutar instrucciones cuando el usuario inserta un renglón y lo abandona.

2.6.2 USO DE TECLAS ESPECIALES

Cuando INFORMIX-4GL encuentra la orden de capturar en el arreglo, éste posiciona el cursor en el primer campo de la pantalla y permite la entrada de datos.

Mientras INPUT ARRAY se esté ejecutando el usuario dispone de varias teclas para realizar diversas funciones :

[ESC] termina la captura del arreglo.

F1 inserta un nuevo renglón.

F2 borra el renglón actual.

F3 despliega la siguiente página de renglones del arreglo.

F4 despliega la página anterior de renglones del arreglo.

← funciona dentro de los campos del arreglo y mueve el cursor a través de este de derecha a izquierda.

→ de manera similar a la tecla anterior ésta mueve el cursor de izquierda a derecha.

↑ mueve el cursor sobre el mismo campo, pero un renglón arriba.

↓ mueve el cursor sobre el mismo campo, pero un renglón abajo.

2.6.3 FUNCIONES

Se dispone de cuatro funciones para controlar los arreglos.

set_count (x) inicializa el valor de arr_count ( ) con el número de renglones actualmente almacenados (que es x).

scr_line ( ) devuelve el número del renglón actual dentro del arreglo de pantalla.

arr_curr ( ) retorna el número del renglón actual dentro del arreglo de programa.

arr_count ( ) regresa el número de renglones ocupados en el arreglo de programa.

2.6.4 DESPLEGADO DE ARREGLOS.

La forma de desplegar información en un arreglo de pantalla para su consulta es :

DISPLAY ARRAY arreglo_programa **TO** arreglo_pant.*

Como requisito de la instrucción es necesario llamar la función *set_count(x)*, para indicar el número de elementos guardados en el arreglo.

Al ejecutarse la instrucción se llenará el área con los primeros elementos del arreglo y podemos entonces empezar a recorrer la información a través de ésta utilizando las teclas de recorrimiento.

CAPITULO 3 GENERACION DE REPORTES

Para la generación de reportes INFORMIX-4GL cuenta con manejo de reportes que nos facilitan el procesamiento de la información. Permite agrupar datos, obtener totales, poner encabezados y dar formato a la información. La forma de manejar estos reportes es definiendo un módulo **REPORT**.

3.1 ESTRUCTURA

La estructura general de un reporte es la siguiente:

REPORT nombre_rep ([parámetros])

[**DEFINE** parámetros tipo,

variables tipo]

[sección **OUTPUT**]

[sección **ORDER BY**]

FORMAT EVERY ROW / FORMAT instrucciones

END REPORT

En donde :

nombre_rep es el nombre que identifica al reporte.

parámetros es opcional y es una lista de valores que entran al reporte para que los procese e imprima. Comúnmente hay que transferir al reporte todos los datos a imprimir.

DEFINE debe declararse los parámetros y las variables locales.

La sección **OUTPUT** sirve para cambiar el formato de la página, así como para indicar donde se desea dejar el reporte. En la sección **ORDER BY** se indica si deseamos ordenar los registros de impresión o si ya vienen ordenados.

La cláusula más importante es la **FORMAT** y aquí se especifica como saldrán impresos los registros. La primer forma es la más sencilla e indica que lo que entra en las variables saldrá tal y como entra en sus formatos de default. La segunda modalidad permite definir encabezados y pies de página, cortes de control antes y después, impresión forrajeada, cálculos y otras características más.

3.1.1 SECCION OUTPUT

En esta sección opcional se asigna un formato a las páginas del reporte y se direcciona ya sea hacia un archivo o hacia una impresora. El formato de esta sección es:

OUTPUT

[REPORT TO ("archivo"| PRINTER | PIPE "programa")]

[LEFT MARGIN entero **]**

[RIGHT MARGIN entero **]**

[TOP MARGIN entero **]**

[BOTTOM MARGIN entero **]**

[PAGE LENGTH entero **]**

En donde:

REPORT TO indica a donde mandar el reporte. Si no especifica se asume la pantalla. PIPE indica que el reporte se pasará como entrada al

programa.

LEFT MARGIN define el margen izquierdo de la hoja de reporte.

RIGHT MARGIN permite definir el margen derecho de la hoja de reporte.

TOP MARGIN define el margen superior de la hoja.

BOTTOM MARGIN define el margen inferior del reporte.

PAGE LENGTH define el número de líneas en cada página de un reporte.

Los valores por default para los márgenes son:

LEFT MARGIN 5

RIGHT MARGIN 132

TOP MARGIN 3

BOTTOM MARGIN 3

PAGE LENGTH 66

3.1.2 SECCION ORDER BY

Esta sección del reporte es opcional y especifica cómo debe estar ordenada la información. El formato de la instrucción es como sigue:

ORDER [EXTERNAL] BY lista_variables.

ORDER BY especifica el orden en el cual se ordenará los registros.

EXTERNAL indica que los registros ya vienen ordenados. INFORMIX no los reordena. También sirve para indicar jerarquía de ordenamiento sólo si los renglones ya han sido ordenados previamente.

lista_variables es una o más variables que se recibieron en el reporte por los cuales se ordenara la información.

3.1.3 SECCION FORMAT

En esta sección del reporte se tienen dos opciones para indicar como debe imprimirse el reporte; el primero indica que no hay formatos establecidos. El reporte constara sólo de un encabezado que es el nombre de la columna y el valor de ella. El segundo, establece una serie de modalidades para optimizar la presentación del reporte.

Los formatos son:

1. **FORMAT EVERY ROW.**

2. **FORMAT**
   [PAGE HEADER
   instrucciones]
   [FIRST PAGE HEADER
   instrucciones]
   [PAGE TRAILER
   instrucciones]
   ON EVERY ROW
   [ON LAST ROW
   instrucciones]
   [AFTER GROUP parámetro
   instrucciones]
   [BEFORE GROUP variable
   instrucciones]

FIRST PAGE HEADER

Es para especificar una serie de instrucciones a ejecutar sobre el encabezado de la primer página.

Ejemplo:

```
FIRST PAGE HEADER

PRINT "Biblioteca Publica Municipal"

PRINT "Usuario      Domicilio      Escuela"
```

PAGE HEADER

En esta parte se especifican una serie de instrucciones que serán ejecutadas en cada encabezado de las páginas.

Ejemplo:

```
PAGE HEADER

PRINT "Usuario      Domicilio      Escuela"
```

BEFORE GROUP OF variable

Donde variable pertenece a la lista de argumentos y prevalece un ordenamiento sobre ella. Son instrucciones que se ejecutan antes de iniciar el procesamiento de un nuevo grupo de valores similares a la variable.

Ejemplo:

```
BEFORE GROUP OF c_nus

PRINT "Número de usuario : ", c_nus

SKIP 1 LINE
```

ON EVERY ROW

En esta parte se indica como se quiere cada línea de información.

Ejemplo:

```
ON EVERY ROW

PRINT c_nus, 5 spaces, dom_u, 5 spaces, esc_t
```

AFTER GROUP OF parámetro

Son instrucciones que se ejecutan cada vez que hay un cambio de valor en "parámetro". Se usa para totalizar.

Ejemplo:

```
AFTER GROUP OF c_nus

PRINT "El número de usuarios con credencial es : " GROUP COUNT (*).
```

PAGE TRAILER

Este bloque especifica la información que aparece al pie de cada página del reporte.

Ejemplo:

```
PAGE TRAILER

PRINT 30 spaces, PAGENO.
```

ON LAST ROW

Al final del reporte cuando el último registro ha sido impreso, podemos ejecutar unas instrucciones de conclusión.

Ejemplo:

```
ON LAST ROW

PRINT COLUMN 30, "Número total de usuarios : " COUNT (*)
```

INSTRUCCIONES

Las instrucciones que se pueden usar dentro de la sección FORMAT de un reporte son cualquiera de las que se usan normalmente con LET, IF, FOR, WHILE además de las siguientes:

NEED exp_ent LINES

Causa que el desplegado siguiente sea impreso en la siguiente página si no quedan exp_ent líneas libres en la página actual; exp_ent es una expresión entera.

```
NEED 5 LINES
```

PAUSE [cadena]

Causa que se detenga la salida a la terminal hasta que se oprima *RETURN*.

```
PAUSE "Oprime RETURN para continuar"
```

SKIP { entero LINES | TO TOP OF PAGE }

Sirve para saltar líneas en un reporte o para saltar a la página siguiente.

```
SKIP 10 LINES
```

PRINT expresión

Sirve para mandar una línea de impresión al reporte y puede conformarse de una combinación de los siguientes elementos:

CONSTANTES.

VARIABLES.

OPERADORES ARITMETICOS.

COLUMN entero	se escribe a partir de esa columna.
número **SPACES**	se dejan número espacios en blanco.
PAGENO	escribe el número de la página actual.
cadena **CLIPPED**	escribe la cadena de carácteres sin los espacios que pudiera tener al final.
[GROUP] COUNT (*)	escribe el total de registros procesados en el grupo o reporte.
[GROUP] SUM (columna)	se escribe la suma de la columna.
[GROUP] MIN (columna)	se escribe el valor mínimo de la columna.
[GROUP] MAX (columna)	se escribe el valor máximo de la columna.
[GROUP] AVG (columna)	se escribe el valor promedio de la columna.
USING "formato"	formatea el dato. Los carácteres válidos son:

& un dígito. Llena con ceros las posiciones en blanco.

no cambia ninguna posición que tenga blanco.

< los números se justifican a la izquierda.

$ despliega signo de dinero.

( ) si el número es negativo aparece entre paréntesis. Si es positivo se imprime normalmente.

, es una literal; se despliega a menos de que no haya números a su izquierda.

***** este carácter llena con asteriscos cualquier posición que el campo sea un blanco.

Los carácteres para fechas son los mismos que los vistos en el atributo FORMAT de las formas.

