UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
A C A T L A N

CONSTRUCCION E IMPLEMENTACION DE UN LENGUAJE ICONOGRAFICO PARA EL DESARROLLO DE APLICACIONES ORIENTADO AL MANEJO DE BASES DE DATOS RELACIONALES

T E S I S P R O F E S I O N A L

QUE PARA OBTENER EL TITULO DE

LICENCIADO EN MATEMATICAS APLICADAS Y COMPUTACION

P R E S E N T A N

RUBEN GONZALEZ RAMIREZ
HECTOR RUBEN SANCHEZ PEREZ

MEXICO, D. F. 1993

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE

INTRODUCCION

Desde siempre, el hombre ha transformando su medio para generar nuevos bienes materiales que le prmitan mejorar su modo de vida. Uno de los avances conseguidos por este proceso, fue el invento de la computadora: su impacto transformó en su mayor parte, las actividades productivas que se basan en el proceso de grandes cantidades de información. A la par, toda una serie de teorías fueron surgiendo para enriquecer y ampliar sus áreas de aplicación. Por ello, se da cada vez más la necesidad de desarrollar lenguajes de programación, mediante los cuales el ser humano tenga la posibilidad de dar instrucciones y planes de trabajo a la computadora, tomando en cuenta los requerimientos de cada una de esas áreas de aplicación, que a veces resultan ser muy especializadas.

Al parecer, la tendencia de hoy día es el diseño de lenguajes de muy alto nivel con características de no-procedimiento, es decir lenguajes que permitan al usuario "decirle" a la máquina lo que debe realizar, y no cómo lo debe realizar (como comunmente se hace con los lenguajes de alto nivel).

El problema principal con que se encuentran los desarrolladores de software, es el de diseñar un ambiente apropiado en que se pueda implantar un lenguaje. Entre más sencillo y de fácil entendimiento sea, más aceptación tendrá entre los usuarios finales.

El desarrollo de una aplicación por computadora no necesariamente requiere en todo momento del conocimiento detallado de un lenguaje de programación, puesto que sólo es en la etapa de implantación cuando éste se utiliza. Realmente la solución de un problema en general, comienza cuando se define qué es lo que debe hacerse, y posteriormente cómo hacerse y con qué elementos; así que lo óptimo en computación, es construir un lenguaje que permita sólo definir el procedimiento de la aplicación, sin necesidad de escribir un código especializado.

Como se sabe, una de las etapas básicas para el diseño de un sistema de información, consiste en la elaboración de diagramas de bloque y diagramas de flujo, que permiten definir de una manera gráfica la solución del problema. Y es aquí donde surge la pregunta clave que enmarca las tendencias futuras de la ingeniería de software: ¿por qué no desarrollar un lenguaje de muy alto nivel, que permita la solución de problemas con la simple definición de un diagrama.

En su totalidad los lenguajes de muy alto nivel, también llamados *lenguajes de cuarta generación*, trabajan en relación a un concepto denominado *Base de Datos*, que es una forma de estructurar un gran volumen de información, para que su almacenamiento y acceso se realice de manera ordenada.

Existen varias formas de organizar una Base de Datos. Una de ellas es la que sigue el *Modelo Relacional*, planteado por el doctor *E. F. Codd*, y que se basa en principios matemáticos sobre la *Teoría de las Relaciones*. Este es uno de los modelos más poderosos que se tienen en términos de *Sistemas de Bases de Datos*, y que ha marcado la pauta para el origen de una gran cantidad de lenguajes que tratan de explotar sus características propias.

Pues bien, la idea fundamental de la presente tesis, es precisamente la definición e implementación de un lenguaje gráfico que permita manipular *Bases de Datos Relacionales*.

CAPITULO 1 : ANTECEDENTES

La idea de incluir la computadora en el ámbito industrial, fue impulsada por las necesidades surgidas de administrar lo que la tecnología incrementaba a una velocidad impresionante. Este hecho permitió que algunas empresas volcaran su atención hacia el diseño y fabricación de computadoras, y que otras más lo hicieran hacia el desarrollo de programas para generar aplicaciones computarizadas; todo esto, conjuntamente con las diferentes instituciones de investigación, dió un gran avance a las Ciencias de la Computación, y en consecuencia su campo de acción se acrecentó todavía más.

Los sistemas de bases de datos, vienen a apoyar al procesamiento de información, y toda su teoría es antecedente del desarrollo de múltiples aplicaciones y conceptos que hoy por hoy, conforman uno de los campos más fuertes de la ingeniería de software.

1.1 El MODELO DE BASE DE DATOS RELACIONAL

De los diferentes tipos de bases de datos, que se manejan parece ser que el modelo presentado por el doctor Codd, es el más fuerte ya que se basa en conceptos matemáticos lo suficientemente formales. Para comprender mejor el modelo de base de datos relacional, es necesario comentar acerca de lo que es un sistema de bases de datos, y sus componentes, además de dar un vistazo al álgebra relacional y sus definiciones.

1.1.1 Bases de Datos

El concepto de *Base de Datos* es uno de los más estudiados: sobre él se han desarrollado una gran cantidad de teorías que han permitido el surgimiento de nuevos campos de análisis, a la par de nuevas herramientas de software que las implementan en la solución de problemas reales.

Una *Base de Datos* [DATE] es un conjunto de archivos lógicos interrelacionados entre sí, y su acceso se realiza en forma canalizada. La manipulación de una base de datos se logra generalmente a través de un sistema, que dentro del contexto de las bases de datos se ha definido como un sistema de mantenimiento de registros basado en computadoras o bien *Sistema de Bases de Datos* (**SBD**) , y está compuesto por cuatro partes fundamentales: los datos, el hardware, el software y los usuarios. Con un sistema como estos, es posible manipular más de una base en forma simultánea.

Los datos le dan el carácter integral y compartido a la base que los contiene, el decir que una base de datos es integrada, hace referencia a que los datos no se encuentran aislados, sino perfectamente relacionados eliminando parcial o totalmente toda redundancia entre estos, y decir que es compartida significa que los datos de la base no son de acceso único de un usuario. Realmente ésta es una característica de las bases de datos: los usuarios tienen acceso a una parte de la base, incluso a la misma, realizando cada uno la

aplicación de su conveniencia, esto se logra por medio de un sistema de Administrador de Bases de Datos, al que se hace referencia en seguida.

El *hardware* son todos los dispositivos de almacenamiento externo en donde se almacena la base de datos, además del equipo central de proceso y mantenimiento.

El *software* lo constituyen varias partes, de las cuales hablaremos más adelante. Por ahora mencionaremos la más importante, que es un *Sistema Administrador de Bases de Datos* (**DBMS** abreviatura de *Data Base Manager System*), y que representa el canal de comunicación entre el usuario y el hardware.

Los usuarios se encuentran divididos en tres tipos: el *programador de aplicaciones*, que es quien escribe los programas de aplicación o explotación de la base, utilizando un lenguaje de propósito especial como COBOL o Pascal, y un sublenguaje de datos llamado **DSL** (*Data SubLanguage*); el *usuario final* que accesa la información, ya sea por medio de un programa o bien un procedimiento en línea; y por último el usuario considerado como *administrador de las bases de datos*, quien es el encargado de optimar los recursos de hardware y software que se tienen, así como de proteger el Sistema de Bases de Datos en cuanto al acceso a las bases disponibles por los demás usuarios.

El utilizar un *Sistema de Información sustentado en una Bases de Datos* ofrece ciertas ventajas importantes respecto a un *Sistema de Información*, ya que es posible reducir la redundancia e inconsistencia de los datos; la información puede compartirse, dependiendo de los requerimientos de las aplicaciones y los privilegios que el administrador de las bases de datos haya concedido a cada usuario; permite seguridad, en cuanto a que los datos estén dispuestos a accesarse por el usuario correcto; se conserva la integridad de las bases, y adicionalmente se pueden seguir ciertas normas y reglas para la utilización del sistema, que irán desde el desarrollo de aplicaciones para la explotación de las bases, hasta el procedimiento adecuado para utilizar alguna de éstas.

Además de todas las características mencionadas, existe una adicional y que por su importancia la hemos separado: se trata de la *independencia de los datos*. La mayoría de las aplicaciones actuales (programas) dependen de la estructura y tipo de acceso de los datos (archivos), esto implica que las aplicaciones sean dependientes de los datos, por lo que un cambio en estos últimos afecta de manera directa las aplicaciones.

La *independencia de los datos* se puede definir como la inmunidad de las aplicaciones a los cambios de la estructura de almacenamiento y acceso de los datos.

La construcción de un *Descriptor de Archivos*, como una herramienta de soporte para una base de datos, permite manejar la independencia de los datos, y consiste en un registro donde se encuentra la información de un determinado archivo, de tal manera que el DBMS puede accesarlo para conocer su estructura,

permitiendo así que los cambios en ella, se registren en el descriptor sin necesidad de modificar las aplicaciones.

En términos generales, la arquitectura de un DBMS se divide en tres niveles: el nivel *externo*, el nivel *conceptual* y el nivel *interno*; con subsistemas que soportan los siguientes lenguajes:

- El DDL o Lenguaje de Definición de Datos (Data Definition Language);

- El DML o Lenguaje de Manipulación de Datos (Data Manipulation Language);

- Un QL o Lenguaje de Consultas no Previstas (Query Language);

- Un Lenguaje de Desarrollo de Aplicaciones, que es un lenguaje convencional como COBOL o Pascal, en el que pueden quedar inmersos los tres lenguajes anteriores.

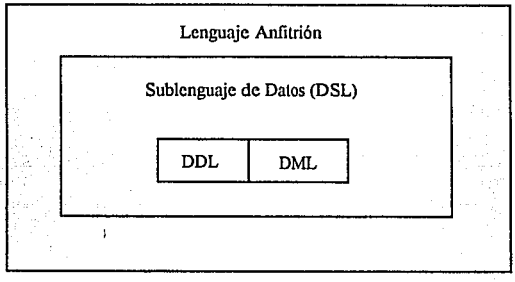


Figura 1.1 Lenguaje de usuario para un programador de aplicaciones

En primer lugar, para un usuario programador de aplicaciones, estos lenguajes son tratados con la siguiente estructura: Con un Lenguaje de Desarrollo de Aplicaciones denominado anfitrión por ejemplo COBOL (Figura 1.1), dentro de éste es declarado un sublenguaje de datos DSL, que es un subconjunto del lenguaje total, comprendiendo los procesos y operaciones de bases de datos. El DSL se compone a su vez de dos partes, un DDL y un DML que ya hemos mencionado: el DDL permite la definición o descripción de los objetos de la base de datos, y el DML ayuda al manejo y procesamiento de dichos objetos. Para un usuario final, que generalmente opera con terminales en línea, representan un lenguaje de consulta o de propósito especial, apoyado por un programa de aplicación en línea; para un usuario administrador es un caso especial, ya que pueden ser cualquiera de los dos anteriores.

En la mayoría de los casos, el DSL es muy semejante al lenguaje anfitrión, ya que en el DDL se definen los datos en forma parecida a como se hace en el lenguaje anfitrión, y en algunos casos, el DDL representa una extensión del anfitrión en la parte que corresponde a la definición de datos. La mayoría de las veces, el DML está compuesto por las llamadas a subrutinas o procedimientos que el sistema administrador de bases de datos contiene, siendo ésta la diferencia entre el DSL y el anfitrión. No obstante, entre el lenguaje anfitrión y el DSL existe un gran acoplamiento, que para un usuario representa comodidad en la programación por lo uniforme de su estructura.

Siguiendo con la arquitectura de un Sistema de Bases de Datos, el **nivel externo** lo constituye la parte donde se encuentran los usuarios, y cada uno de ellos, como se señaló con anterioridad, está restringido al acceso de la base de datos. Por eso cada usuario tiene una vista individual de la base, es decir, una vista externa.

La vista externa de un usuario está constituída por registros externos, un registro externo tiene el mismo concepto que un registro lógico, es decir, un registro externo no es precisamente un registro almacenado. El nivel externo es definido por medio de un esquema llamado esquema externo, dentro de este esquema se declaran los usuarios y su respectiva parte a la que tienen acceso.

El **nivel conceptual** es una representación del contenido total de la base de datos, donde se intenta reconocer a la base completa y no en términos de usuarios o vistas externas, sino vistas conceptuales. En este caso la vista se enfocaría a poder contemplar las bases en su totalidad. Nuevamente, la vista conceptual al igual que la vista externa, está conformada por registros, en este caso, registros conceptuales. Un registro conceptual es aquel que contempla un registro completo de toda la base de datos, la vista conceptual también es definida por medio de un esquema, ahora conocido como esquema conceptual.

El concepto de Independencia de Datos es importante mencionarlo en este momento, ya que las definiciones de los registros tanto externos como conceptuales, no deben incluir la declaración de su forma de acceso o estructura de datos, sino sólo de objetos que lo componen. En resumen, la vista del contenido de la base está definido de acuerdo al esquema que se esté manejando, independientemente si éste es conceptual o externo.

El **nivel interno** es la parte del sistema en donde son representados y declarados los registros almacenados, los archivos de índices, de qué manera se representan los campos almacenados, en qué secuencia física se encuentran los registros almacenados, etc. En este nivel se tiene una vista de las estructuras de datos, y supone que el espacio de almacenamiento es infinito, es decir, esta vista no contempla los espacios de almacenamiento reales o capacidades de registros físicos, bloques, cilindros y pistas de un disco.

La vista interna se describe por medio de un esquema denominado esquema interno.

Entre los tres niveles mencionados, hay dos niveles que asocian o relacionan a los primeros entre una vista externa específica y la vista conceptual, de manera que existe un nivel de correspondencia externo/conceptual que define la relación que hay entre una vista y otra. Por un lado, la vista externa contempla una parte de la base de datos total, por lo que también, considera una parte de la vista conceptual; y por el otro, el nivel de correspondencia identifica a cada registro externo, con su respectiva parte de registro conceptual. El otro nivel se encuentra entre la vista conceptual y la vista interna, y es donde se determina cómo los registros conceptuales se corresponden con sus contrapartes almacenados. Cabe señalar que un cambio en la estructura de la base de datos, afecta directamente el nivel conceptual/interno, por tanto, siempre debe modificarse este nivel por cada cambio en la estructura de la base, esto se debe realizar con el fin de no alterar la independencia de los datos.

Analizando más de cerca el software de un Sistema de Bases de Datos, encontramos que aparte del *Sistema Administrador de Bases de Datos*, está el procesador del lenguaje para la definición de las bases de datos físicas; un procesador del lenguaje para la definición de bases de datos lógicas; un procesador de lenguaje para la manipulación de los datos, que generalmante se encuentra dentro del compilador del lenguaje anfitrión; un procesador del lenguaje para la definición del diccionario de datos, el cual veremos posteriormente; programas de utilería para los respaldos, reinicios, estadísticas, creaciones, reorganizaciones, etc.; un generador de consultas y reportes no previstos y un sistema manejador de transacciones, utilizado para realizar transferencias de archivos que se encuentran en un código determinado a otro.

El *Sistema Administrador de Bases de Datos*, es el software que maneja todos los accesos a la base de datos, y que utiliza los esquemas y las correspondencias entre estos para dar solución a cualquier solicitud de los usuarios. Realmente el usuario administrador de bases de datos es o debe ser la única persona autorizada para tener una vista externa igual a una vista conceptual, y utilizar todo el software que se tiene para poder decidir sobre el contenido de la información de las bases de datos, así como de su estructura de almacenamiento y estrategia de acceso (organización y acceso de archivos), diseñar los esquemas interno, conceptual y externos de todas las bases con sus respectivas correspondencias, y compilar cada uno de estos para que el DBMS pueda operar sobre ellos. Asimismo, el administrador define los privilegios de acceso y procedimientos de validación de los datos, y declara los procesos a seguir para respaldar la información, ante todo controla el desempeño del sistema, respondiendo oportunamente a los cambios de requerimientos.

El administrador, también hace uso del diccionario de datos, como apoyo de información acerca de la estructura de las bases. El diccionario es propiamente una base de datos que contiene la información de todos los datos: este incluye los descriptores de archivos, referencias cruzadas de los datos contra los programas que los utilizan, y los esquemas tanto en su forma fuente como objeto. Realmente el diccionario de datos es una de las herramientas más importantes para el administrador, debido a que la información que proporciona, le ayuda a dar un mantenimiento más eficiente a las bases de datos.

1.1.2 El enfoque relacional

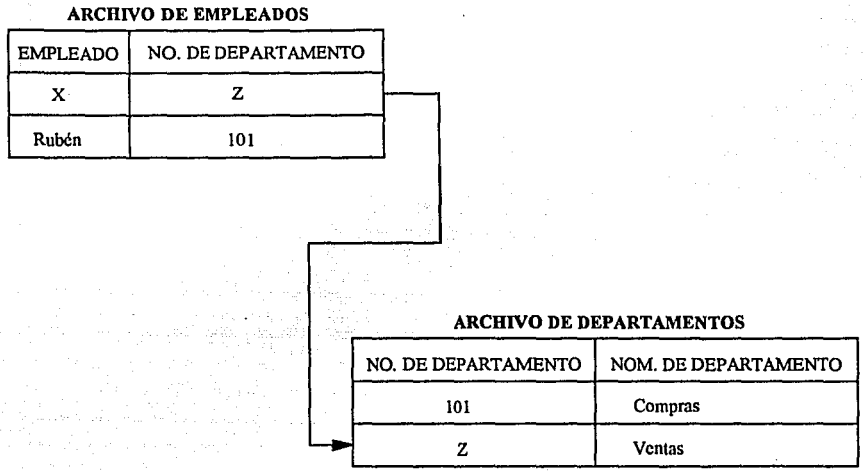
En el diseño de bases de datos se han tomado tres enfoques básicos, el relacional, el jerárquico y el de red, de los cuales nuestro estudio se basa en el primero, y los restantes pueden llegar a ser tema de otras investigaciones, ya que considerarlos saldría de los alcances del presente trabajo.

Los estudios de el doctor Edgar F. Codd sobre el tema, lo condujeron a considerar que un archivo de datos podría trabajarse como una relación , esto es, si nos adentramos en la estructura de un archivo de datos tal y como se define en computación, nos encontramos con que se trata de un conjunto de registros del mismo tipo, y cada registro es compuesto por una serie de valores, que corresponden a campos definidos por el registro. Cada campo es en sí un atributo que puede calificar a un registro completo, teniendo así la posibilidad de accesar la información contenida en los registros de un archivo cuyos campos cumplan con cierta característica definida por un atributo.

En términos de Bases de Datos Relacionales, un archivo es visto como una tabla de m renglones por n columnas, donde cada renglón corresponde a un registro y cada columna a un campo. Una tabla es en realidad un caso especial de la construcción conocida en matemáticas como relación, cuya definición es mucho más amplia y precisa que la definición tradicional de archivo.

El hecho que los archivos de datos puedan tratarse como una relación, obliga a considerar que la teoría de las relaciones puede también aplicarse a dichos archivos, resolviendo de esta manera algunos problemas prácticos que tengan que ver con el manejo de datos. Y si esto es proyectado a un nivel de base de datos, entonces tendremos una herramienta suficientemente poderosa para el diseño de un SBD con características relacionales.

Un aspecto importante de la estructura de datos relacional, es que se pueden asociar registros o tuplas de relaciones distintas mediante la correspondencia que tienen de pertenecer a un dominio común, es decir, los valores o datos de un registro pueden ligar archivos, si estos datos corresponden a un mismo *atributo* o *campo* común. Por ejemplo, supongamos que se tienen dos archivos denominados EMPLEADOS y DEPARTAMENTOS respectivamente (Figura 1.2), y en el primer archivo encontramos el campo "No. de Departamento", que indica el número de departamento al que pertenece el empleado X, y en el segundo archivo encontramos el mismo campo, pero esta vez correspondiendo a un registro con otro campo denominado "Nom. Departamento", que corresponde al nombre del departamento. En este caso, es posible asociar los registros del archivo EMPLEADOS con los registros del archivo DEPARTAMENTOS en base a un atributo común "No. de Departamento", y conocer el nombre del departamento al que pertenece un determinado empleado.



Fugura 1.2 Relación entre archivos

La uniformidad en la representación de los datos provoca asimismo una uniformidad en las operaciones con ellos, logrando operadores de alto nivel: como la información se presenta de una y sólo una forma, se requiere un sólo operador para cada una de las funciones básicas que se pueden realizar con un archivo de datos, tales como leer, borrar, insertar o modificar registros. Esto es esencial y contrasta con toda una serie de estructuras variadas y más complejas, que por consiguiente requieren de más operadores. De esta manera, podemos decir que seleccionar una estructura de datos relacional es de grán ayuda, dados su potencial y sencillez.

En la definición de un modelo de estructura de datos relacional, se presentan tres aspectos importantes:

- *Un conjunto de tipos de estructura de datos*

- *Un conjunto de operadores*

- *Un conjunto de reglas de integridad, que permiten definir el conjunto de valores de la base de datos y las operaciones permitidas entre los mismos.*

1.1.3 Estructura de datos relacional

La estructura fundamental de datos es la Relación, para la cual existen dos definiciones, que en términos matemáticos se manejan como sigue [DATE]:

1a. Definición de Relación.

Dada una serie de conjuntos D_1, D_2, ... , D_n (no necesariamente distintos) se dice que **R** es una relación sobre estos **n** conjuntos, si es un conjunto de **m** tuplas ordenadas $(d_1,d_2,...,d_n)$ tal que d_1 pertenece a D_1, d_2 pertenece a D_2,..., d_n pertenece a D_n.

2a. Definición de Relación.

Dada una serie de conjuntos D_1, D_2, ... , D_n (no por fuerza distintos), el producto cartesiano de estos **n** conjuntos, denotados por $D_1 \times D_2 \times ... \times D_n$, es el conjunto de todas las **m** tuplas ordenadas posibles $(d_1,d_2,...,d_n)$ tal que d_1 pertenece a D_1, d_2 pertenece a D_2,..., d_n pertenece a D_n.

Luego, se define **R** como una relación sobre los conjuntos D_1, D_2, ... , D_n si es un subconjunto del producto cartesiano $D_1 \times D_2 \times ... \times D_n$.

Para las definiciones anteriores, se dice que los conjuntos D_1, D_2, ... , D_n son los *dominios* de **R**, el valor **n** es el *grado* de **R**, llamándose *unarias* a las relaciones de grado uno, *binarias* a las de grado dos, ..., *n-arias* a las relaciones de grado **n**; y por último, el número de tuplas de una relación es la *cardinalidad* de una relación.

Para una relación, estrictamente no se considera que las tuplas que la componen lleven un orden, puesto que la relación es vista como un conjunto de tuplas y los conjuntos no son necesariamente ordenados. No obstante, en el contexto de bases de datos, en ocasiones es necesario que las tuplas de una relación lleven algún orden, para facilitar la consulta sobre la base.

Ahora bien, en todo momento los dominios de una relación conservan un orden definido, esto es, una relación como la habíamos mencionado, es un conjunto de tuplas ordenadas, con el k-ésimo elemento correspondiente al k-ésimo dominio, por lo cual, se intuye que el orden es único y que una relación pudiera contener la misma información que otra y no llegar a ser la misma.

Sin embargo, existe una convención que consiste en generar diferentes nombres de atributos, prefijando de antemano un dominio único para dichos nombres, con el fin de poder referirse de alguna manera, a una parte de un dominio según su significado dentro de la relación. Generalmente el nombre que se le da a un

dominio cualquiera, corresponde al atributo que asocia a ese dominio dentro de una relación. Un *dominio* es un universo de elementos que se pueden considerar, y el atributo es de alguna manera el nombre de un conjunto de valores de ese dominio (un dominio define un conjunto de valores posibles para un atributo). Supóngase, por ejemplo, que se tiene el dominio de "marcas de automóviles", y en algún momento este dominio podría tener el atributo "compactos" o "grandes", por lo que los atributos pueden indicar roles diferentes, representados por un dominio común en cada una de sus apariciones.

Los valores de los dominios deben estar normalizados. El concepto de *normalización* significa que todo valor de una relación, o bien, cada valor de atributo en cada tupla debe ser atómico (indivisible). En cada intersección renglón-columna debe existir uno y sólo un dato, y nunca un conjunto de ellos. Cabe señalar que los valores pueden llegar a ser nulos. Se considera que una relación que satisface esta condición, se encuentra normalizada. Matemáticamente una relación no normalizada es una relación cuyos dominios no son simples. Un dominio simple es aquel que tiene todos sus elementos o valores atómicos.

Según el enfoque relacional, todas las relaciones deben encontrarse normalizadas, ya que la simplificación de la estructura de datos que se contempla para tratar un archivo como una tabla, a su vez simplifica tanto los operadores como el almacenamiento físico de los archivos.

Con respecto a los valores de los atributos, se considera que uno o varios atributos específicos pueden constituir una *llave* para poder accesar los registros de un archivo, esto es con el fin de identificar cada tupla de la relación. La característica de una llave, es que debe ser compuesta por uno o varios atributos, cuyos valores permitan diferenciar cada ocurrencia.
La existencia de una llave al menos, está garantizada, puesto que una relación es un conjunto de tuplas, y debido a la caraterística de conjunto, mínimamente cada tupla es llave.

Probablemente, más de un atributo o un conjunto de estos sea candidato para ser llave. Cuando existe más de un atributo llave, se puede tomar la apropiada y las restantes se pueden considerar llaves alternas a ésta que se tomará como *principal* o *primaria*.

Una tupla representa o hace referencia a un ente del mundo real, y la llave primaria sirve a su vez para referenciar a esa tupla. Este concepto lleva a considerar ciertas reglas de integridad, que a continuación se presentan:

- Primera regla de integridad (Integridad de la entidad)

Ninguna llave primaria puede tener el valor nulo.