Ejemplos:

Cadena de formato	valor numérico	formateado
"##,###"	12345	12,345
"**,***"	12345	12,345
	1234	*1,234
"$***,***,&&"	12345.67	$*12,345.67
	1234.56	$**1,234.56
"($$$,$$$.&&)"	-12345.67	($12,345.67)

3.2 LLAMADO AL REPORTE

Una vez construido el reporte es necesario llamarlo y enviarle la información que imprimirá. El llamado debe hacerse desde una función o desde MAIN para ejecutarlo. Hay varios pasos a seguir para esto:

1. Seleccionar la información deseada para el reporte. Debe declararse un cursor y en la parte SELECT de preferencia ordenar la información como la requiera el reporte.

2. Abrir el reporte. Esto a través de:

START REPORT nombre_rep [**TO** { **PRINTER** | "archivo" | **PIPE** "programa" }]

En donde:

nombre_rep es el nombre que se dio al reporte.

TO es opcional y es para indicar a donde debe mandarse el reporte. Si se omite esta parte entonces el reporte se enviará a donde se especifique en la sección OUTPUT del reporte en cuestión. Si no se especifica en ningún lado la dirección, el reporte saldrá en pantalla.

3. Mandar información al reporte. Hay que mandar sucesivamente cada uno de los renglones de información. Para ello podemos usar el ciclo FOREACH:

```
FOREACH cur0 INTO lista_variables.
        OUTPUT TO REPORT nombre_rep (parámetros)
END FOREACH.
```

4. Cerrar el reporte. Se hace a través de:

FINISH REPORT nombre_rep.

CAPITULO 4 EJEMPLO DE APLICACION

4.1 ANTECEDENTES

En los años que cursé la carrera de Ingeniería en Computación recurrí los fines de semana a la Biblioteca Pública Municipal "Alfonso Cravioto" perteneciente a la población de Atitalaquia, Hgo. Nunca imagine que podía contribuir en forma directa al mejoramiento y funcionamiento de esta.

Con el deseo de gratificar el espacio, y la información brindada decidí desarrollar un sistema que permitiera automatizar las actividades más comunes que se realizan en la biblioteca (registro de usuarios, consulta, apartado, etc.). Observe que gran parte de ellas son rutinarias y que realizarlas manualmente ocupa demasiado tiempo del personal. La automatización permitira realizarlas en menor tiempo y con precisión; en beneficio de los usuarios que diariamente acuden, en busca de información y entretenimiento.

En agosto de 1983 el presidente de la Republica, Lic. Miguel de la Madrid, puso en marcha el Programa Nacional de Bibliotecas Públicas, la Federación, los Gobiernos Municipales y Estatales, y los propios ciudadanos, tanto de grandes centros urbanos como de pequeñas comunidades, se comprometieron, en un esfuerzo común, a establecer bibliotecas que proporcionaran el acceso gratuito a la lectura para todos.

La Biblioteca pública esta destinada a servir a los habitantes de la comunidad en la que se encuentra ubicada. A ella acuden personas de todas las edades y de cualquier ocupacion con el propósito de aprender y superarse valiéndose de las obras que se les ofrecen.

La Biblioteca pública Municipal "Alfonso Cravioto" quedo integrada a la Red Nacional de Bibliotecas Públicas con el número 1774, el 18 de Agosto de 1988.

Inicio con acervo bibliográfico catalogado y clasificado, compuesto por 6,364 volúmenes y asi como 56 estantes sencillos de 1.90 mts., 259 charolas normales, 5 charolas inclinadas, 3 tarjeteros bibliográficos, 2 carros transporta libros, 2 bancos móvil, 6 cestos de basura, 1 escritorio, 1 paquete de señalamientos y 3 tablas periódicas.

Actualmente el número de libros es de 8,407, la comunidad, institutos públicos y privados han donado 1,257 libros. Cuenta con una computadora personal con las siguientes características:

- Computadora personal (unidad completa) modelo at 6386/25
- Arquitectura bus: isa
- Procesador: 80386 de 25 Mhz
- Memoria Principal: 640 kb
- Copresador : Intel 80387
- Unidad de disco duro 80 Mb
- Disco Flexible 3.5" 1.44 Mb
- Disco Flexible 5.25" 1.2 Mb
- Dos Puertos seriales
- Un puerto paralelo

4.2 AREAS DE LA BIBLIOTECA

El acervo de la biblioteca esta integrada por tres colecciones básicas: general, de consulta e infantil.

4.2.1 SALA GENERAL

La coleccion general es el conjunto organizado de libros que tratan sobre temas específicos, los cuales constituyen la mayor parte del acervo de la biblioteca.

La biblioteca pública municipal "Alfonso Cravioto" emplea el Sistema de Clasificación Decimal de Dewey. Este sistema de clasificación bibliográfico fue creado por el estadounidense Melvil Dewey (1851-1931) hacia el año 1875, y posteriormente ha sido desarrollado para adaptarlo a las necesidades actuales.

El sistema se llama decimal porque divide el conocimiento humano en 10 clases (000-900):

000	Generalidades	500	Ciencias puras
100	Filosofía y disciplinas afines	600	Tecnología (Ciencias aplicadas)
200	Religión	700	Bellas artes
300	Ciencias sociales	800	Literatura
400	Lenguas	900	Geografía e historia

Cada una de estas clases se fracciona a su vez en 10 subclases y cada una de éstas en diez divisiones; esto permite clasificar temas cada vez más específicos. Los tres primeros dígitos siempre son números enteros; después del punto se utilizan decimales. A medida que aumenta el número de dígitos después del punto, la clasificación es más específica.

4.2.2 SALA INFANTIL

La colección infantil es el conjunto organizado de materiales diversos, como libros de estudio, recreativos y de consulta, y revistas especialmente destinadas a niños entre los cinco y los doce años.

4.2.3 CONSULTA

La colección de consulta es el conjunto organizado de diccionarios, enciclopedias, bibliografías, manuales, directorios y otros materiales, como folletos, recortes, láminas, etc. . Todos ellos ofrecen información breve y precisa sobre las diferentes áreas del conocimiento.

4.3 SERVICIOS

Los servicios básicos que la Biblioteca Pública Municipal otorga a todos sus usuarios son préstamo con estantería abierta, préstamo a domicilio, consulta y orientación a usuarios.

4.3.1 ESTANTERIA ABIERTA

El servicio de estantería abierta consiste en proporcionar a los usuarios el libre acceso en el propio recinto, a los materiales que componen las colecciones básicas de la biblioteca.

4.3.2 CONSULTA

El servicio de consulta consiste en utilizar la colección de materiales de consulta, para responder a preguntas específicas de los usuarios.

4.3.3 ORIENTACION

El servicio de orientación a usuarios consiste en proporcionar a estos el conocimiento de los contenidos y servicios de la biblioteca, de tal manera que puedan usarla en su conjunto con habilidad.

4.3.4 PRESTAMO A DOMICILIO

El servicio de prestamo a domicilio consiste en la autorización que se otorga a los usuarios para llevar fuera de la biblioteca los libros de su interés, a fin de que puedan leerlos en el momento y lugar que deseen.

Con este servicio se amplia la posibilidad de que los libros se lean, meta de toda biblioteca, pues sabemos que a la mayor parte de los miembros de la comunidad (empleados, amas de casa, obreros, minusválidos, etc.) no les es posible permanecer en la biblioteca durante todo el tiempo que se requiera su lectura, además de que probablemente prefieran leer en un lugar distinto.

4.4 FORMAS DE PAPELERIA Y ARCHIVOS

Los archivos de préstamo a domicilio son de suma importancia, pues controlan la información que permite localizar las obras y obtener los datos para la estadística.

Cada archivo está compuesto por distintas formas de papelería impresa, las cuales cumplen una función especifica cuando se realizan las rutinas de préstamo a domicilio.

Las formas de papeleria impresa de prestamo a domicilio son:

1. *Registro*. Contiene la información que se requiere conocer sobre el usuario y su fiador. Se conserva siempre en la biblioteca.

2. *Credencial*. Acredita al usuario como lector registrado para hacer uso del servicio de préstamo a domicilio. Sólo se mantiene en la biblioteca cuando el usuario tiene libros en préstamo.

3. *Tarjeta de préstamo.* Incluye los datos básicos de la obra, el nombre del lector y la fecha de devolución. Esta tarjeta se conserva en la biblioteca cuando el libro se presta a domicilio.

4. *Papeleta de devolución.* En ella se indica al usuario la fecha en que se deberá devolver el libro. Se encuentra pegada en la guarda posterior de la obra.

5. *Papeleta de préstamo a domicilio.* En esta papeleta se anota la colocación del libro y el nombre del lector y se sella con la fecha de devolución. Se conserva junto con la credencial.

6. *Solicitud de apartado.* Si el libro que necesita el usuario se encuentra prestado, el lector llena la solicitud por duplicado; una la conserva él y la otra se archiva junto con la tarjeta del libro.

7. *Aviso de retraso.* Se envía al lector si, después de dos semanas de la fecha de vencimiento del préstamo, no ha devuelto el material. En caso de que no responda, se remite un segundo aviso (utilizando el mismo formato).

8. *Notificación al fiador.* Se envía al fiador del usuario cuando, a pesar de los avisos de retraso, no se hayan recuperado el o los libros en cuestión.

4.4.1 ARCHIVOS

Los archivos básicos del servicio de préstamo a domicilio son:

- Tarjetero de usuarios
- Tarjetero de préstamos
- Tarjetero de credenciales
- Tarjetero auxiliar

4.4.2 TARJETERO DE USUARIOS

Este tarjetero se compone de las formas de registro que ha llenado el usuario al solicitar su credencial. Las formas de registro se ordenan en el tarjetero alfabéticamente por el apellido del usuario.

4.4.3 TARJETERO DE PRESTAMO

En este tarjetero se ordenan las tarjetas de los libros prestados a domicilio por fecha de vencimiento.