Esta regla indica que si alguna entidad llegase a tener el valor de su llave primaria nulo, entonces ésta perderá su identidad si alguna otra tupla tuviera esta misma característica, puesto que si entre dos o más entes no hay nada que los distinga, entonces por definición no son dos o más entidades, sino una sola.

- Segunda regla de integridad (Integridad de referencia)

Sea D *un dominio primario, sí y solo sí, existe alguna llave primaria de un sólo atributo definida sobre ese dominio, y sea* R1 *una relación con un atributo* A *que se define sobre* D. *Entonces en cualquier instante dado, cada valor de* A *en* R1 *debe ser o bien nulo, o bien igual a* v, *donde* v *es el valor de la llave primaria de alguna tupla en la relación* R2 *(*R1 *y* R2 *no por fuerza distintas) con llave primaria definida sobre* D.

Nótese que R2 debe existir, por definición de dominio primario y también la restricción se satisface de modo trivial si A es la llave primaria de R1.

Como se puede observar A en R1 nunca sera nulo, pero se debe de mencionar, porque en ningún momento se ha dicho que A es llave primaria en R1, sino como un simple atributo, por ello se dice que pudiera ser nulo o ser blanco, además a A se le denomina *llave foránea*, es decir, es un atributo en una relación que puede existir o servir como llave primaria en otra relación.

Esta segunda regla nos dice que si una relación tiene una llave primaria, entonces desde otra relación puede hacerse referencia, a alguna tupla de la primera relación si ésta contiene el mismo atributo utilizado como llave primaria, no importando si para esta segunda relación el atributo es llave o no.

De lo anterior se desprende que una relación está compuesta por dos partes, la extensión y la comprensión: La *extensión* de una relación dada es el conjunto de tuplas que aparecen en determinado momento dentro de ella; es el contenido de la relación, por lo tanto varía con el tiempo, y equivale a la vista. Por otra parte, la *compresión* de una relación especifica no depende del tiempo, pero corresponde a lo que se define en el esquema relacional; es la descripción de la relación en términos de una estructura nominadora que la constituye la propia relación y los nombres de los atributos, cada uno de ellos con su dominio asociado; y además de las restricciones estipuladas en las reglas de integridad.

Por último, los archivos que considera el enfoque relacional deben tener ciertas características para una manipulación apropiada, y ya que los archivos son considerados análogamente a las relaciones, estos deben estructurarse de manera similar, así que:

- Cada archivo sólo contiene un tipo de registro.
- Cada ocurrencia de registro en un archivo debe contener el mismo número de campos.

- Cada ocurrencia tiene identificador o llave única.

- Las ocurrencias de registro dentro de un archivo se encuentran desordenadas, o bien, ordenadas de acuerdo a una llave de orden. Esta llave de orden no necesariamente debe ser la llave primaria.

1.1.4 El álgebra relacional

El *álgebra relacional* esta definida como un conjunto de operaciones sobre las relaciones.

En un sentido estricto cada operación está constituida por un operador y uno o dos operandos que son relaciones, en términos matemáticos, las relaciones constituyen un sistema cerrado bajo el álgebra, esto es, que las operaciones darán una relación como resultado.

El álgebra relacional se compone de dos grupos de operadores, los operadores tradicionales de la teoría de conjuntos: unión, diferencia, producto cartesiano e intersección, siendo los tres primeros los operadores primitivos; y los operadores especiales: selección, proyección, junta y división; siendo los dos primeros también operadores primitivos que dan origen a las operaciones que a continuación se definen:

Definición de las operaciones tradicionales de la teoría de conjuntos:

Sean A y B relaciones, tales que tienen el mismo grado y tuplas ordenadas, donde el j-ésimo atributo de cada una de ellas corresponde al mismo dominio bajo el cual se definen las relaciones (los atributos no necesariamente con el mismo nombre), entonces se dice que A y B son compatibles.
Sean A y B relaciones compatibles, con tuplas *t*, se define:

$A \cup B = \{ t \mid t \in A \text{ o } t \in B \}$

como la *unión* entre A y B

$A \cap B = \{ t \mid t \in A \text{ y } t \in B \}$

como la *intersección* entre A y B

$A - B = \{ t \mid t \in A \text{ y } t \notin B \}$

como la *diferencia* entre A y B.
Sean A y B relaciones no necesariamente compatibles, y con *a*, *b* tuplas respectivamente de cada relación, se define:

$A \times B = \{ (a,b) \mid a \in A , b \in B \}$

como el *producto cartesiano* entre A y B

Tanto la unión, intersección, y producto cartesiano son operaciones asociativas, es decir, si A, B y C son relaciones, entonces:

$$A \cup (B \cup C) = (A \cup B) \cup C$$
$$A \cap (B \cap C) = (A \cap B) \cap C$$
$$A \times (B \times C) = (A \times B) \times C$$

Definición de las operaciones especiales:

Sean a,b atributos cualesquiera de la relación A, sea # un operador de relación (=, <, >, <=, >=, <>), y sea c una constante con valor definido sobre el dominio de a, entonces (a#b), es un predicado y (a#c) también es un predicado.

Sean *p* y *q* predicados, y sea @ un operador booleano (*Y*,*O*), entonces (*p*@*q*) es también un predicado.

Sea *p* predicado y el operador booleano *NO*, entonces (*NO p*) es también un predicado.

Dadas las definiciones anteriores, podemos concluir que un predicado es una combinación booleana de términos, donde cada término es una evaluación sobre todas las tuplas de una relación.

La operación de *selección*, se define como el subconjunto vertical de tuplas de una relación dada que cumplen un predicado específico.

La operación de *proyección*, se define como el subconjunto horizontal obtenido al escoger los atributos dados en un orden específico, de izquierda a derecha, y eliminando luego las tuplas duplicadas.

La operación de *junta* de dos relaciones A y B se define como el conjunto de tuplas t, tal que t está compuesta por la unión de dos tuplas, a y b, donde a pertenece a A y b pertenece a B. Si x1,x2,...,xn son atributos contenidos en a y en b, entonces t pertenecerá a A junta B cuando todos los valores de x1,x2,...,xn en ambas tuplas cumplan uno a uno alguna igualdad o desigualdad predefinida.

Dependiendo del criterio de selección la operación junta recibe varios nombres: si el criterio fue la igualdad, la junta es llamada *equijunta*; cuando el criterio fue un mayor que, la junta es llamada junta mayor que, análogamente sucede con las demás desigualdades.

Cuando se produce una equijunta existen dos o más atributos en la relación resultante que tienen el mismo nombre y valores, es decir, siempre se encontrarán dos o más columnas idénticas. Si en la equijunta se eliminan las columnas redundantes, entonces la operación es llamada *junta natural*.

La operación de junta entre las relaciones A y B se puede realizar calculando primero el producto cartesiano entre ellas, y posteriormente aplicando al resultado la operación de selección, con un predicado de la forma:

$$(x_1 \# y_1) \wedge (x_2 \# y_2) \wedge \ldots \wedge (x_n \# y_n)$$

donde:

$x_1, x_2, \ldots, x_n$ son atributos de A;

$y_1, y_2, \ldots, y_n$ son atributos de B;

el dominio y el nombre del atributo de x_i son los mismos que los de y_i, para i = 1,...,n; y # representa un operador booleano del conjunto { =,<,>,<>,<=,>=}, dependiendo del tipo de junta.

Definición:

Sean $X_1,X_2,\ldots,X_n$, $Y_1,Y_2,\ldots,Y_m$ atributos que definen la relación A, y sean $W_1,W_2,\ldots,W_m$ atributos que definen la relación B, donde Y_i, W_i i=1,2, ... , m en el mismo dominio. Entonces se define:

$$A \div B = \{ (x_1, x_2, \ldots, x_n) \mid \forall (x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m) \exists (w_1, w_2, \ldots, w_m) : y_1=w_1, y_2=w_2, \ldots, y_m=w_m \}$$

donde: $x_1 \in X_1$, $x_2 \in X_2$, ..., $x_n \in X_n$;

$y_1 \in Y_1$, $y_2 \in Y_2$, ..., $y_m \in y_m$;

$w_1 \in W_1$, $w_2 \in W_2$, ..., $w_m \in W_m$;

Esta operación divide una relación dividendo A de grado n+m entre una relación divisor B de grado m. Por definición, en el resultado no debe haber tuplas repetidas.

1.2 LENGUAJES DE MODELO RELACIONAL

A partir de la propuesta de Codd del Modelo Relacional, diversos lenguajes de consulta han sido desarrollados. Algunos se basan en el Algebra de Relaciones y otros en el Cálculo de Tuplas y Dominios. Cada sistema propone un enfoque con facilidades de uso y aplicaciones. En el artículo Lenguaje de Flujograma para la Consulta en Base de Datos (Un estudio comparativo) [CHAPA89], se presenta una discusión que compara, SEQUEL: lenguaje basado en el Cálculo de Tuplas, QBE: lenguaje basado en el Cálculo de Dominios, y Lenguaje de Flujograma basado en el Algebra con enfoque gráfico.

En seguida presentamos algunos lenguajes, con aplicación a una base de datos. Uno de estos lenguajes es **SEQUEL** (*Structured English Query Languaje*). Como su nombre lo indica, es un lenguaje de consulta que utiliza palabras en inglés, y cuya estructura semántica es parecida a este idioma. SEQUEL está diseñado tanto para inexpertos como para programadores profesionales. Otros lenguajes que se conocen con esta misma orientación incluyen a **QUEL** y **SQUARE**.

Por su parte, **QBE** (*Query by Example*), es un lenguaje relacional desarrollado por M. M. Zloof en el Yorktown Heights Laboratory de IBM, y su diseño corresponde a un sistema de despliegue visual, es decir, un sistema en el cual la interacción usuario-máquina se realiza por medio de imágenes. En el caso de QBE, las imágenes son tablas que el usuario debe llenar en la pantalla de video, para realizar una solicitud de consulta a una base de datos, y los resultados son igualmente presentados.

1.2.1 SEQUEL

SEQUEL es un lenguaje de consulta con una tendencia a unificar el DML, usando la misma sintaxis para insertar, borrar y actualizar. Además el DDL permite la definición de datos, permitiendo a su vez la definición de relaciones y de usuarios alternos para las vistas de las relaciones; de igual manera, posee herramientas para el control de datos, con el fin de que cada usuario autorice a otros el acceso a sus bases de datos, asi como ofrecer completa integridad de información y la opción para ejecutar varios procedimientos a la vez. Todas estas herramientas fueron incorporadas a SEQUEL de tal manera que pudiese conformar un sublenguaje inmerso en algún lenguaje de alto nivel como PL/1 o COBOL.

Un aspecto importante de este lenguaje, es que por su poder y su facilidad de uso, se ha convertido en un estándar en toda la industria del software. De esta manera, desde las computadoras personales hasta las mainframes, poseen aplicaciones que lo incluyen para permitir consultas a sus bases de datos. El nombre que se ha dado en los últimos años a este lenguaje estándar, es **SQL** (*Structured Query Languaje*), siendo esto una abreviatura del original, surgido a principios de los años setenta.

SEQUEL maneja toda la información de las bases de datos en forma de tablas identificadas con un nombre. Estas tablas son propiamente relaciones que en su conjunto forman una base de datos relacional.

Considérese la siguiente base de datos definida por las relaciones que a continuación se describen:

RELACION EMPLEADOS

NOEMP	NOMBRE	NODEP	PUESTO	NOJEFATURA	SALARIO	COMISION

RELACION DEPARTAMENTOS

NODEP	DEPARTAMENTO	LOCALIZACION

RELACION INSUMOS

NODEP	MATERIAL

RELACION PROVEEDORES

PROVEEDOR	MATERIAL

De tal manera que la relación *EMPLEADOS* cuenta con los atributos:

NOEMP	número de empleado
NOMBR	nombre del empleado
NODE	número del departamento al que pertenece
PUESTO	puesto que ocupa
NOJEFATURA	número de empleado de su jefe inmediato
SALARIO	salario mensual
COMISION	comisión respectiva al salario

La relación *DEPARTAMENTOS* tiene sólo tres atributos:

NODEP	número del departamento
DEPARTAMENTO	nombre del departamento
LOCALIZACION	nombre del lugar donde se ubica el departamento

La relación *INSUMOS* presenta como atributos:

NODEP	número del departamento al que se le provee un material
MATERIAL	nombre del material que consume el departamento

Por último, la relación *PROVEEDORES* contiene:

PROVEEDOR	nombre del proveedor
MATERIAL	nombre del material que surte

A continuación se describen las operaciones que son consideradas, por su importancia, como la parte medular de SEQUEL:

- Recuperación Mapeo

Para lograr una recuperación de información de la base que se ha descrito, se debe formar una petición de consulta conocida como query. Por ejemplo, para obtener los nombres de los empleados del departamento 50, debe hacerse la consulta:

```
SELECT NOMBRE
FROM EMPLEADOS
WHERE NOEMP = 50
```

Esta operación es llamada mapeo, y constituye la más básica de SEQUEL. La cláusula **SELECT** (selecciona), regresa el valor de los atributos que se enlistan, en este caso el atributo es *NOMBRE*; si se desea toda una tupla, entonces debera aparecer un asterisco delante de la clausula SELECT (SELECT *). La cláusula **WHERE** (donde), puede contener toda una colección de predicados que comparan atributos y valores de las tuplas; y la cláusula **FROM** (de), sólo especifica la relación de la cual serán extraídos los datos.

En general, un mapeo proporciona una colección de valores, que corresponden a los atributos seleccionados de las tuplas que satisfacen la cláusula **WHERE**. Los valores duplicados no son eliminados a menos que se especifique lo contrario utilizando la sentencia **UNIQUE** delante de **SELECT**.

- Recuperación de Mapeo Anidado

Es posible usar el resultado de un mapeo en la sentencia **WHERE** de otro mapeo. Esta operación es llamada mapeo anidado.

En el siguiente ejemplo, el mapeo interno regresa la colección de valores de los números de departamento *NODEP*, localizados en Guadalajara, y el mapeo externo entonces opera sobre este conjunto de valores. Los mapeos pueden estar anidados en cualquier número de niveles. En este caso, el operador de comparación = puede ser utilizado en lugar del operador **IN** (operador de pertenencia) sin cambiar el significado del query:

```
SELECT NOMBRE
FROM EMPLEADOS
WHERE NODEP IN
      SELECT NODEP
      FROM DEPARTAMENTOS
      WHERE LOCALIZACION = 'Guadalajara'
```

- Recuperación con Ordenamiento

El resultado de una consulta se regresa en el orden en que se encuentre almacenada la información. Si se desea evitar esto, SEQUEL cuenta con un operador que permite indicar el orden en que deben enlistarse los resultados, ya sea de manera ascendente o descendente, según sea el caso, con respecto a uno o varios atributos. La consulta:

```
SELECT NOEMP,NOMBRE,SALARIO
FROM EMPLEADOS
WHERE NOEMP = 50
ORDER BY NOEMP ASC
```

permite obtener el número de empleado, el nombre y el salario de los trabajadores del departamento 50, ordenados ascendentemente con respecto a la clave de empleado.

Si la ordenación debe cumplirse de manera ascendente, puede cambiarse **ASC** (ascendente) por **DESC** (descendente). Si ninguna de estas palabras reservadas aparece, entonces el default sera **ASC**.

- Recuperación usando funciones y operaciones aritméticas

SEQUEL utiliza una serie de funciones en la cláusula **SELECT** que permiten obtener datos numéricos:

AVG Determina el promedio sobre un atributo.

SUM Efectúa la suma sobre un atributo.

COUNT Contabiliza todos los valores de un atributo. **COUNT(*)** implica que cuente el número de tuplas que satisfacen la cláusula **WHERE**.

MAX Obtiene el máximo valor sobre un atributo.
MIN Obtiene el mínimo valor sobre un atributo.

Igualmente, se pueden realizar expresiones aritméticas, incluyendo **+, -, *, /, (,)**.

Así, para obtener el promedio del salario de los programadores, se tiene:

```
SELECT AVG(SALARIO)
FROM EMPLEADOS
WHERE PUESTO = 'PROGRAMADOR'
```

- Recuperación por grupos

Una relación puede agruparse de acuerdo a los valores de algún atributo, y entonces aplicar alguna operación sobre cada grupo generado. Esto se logra con la cláusula **GROUP BY** (agrupado por) y casi siempre es usada conjuntamente con alguna función. Cuando la cláusula **GROUP BY** sea utilizada, cada campo en la cláusula **SELECT** debe tener una única propiedad de grupo, en lugar de una característica de una tupla individual. Por ejemplo, para obtener la lista de todos los departamentos y el promedio de los salarios de cada uno de ellos, puede hacerse la consulta:

```
SELECT NODEP,AVG(SALARIO)
FROM EMPLEADOS
GROUP BY NODEP
```

Es posible seleccionar varios grupos por medio de un predicado a nivel grupo, y esto se logra con la cláusula **HAVING** (teniendo). Así la consulta:

```
SELECT NODEP
FROM EMPLEADOS
GROUP BY NODEP
HAVING AVG(SALARIO) < 1000000
```

permite obtener los departamentos en los cuales el promedio del salario (*AVG(SALARIO)*) es menor que $1,000,000

Puede notarse que el predicado *AVG(SALARIO) < 1000000* se toma a nivel de grupos y no de tuplas.

- Recuperación como conjunto

La sentencia **SET**, reune los valores de un atributo y los toma como un conjunto que servirá para calificar a todo un agrupamiento. Esta comparación, como es entre grupos, se convierte en una operación a nivel de teoría de conjuntos. Por ejemplo, para obtener la lista de departamentos que contienen todos los puestos existentes, puede hacerse la consulta:

```
SELECT NODEP
FROM EMPLEADOS
GROUP BY NODEP
HAVING SET(PUESTO) =
      SELECT PUESTO
      FROM EMPLEADOS
```

El mapeo interno obtiene todos los puestos de la relación empleados (con duplicados). El mapeo externo agrupa a toda esas tuplas por departamento. El **SET** sobre los puestos genera un conjunto con todos los

puestos que existen. Finalmente se compara a nivel conjunto, el conjunto de todos los puestos con cada agrupación por departamento.
También se tiene la posibilidad de efectuar operaciones de unión, intersección, y diferencia entre relaciones vistas como conjuntos. Por definición una relación es un subconjunto, por lo tanto, una tabla puede tratarse como conjunto y efectuar operaciones sobre éstas.

```
SELECT NODEP
FROM DEPARTAMENTOS
MINUS
SELECT NODEP
FROM EMPLEADOS
```

es una consulta que permite conocer todos los departamentos que no tienen empleados.

Análogamente se utilizan las palabras reservadas **UNION** e **INTERSEC** para las operaciones de unión e intersección respectivamente.

Otros operadores entre conjuntos son **IS NOT IN** (no pertenece a), **CONTAINS** (contiene a) y **DOES NOT CONTAIN** (no contiene a), además del ya conocido **IN** (pertenece a) y de los operadores relacionales = y **<>**.

- Recuperación con más de una relación

En ocasiones es necesario considerar más de una relación en una consulta, un ejemplo de ello son las operaciones vistas anteriormente, pero como puede observarse, se requiere más de un mapeo.
Existe la forma de considerar más de una relación en un sólo mapeo, y esto se logra referenciando los atributos bajo la relación a la que pertenecen. Por ejemplo, un tipo de operación relacional *junta*, puede requerirse en la consulta:

```
SELECT EMPLEADOS.NOMBRE,DEPARTAMENTOS.LOCALIZACION
FROM EMPLEADOS,DEPARTAMENTOS
WHERE EMPLEADOS.NODEP = DEPARTAMENTOS.NODEP
```

que obtiene la lista de los nombres de todos los empleados y las localizaciones de sus departamentos.

La referencia se logra con el nombre de la relación, un punto y en seguida el atributo correspondiente.

Algunas veces se requiere efectuar operaciones entre la misma relación. Esto puede lograrse mencionando más de una vez la relación en la sentencia **FROM**, pero asociando una etiqueta para referenciar las relaciones. Las etiquetas pueden entonces ser usadas en lugar del nombre de la relación. Por ejemplo, para conocer los nombre de los empleados y de sus respectivos jefes, donde los empleados tienen un salario mayor al de sus jefes, puede hacerse la consulta:

```
SELECT X.NOMBRE,Y.NOMBRE
FROM EMPLEADOS X,EMPLEADOS Y
WHERE (X.NOJEFATURA = Y.NOEMP)
       AND
       (X.SALARIO > Y.SALARIO)
```

- Recuperación utilizando subtuplas

Una tupla constante es denotada como el siguiente ejemplo:

```
('ANALISTA',50)
```

Un conjunto de tuplas constantes puede ser representado como sigue:

```
({'ANALISTA',50},{'PROGRAMADOR',45},{'CAPTURISTA',23})
```

Las llaves pueden ser utilizadas para denotar una subtupla de atributos seleccionados de una tupla de la base de datos. Por ejemplo, para conocer los nombres de los empleados que tengan el mismo puesto y el salario de Arturo Barrios, puede usarse el query:

```
SELECT NOMBRE
FROM EMPLEADOS
WHERE {PUESTO,SALARIO} =
      SELECT PUESTO,SALARIO
      FROM EMPLEADOS
      WHERE NOMBRE = 'Arturo Barrios'
```

1.2.2. Query by Example

El lenguaje fue llamado *Query by Example* por sus propias características, ya que se utilizan ejemplos para relizar una consulta: el usuario debe formular un query insertando un ejemplo de una respuesta posible en el sitio adecuado de una tabla vacía. Supongamos que se tiene la relación *EMPLEADOS* definida por los siguientes atributos:

RELACION EMPLEADOS

NOMBRE	CLAVE	DEPARTAMENTO	SUELDO

Ahora bien, si quisiéramos consultar las claves de los empleados cuyo departamento es *COMPRAS* tendríamos la consulta de la figura siguiente:

EMPLEADOS	NOMBRE	CLAVE	DEPARTAMENTO	SUELDO
		P._C07	COMPRAS	

Figura 1.3 Tabla de consulta QBE

Primero, el usuario deberá accesar la tabla oprimiendo alguna tecla o conjunto de ellas para que sea presentada una tabla parecida a la de la figura 1.3, pero la tabla estará en blanco. Para obtener la tabla (relación) correcta sobre la cual se va a generar la consulta, el usuario debe teclear el nombre de la relación en el lugar apropiado (en este caso *EMPLEADOS*). Esto provoca que **QBE** responda colocando los atributos en los lugares correspondientes sobre la tabla, de manera que el usuario podrá ver la tabla como se presenta en la figura 1.3, pero sin los campos de consulta aún. Para dar solución a las consultas, deberán ser expresadas haciendo entradas en dos posiciones de la tabla. La **P.** significa *imprime*, es decir, *presenta los valores que se obtengan de la consulta*; *C07* es un elemento ejemplo, ya que toda cadena que tiene como prefijo un _ se toma como una respuesta posible; *COMPRAS* no tiene el dicho prefijo, por lo que es considerado como una constante. De esta manera, la tabla contiene la consulta: *Imprimir todos los valores de CLAVE (o todas las claves), tales como C07, cuyo departamento sea COMPRAS*. Realmente *C07* es simplemente un valor cualesquiera, y esto es arbitrario, pues nótese que en lugar de *C07* pudieramos haber puesto cualquier otra cosa como *PPL* o *S0005*, sin alterar el significado de la consulta. Como esto es meramente simbólico, bien puede colocarse **P.** simplemente en lugar de *P._C07*, pero se hace con fines de claridad de la consulta.

QBE ofrece además una serie de características de manejo y modificación de despliegue de tablas, entre las más importantes se consideran aquéllas que permiten al usuario, mediante ciertas teclas o funciones de la terminal, agregar columnas a una tabla, o bien eliminarlas; además, también permite ampliar o reducir el espacio de las columnas, para que la información sea desplegada y capturada adecuadamente. De la misma manera, estas operaciones pueden realizarse sobre los renglones.
Lo que realmente interesa a nuestro análisis, es la forma en que QBE realiza las recuperaciones de información, para lo cual citaremos sus formas más importantes.

- Recuperación simple

EMPLEADOS	NOMBRE	CLAVE	DEPARTAMENTO	SUELDO
		P._X	VENTAS	

Esta consulta permite conocer todos los nombres de los empleados del departamento de *VENTAS*.
QBE elimina cualquier duplicado de información en sus respuestas, y si por algún motivo se requiere que no se realice esta operación, debe entónces agregarse la palabra **ALL**.

EMPLEADOS	NOMBRE	CLAVE	DEPARTAMENTO	SUELDO
		P.ALL._X	VENTAS	

Si se desea obtener toda la información almacenada en *EMPLEADOS*, podría realizarse la consulta

EMPLEADOS	NOMBRE	CLAVE	DEPARTAMENTO	SUELDO
	P._W	P._X	P._Y	P._Z

o bien,

EMPLEADOS	NOMBRE	CLAVE	DEPARTAMENTO	SUELDO
P.				

- Recuperación calificada

EMPLEADOS	NOMBRE	CLAVE	DEPARTAMENTO	SUELDO
	P._X		VENTAS	>1000000

Sirve para conocer los nombres de los empleados que perciben más de *$1,000,000* y que pertenecen al departamento de *VENTAS*.

Como puede observarse, se utilizan operadores relacionales =, <>, >, <, >= y <= para auxiliar las consultas, de manera que *>1000000*, indica que la selección sea hecha para los valores de esa columna que cumplan con la condición.

EMPLEADOS	NOMBRE	CLAVE	DEPARTAMENTO	SUELDO
	P._X		CAPTURA	>1000000
	P._Y			>1200000

Es una consulta que permite obtener los nombres de los empleados que ganan más de *$1,200,000* o bien, que pertenecen al departamento de *CAPTURA* y ganan más de *$1,000,000* o ambos.