4.4.4 TARJETERO DE CREDENCIALES

Este tarjetero está formado por las credenciales de los usuarios a las que se les ha adherido con un clip la papeleta de préstamo a domicilio de la obra prestada.

La credencial se mantiene en la biblioteca mientras el lector tiene libros en préstamo, cuando los devuelve, se le regresa para que él la conserve.

El ordenamiento de las credenciales en el tarjetero es alfabético, por apellido del usuario.

4.4.5 TARJETERO AUXILIAR

En este tarjetero se mantienen las tarjetas de:

- Prestamos vencidos (1er. y 2do. aviso).
- Obras en reparación o encuadernación.
- Préstamo interbibliotecario.
- Registros cancelados.

4.5 CREACION DE LA BASE DE DATOS

La base de datos se llamará BIBMUN y contendrá las siguientes tablas:

USUARIO. Esta tabla nos permite almacenar información que se requiere conocer sobre el usuario y su fiador. Además los retrasos, avisos y suspensiones a los cuales se hacen acreedores los lectores.

CREDENCIAL. En esta tabla tendremos los datos generales que acreditan al usuario como lector registrado para hacer uso del servicio de préstamo a domicilio.

LIB01. Esta tabla contendrá todos los números de adquisición, volúmenes y ejemplares de cada libro de la biblioteca. Así como el estado actual de cada obra.

LIBRO. Esta tabla nos permite almacenar los elementos más importantes del libro. Estos datos se toman principalmente de la portada y del reverso de la portada del libro.

TEMA. En esta tabla se guardaran las materias de cada libro.

PRESTA. Esta tabla incluye los datos básicos de la obra, nombre del lector, fecha de devolución, fecha de préstamo, plazo de entrega y días de atraso.

APARTA. Esta tabla contendrá los datos básicos de la obra, del lector, fecha de apartado y fecha de validez.

4.5.1 MODULO DE LA CREACION DE LA BASE DE DATOS

```
main
create database bibmun
create table usuario
( fereg date,
 feven date,
 nus char (8),
 nom_u char (30),
 edad char (2),
 dom_u char (40),
 cp_u char (5),
 tel_u char (10),
 ocup_u char (30),
 esc_t char (20),
 dir_et char (30),
 tel_et char (10),
 nom_f char (30),
 dom_f char (40),
 cp_f char (5),
 tel_f char (10),
 ocup_f char (20),
 nom_t char (20),
 dir_t char (30),
 tel_t char (10),
 ret char (2),
 avis char (2),
 susp char (2) )
```

```
create table credencial
( c_nbib smallint,
  c_nus char (8),
  c_nom_u char (30),
  c_dom_u char (40),
  c_feven date )

create table lib01
( n_tar integer,
  n_adq integer,
  vols smallint,
  ejems smallint,
  edo_lib char(1) )

create table libro
( l_ntar integer,
  l_clas char (28),
  l_aut char (60),
  l_tit char (75),
  l_edi char (2),
  l_lug char (20),
  l_edit char (30),
  l_apub char (10),
  l_isbn char (13) )

create table tema
( t_ntar integer,
  t_mat char (60) )

create table presta
(
 p_nus char (8),
 p_nomu char (30),
 p_nadq integer,
 p_clas char (28),
 p_aut char (60),
```

```
 p_tit char (75),
 p_fep date,
 p_plc smallint,
 p_fed date,
 p_atra smallint )

create table aparta
( a_nadq integer,
  a_clas char (28),
  a_aut char (60),
  a_tit char (75),
  a_nus char (8),
  a_nomu char (30),
  a_fea date,
  a_fev date )

create unique index i_l_ntar on libro (l_ntar)
create index i_l_aut on libro (l_aut)
create index i_l_tit on libro (l_tit)
create unique index i_clas on libro (l_clas)
create index i_t_ntar on tema (t_ntar)
create unique index i_nus on usuario (nus)
create unique index i_c_nus on credencial (c_nus)
create index i_ntar on lib01 (n_tar)
create unique index i_nadq on lib01 (n_adq)
create unique index inta_3 on lib01 (n_tar,n_adq)
create unique index i_p_nadq on presta (p_nadq)
create unique index i_a_nadq on aparta (a_nadq)
end main
```

4.6 FORMA DE CAPTURA PARA LA TABLA USUARIO

```
database BIBMUN
screen
{
                BIBLIOTECA PUBLICA MUNICIPAL
                    REGISTRO DE USUARIOS

  FE.REG:[f000    ]                    FE.VENC:[f001    ]
  NUMERO      NOMBRE                        EDAD
  [f002   ]      [f003                ]       [a0] AÑOS
  DOMICILIO                          C.P.[f005 ]  TELEFONO
  [f004                        ]                 [f006    ]
  OCUPACION                ESCUELA O TRABAJO [f008          ]
  [f007                ]      DIRECCION:[f009                ]
  TELEFONO [f010    ]
                    F I A D O R
  NOMBRE:[f011                ]          C.P. [f013 ]
  DOMICILIO:[f012                          ] TELEFONO:[f014    ]
  OCUPACION:[f015          ]    NOMBRE TRABAJO:[f016          ]
  DIRECCION TRABAJO:[f017                  ]  TELEFONO [f018    ]
  RETRASO:[a1]          AVISO:[a2]            SUSPENSION:[a3]
}
end
tables
usuario
attributes
f000 = usuario.fereg,format="dd/mm/yyyy",color=green,default=today,queryclear;
f001 = usuario.feven,format="dd/mm/yyyy",color=green,queryclear;
f002 = usuario.nus,autonext,upshift,picture="A#######",color=green,
comments=" Opcion (E)studiante (A)ma de Casa (P)rofesionista (T)rabajador (X) otros";
f003 = usuario.nom_u,autonext,upshift,color=green,queryclear;
a0 = usuario.edad,picture="##",color=green,autonext,queryclear;
f004 = usuario.dom_u,autonext,color=green,upshift,queryclear;
f005 = usuario.cp_u,autonext,picture="#####",color=green,queryclear;
f006 = usuario.tel_u,autonext,color=green,queryclear;
f007 = usuario.ocup_u,autonext,upshift,color=green,queryclear;
f008 = usuario.esc_t,autonext,upshift,color=green,queryclear;
f009 = usuario.dir_et,autonext,color=green,upshift,queryclear;
f010 = usuario.tel_et,autonext,color=green,upshift,queryclear;
f011 = usuario.nom_f,autonext,upshift,color=green,queryclear;
```

```
f012 = usuario.dom_f,autonext,color=green,upshift,queryclear;
f013 = usuario.cp_f,autonext,picture="#####",color=green,upshift,queryclear;
f014 = usuario.tel_f,autonext,color=green,upshift,queryclear;
f015 = usuario.ocup_f,autonext,upshift,color=green,queryclear;
f016 = usuario.nom_t,autonext,color=green,upshift,queryclear;
f017 = usuario.dir_t,autonext,color=green,upshift,queryclear;
f018 = usuario.tel_t,autonext,color=green,upshift,queryclear;
a1 = usuario.rct,autonext,picture="#A",color=green,upshift,
comments=" Opcion (1R) PRIMER RETRASO (2R) SEGUNDO RETRASO (3R) TERCER
RETRASO",queryclear;
a2 = usuario.avis,autonext,picture="A#",color=green,upshift,
comments=" Opcion (A1) PRIMER AVISO (A2) SEGUNDO AVISO ",queryclear;
a3 = usuario.susp,autonext,picture="AA",color=red,upshift,queryclear;
end
```

4.7 FORMA DE CAPTURA PARA LA TABLA CREDENCIAL

```
database BIBMUN
screen
{
                BIBLIOTECA PUBLICA MUNICIPAL
          CREDENCIAL DE PRESTAMO A DOMICILIO

                                                BIBLIOTECA N°[f000 ]
  N° USUARIO:  [f001  ]
  NOMBRE:      [f002                ]
  DOMICILIO:   [f003                      ]
  FECHA VENC: [f004    ]
}
end
tables
credencial
attributes
f000 = credencial.c_nbib,autonext,color=yellow,queryclear,default=1774;
f001 = credencial.c_nus,picture="A#######",upshift,queryclear,autonext,color=yellow;
f002 = credencial.c_nom_u,color=yellow,queryclear,upshift;
f003 = credencial.c_dom_u,color=yellow,queryclear,upshift;
f004 = credencial.c_feven,format="dd/mm/yyyy",color=yellow,queryclear;
end
```

4.8 FORMA DE CAPTURA PARA LA TABLA LIB01

```
database BIBMUN
screen
{

            BIBLIOTECA PUBLICA MUNICIPAL
        CAPTURA DE Nº ADQUSICION, VOLS. Y/O EJEMPLAR

                    NUMERO DE TARJETA: [f000     ]

    NUMERO DE ADQUISICION   VOLUMEN    EJEMPLAR
    [f001     ]             [f002 ]     [f003 ]

    SITUACION ACTUAL LIBRO: [a]
}
end
tables
lib01
attributes
f000 = lib01.n_tar,autonext,queryclear,color=green;
f001 = lib01.n_adq,autonext,queryclear,color=green;
f002 = lib01.vols,autonext,queryclear,color=green,default=0;
f003 = lib01.ejems,autonext,queryclear,color=green,default=0;
a = lib01.edo_lib,upshift,comments=" P= Perdido R= Reparación E= Encuadernación M= Mutilado
O= Optimo",
include=("P","R","E","M","O"),default="O",queryclear,color=green;
end
instructions
screen record sc_lib01 (lib01.n_adq THRU lib01.edo_lib)
end
```