Puede notarse que lo que se indica sobre un renglón, obliga a que las columnas se asocien mediante el operador booleano **Y**, y entre los renglones se relacionen por el operador **O**.

- Recuperación usando enlace

Este tipo de recuperación se realiza sobre dos o más tablas, las cuales son ligadas por medio de un atributo llave (recuérdese las reglas de integridad del modelo de estructura de datos relacional).

Supóngase ahora que son definidas las siguientes relaciones:

RELACION DEPARTAMENTOS

DEPARTAMENTO	LOCALIZACION	CLAVEDEP

RELACION PUESTOS

PUESTO	CLAVEDEP

Para obtener los empleados y sus puestos respectivos, se puede hacer la consulta

EMPLEADOS	NOMBRE	CLAVE	DEPARTAMENTO	SUELDO
	P._X		_DEP	

DEPARTAMENTOS	DEPARTAMENTO	LOCALIZACION	CLAVEDEP
	_DEP		_CDEP

PUESTOS	PUESTO	CLAVEDEP
	P._PUESTO	_CDEP

De esta forma, el enlace de los archivos se realiza a través de las variables *_DEP* y *_CDEP*, que representan los campos llaves de cada tabla.

- Recuperación usando un enlace dentro de una sola tabla

Lo que realmente se hace en este tipo de recuperación, es efectuar una operación Y entre dos renglones. Así por ejemplo, para conocer los departamentos que se localizan en más de un área de la empresa, hay que hacer una consulta como la siguiente:

DEPARTAMENTOS	DEPARTAMENTO	LOCALIZACION	CLAVEDEP
	P._X	_AREA	
	_X	<>AREA	

que se traduciría como: *Imprimir todos los departamentos X en AREA, tales que tienen otra localización diferente a AREA.* Para este caso se utilizan las variables *X* y *AREA* como llaves.

- Recuperación utilizando una caja de condiciones

Para dar una facilidad en la recuperación de condiciones entre los campos de una relacion, QBE ofrece la opción de abrir una *caja de condiciones* que son requeridas por una consulta. Por ejemplo, supóngase que se define la relación

RELACION SUELDOS

CLAVE	CATEGORIA	SUELDOTOPE

y se desea conocer los nombres y claves de los empleados que superan en 5% su sueldo tope.

EMPLEADOS	NOMBRE	CLAVE	DEPARTAMENTO	SUELDO
	P.	P._CLV		_SUEL

SUELDOS	CLAVE	CATEGORIA	SUELDOTOPE
	_CLV		_SUELTOP

CONDITIONS
_SUEL > _SUELTOP * 1.05

Esta última tabla corresponde a la caja de condiciones, y en ella se establece un predicado donde se involucran las variables *SUEL* y *SUELTOP*.

1.3. SISTEMAS GRAFICOS

El desarrollo e implementación de sistemas gráficos por computadora ha requerido de muchos años de investigación para lograr los alcances que se dan en la actualidad. Analizando un poco los orígenes de los sistemas gráficos, encontramos que desde hacía tiempo se ha intentado desarrollar hardware que permita una interfase lo suficientemente amigable para el usuario; paralelamente, los desarrolladores de software han realizado un esfuerzo para explotar todo este hardware, e incluso balancear lo que la tecnología adelanta considerablemente.

1.3.1 Orígenes de la graficación por computadora

Es en 1950 cuando en el Massachusetts Institute of Technology (*MIT*) se logran las primeras imágenes computarizadas utilizando un tubo de rayos catódicos (*CRT*) similar al de un aparato de televisión. Durante la década de los años 50's no se obtuvieron muchos avances en esta área, debido a las restricciones tecnológicas; y no es sino hasta el final de esa década cuando a las computadoras se les da la capacidad de realizar funciones interactivas con el ser humano. Este suceso abrió la brecha para que surgieran las primeras investigaciones serias sobre la graficación asistida por computadora, pero su aplicación y desarrollo sólo fue posible en las grandes universidades y en las industrias y laboratorios que poseían los recursos indispensables para ello. Así es como transcurrió la década de los años 60's.

Con los avances de la tecnología, el costo de los equipos de cómputo fue disminuyendo, mientras que su rapidez y eficiencia aumentaba. De esta manera, es en la década de los 70's cuando los años de investigación en la rama de la graficación computarizada comienza a dar grandes frutos.

En la actualidad, es común encontrar sistemas basados en la graficación por computadora:

El diseño con ayuda de la computadora (*Computer Aided Design* **CAD**), ofrece una extensa gama de herramientas de software, que permiten lograr el diseño de partes y el dibujo mecánico de manera interactiva, una vez que se han dado las dimensiones del objeto al sistema CAD, el usuario puede por ejemplo, observar cualquier lado del objeto para apreciar como se verá después de su construcción. Además, el usuario puede cambiar las dimensiones a placer, lo que contrasta con un dibujo realizado manualmente. Estos sistemas generalmente son auxilio de diseñadores de herramientas y maquinaria, automóviles, barcos, aviones, circuitos electrónicos, edificios, naves espaciales, etc.

Existe otro tipo de software, que se ha desarrollado para generar gráficas y diagramas. Muchas veces este software puede presentar, una variedad de tipos de gráficas, como diagramas de barras, barras tridimensionales, de pastel, gráficas de funciones, así como gráficas de superficies, etc. Las gráficas y los diagramas, se usan generalmente para resumir datos financieros, estadísticos, matemáticos, científicos o económicos, y algunas gráficas se combinan en una misma presentación. En la mayor parte de los casos,

este tipo de gráficas son generadas para dar a conocer informes administrativos, boletines y como herramienta de apoyo didáctico.

La animación de imágenes, es uno de los aspectos de interés para los diseñadores de software, ya que por medio de programas como estos, el cine y la televisión crean efectos especiales y dibujos animados; en proyectos de sistemas espaciales y de navegación, son usados en aplicaciones de capacitación: algunos ejemplos de estos son simuladores para el entrenamiento de pilotos de aeronaves, o bien, capitanes de barcos e incluso conductores de automóviles. Una utilización más, es vista en el desarrollo de juegos por computadora.

La técnica, que se usa para producir gráficas computarizadas a partir de fotografías o exploraciones de TV, se denomina *Procesamiento Digital de Imágenes.*

El procesamiento de imágenes, es muy utilizado en aplicaciones de arte comercial, donde se realizan retoques y reacomodo de secciones fotográficas y otros trabajos artísticos. En medicina es utilizada esta técnica para obtener imágenes del cuerpo humano, como por ejemplo en la tomografía.

Las operaciones de entrada a muchos programas de computadora se diseñan como un conjunto de iconos o símbolos gráficos que representan el procesamiento que se desea realizar. Los usuarios seleccionan las opciones de procesamiento, señalando el icono adecuado. La ventaja de estos sistemas es que los iconos, pueden ocupar menos espacio en la pantalla, que la descripción textual correspondiente a las funciones, y pueden entenderse más rápidamente si están bien diseñados.

Muchos dispositivos de hardware han surgido con el fin específico de manipular grandes cantidades de información gráfica en un tiempo considerablemente bajo, y gracias a ello, son cada vez más los programas de aplicación que presentan una interfase gráfica, facilitando la presentación detallada de imágenes y agilizando su manejo y comprensión.

Por todo lo anteriormente descrito, no es difícil predecir que la graficación por computadora será cada vez más utilizada, como una nueva opción contra los métodos tradicionales.

1.3.2 Filosofía de los sistemas gráficos

Cuando surgen los sistemas interactivos que permiten la comunicación hombre-máquina de manera directa, es utilizado el método tradicional de visulizar la información en un monitor, que consiste en dividir la pantalla en un conjunto de renglones y columnas para formar una matriz, donde cada elemento o celda tiene las mismas diemensiones apropiadas para dar cabida a un caracter, y de esta forma, poder presentar textos, números y aún símbolos especiales de uso común. De hecho, esto da una gran versatilidad en la captura y visualización de datos, por lo cual el desarrollo de hardware y software se fue orientando hacia este modelo de trabajo. Sin embargo, con los enormes avances logrados en la tecnología digital, se fueron

encontrando cada vez más aplicaciones para las computadoras, y algunas de ellas, requerían más que la simple visualización de símbolos de tamaño uniforme y número limitado: las gráficas computarizadas se hacen cada vez más necesarias en las aplicaciones, debido a que son un medio efectivo de comunicación entre el hombre y la computadora; la mente humana asimila mucho más rápido las imágenes de un diagrama u objeto, que los números y nombres que aparecen en una tabla. Por ello surge la necesidad de dividir la pantalla de un monitor en entes mucho más pequeños que un caracter: estos entes son denominados pixeles (plural de pixel), y consisten en rectángulos o puntos que pueden ser manipulados individualmente, de tal forma que permitan generar imágenes detalladas.

De todo lo anterior, surge una de las múltiples clasificaciones que se le dan a los programas de aplicación: los sistemas no-gráficos o de texto, que son aquéllos que sólo permiten el uso de caracteres predefinidos, ya sea a causa de las limitaciones del equipo, o debido a su naturaleza misma; y los sistemas gráficos, que utilizan una matriz de pixeles para visualizar imágenes variadas y de mayor detalle.

Una gráfica por computadora puede ser utilizada para una gran variedad de propósitos: puede ser un dibujo de ingeniería, una ilustración de un libro o revista, una gráfica de estadística, o incluso un dibujo artístico.

Dependiendo de la aplicación, se utilizan una gran variedad de algoritmos para representar una gráfica en la pantalla de una computadora. Los más comunes son los orientados al manejo de vectores o líneas [ROGERS], las cuales pueden trazarse con el conocimiento de las coordenadas de dos puntos en el espacio. Un aspecto importante en el trazo de una gráfica de este tipo, es el orden y la manera en que deben manipularse dichos puntos. Por ejemplo, una figura rectangular puede tratarse como un conjunto de cuatro puntos:

P1(0,0), P2(0,10), P3(10,10), P4(10,0)

y el algoritmo de trazado podría ser:

Conectar los puntos P1 P2 P3 P4 P1 en secuencia

Otra manera de abstracción del mismo rectángulo puede darse a través de cuatro líneas:

L1 = P1 P2 L2 = P2 P3 L3 = P3 P4 L4 = P4 P1

y el algoritmo para trazarlo:

Dibujar las líneas L1 L2 L3 L4 en secuencia.

Finalmente, el rectángulo en sí puede tratarse como una unidad a través de un polígono definido como:

G1 = P1 P2 P3 P4 P1 o G1 = L1 L2 L3 L4

Sin importar cuál de estas formas sea utilizada, podemos deducir que los elementos fundamentales de la gráfica siguen siendo los puntos, lo único que varía es el método de ordenarlos y manipularlos.

De esta manera, (x1,y1) o (x2,y2,z2) pueden representar dos puntos en 2 y 3 dimensiones respectivamente. Dos puntos representarán una línea o contorno, y tres o más puntos formarán un polígono. Esos puntos, líneas o polígonos son reunidos y almacenados en una base de datos para su posterior uso. Es importante recalcar que rara vez se utiliza esta base de datos directamente para mostrar la gráfica final; normalmente se utiliza una estructura de datos interna, que es la que contiene la información ordenada y clasificada de acuerdo a la porción de la gráfica que esté visualizándose en un momento determinado.

1.3.3 Componentes de un sistema con interfase gráfica

Un *Sistema con Interfase Gráfica* es aquel que permite al usuario visualizar y manejar información representada a través de dibujos o diagramas. Para ello, deben utilizarse las herramientas de hardware y software necesarias que faciliten su manejo y comprensión.

Las herramientas de hardware son muchas y de tipos muy variados, pero básicamente, se debe contar con por lo menos dos dispositivos fundamentales: un monitor con capacidades gráficas aunado a un controlador de video que permita tales funciones; y un dispositivo que permita al usuario comunicarse con la computadora, tanto para introducir información, como para adecuarla a sus necesidades. Normalmente, un teclado convencional cumple con los requerimientos de este último tipo de dispositivos.

En el diseño del software que permita este tipo de interfases, deben considerarse por principo los objetivos para los cuales será creado el sistema, y de acuerdo a ello, elaborar una serie de herramientas que ayuden de manera fácil y eficiente al manejo de datos y entes gráficos.

Básicamente, la interfase que permite la comunicación hombre- máquina se divide en cuatro componentes: Modelo del Usuario, Lenguaje de Comandos, Retroalimentación y Visualización de Información [NEWMAN]: el primero y más importante de ellos, el *Modelo del Usuario*, que no es sino el modelo conceptual que el usuario tiene del sistema, conociendo previamente el tipo de información que éste debe manejar, y los procesos que pueden ser aplicados a tal información. Sin este modelo, el usuario no puede mas que seguir ciegamente un conjunto de instrucciones que en su mayoría permanecen incomprensibles. El modelo debe ser capaz de permitir al usuario el entendimiento de lo que la computadora está realizando, sin importar si tiene o no conocimiento sobre tecnología en computación.