4.9 ARREGLO DE PANTALLA PARA LAS TABLAS LIBRO Y TEMA

```
database BIBMUN
screen
{
                BIBLIOTECA PUBLICA MUNICIPAL
                    REGISTRO DE LIBROS

 No TARJETA: [f000     ] CLASIFICACION
                          [f001              ]
 AUTOR:[f002                                 ]
 TITULO
 [f003                                       ]
 EDICION  LUGAR            EDITORIAL              AÑO PUB.
 [a0]     [f004          ] [f005               ]  [f006    ]
 ISBN [f007      ]   M A T E R I A S
          [f008                              ]
          [f008                              ]
}
end
tables
libro
tema
attributes
f000 = libro.l_ntar,autonext,upshift,color=cyan;
f001 = libro.l_clas,autonext,upshift,color=cyan;
f002 = libro.l_aut,autonext,upshift,color=cyan;
f003 = libro.l_tit,autonext,upshift,color=cyan;
a0 = libro.l_edi,autonext,upshift,color=cyan;
f004 = libro.l_lug,autonext,upshift,color=cyan;
f005 = libro.l_edit,autonext,upshift,color=cyan;
f006 = libro.l_apub,autonext,upshift,color=cyan;
f007 = libro.l_isbn,autonext,upshift,color=cyan;
f008 = tema.t_mat,autonext,upshift,color=cyan;
end
instructions
screen record sc_lib (libro.l_ntar THRU libro.l_isbn)
screen record ps_mat[2] (t_mat)
end
```

4.10 FORMA DE CAPTURA PARA LA TABLA PRESTA

```
database BIBMUN
screen
{

                BIBLIOTECA PUBLICA MUNICIPAL
                  PRESTAMO A DOMICILIO

 Nº USUARIO [f000   ]          NOMBRE [f001                    ]

 Nº ADQUISICION [f002     ]  CLASIFICACION
                               [f003                ]
 AUTOR [f004                              ]
 TITULO
 [f005                                    ]

 FECHA PRESTAMO    [f006    ]    PLAZO ENTREGA [a    ]

 FECHA DEVOLUCION [f007     ]    DIAS DE ATRASO [b    ]
}
end
tables
presta
attributes
f000 = presta.p_nus,autonext,upshift,picture="A#######",color=green;
f001 = presta.p_nomu,noentry,reverse,color=green;
f002 = presta.p_nadq,autonext,color=blue;
f003 = presta.p_clas,noentry,color=yellow;
f004 = presta.p_aut,noentry,color=yellow;
f005 = presta.p_tit,noentry,color=yellow;
f006 = presta.p_fep,format="dd/mm/yyyy",autonext,default=today,color=green;
a = presta.p_ple,autonext,color=green;
f007 = presta.p_fed,format="dd/mm/yyyy",autonext,color=green,noentry;
b = presta.p_atra,noentry,color=red;
end
```

4.11 FORMA DE CAPTURA PARA LA TABLA APARTA

```
database BIBMUN
screen
{
                BIBLIOTECA PUBLICA MUNICIPAL
                  APARTADO DE LIBROS


 N° ADQUISICION [f000     ]    CLASIFICACION
                               [f001              ]

AUTOR  [f002                              ]
TITULO
[f003                                     ]

   N° USUARIO [f004   ]          NOMBRE [f005                ]

   FECHA DE APARTADO [f006    ] FECHA DE VALIDEZ  [f007    ]
}
end
tables
aparta
attributes
f000 = aparta.a_nadq,autonext,queryclear,color=green;
f001 = aparta.a_clas,autonext,queryclear,color=green,noentry;
f002 = aparta.a_aut,autonext,queryclear,color=green,noentry;
f003 = aparta.a_tit,autonext,queryclear,color=green,noentry;
f004 = aparta.a_nus,autonext,queryclear,color=green,upshift,picture="A#######";
f005 = aparta.a_nomu,autonext,queryclear,color=green,noentry;
f006 = aparta.a_fea,autonext,queryclear,color=green,format="dd/mm/yyyy",default=today;
f007 = aparta.a_fev,autonext,queryclear,color=green,format="dd/mm/yyyy";
end
```

4.11 FORMA DE CAPTURA PARA LA TABLA APARTA

```
database BIBMUN
screen
{
                BIBLIOTECA PUBLICA MUNICIPAL
                  APARTADO DE LIBROS


 Nº ADQUISICION [f000     ]     CLASIFICACION
                                [f001              ]

AUTOR  [f002                              ]
TITULO
[f003                                     ]

   Nº USUARIO [f004   ]              NOMBRE [f005                ]

   FECHA DE APARTADO [f006    ] FECHA DE VALIDEZ  [f007    ]
}
end
tables
aparta
attributes
f000 = aparta.a_nadq,autonext,queryclear,color=green;
f001 = aparta.a_clas,autonext,queryclear,color=green,noentry;
f002 = aparta.a_aut,autonext,queryclear,color=green,noentry;
f003 = aparta.a_tit,autonext,queryclear,color=green,noentry;
f004 = aparta.a_nus,autonext,queryclear,color=green,upshift,picture="A#######";
f005 = aparta.a_nomu,autonext,queryclear,color=green,noentry;
f006 = aparta.a_fea,autonext,queryclear,color=green,format="dd/mm/yyyy",default=today;
f007 = aparta.a_fev,autonext,queryclear,color=green,format="dd/mm/yyyy";
end
```