Con la ayuda del modelo, el usuario debe poder anticipar los efectos de sus acciones, y así definir sus propias estrategias para operar el programa.

Una vez que el usuario ha entendido el modelo, surge la necesidad de introducir órdenes o comandos para manipularlo. Precisamente, el segundo componente de una interface es el llamado *Lenguaje de Comandos.* La mayoría de nosotros está familiarizado con alguno de estos lenguajes, ya que casi todos los aparatos y máquinas que utilizamos en la vida cotidiana lo poseen explícita o implícitamente. El hecho de que esos comandos permitan el uso de tales máquinas sin requerir gran esfuerzo de aprendizaje, nos indica el cuidado con el que fueron diseñados e implementados. Idealmente, en la interfase de los programas por computadora, debe existir un lenguaje que se asemeje e estos en facilidad y funcionalidad.

El tercero de los componentes de una interfase, es la llamada *Retroalimentación*, con la cual la computadora asiste al usuario en la operación adecuada del programa. Esta retroalimentación se da de maneras muy diversas: en la entrada de textos y números, en el envío de mensajes, en la indicación de objetos o figuras seleccionadas, etc. Algunas formas de retroalimentación son implementadas principalmente para ayudar a usuarios sin experiencia y pueden ser ignoradas por los expertos. Pero por otro lado, algunos lenguajes de comandos son inherentemente dependientes de la retroalimentación; por ejemplo, la manera de posicionarse en un determinado lugar o elemento de una gráfica, casi siempre requiere de un señalamiento (cursor) en la pantalla que se desplace según las indicaciones realizadas a través del dispositivo de posicionamiento.

La retroalimentación ayuda al usuario a estar seguro de que sus comandos fueron correctamente recibidos y completamente entendidos por el programa.

La *Visualización de Información* es el último componente de una interfase, y es indispensable para mostrar el estado actual de los datos que se están manipulando. Es aquí donde se debe poner un especial cuidado en la organización de las imágenes mostradas, con el fin de transmitir la información de la manera más efectiva posible. La imagen es la confirmación que indica al usuario si su modelo es correcto, y ésta debe darse en el apego más estricto al modelo que fue seleccionado. Cuando el modelo depende del realismo, como en un simulador de vuelo, se debe poner especial énfasis en el detalle de la imagen visualizada; cuando se escoge un modelo más sintético, debe implementarse un conjunto adecuado de símbolos gráficos y de imágenes, que permitan visualizar la información de acuerdo a como el usuario lo espera.

1.4 CASE

CASE (*Computer Aided Software Engineering/ Ingeniería de Software Asistida por Computadora*), es una herramienta para el desarrollo de Software, que tiene el objetivo de asistir al usuario tanto en la implementación de sistemas, como para documentar su trabajo y funcionamiento [LUCAS].

1.4.1 La filosofía CASE

La mayoría de las veces, la comprensión del funcionamiento de las empresas y de los servicios de información reside en la mente de un reducido número de personas. A través de los años, las compañías han intentado construir modelos para documentar su propia administración. Esos modelos fueron creados con el fin de simbolizar de alguna forma el conocimiento y la experiencia del personal calificado, en lo referente al funcionamiento de la empresa, y a su vez para que todo el organismo conociera en términos generales la administración de la empresa.

Se han desarrollado diversas metodologías que permiten la construcción de modelos, mediante la combinación de técnicas de diagramación con descripciones textuales.

Los diagramas son expresiones gráficas que pueden mostrar actividades de la empresa, así como el flujo de la información que se tiene en la organización.

Las técnicas de diagramación usan iconos para representar varios de los componentes de una empresa: personal, recursos, datos, etc. Cada icono por lo regular, va acompañado de un texto que describe su propósito, su relación con las demás actividades, y su relación con otros iconos del diagrama. Las descripciones textuales incluyen la información necesaria para describir con claridad el objeto del mundo real al que representa el icono.

Las metodologías para la construcción de modelos, consideran alguna guía para la elaboración de modelos que representan actividades corporativas y desarrollo de sistemas. Sin embargo los modelos de problemas requieren modificaciones frecuentes de acuerdo al cambio del problema original. Si los modelos son construídos a mano, los cambios requerirán de un gran esfuerzo. Como resultado, la elaboración a mano de los modelos sofisticados es obsoleto. Además, si se va a realizar un gran esfuerzo en construir manualmente un modelo, no se deseará hacer un gasto extra en modificarlo. Por ello es necesario construir una representación simbólica de varias facetas del problema para su revisión y sus posibles mejoras en el futuro. De esta manera, podemos ver que el cambio es inherente en la construcción de modelos, por lo que debe darse una forma de realizarlo con la mayor facilidad posible.

Las descripciones textuales en la CASE, forman parte de lo que se conoce como *Diccionario*. El diccionario está constituído por formatos predefinidos asociados con diferentes iconos. Estos formatos suministran información importante relacionada con la parte del mundo real que representa el icono. Además, en el

diccionario se almacena un directorio que indica la relación existente entre diagramas específicos, iconos, y descripciones textuales.

La CASE representa una filosofía completa para el desarrollo de sistemas de información, y a su vez para modelar el funcionamiento y organización de cualquier empresa o negocio. La filosofía CASE representa tanto la computadora, como las herramientas de desarrollo para construir los modelos que describan una empresa, y para documentar y desarrollar sus sistemas computacionales, desde la planeación hasta su implemantación.

Para comprender de manera sencilla la CASE, es necesario revisar por separado sus componenetes, para ello se considera una división en tres niveles [LUCAS]: superior, intermedio e inferior.

La *CASE Superior*, a veces llamada *Planeación Asistida por Computadora*, se refiere a la parte que soporta la planeación de la empresa o planeción corporativa. La *CASE Intermedia* se centra en el análisis y diseño de sistemas. La *CASE inferior* es el componente que soporta el desarrollo de sistemas.

Uno de los principales problemas de la CASE, es que no existen estándares de productos CASE en el mercado. Cada desarrollador de Software CASE, incluye y omite lo que desea, basado en su propio criterio, y lucha además, por que su producto sea aceptado como un estándar.

CASE Superior. Uno de los problemas con que la mayoría de las empresas se enfrenta, es la pérdida de tiempo por parte de sus administradores para comprender y crear planes de las actividades centrales de su compañía. Como lo marca la administración, los planes empresariales fijan metas y objetivos, generan estrategias y políticas que permitan alcanzarlos, y establecen estándares para tener niveles aceptables de desempeño en la operaciones.

La CASE superior utiliza software para describir la compañía y sus planes. Por medio de diagramas, permite descomponer y describir sus partes importantes. Dichas partes generalmente son los departamentos que la constituyen, conjuntamente con sus funciones y procesos; asimismo, es posible representar las metas, objetivos, responsabilidades, recursos, problemas, etc. Con ello es fácil distinguir y entender la estructura y condiciones de la empresa.

Con esas descripciones es posible crear mejores planes estratégicos, que conjuntamente con los recursos y tareas, pueden ayudar a alcanzar todos los objetivos y metas fijadas. Una empresa puede tomar lo anterior como atributos de planeación sobre la estructura de la empresa, de tal suerte que la estructura no cambie, mientras que paramétricamente los atributos de planeación sí. De hecho, esto suele suceder en las empresas, no obstante, también su estructura puede llegar a cambiar dadas las condiciones para alcanzar un objetivo, pero lo que siempre se busca, es el mejoramiento de su funcionalidad no de su estructura.

Algunos de estos modelos de representación, requieren un gran trabajo. Sin embargo, después de que se construye el modelo de un problema y sus modelos de planeación, es factible que se reutilicen algunas de

las especificaciones dadas, asi que estas descripciones forman parte de un diccionario. Por ejemplo, la descripción de clientes, que se puede generar en el diseño de la planeación del padrón de ventas, puede ser utilizado, en el desarrollo de un modelo de planeación de estudio de mercado.

El éxito en la planeación de una compañía, depende mucho de la cantidad y disponibilidad de la información. Los sistemas CASE superior, contienen especificaciones de planeación, muy útiles para el desarrollo de sistemas de información posteriores.

CASE Intermedia. En la CASE intermedia se analizan los problemas de información, y se diseñan sus soluciones. Los sistemas CASE intermedia consisten en componenetes de diagramación y un diccionario, que operan de manera similar a sus correspondientes en la CASE superior. Sin embargo, la metodología involucrada en la CASE intermedia es diferente. La combinación de diagramas y entradas al diccionario, sirven a los analistas de sistemas para automatizar su trabajo. Esos sistemas también tienen iconos de propósito especial, y pantallas preformateadas del diccionario, para describir cómo esos iconos representan objetos del mundo real.

Para los desarrolladores de software, la CASE intermedia reduce de gran manera el ciclo de vida del desarrollo de un proyecto. Otros beneficios que se obtienen, es el almacenamiento de conocimientos de los propios analistas, que generalmente la mayor parte de las veces sólo queda en su mente, como lo pueden ser el entendimiento de procedimientos de la compañía y necesidades de información.

No obstante, es relativamente poca la información que se captura con el software que soporta las operaciones de la empresa, por lo que utilizando la CASE intermedia, es posible describir qué operaciones departamentales existen, y cual es su importancia; por qué las operaciones se realizan de cierta manera, qué información necesitan y de qué forma es utilizada esa información; por qué ciertas condiciones afectan las operaciones, qué información acerca de esas condiciones se necesita y para qué sirve; las responsabilidades de cada operación; las tareas del personal y cómo y para qué sirve la información de esas tareas; etc.

Cabe señalar que sólo del 25 al 30% de las especificaciones de la CASE intermedia es transportable a la CASE inferior. Los sistemas de la CASE inferior crean especificaciones propias de los programas. Dado que las especificaciones de la CASE intermedia involucran, predominantemente, la documentación de las actividades de una compañía y las formas en las que le sirve la información, sólo un pequeño porcentaje de esas especificaciones es vaciada directamente dentro de los sistemas inferiores. El diseño y desarrollo de especificaciones es un trabajo arduo, pero una vez que se han modelado varios sistemas, es posible reutilizarlas convenientemente.

CASE Inferior. La CASE inferior utiliza software para crear un conjunto de especificaciones de desarrollo orientadas a la generación de programas y de documentación para el usuario de los sistemas. El software de desarrollo también incluye un diccionario, sin embargo, en la mayor parte de las veces no provee un componente gráfico, dado que las especificaciones de desarrollo de software no lo necesitan: estas

especificaciones se relacionan directamente con los programas del sistema. El diccionario documenta las características de los programas que en sí, representan el mundo real que se está modelando, por lo que el diccionario sólo proporciona una referencia. El diccionario de desarrollo CASE, es denominado Diccionario Activo, ya que permite introducir especificaciones que describen el desarrollo de los programas, dando criterios y referencias para sus atributos.

Un diccionario activo comprende tres partes principales:

- Una ***Base de Datos***, en la cual se almacenan las características del ambiente computacional, y las características de los sistemas de aplicación.

- ***Esquemas*** de la lógica de procedimientos y módulos que componen los programas que son parte de los sistemas de aplicación.

- Un ***activador*** capaz de combinar características de aplicación, con esquemas de lógica de procedimientos y módulos, para producir programas de aplicación.

La creación de especificaciones de desarrollo, y expresamente la creación de programas, es una tarea que en un principio puede ser ardua, pero conforme se desarrollen más sistemas, dichas especificaciones podrán ser reutilizadas.

Finalmente, el sistema de desarrollo CASE, puede generar documentación de desarrollo y operación que puede obtenerse en varios formatos.

1.4.2 Los beneficios de usar CASE

Los beneficios de la CASE superior, son mucho mayores si se utilizan por completo en la planeación de la empresa. Si se utiliza un sistema de CASE superior para construir el modelo de una empresa, se obtiene mayor penetración para conocer la importancia de ciertas funciones y de cómo afectan a la organización.

Con CASE superior se pueden comprender mejor:

1. Los **mecanismos y las responsabilidades** de la empresa y de sus departamentos,

2. Las **metas** y objetivos de la compañía y de sus departamentos,

3. La **influencia de las operaciones** en el trabajo para alcanzar sus metas,

4. El **organigrama** funcional de la empresa,

5. Los **factores** que afectan las operaciones de la empresa,

6. La **secuencia** de las operaciones,

7. La **obtencion de recursos** para el soporte de operaciones,

8. El **efecto** de influencias externas en la organización,

9. Los **problemas** a los que se enfrenta la empresa, y

10. La **importancia de la información** en el éxito de la administración.

El uso de CASE superior para desarrollar modelos de planeación, da un entendimiento más claro de la dirección de la compañía, y de cómo puede contribuirse a su éxito. Esos modelos de planeación permiten estimar el impacto que puede causar el cambio de valores de ciertas especificaciones de planeación de la empresa, puede realizarse el análisis de "qué pasa si ..." , y fijar los peores y mejores casos. De esta manera es posible estimar que tanto pueden afectar los cambios antes de realizarlos.