4.12 MODULO DE PRESTAMO A DOMICILIO


```
Database BIBMUN
 Define
    p_presta record like presta.*,
    p_lib01 record like lib01.*,
    p_libro record like libro.*,
    p_credencial record like credencial.*,
    p_usuario record like usuario.*,
    s_n char (1),
    apunta array [100] of integer,
    napunta integer
 main
    define cont integer
    call startlog ("bibpub.err")
    defer interrupt
    whenever warning continue
    whenever error call _error
    options prompt line 24,
          message line 24,
          comment line 24,
          error line 24,
          form line 4,
          accept key esc
     open form panpre from "panpre"
    display form panpre
    call en_cab(0)
    initialize p_presta.* to null
    let napunta=0
    let cont=0
    menu " PRESTAMO "
          command "Préstamo" "Préstamo de libros"
                call en_cab(1)
                call alta()
                let napunta=0
                let cont=0
```

```
    call en_cab(0)
command "Devolución" "Devolución de libros"
    call en_cab(1)
    if cont=0 then
        call anuncio ("NO HAY REGISTROS")
    else
      let cont=borrar(cont)
    end if
    call en_cab(0)
command "Consulta" "Buscar libros prestados"
    call en_cab(2)
    let napunta=0
    let cont=buscar()
    call en_cab(0)
command "Fin" "Salir del Programa"
    exit menu
command "Inicial" "Primer registro de la Ultima consulta"
    call en_cab(0)
    if cont=0 then
        call anuncio ("NO HAY REGISTROS")
      else
        let cont=1
        call despliega(cont)
    end if
    call en_cab(0)
command "Renovación" "Renovación de prestamo"
     call en_cab(1)
        call modif()
     let napunta=0
     let cont=0
    call en_cab(0)
command "Anterior" "Da el registro anterior de la ultima consulta"
    call en_cab(0)
    if cont=0 then
        call anuncio("NO HAY REGISTROS")
      else if cont=1 then
```

```
                call anuncio("Primer registro")
                 else
                 let cont= cont-1
                 call despliega(cont)
               end if
          end if
     command "Siguiente" "Da el siguiente registro de la ultima consulta"
          call en_cab (0)
          if cont=0 then
            call anuncio("NO HAY REGISTROS")
            else if cont=napunta then
                  call anuncio ("Ultimo registro")
                 else
                 let cont=cont+1
                 call despliega(cont)
               end if
          end if
          call en_cab(0)
     command key(B) "rediBujar" "Volver a dibujar toda la pantalla"
          message ""
          clear screen
          display form panpre
          if (cont>0) then
                call despliega(cont)
          end if
          call en_cab(0)
     command "Ultimo" "Ultimo registro de la ultima consulta"
          call en_cab(0)
          if cont=0 then
            call anuncio("NO HAY REGISTROS")
          else
            let cont=napunta
            call despliega(cont)
          end if
end menu
close form panpre
```

```
    clear screen
    update statistics for table presta
    close database
    exit program 0
  end main
  Function alta()
    clear form
    initialize p_presta.* to null
input by name p_presta.*
  after field p_nus
if p_presta.p_nus is null then
   call anuncio ("Anote número de usuario")
   next field p_nus else
   select c_nus,c_nom_u,c_feven into p_credencial.c_nus,
     p_credencial.c_nom_u,p_credencial.c_feven
     from credencial
     where credencial.c_nus=p_presta.p_nus
   if status=notfound
     then call anuncio ("USUARIO NO TIENE CREDENCIAL DE PRESTAMO")
     clear form
     return
      else
     display p_credencial.c_nom_u to p_nomu
     let p_presta.p_nomu=p_credencial.c_nom_u
       select nus,susp into p_usuario.nus,p_usuario.susp
          from usuario
       where usuario.nus=p_presta.p_nus
     if p_usuario.susp="SP" or p_usuario.susp="ST" THEN
     display "Clave de cancelación ",p_usuario.susp at 10,20 attribute (reverse,magenta)
     sleep 3
     display "" at 10,20
     call anuncio ("U S U A R I O  S U S P E N D I D O -----")
     clear form
     return
     end if
    if p_credencial.c_feven <= today then
```

```
    display "Fecha de vencimiento: ",p_credencial.c_feven at 10,20 attribute (reverse,red)
      sleep 3
      display "" at 10,20
      call anuncio ("Usuario no tiene credencial vigente")
      clear form
      return
     end if
   next field p_nadq
  end if
end if
 after field p_nadq
  select unique n_tar, n_adq,edo_lib,l_ntar,l_clas,l_aut,l_tit into
    p_lib01.n_tar,p_lib01.n_adq,p_lib01.edo_lib,p_libro.l_ntar,
    p_libro.l_clas,p_libro.l_aut,p_libro.l_tit
     from lib01,libro
    where lib01.n_tar=libro.l_ntar
     and lib01.n_adq=p_presta.p_nadq
      if status=notfound then
            call anuncio ("Número de adquisición no existe")
           clear form
           return
         else
          select p_nadq from presta
          where p_nadq=p_presta.p_nadq
           if status=0 then
             call anuncio ("L I B R O  P R E S T A D O")
             clear form
             return
           end if
       if p_lib01.edo_lib="P" OR p_lib01.edo_lib="R"
       OR p_lib01.edo_lib="E" then
       display "Estado actual del libro : ",p_lib01.edo_lib at 10,20 attribute (reverse,yellow)
       sleep 3
       display "" at 10,20
       call anuncio ("Libro (P)erdido, Libro en (R)eparación, Libro en (E)ncuadernación")
       clear form
```

```
        return
        end if
      display p_libro.l_clas,p_libro.l_aut,p_libro.l_tit to p_clas,p_aut,p_tit
      let p_presta.p_clas=p_libro.l_clas
      let p_presta.p_aut=p_libro.l_aut
      let p_presta.p_tit=p_libro.l_tit
      next field p_fep
      end if
    after field p_ple
    let p_presta.p_fed= p_presta.p_fep + p_presta.p_ple
    display p_presta.p_fed to p_fed
    let p_presta.p_atra= today - (p_presta.p_fep + p_presta.p_ple)
    display p_presta.p_atra to p_atra
end input
  if int_flag=0 then
      insert into presta values (p_presta.*)
      call anuncio ("PRESTAMO DE LIBRO REALIZADO")
  end if
end function
Function borrar (cont)
  define cont integer, i integer
  let s_n="N"
  prompt "SE CANCELA PRESTAMO, SEGURO (S/N)? -> " for s_n
  if int_flag= 0 then
     if s_n= "S" or s_n= "s" then
        delete from presta where rowid=apunta[cont]
        if status=0 then
             let napunta=napunta-1
           for i= cont to napunta
            let apunta[i]=apunta[i+1]
           end for
           call anuncio("PRESTAMO CANCELADO")
           if napunta= 0 then
            clear form
            return 0
           end if
```

```
            if cont > napunta then
              let cont=napunta
            end if
            call despliega(cont)
          else
            call anuncio ("NO SE PUEDE BORRAR EL REGISTRO")
          end if
        end if
      end if
      return cont
end function
function modif ()
      clear form
      input by name p_presta.p_nadq
       after field p_nadq
        if p_presta.p_nadq is null then
         call anuncio ("Por Favor anote número de Adquisición")
         next field p_nadq else
         select * into p_presta.* from presta
          where p_nadq=p_presta.p_nadq
          if status=notfound then
           call anuncio ("Número de Adquisición no existe")
           clear form
           return
            else
           select a_nadq from aparta where a_nadq=p_presta.p_nadq
            if status=0 then
            call anuncio ("L I B R O  A P A R T A D O")
            clear form
            return
            end if
          end if
        end if
      end input
  if int_flag=0 then
input by name p_presta.p_nus,p_presta.p_nomu,
```

```
    p_presta.p_clas,p_presta.p_aut,
    p_presta.p_tit,p_presta.p_fep,
    p_presta.p_ple,p_presta.p_fed,
    p_presta.p_atra without defaults
    after field p_ple
     let p_presta.p_fed= p_presta.p_fep + p_presta.p_ple
     display p_presta.p_fed to p_fed
     let p_presta.p_atra= today - (p_presta.p_fep + p_presta.p_ple)
     display p_presta.p_atra to p_atra
  end input
end if
      if int_flag=0 then
       update presta set presta. * =p_presta.* where p_nadq=p_presta.p_nadq
       if status= 0 then
         call anuncio("RENOVACION DE PRESTAMO")
       else
         call anuncio("NO SE PUEDE MODIFICAR EL REGISTRO")
       end if
      end if
end function
function buscar()
      define linea char (100), aux char (100)
      clear form
      initialize p_presta.* to null
      construct by name aux on presta.*
      if int_flag <> 0 then
        return 0
      end if
      let linea="select presta.rowid from presta where ", aux clipped
      prepare s1 from linea
      declare cur cursor for s1
      let napunta =1
      foreach cur into apunta[napunta]
           if napunta >100 then
             call anuncio("DEMASIADOS ELEMENTOS EN LA BUSQUEDA")
             exit foreach
```

```
            end if
            let napunta = napunta +1
        end foreach
        let napunta=napunta - 1
        if napunta>0 then
          call despliega(1)
          return 1
        end if
        call anuncio("NO HAY REGISTROS")
        return 0
end function
Function despliega(cont)
        define cont smallint
        clear form
        initialize p_presta.* to null
        select * into p_presta.* from presta where rowid=apunta[cont]
        if status=0 then
          let p_presta.p_fed = p_presta.p_fep + p_presta.p_ple
          let p_presta.p_atra = today - (p_presta.p_fep + p_presta.p_ple)
          display by name p_presta.*
          message "REGISTRO : ", cont using "####", "/",napunta using"<<<<"
          sleep 2
          message " "
        else
          call anuncio("NO SE PUEDE LEER UN REGISTRO")
        end if
end function
Function anuncio(linea)
        define linea char(79)
        message linea clipped attribute(reverse,cyan)
        sleep 2
        message " "
end function
Function en_cab (val)
        define linea char (79), val smallint
        let int_flag=0
```

```
        if val=0 then
          let linea=fecha()
          let linea=linea[1,70], time clipped
        else if val=1 then
          let linea="TECLAS: [ESC] PARA EJECUTAR; [CTRL-C] PARA ABORTAR"
           else
          let linea="TECLAS: [ESC] PARA EJECUTAR; [CTRL-C] PARA ABORTAR"
           end if
        end if
        display linea at 3,1 attribute(reverse,cyan)
end function
Function fecha ()
        define linea char (79)
        let linea = day(today) using "##", " DE ", mes(today)," DE ",year(today)
        return linea
end function
Function mes (dia)
        define dia date
        case month (dia)
           when 1 return "ENERO" exit case
           when 2 return "FEBRERO" exit case
           when 3 return "MARZO" exit case
           when 4 return "ABRIL" exit case
           when 5 return "MAYO" exit case
           when 6 return "JUNIO" exit case
           when 7 return "JULIO" exit case
           when 8 return "AGOSTO" exit case
           when 9 return "SEPTIEMBRE" exit case
           when 10 return "OCTUBRE" exit case
           when 11 return "NOVIEMBRE" exit case
           when 12 return "DICIEMBRE" exit case
        end case
        return " "
end function
Function _error ()
        call anuncio ("ERROR : AVISE A SU ADMINISTRADOR")
```

```
        exit program
end function
```