Los modelos también proporcionan las bases para las especificaciones de proyectos. Muchas especificaciones de planeación puede ser mapeadas dentro de los programas del proyecto, las descripciones de las actividades y sus duraciones, y la adquisición, utilización y costo de recursos.

Los fabricantes de sistemas de CASE superior están intentando integrar su software a otros programas, con el fin de que por ejemplo, puedan pasarse las especificaciones desarrolladas en la CASE superior, directamante hacia un sistema manejador de proyectos, con el fin de que muchas actividades iniciales, desarrolladas durante la administración de proyectos, sean hechas sin la intervención del humano.

El mayor beneficio de la CASE intermedia, es que proporciona métodos más fáciles de actualización y modificación en el diseño de los sistemas. Es también más sencillo visualizar como el analista de sistemas, comprende y resuelve los problemas. El proceso de diseñar, lleva consigo procesos interactivos:

- Los ususarios **discuten** sus necesidades de información con los analistas,

- Los analistas usan **diagramas** y entradas del diccionario para documentarlas,

- Los usuariors **revisan** los diagramas y las entradas del diccionario, y sugieren modificaciones; y

- Los analistas **responden** a esas sugerencias, haciendo cambios apropiados a las especificaciones de análisis y diseño.

Los últimos dos pasos continúan hasta que las especificaciones de análisis y diseño satisfacen las necesidades requeridas.

Un segundo beneficio de la CASE intermedia, es que facilita los trabajos de diseño-aplicación; en esta parte, los profesionales en sistemas y los usuarios finales, interactúan rápidamente, y documentan los requerimientos para los sistemas de aplicación. En un principio dentro del desarrollo de proyectos, esos trabajos representan el salto inicial para los mismos. Conforme continúa la interacción, sus resultados son grabados en diagramas y entradas preliminares del diccionario. Así, los usuarios finales pueden influenciar el análisis y el diseño, de forma rápida y directa.

Un tercer beneficio, es la facilidad que se da al analista para crear pantallas que simulen la entrada y salida de datos, y los reportes que tendrá el usuario final. Esas patallas prototipo son producidas en los inicios del análisis y el diseño del proyecto. Pueden usarse para simular la captura de datos, y para determinar de forma global las necesidades de información.

Los sistemas de CASE inferior generan del 60 al 80% del código del programa en el sistema. Como resultado, el mayor beneficio de la CASE inferior es la reducción substancial del tiempo requerido para desarrollar un sistema. Cuando se utilizan los sistemas de CASE inferior, la mayoría del tiempo consumido en el desarrollo de sistemas es el relacionado con la escritura del código.

Otro beneficio consiste en la facilidad de modificar los sistemas generados por la CASE inferior. Estos últimos generan la mayoría del código, restando sólamente la implementación y mantenimiento de los programas, lo cual es menos complicado y lleva menor tiempo. En adición, las actividades de mantenimiento son llevadas a cabo al través del diccionario activo, que mantiene un directorio de programas y módulos de aplicación.

Un beneficio final está relacionado con las capacidades de creación de prototipos. Los prototipos producidos por la mayoría de los sistemas de CASE, requieren ser cargados en una fase intermedia de éste para su ejecución, lo cual significa la familiarización con algún sistema de CASE intermedia. Los sistemas de CASE inferior producen prototipos que funcionan como sistemas individuales, lo cual no requiere adiestramiento especializado para usarlos.

La CASE es benéfica en muchas formas. Los modelos creados usando sistemas de CASE permiten entender mejor a la compañía y a las condiciones que enfrenta. Se pueden evaluar las situaciones con mayor juicio, y tomar desiciones apropiadas para el mejoramiento de la empresa. Puede verse el desarrollo de los sistemas e influenciar su progreso. Y ya completos los sistemas, serán consistentes con la administración actual de la empresa.

Con la evolución constante de la CASE, se tendrán las herramientas necesarias para la planeación y desarrollo de sistemas empresariales, oportunamente y de manera más adecuada a las necesidades.

CAPITULO 2: LENGUAJE ICONOGRAFICO PARA EL DESARROLLO DE APLICACIONES (LIDA)

Uno de los principales problemas en la Ingeniería de Software, es precisamente el desarrollo de un lenguaje que facilite la programación. Desde la aparición de la computadora, los lenguajes han venido progresando en este sentido.

Considerando esta necesidad actualmente los lenguajes de cuarta generación intentan resolver el problema, pero a pesar de todas las afirmaciones respecto a que las líneas de programación se han reducido, y que a su vez se ha incrementado el nivel de procedimientos encapsulados, llegando a la generación de aplicaciones complejas, los usuarios finales todavía tienen dificultad para trabajar con los métodos de acceso de estos lenguajes, debido a que aún les es necesario conocer, tanto teoría de bases de datos, como la manera en que se manipulan, sin embargo, estos lenguajes constituyen una opción de programación para el usuario final.

2.1 LIDA COMO UN LENGUAJE FORMAL

Se ha incursionado últimamente en el proceso de producir y trabajar con gráficas, imágenes y diagramas como una forma de interacción entre el usuario y la computadora, creando un nuevo concepto denominado programación visual.

Hoy día, se puede advertir que el elemento gráfico por computadora, es posible utilizarlo en áreas diversas como lo son: la administración, la industria, el gobierno, el arte, el entretenimiento, la publicidad, la educación, la ingeniería, la arquitectura y la medicina entre otros.

El diseño de un lenguaje visual, con el poderío de un lenguaje de cuarta generación, parece ser también una respuesta al problema interfase; esto es, se sabe de antemano que el valor de una imagen como un medio eficaz de comunicación, y la capacidad de conversar en forma gráfica con una computadora, está revolucionando la manera de trabajar en muchas áreas, como lo hemos advertido antes; además realmente a los lenguajes de cuarta generación se les utiliza en los generadores de aplicaciones, es decir, herramienta de software para dar solución a problemas por medio de la computadora, sin que necesariamente se tenga que programar, como comúnmente lo hacemos con los lenguajes de tercera generación.

LIDA es un lenguaje iconográfico que ocupa el tema central de nuestro trabajo, y que siguiendo las características de un lenguaje de cuarta generación, se diseña como un lenguaje lo suficientemente formal como para considerarse dentro del ámbito CASE, a continuación describiremos el concepto de un

generador de aplicaciones, y las características más relevantes de los lenguajes iconográficos que son conceptos básicos para la definición de LIDA.

2.1.1 Generadores de aplicaciones

En el último cuarto de siglo se ha dado testimonio, a una asombrosa mejoría en nuestra capacidad para fabricar sofisticados dispositivos de hardware. El crecimiento de elementos construídos por chip, se duplica por año. Este crecimiento exponencial desmedido, sólo se ha podido compensar por el mejoramiento en el nivel de software [HOROWITS]. Los desarrolladores de software actuales, se encuentran involucrados directamente en los siguientes aspectos:

- *Incremento en la demanda de nuevas aplicaciones*

- *Incremento en la complejidad de las nuevas aplicaciones, e*

- *Incremento en el costo y decremento de disponibilidad de gente calificada.*

Es extremadamente improbable, que estas demandas puedan ser cubiertas completamente, por la aplicación cuidadosa de herramientas convencionales de software.

Una aproximación a la solución de estos retos, es el uso de generadores de aplicaciones. Los GA (*Generadores de Aplicaciones*), son sistemas de software que forman parte del concepto CASE de nivel inferior y están diseñados para soportar el desarrollo de aplicaciones de forma automática. Los GAs proveen un lenguaje de muy alto nivel de propósito especial de una sintaxis que pretende ser sencilla para el usuario. Por lo tanto los GAs pueden ser fácilmente utilizados por usuarios finales, quienes generalmente no son expertos conocedores del área de computación.

Un juicio convencional dicta que el desarrollo de software, puede ser visto como una serie de pasos, que consisten en el análisis y diseño, programación, pruebas, depuración, documentación, instalación y mantenimiento. Cuando se utiliza un GA, el usuario debe comenzar por el análisis y el diseño de la aplicación, entonces incrementar su prototipo y extenderlo hasta que reúna todos los requerimientos. Una vez que se tiene esto, deberá saltar la fase de programación, que será generada por el GA, posteriormente las siguientes fases se simplifican de manera considerable.

El decir que la fase de programación se omite, no es del todo cierto, el GA construirá un programa o bien, será capaz de generar un código que conformará la aplicación, pero en base a un lenguaje de cuarta generación, así que el usuario tendrá una herramienta para la fácil manipulación de este

lenguaje, y prácticamente la programación se convierte en una actividad simple, que difiere en mucho a la programación en algún lenguaje convencional de tercera generación.

2.1.2 Características de un lenguaje iconográfico

Como puede verse, el uso de herramienta de software de tipo visual, ha diversificado su aplicación en muchas ramas de otras ciencias, lo importante es que este tipo de aplicaciones, permiten una comunicación más clara entre los usuarios finales y las computadoras. Realmente el diseño de lenguajes de programación de características puramente visuales, marcan un nuevo estilo de programación.

En contraste de los ambientes de programación de los lenguajes tradicionales de tercera generación, en los que prevalece su presentación de texto como Pascal, Fortran, COBOL, Lisp, etc.; un lenguaje de iconos, contiene un alto grado de elementos pictóricos. Además se sabe que la información que asimila el ser humano en un 80%, proviene de su sentido de la vista, por lo que un lenguaje iconográfico representa el mejor modelo de percepción para el usuario, así como para su aprendizaje. En el fondo, las dos categorías texto e iconos, representan una simbología, pero en diferente escala, un icono puede representar muchas palabras, y en esto estriba la diferencia.

Se precisa [CHAPA86] que para que un lenguaje de programación esté diseñado en términos visuales, opuesto a uno de texto, debe cumplir lo siguiente:

1) Cada icono debe diseñarse a un nivel alto, esto quiere decir, que el icono represente una acción definida, de manera que el usuario lo trate como un átomo.

2) Cada átomo podrá ser manejado en la programación. Las entidades gráficas de alto nivel, se construyen de elmentos geométricos y formarán un alfabeto estándar de programación.

3) Los elementos gráficos pueden contener texto y números como componentes paramétricos, formando una parte integral del lenguaje, y no sólo como parte decorativa.

Un lenguaje de programación se denomina **iconográfico**, si el proceso de programación se basa esencialmente en seleccionar y/o componer iconos, colocándolos en la pantalla de acuerdo a una apropiada yuxtaposición [CHAPA86]. Los iconos son imágenes que tienen asociada una semántica y se organizan con reglas sintácticas, por lo que pueden usar cualquier representación como elementos del lenguaje.

Dentro del ambiente de la administración, el uso de lenguajes visuales y graficación por computadora, han mostrado bastante poderío para la documetación, entendimiento y desarrollo de sistemas. En principio uno de los procesos que es de suma importancia para la administración, es el desarrollo de

sistemas de información, y con el advenimiento de lenguajes que manipulan bases de datos, su interés por asimilarlas se ha incrementado.

Inicialmente la construcción de un sistema para bases de datos el cual contiene un interfaz de tipo icónico, fue propuesta por los diseñadores del lenguaje QBE, esta interfaz como ya hemos visto, permite a los usuarios finales programar en un ambiente gráfico. El sistema, contiene un lenguaje no procedural (de no procedimiento), para el manejo de una base de datos. Como se sabe en este lenguaje la recuperación de la información se realiza por el usuario llenando objetos gráficos que semejan tablas.

El sistema Xerox Start es también un ejemplo claro, en donde se utilizan iconos para representar varias funciones y objetos de datos, que normalmente se encuentran en los ambientes de la administración.

2.1.3 Definición de LIDA

El presente proyecto intenta entonces, diseñar un lenguaje iconográfico, donde cada elemento átomico, represente una operación sobre una base de datos relacional, dichas operaciones, se basan en las operaciones del álgebra relacional, que por tanto en cada una de ellas, concurren objetos que en este caso, son archivos de datos; por ello este lenguaje lo hemos denominado **LIDA** (*Lenguaje Iconográfico para el Desarrollo de Aplicaciones*) y se basa en el concepto de flujograma dado en [CHAPA87] y [CHAPA89], además como parte complementaria a nuestro trabajo de tesis, desarrollamos un sistema de bases de datos denominado **SABRE** (*Sistema para el Desarrollo de Aplicaciones con Bases de Datos Relacionales*), es un sistema en el que se implementan algunas herramientas de software que apoyan al lenguaje para la generación de aplicaciones, y que de hecho, desde el punto de vista CASE, es un generador de aplicaciones en el que se utiliza un lenguaje de muy alto nivel, en este caso LIDA.

Cada proceso primitivo, está representado por un símbolo o icono de diseño geométrico sencillo. Los iconos se disponen para dar un encapsulamiento más complejo dentro de una estructura, en la cual se conectarán uno entre otro, por medio de líneas de flujo de datos, resolviendo también el paso de parámetros.