4.13 MODULO DE REGISTRO DE LIBROS

```
Database BIBMUN
 Define
    p_lib record like libro.*,
    ap_mat array[10] of record
    t_mat like tema.t_mat
    end record,
    i smallint,
    cont smallint,
    s_n char (1),
    cuenta smallint,
    pa_curr smallint,
    apunta array [100] of integer,
    napunta integer
main
    define cont integer
    call startlog ("bibpub.err")
    defer interrupt
    whenever warning continue
    whenever error call _error
    options prompt line 24,
          message line 24,
          comment line 24,
          error line 24,
          form line 4,
          accept key esc
    open form caplib from "caplib"
    display form caplib
    call en_cab (0)
    initialize p_lib.* to null
    let napunta=0
    let cont=0
```

```
menu "REGISTRO DE LIBROS"
    command "Alta" "Alta de Libros"
        call en_cab (1)
        call alta ()
        let napunta=0
        let cont=0
        call en_cab (0)
     command "Baja"
        call en_cab (1)
        call borrar ()
        call en_cab (0)
    command "Consulta" "Consulta de libros"
        call en_cab (2)
        let napunta=0
        let cont=buscar()
        call en_cab (0)
     command "Inicial" "Primer registro de la Ultima consulta"
        call en_cab (0)
        if cont=0 then
            call anuncio ("NO HAY REGISTROS")
          else
           let cont=1
           call despliega(cont)
           call limpia()
           call dis_mat ()
        end if
        call en_cab(0)
    command "Previo" "Da el registro anterior de la ultima consulta"
        call en_cab(0)
        if cont=0 then
            call anuncio("NO HAY REGISTROS")
          else  if cont=1 then
              call anuncio("Primer registro")
              else
              let cont= cont-1
              call despliega(cont)
```

```
                call limpia ()
                call dis_mat ()
              end if
          end if
    command "Siguiente" "Da el siguiente registro de la ultima consulta"
          call en_cab (0)
          if cont=0 then
            call anuncio ("NO HAY REGISTROS")
            else if cont=napunta then
                  call anuncio ("Ultimo registro")
                 else
                  let cont=cont+1
                  call despliega(cont)
                  call limpia()
                  call dis_mat ()
                end if
          end if
          call en_cab(0)
    command "Ultimo" "Ultimo registro de la ultima consulta"
          call en_cab(0)
          if cont=0 then
            call anuncio ("NO HAY REGISTROS")
          else
            let cont=napunta
            call despliega(cont)
            call limpia ()
            call dis_mat ()
          end if
    command "Modificación" "Actualización de Libros"
          call en_cab(1)
          call modif()
          let napunta=0
          let cont=0
          call en_cab(0)
    command "Fin" "Salir del Programa"
          exit menu
```

```
    end menu
    close form caplib
    clear screen
    update statistics for table libro
    close database
    exit program 0
end main
Function alta()
 clear form
  initialize p_lib.* to null
  input p_lib.l_ntar THRU p_lib.l_isbn
    from sc_lib.*
   after field l_ntar
    if p_lib.l_ntar is null then
     call anuncio ("Anote Por Favor Número de Tarjeta")
     next field l_ntar
     else
     select unique n_tar from lib01
      where lib01.n_tar=p_lib.l_ntar
      if status=notfound then
       call anuncio ("Número de Tarjeta no existe")
       clear form
       return
       else
        select l_ntar from libro
         where l_ntar=p_lib.l_ntar
          if status=0 then
           call anuncio ("Número de Tarjeta registrado")
           next field l_ntar
           end if
      end if
    end if
  end input
 if int_flag=0 then
  insert into libro values (p_lib.*)
  call al_mat ()
```

```
      call anuncio ("L I B R O  R E G I S T R A D O")
    end if
end function
function al_mat ()
 input array ap_mat from ps_mat.*
 let cuenta=arr_count ()
  if int_flag=0 then
   for i=1 to cuenta
    insert into tema values (p_lib.l_ntar,ap_mat[i].t_mat)
   end for
  end if
end function
function buscar()
        define linea char(100), aux char(100)
        call limpia ()
        clear form
        call lim_mat ()
        initialize p_lib.* to null
        construct by name aux on libro.*
        if int_flag <> 0 then
          return 0
        end if
        let linea=" select libro.rowid from libro where ", aux clipped
        prepare s1 from linea
        declare cur cursor for s1
        let napunta =1
        foreach cur into apunta[napunta]
             if napunta >100 then
               call anuncio ("DEMASIADOS ELEMENTOS EN LA BUSQUEDA")
               exit foreach
             end if
            let napunta = napunta +1
        end foreach
        let napunta=napunta - 1
        if napunta>0 then
          call despliega(1)
```

```
            call dis_mat ()
            return 1
          end if
          call anuncio("NO HAY REGISTROS")
          return 0
 end function
  function dis_mat()
 call lim_mat()
declare cur1 cursor for
 select t_mat from tema
      where tema.t_ntar=p_lib.l_ntar
  let cuenta = 1
foreach cur1 into ap_mat[cuenta].*
         let cuenta = cuenta+1
end foreach
call set_count(cuenta)
message "Oprima [Esc] cuando termine de consultar la información"
display array ap_mat to ps_mat.*
end function
function limpia()
define i smallint
for i = 1 to 10
   initialize ap_mat[i].* to null
end for
end function
function lim_mat()
define i smallint
for i = 1 to 2
   clear ps_mat[i].*
end for
end function
function modif()
 clear form
 input by name p_lib.l_ntar
  after field l_ntar
   if p_lib.l_ntar is null then
```

```
 call anuncio ("Por Favor anote Número de Adquisición")
 next field l_ntar else
  select * into p_lib.* from libro
   where libro.l_ntar=p_lib.l_ntar
   if status=notfound then
    call anuncio ("Número de Tarjeta no existe")
     clear form
     return
    end if
   end if
 end input
 if int_flag=0 then
  input by name p_lib.l_clas,p_lib.l_aut,
  p_lib.l_tit,p_lib.l_edi,p_lib.l_lug,
  p_lib.l_edit,p_lib.l_apub,p_lib.l_isbn
  without defaults
 end if
 if int_flag=0 then
  update libro set * =p_lib.* where l_ntar=p_lib.l_ntar
  call anuncio ("Elementos principales actualizados")
  prompt "Desea Actualizar los temas (s/n) -> " for char s_n
  if s_n="s" or s_n="S"
   then call mo_mat ()
   else
   clear form
   return
  end if
 end if
end function
function mo_mat ()
call dis_mat1 ()
call bor_mat ()
call set_count (cuenta)
input array ap_mat without defaults from ps_mat.*
 let pa_curr=arr_curr ()
 for i=(pa_curr+1) to cuenta
```

```
 initialize ap_mat[cuenta].* to null
 end for
 let cuenta=pa_curr
 if int_flag=0 then
  for i=1 to cuenta
  insert into tema values (p_lib.l_ntar,ap_mat[i].t_mat)
 end for
 call anuncio ("Materias modificadas")
 end if
 end function
function dis_mat1 ()
 declare cur0 cursor for
  select t_mat from tema
   where tema.t_ntar=p_lib.l_ntar
   let cuenta=1
   foreach cur0 into ap_mat[cuenta].*
   let cuenta=cuenta+1
   initialize ap_mat[cuenta].* to null
   end foreach
   let cuenta=cuenta-1
 end function
 function bor_mat ()
 delete from tema
  where t_ntar=p_lib.l_ntar
 end function
Function despliega(cont)
      define cont smallint
      clear form
      initialize p_lib.* to null
      select * into p_lib.* from libro where rowid=apunta[cont]
      if status=0 then
        display by name p_lib.*
        message "REGISTRO : ", cont using "####", "/",napunta using"<<<<"
        sleep 2
        message " "
       else
```

```
        call anuncio ("NO SE PUEDE LEER UN REGISTRO")
      end if
end function
Function borrar()
 clear form
 while (true)
  input by name p_lib.l_ntar
  if int_flag=0 then
   select * into p_lib.*
    from libro
     where l_ntar=p_lib.l_ntar
     if status=notfound then
     call anuncio ("El número de Tarjeta no existe")
     continue while
     end if
    display by name p_lib.*
    Prompt "Dar de baja el Número de Tarjeta (s/n) ?  " for char s_n
    if s_n matches "[sS]"
     then
     delete from libro where l_ntar=p_lib.l_ntar
     delete from tema where t_ntar=p_lib.l_ntar
     call anuncio ("Número de Tarjeta dado de baja")
     else
     call anuncio ("Número de Tarjeta no se dio de baja")
    end if
    else
    exit while
   end if
   clear form
   return
end while
 end function
Function anuncio(linea)
        define linea char(79)
        message linea clipped attribute(reverse,cyan)
        sleep 2
```

```
        message " "
end function
Function en_cab(val)
        define linea char(79), val smallint
        let int_flag=0
        if val=0 then
          let linea=fecha()
          let linea=linea[1,70], time clipped
        else if val=1 then
          let linea="TECLAS: [ESC] PARA EJECUTAR; [CTRL-C] PARA ABORTAR"
           else
          let linea="TECLAS: [ESC] PARA EJECUTAR; [CTRL-C] PARA ABORTAR"
           end if
        end if
        display linea at 3,1 attribute(reverse,cyan)
end function
Function fecha ()
        define linea char (79)
        let linea = day(today) using "##", " DE ", mes(today)," DE ",year(today)
        return linea
end function
Function mes(dia)
        define dia date
        case month (dia)
           when 1 return "ENERO" exit case
           when 2 return "FEBRERO" exit case
           when 3 return "MARZO" exit case
           when 4 return "ABRIL" exit case
           when 5 return "MAYO" exit case
           when 6 return "JUNIO" exit case
           when 7 return "JULIO" exit case
           when 8 return "AGOSTO" exit case
           when 9 return "SEPTIEMBRE" exit case
           when 10 return "OCTUBRE" exit case
           when 11 return "NOVIEMBRE" exit case
           when 12 return "DICIEMBRE" exit case
```

```
        end case
        return " "
end function
Function _error()
        call anuncio ("ERROR : AVISE A SU ADMINISTRADOR")
        exit program
end function
```

4.14 MODULO DE REGISTRO DE USUARIOS

```
Database BIBMUN
 Define
    p_usuario record like usuario.*,
    s_n char (1),
    apunta array [100] of integer,
    napunta integer
 main
    define cont integer
    call startlog ("bibpub.err")
    defer interrupt
    whenever warning continue
    whenever error call _error
    options prompt line 24,
          message line 24,
          comment line 24,
          error line 24,
          form line 4,
          accept key esc
    open form panusua from "panusua"
    display form panusua
    call en_cab(0)
    initialize p_usuario.* to null
    let napunta=0
    let cont=0
    menu "USUARIOS"
```

```
command "Alta" "Alta de Usuarios"
    call en_cab(1)
    call alta()
    let napunta=0
    let cont=0
    call en_cab(0)
command "Baja" "Borrar el registro activo"
    call en_cab(1)
    if cont=0 then
        call anuncio ("NO HAY REGISTROS")
    else
      let cont=borrar(cont)
    end if
    call en_cab(0)
command "Consultar" "Consultar usuario"
    call en_cab(2)
    let napunta=0
    let cont=buscar()
    call en_cab(0)
command "Inicial" "Primer registro de la Ultima consulta"
    call en_cab(0)
    if cont=0 then
        call anuncio ("NO HAY REGISTROS")
      else
        let cont=1
        call despliega(cont)
    end if
    call en_cab(0)
command "Modificacion" "Actualización de Datos del Usuario"
    call en_cab(1)
        call modif()
        let napunta= 0
        let cont= 0
    call en_cab(0)
command "Previo" "Da el registro anterior de la ultima consulta"
    call en_cab(0)
```

```
    if cont=0 then
          call anuncio ("NO HAY REGISTROS")
      else  if cont=1 then
           call anuncio("Primer registro")
           else
           let cont= cont-1
           call despliega(cont)
          end if
     end if
command "Siguiente" "Da el siguiente registro de la ultima consulta"
    call en_cab (0)
    if cont=0 then
      call anuncio ("NO HAY REGISTROS")
      else if cont=napunta then
           call anuncio ("Ultimo registro")
          else
           let cont=cont+1
           call despliega(cont)
         end if
    end if
    call en_cab(0)
command "Redibujar" "Volver a dibujar toda la pantalla"
    message ""
    clear screen
    display form panusua
    if (cont>0) then
          call despliega(cont)
    end if
    call en_cab(0)
command "Ultimo" "Ultimo registro de la ultima consulta"
    call en_cab(0)
    if cont=0 then
      call anuncio("NO HAY REGISTROS")
    else
      let cont=napunta
      call despliega(cont)
```

```
            end if
          command "Fin" "Salir del Programa"
            exit menu
    end menu
    close form panusua
    clear screen
    update statistics for table usuario
    close database
    exit program 0
end main
Function alta()
   clear form
   initialize p_usuario.* to null
 input by name p_usuario.*
  after field fereg
  let p_usuario.feven=p_usuario.fereg + 730
  display p_usuario.feven to feven
  next field nus
  before field nus
   open window v1 at 12,20
   with 1 rows, 42 columns
   attribute (reverse)
   message " Use AMMYY### Donde: A= Tipo de Usuario  "
   sleep 2
   message " MM= Mes de registro YY= Afio de registro "
   sleep 2
   message " ###= Consecutivo    "
   sleep 2
   close window v1
  after field nus
     if p_usuario.nus is null then
       call anuncio ("Por Favor anote número de Usuario")
         next field nus  else
    select nus from usuario where nus=p_usuario.nus
       if status=0 then
           call anuncio ("Número de Usuario duplicado")
```

```
            next field nus
          else
            if p_usuario.nus [1,1] <>"E" and p_usuario.nus [1,1] <>"P" and
              p_usuario.nus [1,1] <>"A" and p_usuario.nus [1,1] <>"X" and
              p_usuario.nus [1,1] <>"T" then
            call anuncio ("El Primer carácter no corresponde al Tipo de usuario")
            next field nus
            end if
          end if
        end if
    end input
    if int_flag=0 then
        insert into usuario values(p_usuario.*)
        call anuncio("U S U A R I O  D A D O  D E  A L T A")
    end if
end function
Function borrar(cont)
    define cont integer, i integer
    let s_n="N"
    prompt "SE CANCELA, SEGURO (S/N)? -> " for s_n
    if int_flag= 0 then
       if s_n= "S" or s_n= "s" then
          delete from usuario where rowid=apunta[cont]
          if status=0 then
              let napunta=napunta-1
            for i= cont to napunta
              let apunta[i]=apunta[i+1]
            end for
            call anuncio("USUARIO BORRADO")
            if napunta= 0 then
              clear form
              return 0
            end if
            if cont > napunta then
              let cont=napunta
            end if
```

```
            call despliega(cont)
          else
            call anuncio ("NO SE PUEDE BORRAR EL REGISTRO")
          end if
        end if
      end if
      return cont
end function
function modif ()
    clear form
    input by name p_usuario.nus
     after field nus
     if p_usuario.nus is null then
     call anuncio ("Por Favor anote Número de Usuario")
      next field nus else
     select * into p_usuario.* from usuario
        where nus=p_usuario.nus
     if status=notfound then
       call anuncio ("Número de Usuario no existe")
       clear form
       return
     end if
    end if
   end input
  if int_flag=0 then
 input by name p_usuario.fereg,
  p_usuario.feven,p_usuario.nom_u,p_usuario.edad,
  p_usuario.dom_u,p_usuario.cp_u,
  p_usuario.tel_u,p_usuario.ocup_u,
  p_usuario.esc_t,p_usuario.dir_et,
  p_usuario.tel_et,p_usuario.nom_f,
  p_usuario.dom_f,p_usuario.cp_f,
  p_usuario.tel_f,p_usuario.ocup_f,
  p_usuario.nom_t,p_usuario.dir_t,
  p_usuario.tel_t,p_usuario.ret,
  p_usuario.avis,p_usuario.susp without defaults
```

```
 end if
      if int_flag=0 then
        update usuario set usuario.* =p_usuario.* where nus=p_usuario.nus
        if status= 0 then
          call anuncio("DATOS ACTUALIZADOS")
        else
          call anuncio("NO SE PUEDE MODIFICAR EL REGISTRO")
        end if
      end if
end function
function buscar()
      define linea char(100), aux char(100)
      clear form
      initialize p_usuario.* to null
      construct by name aux on usuario.*
      if int_flag <> 0 then
        return 0
      end if
      let linea="select usuario.rowid from usuario where ", aux clipped
      prepare s1 from linea
      declare cur cursor for s1
      let napunta =1
      foreach cur into apunta[napunta]
           if napunta >100 then
             call anuncio("DEMASIADOS ELEMENTOS EN LA BUSQUEDA")
             exit foreach
           end if
          let napunta = napunta +1
      end foreach
      let napunta=napunta - 1
      if napunta>0 then
        call despliega(1)
        return 1
      end if
      call anuncio("NO HAY REGISTROS")
      return 0
```

```
end function
Function despliega(cont)
      define cont smallint
      clear form
    - initialize p_usuario.* to null
      select * into p_usuario.* from usuario where rowid=apunta[cont]
      if status=0 then
        display by name p_usuario.*
        message "REGISTRO : ", cont using "####", "/",napunta using"<<<<"
        sleep 2
        message " "
      else
        call anuncio ("NO SE PUEDE LEER UN REGISTRO")
      end if
end function
Function anuncio(linea)
      define linea char(79)
      message linea clipped attribute(reverse,cyan)
      sleep 2
      message " "
end function
Function en_cab(val)
      define linea char(79), val smallint
      let int_flag=0
      if val=0 then
        let linea=fecha()
        let linea=linea[1,70], time clipped
      else if val=1 then
        let linea="TECLAS: [ESC] PARA EJECUTAR; [CTRL-C] PARA ABORTAR"
         else
        let linea="TECLAS: [ESC] PARA EJECUTAR; [CTRL-C] PARA ABORTAR"
         end if
      end if
      display linea at 3,1 attribute(reverse,cyan)
end function
Function fecha ()
```

```
        define linea char (79)
        let linea = day(today) using "##", " DE ", mes(today)," DE ",year(today)
        return linea
end function
Function mes(dia)
        define dia date
        case month (dia)
            when 1 return "ENERO" exit case
            when 2 return "FEBRERO" exit case
            when 3 return "MARZO" exit case
            when 4 return "ABRIL" exit case
            when 5 return "MAYO" exit case
            when 6 return "JUNIO" exit case
            when 7 return "JULIO" exit case
            when 8 return "AGOSTO" exit case
            when 9 return "SEPTIEMBRE" exit case
            when 10 return "OCTUBRE" exit case
            when 11 return "NOVIEMBRE" exit case
            when 12 return "DICIEMBRE" exit case
        end case
        return " "
end function
Function _error()
        call anuncio ("ERROR : AVISE A SU ADMINISTRADOR")
        exit program
end function
```

4.15 MODULO DE ASIGNACION DE NUMEROS DE ADQUISICION, VOLUMEN Y EJEMPLAR.

```
Database BIBMUN
 Define
    p_lib01 record like lib01.*,
    s_n char (1),
    apunta array [100] of integer,
    napunta integer
 main
    define cont integer
    call startlog ("bibpub.err")
    defer interrupt
    whenever warning continue
    whenever error call _error
    options prompt line 23,
         message line 23,
         comment line 23,
         error line 23,
         form line 4,
         accept key esc
    open form captcat from "captcat"
    display form captcat
    call en_cab(0)
    initialize p_lib01.* to null
    let napunta=0
    let cont=0
    menu "Numeros Adq. Vols. y Ejem."
         command "Alta" "Alta de Numeros ..."
              call en_cab(1)
              call alta()
              let napunta=0
              let cont=0
              call en_cab(0)
         command "Baja" "Borrar registro activo"
```

```
    call en_cab(1)
    if cont=0 then
        call anuncio ("NO HAY REGISTROS")
    else
      let cont=borrar(cont)
    end if
    call en_cab(0)
command "Consulta" "Buscar una registro"
    call en_cab(2)
    let napunta=0
    let cont=buscar()
    call en_cab(0)
command "Inicial" "Primer registro de la Ultima consulta"
    call en_cab(0)
    if cont=0 then
        call anuncio ("NO HAY REGISTROS")
      else
       let cont=1
       call despliega(cont)
    end if
    call en_cab(0)
command "Previo" "Da el registro anterior de la ultima consulta".
    call en_cab(0)
    if cont=0 then
        call anuncio("NO HAY REGISTROS")
      else if cont=1 then
          call anuncio("Primer registro")
          else
          let cont= cont-1
          call despliega(cont)
        end if
    end if
command "Siguiente" "Da el siguiente registro de la ultima consulta"
    call en_cab (0)
    if cont=0 then
      call anuncio("NO HAY REGISTROS")
```

```
            else if cont=napunta then
                 call anuncio ("Ultimo registro")
                else
                 let cont=cont+1
                 call despliega(cont)
                end if
          end if
          call en_cab(0)
     command "Ultimo" "Ultimo registro de la ultima consulta"
          call en_cab(0)
          if cont=0 then
            call anuncio("NO HAY REGISTROS")
          else
            let cont=napunta
            call despliega(cont)
          end if
     command "Modificación" "Actualización de registro"
          call en_cab(1)
               call modif()
          let napunta=0
          let cont=0
          call en_cab(0)
     command "Redibujar" "Volver a dibujar toda la pantalla"
          message ""
          clear screen
          display form captcat
          if (cont>0) then
               call despliega(cont)
          end if
          call en_cab(0)
     command "Fin" "Salir del Programa"
          exit menu
end menu
close form captcat
clear screen
update statistics for table lib01
```

```
    close database
    exit program 0
end main
Function alta()
   clear form
   initialize p_lib01.* to null
 input by name p_lib01.*
   after field n_tar
     if p_lib01.n_tar is null then
      call anuncio ("Por Favor anote número de Tarjeta")
      next field n_tar
     end if
   after field n_adq
     if p_lib01.n_adq is null then
      call anuncio ("Por Favor anote número de Adquisición")
      next field n_adq
       else select n_adq from lib01 where n_adq=p_lib01.n_adq
        if status=0 then
           call anuncio ("Número de Adquisición duplicado")
           next field n_adq
        end if
     end if
 end input
  if int_flag=0 then
      insert into lib01 values(p_lib01.*)
      call anuncio("REGISTRO DADO DE ALTA")
      prompt "Desea continuar capturando números de Adquisición (s/n) ? "
            for  char s_n
       if s_n="s" or s_n="S"
       then call alta_1 ()
       end if
  end if
end function
function alta_1 ()
 let s_n="s"
 while (s_n="S" or s_n="s")
```

```
  input  p_lib01.n_adq thru p_lib01.cdo_lib
       from sc_lib01.*
  after field n_adq
     if p_lib01.n_adq is null then
      call anuncio ("Por Favor anote número de Adquisición")
      next field n_adq
       else select n_adq from lib01 where n_adq=p_lib01.n_adq
        if status=0 then
           call anuncio ("Número de Adquisición duplicado")
           next field n_adq
        end if
     end if
  end input
   if int_flag=0 then
       insert into lib01 values(p_lib01.*)
       call anuncio("REGISTRO DADO DE ALTA")
       prompt "Desea continuar capturando números de Adquisición (s/n) ? "
       for char s_n
    else exit while
   end if
  end while
end function
 Function borrar(cont)
   define cont integer, i integer
   let s_n="N"
   prompt "SE BORRA, SEGURO (S/N)? -> " for s_n
   if int_flag= 0 then
      if s_n= "S" or s_n= "s" then
         delete from lib01 where rowid=apunta[cont]
         if status=0 then
              let napunta=napunta-1
            for i= cont to napunta
              let apunta[i]=apunta[i+1]
            end for
            call anuncio("REGISTRO BORRADO")
            if napunta= 0 then
```

```
                clear form
                return 0
              end if
              if cont > napunta then
                let cont=napunta
              end if
              call despliega(cont)
            else
              call anuncio ("NO SE PUEDE BORRAR EL REGISTRO")
            end if
          end if
        end if
        return cont
      end function
    function modif()
        clear form
      input by name p_lib01.n_adq
        before field n_adq
        open window v2 at 14,20
        with 1 rows, 40 columns attribute (border)
        message "ANOTE NUMERO DE ADQUISICION A MODIFICAR"
        sleep 3
        close window v2
       after field n_adq
      if p_lib01.n_adq is null then
        call anuncio ("Por favor anote número de adquisición")
        next field n_adq else
        select * INTO p_lib01.*
         from lib01
         where n_adq=p_lib01.n_adq
        if status=notfound then
        call anuncio ("Número de adquisición no existe")
        next field n_adq
         end if
        end if
    end input
```

```
  if int_flag=0 then
    input by name p_lib01.n_tar, p_lib01.vols, p_lib01.ejems,
    p_lib01.edo_lib without defaults
    after field n_tar
      if p_lib01.n_tar is null then
       call anuncio ("Por Favor anote número de Tarjeta")
       next field n_tar
      end if
   end input
 end if
      if int_flag=0 then
        update lib01 set * =p_lib01.* where n_adq=p_lib01.n_adq
        if status= 0 then
          call anuncio("REGISTRO ACTUALIZADO")
        else
          call anuncio("NO SE PUEDE MODIFICAR EL REGISTRO")
        end if
      end if
end function
function buscar()
      define linea char(100), aux char(100)
      clear form
      initialize p_lib01.* to null
      construct by name aux on lib01.*
      if int_flag <> 0 then
        return 0
      end if
      let linea="select lib01.rowid from lib01 where ", aux clipped
      prepare s1 from linea
      declare cur cursor for s1
      let napunta =1
      foreach cur into apunta[napunta]
           if napunta >100 then
             call anuncio("DEMASIADOS ELEMENTOS EN LA BUSQUEDA")
             exit foreach
           end if
```

```
            let napunta = napunta +1
        end foreach
        let napunta=napunta - 1
        if napunta>0 then
          call despliega(1)
          return 1
        end if
        call anuncio("NO HAY REGISTROS")
        return 0
end function
Function despliega(cont)
        define cont smallint
        clear form
        initialize p_lib01.* to null ,
        select * into p_lib01.* from lib01 where rowid=apunta[cont]
        if status=0 then
          display by name p_lib01.*
          message "REGISTRO : ", cont using "####", "/",napunta using"<<<<"
          sleep 2
          message " "
         else
          call anuncio("NO SE PUEDE LEER UN REGISTRO")
        end if
end function
Function anuncio(linea)
        define linea char(79)
        message linea clipped attribute(reverse,cyan)
        sleep 2
        message " "
end function
Function en_cab(val)
        define linea char(79), val smallint
        let int_flag=0
        if val=0 then
          let linea=fecha()
          let linea=linea[1,70], time clipped
```

```
        else if val=1 then
          let linea="TECLAS: [ESC] PARA EJECUTAR; [CTRL-C] PARA ABORTAR"
           else
          let linea="TECLAS: [ESC] PARA EJECUTAR; [CTRL-C] PARA ABORTAR"
           end if
        end if
        display linea at 3,1 attribute(reverse,cyan)
end function
Function fecha ()
        define linea char (79)
        let linea = day(today) using "##", " DE ", mes(today)," DE ",year(today)
        return linea
end function
Function mes(dia)
        define dia date
        case month (dia)
           when 1 return "ENERO" exit case
           when 2 return "FEBRERO" exit case
           when 3 return "MARZO" exit case
           when 4 return "ABRIL" exit case
           when 5 return "MAYO" exit case
           when 6 return "JUNIO" exit case
           when 7 return "JULIO" exit case
           when 8 return "AGOSTO" exit case
           when 9 return "SEPTIEMBRE" exit case
           when 10 return "OCTUBRE" exit case
           when 11 return "NOVIEMBRE" exit case
           when 12 return "DICIEMBRE" exit case
        end case
        return " "
end function
Function _error()
        call anuncio("ERROR : AVISE A SU ADMINISTRADOR")
        exit program
end function
```

4.16 REPORTE DEL ESTADO ACTUAL DE LOS LIBROS

```
database bibmun
define p_libro record like libro.*
main
 define p_lib01 record like lib01.*,
 edo char (1)
 open window v2 at 10,10 with 1 rows, 37 columns attribute (border)
 prompt " Da el estado actual del libro  : " for edo
 close window v2
 declare c1 cursor for
   select * from lib01
     where edo_lib = edo
     order by n_adq
     initialize p_lib01.* to null
start report redolib to printer
 foreach c1 into p_lib01.*
 output to report redolib(p_lib01.*)
 initialize p_lib01.* to null
 end foreach
 finish report redolib
exit program 0
end main

Report redolib (p_lib01)
   define p_lib01 record like lib01.*
 output
  report to printer #"report.lst"
   left margin 1
   right margin 80
   top margin 1
   bottom margin 3
   page length 66
 format
   page header
   print column 60,"FECHA: ", today
```

```
 skip 1 line
 print column 65,pageno using "Pagina ###"
 skip 1 line
 print column 25, "BIBLIOTECA PUBLICA MUNICIPAL"
 print column 31, "ALFONSO CRAVIOTO"
 print column 31, "ATITALAQUIA, HGO"
 skip 2 lines
 print column 26, "ESTADO ACTUAL DE LOS LIBROS"
 skip 1 line
 print "No ADQUISICION", 2 spaces, "CLASIFICACION",15 spaces, "ESTADO "
 print column 3,"TITULO"
 on every row
  select l_clas, l_tit into p_libro.l_clas, p_libro.l_tit
    from libro where l_ntar=p_lib01.n_tar
 print p_lib01.n_adq,"     ",p_libro.l_clas,
     column 48, p_lib01.edo_lib
 print column 3,p_libro.l_tit clipped
 on last row
 skip 1 line
 print column 20, "TOTAL DE LIBROS : ", COUNT (*) using "#####"
end report
```

CONCLUSIONES

Hemos visto los aspectos generales del INFORMIX-4GL y la elaboración del sistema de control de una bibiloteca pública municipal. La intención de presentar la programación, es con el objetivo de que el lector interesado en aprender a utilizar herramientas de cuarta generación; incursione con mayor rapidez al INFORMIX-4GL.

El poder de la instrucción SELECT es la base para todas las consultas en INFORMIX-4GL, y el resultado de un SELECT junto con otras dos instrucciones: FROM y WHERE, puede ser usado en reportes, desplegados por pantalla y otras opciones más.

Al finalizar la lectura del trabajo de tesis, no dudo que los lectores puedan compartir mi punto de vista de que INFORMIX-4GL:

- Es un lenguaje sencillo y fácil de aprender
- Permite realizar Desarrollos rápidamente
- Facilidad para crear ventanas, menús y formas para interactuar con los datos
- Facilidad para incorporar funciones de otros lenguajes
- Utiliza SQL estándar

Es probablemente el mejor lenguaje de cuarta generación, ya que últimamente se han desarrollado nuevas investigaciones para su mejora continua (aparición de nuevas versiones).

Espero que esta tesis ayude a los lectores a considerar al INFORMIX-4GL como una alternativa para el desarrollo de sus aplicaciones. Así mismo, como material de consulta dirigido a los estudiantes de base de datos.

En lo particular, la razón principal es que se pueda contar con este tipo de herramientas para poder ser competitivos en el mercado laboral; el cuál tiende a emigrar a lenguajes de cuarta generación y manejadores de bases de datos relaciónales como el INFORMIX-4GL.

REFERENCIAS

INFORMIX SOFTWARE.INC

GUIDE TO THE INFORMIX-4GL FOR THE DOS OPERATING SYSTEM

VERSION 1.10

INFORMIX SOFTWARE.INC

REFERENCE MANUAL OF INFORMIX-4GL FOR THE DOS OPERATING SYSTEM

VERSION 1.10

INFORMIX SOFTWARE.INC

A TWENTY-MINUTE GUIDE FOR THE DOS OPERANTING SYSTEM

VERSION 1.10

CONSEJO NACIONAL PARA LA CULTURA Y LAS ARTES (MEXICO). DIRECCION GENERAL DE BIBLIOTECAS

MANUALES DE LA RED NACIONAL DE BIBLIOTECAS PUBLICAS

MAHLER

INTRODUCTION INFORMIX-4GL

ED. PRENTICE HALL