Un programa en el LIDA, se traduce directamente en una gráfica, cuyos nodos representan operaciones relacionales, o bien archivos relacionales, y los arcos representan las incidencias entre los archivos y las operaciones.

El diseño de lenguajes de programación de muy alto nivel, enfocados a la administración, es un hecho que cobra gran importancia, con una visión más ámplia en el desarrollo de nuevas formas de programación y diseño, el LIDA se considera una herramienta para la elaboración de sistemas gráficos; el lenguaje esta dirigido, hacia el área de aplicación denominada procesamiento de datos para la

administración. Para su diseño se han considerado algunos conceptos de ingeniería de software, conjugada con la teoría de las relaciones matemáticas.

El lenguaje tiene el compromiso de ser diseñado, bajo la simplicidad de su utilización y la generalidad de sus aplicaciones, como un lenguaje de cuarta generación, para ello se basa en dos aspectos: La generalidad se resuelve con la aproximación de la programación automática que pretende solucionar una gran cantidad de problemas, que se presentan en el procesamiento de información, de esta forma el usuario sólo tiene que definir su problema en términos de una carta gráfica, y un parser generará la aplicación. El aspecto de simplicidad se enfrenta ante una comunicación más sencilla entre el usuario y la máquina, y esto es resuelto por la misma estructura del lenguaje, que se basa en la programación no procedural y el estilo de programación gráfica, que son los elementos fundamentales de la filosofía del lenguaje.

2.2 DIAGRAMA ICONOGRAFICO (Flujograma)

Los iconos propuestos por el LIDA, son ligados dentro de una carta gráfica denominada **flujograma** [CHAPA86]. La idea que adopta un flujograma es lo que en términos computacionales se le conoce como diagrama de flujo de datos, sólo que en este caso representa operaciones encapsuladas, superiores a las que puede mostrar un diagrama de flujo común. Los fujogramas son una herramienta para representar un sistema administrativo. La filosofía de CASE los propone como un medio para representar la planeación de actividades o procedimientos relacionados con el procesamiento de datos. En suma los flujogramas se pueden utilizar para dos propósitos:

- *Describir la estructura administrativa de una empresa, y*

- *Procesar información*

En este caso LIDA utiliza los flujogramas para encapsular de alguna manera, las diferentes operaciones relacionales que forman parte de un procedimiento o aplicación. LIDA también es considerado un lenguaje de flujograma, debido a las características propias del lenguaje.

Por lo que un flujograma dentro de LIDA, representa una precisa relación funcional entre los objetos que son operandos y operadores relacionales; a su vez dichos objetos se muestran como entes iconográficas. Cada una de las líneas de flujo que conecta un icono y otro, define la semántica del lenguaje, y de alguna manera el flujo de la información, adicionalmente la forma de cada icono resuelve la parte sintáctica.

Las ventajas que ofrece la utilización de un flujograma dentro de un sistema de información, son las descritas por las herramientas CASE de tipo visual. El esqueleto del flujograma, puede mostrar el desarrollo de cualquier aplicación, LIDA utiliza el flujograma como una estructura para transmitir el mensaje semántico.

2.3 SEMANTICA

La semántica de LIDA se define por el mismo significado que se muestra en cada uno de sus iconos primitivos y las relaciones entre estos. Como se mencionó anteriormente LIDA utiliza la herramienta de flujograma para armar una carta gráfica donde su estructura muestre las relaciones entre operandos y operadores que son representados por iconos, estas relaciones se representan gráficamente por arcos dirigidos que conectan entre sí los iconos del lenguaje.

Como se sabe, cada icono representa un operando o un operador relacional; cuando se trata de la representación de un operador, no existe mayor problema en distinguirlos, dado que los operadores son únicos; pero cuando intentamos distinguir los operandos encontramos que estos pueden ser el resultado de un procedimiento (que es una relación), o bien una relación; cuando el operando es una relación, entonces ésta se encontrará representada como un icono, pero aquí se presenta un problema: tal vez debería existir un icono diferente por cada relación para poder distinguir entre una relación y otra, lo que no es posible, dado que el número de relaciones es infinito. Para resolver este problema, LIDA utiliza parámetros dentro de sus iconos.

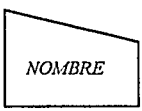
De esta manera, algunos iconos contienen parámetros que pueden ser expresiones aritméticas, booleanas, o cadenas de caracteres; y ya que quedan inscritos dentro de los mismos iconos, la semántica de LIDA se divide en dos partes, la semántica de los parámetros, y la semántica iconográfica.

2.3.1 Semántica paramétrica

Es un hecho que cada icono mantiene intrínsecamente su semántica predefinida, pero la estructura de sus parámetros modifica la semántica del propio icono, así por ejemplo el ICONO ARCHIVO, se ha diseñado para representar una relación pero depende de su parámetro, que en este caso es el nombre de la relación misma, que define el archivo que la contiene; por ejemplo si este icono contiene el parámetro EMPLEADOS, podría representar la relación de los empleados de una empresa, y el mismo icono con el parámetro SUELDOS, podría representar el registro de los sueldos de cada uno de los empleados.

Debido a que no todos los iconos de LIDA llevan parámetros, nos concentraremos en definir la semántica referida a los iconos que si los contienen:

DEFINICION DE ARCHIVO

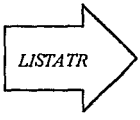


Este icono sólo lleva un parámetro y representa el *NOMBRE* de la relación.

SELECCION

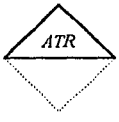
El icono selección requiere de una expresión booleana *EXPBOOL* como parámetro y su semántica obedece a la semántica convencional de una expresión booleana, incluyendo paréntesis y las precedencias de los operadores comunes; los operandos de la expresión son los atributos de la relación en cuestión y constantes numéricas o alfanuméricas que representan valores del dominio válidos.

PROYECCION



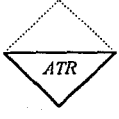
Este icono lleva como parámetro la lista de atributos *LISTATR* sobre los cuales se realizará la operación.

ORDENACION ASCENDENTE



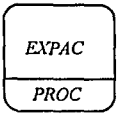
El único parámetro, es el aributo llave *ATR* como referencia para la ordenación.

ORDENACION DESCENDENTE



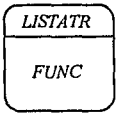
El parámetro para esta operación es análogo al del icono anterior.

ACTUALIZACION



Este icono requiere de dos parámetros, uno de ellos es una expresión de actualización *EXPAC* representada por medio de una ecuación, donde el lado izquierdo de la igualdad es el nombre del atributo a actualizar y el lado derecho una expresión aritmética convencional; y el otro el nombre *PROC* del icono para identificación del procedimiento de actualización.

ICONO FUNCION



El icono función requiere del nombre *FUNC* de la función, y el nombre de los atributos *LISTATR* sobre los cuales opera la función, las funciones agrupadas (sus nombres tienen como sufijo la letra "A"), se aplican sobre relaciones que primeramente se agrupan con respecto a un atributo y posteriormente se aplica la función para cada grupo o subconjunto de tuplas, la salida de estas funciones agrupadas representa una relación, el resto de las funciones devuelven un valor numérico. Los parámetros deben separarse por una coma (en el caso de las funciones agrupadas, en orden primero debe aparecer el nombre del atributo sobre el que se aplica la función y después el nombre del atributo sobre el cual se agrupa la relación, separado por una coma).

A saber se tienen las siguientes posibles funciones:

Función	*Semántica*
MAX	Obtiene el máximo valor sobre un atributo
MIN	Se obtiene el mínimo valor sobre un atributo
PROM	Calcula el promedio de los valores de un atributo
SUM	Suma todos los valores de un atributo

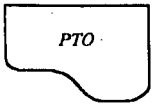
Función	*Semántica*
REGS	Obtiene la cardinalidad de una relación, el parámetro es el nombre de la relación
MAXA	Obtiene el conjunto de tuplas máximas sobre un atributo de una relación agrupada con respecto a un atributo, de manera que cada tupla es el máximo de cada grupo.
MINA	Obtiene las tuplas que con respecto a un grupo contienen el atributo mínimo. La relación sobre la que se ejecuta esta función se agrupa sobre un atributo en específico.
PROMA	Calcula el promedio con respecto a un atributo de cada grupo de tuplas. La relación sobre la que actúa, se agrupa sobre un atributo determinado.
SUMA	Suma todos los valores de un atributo, por grupo de tuplas. La relación se agrupa con respecto a un atributo.
REGSA	Obtiene el número de tuplas de cada grupo, la relación se agrupa con respecto a un atributo dado como parámetro.

DEFINICION DE VARIABLE

VAR

El parámetro para este icono, es el nombre *VAR* de una variable numérica, destinada a contener el valor resultado de una función. A éste tipo de variables se les da el nombre de *Variables de Memoria.*

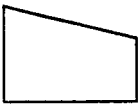
IMPRESION



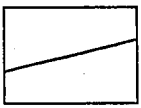
El parámetro *PTO* requerido, es el nombre del puerto o del archivo de salida hacia el cual se dirige la impresión.

2.3.2 Semántica iconográfica.

Cada icono de LIDA, significa una operación relacional o una relación, por lo que semánticamente se tiene:



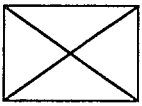
Este icono representa una relación. Si un arco sale de él, entonces implica que la relación a la que simboliza existe, y su estructura se encuentra definida en un descriptor, de lo contrario, si un arco llega al icono, significa que la relación será creada o regenerada al igual que su descriptor.



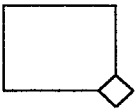
Representa la operación de unión del álgebra relacional. Para ello requiere de dos bases de datos como operandos, las cuales deben ser compatibles bajo la unión (deben tener los mismos atributos), y son dadas por los arcos que llegan al icono. Como resultado produce otra relación que se representa por el arco de salida de este icono.

Significa la diferencia algebraica entre la relación que se conecta mediante el arco de entrada que aparece más a la izquierda, menos la relación que se conecta a la derecha del icono. Como resultado produce otra relación que se representa por el arco de salida de este icono.

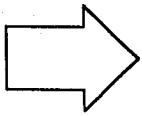
La operación de intersección del álgebra relacional, es indicada a través de este icono. Los arcos de entrada deben provenir de las relaciones operando. Como resultado produce otra relación que se representa por el arco de salida de este icono.



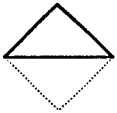
Los arcos que llegan al icono deben provenir de dos relaciones, las cuales se van a juntar. La operación a realizarse es la conocida como *junta natural* del álgebra relacional. Como se sabe, las dos relaciones mencionadas deben coincidir en al menos un atributo. Su salida es nuevamente otra relación y es representada por el arco que sale del icono.



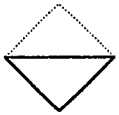
El icono selección, efectúa la operación de selección del álgebra relacional sobre la relación de la que proviene el arco que llega a este icono. La condición de selección origina dos relaciones, una que cumple la condición (expresión booleana), y otra que no la cumple. Para diferenciar la parte verdadera de la falsa, se toma como referencia el tipo de línea de los arcos que salen del icono; las líneas continuas definen la parte verdadera y las punteadas la parte falsa.



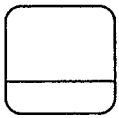
Este icono representa la proyección sobre una relación. El arco que llega a él, hace referencia a la relación a proyectar, y su parámetro es la lista de los atributos proyectados. Como resultado se crea otra relación con columnas idénticas a la relación que le dió origen.



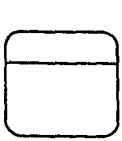
A través de este icono, es posible realizar el ordenamiento de una relación en forma ascendente. El arco que llega al icono representa la relación a ordenar. Como resultado produce una relación ordenada ascendentemente con respecto al atributo declarado como llave.



Su función es análoga a la del icono anterior, pero esta vez la ordenación es descendente.



Este icono representa una operación de actualización sobre todas las tuplas de una relación. El arco que llega al icono referencia la relación a operar. La actualización se realiza, de acuerdo a la instrucción que entra como parámetro; el nombre para el procedimiento sirve para identificar al icono, por ende, los arcos que salen de este icono, representan la relación de entrada con un atributo actualizado sobre la relación.



LIDA define ciertas funciones que operan sobre las bases de datos relacionales. Estas funciones originan un valor numérico o una relación. El arco de entrada al icono representa la relación sobre la cual se aplica la función, y el o los arcos de salida, la relación o el valor resultante de la función.

Este icono tiene una relación muy estrecha con el icono de función, ya que su único significado es crear una variable en memoria (*variable de memoria*), para almacenar el valor que se obtiene al efectuar una función, sobre una relación. El arco de entrada, debe provenir de un icono de función u otro icono de variable. El arco de salida sólo podrá dirigirse hacia otro icono de variable, puesto que el flujo representa un dato.

Existen tres dispositivos para poder enviar un reporte. Este icono permite enviar un reporte hacia el monitor de la computadora o consola, la impresora y hacia un archivo ASCII estándar.

2.4 SINTAXIS

En un lenguaje iconográfico, la sintaxis se basa sobre la forma de dibujar adecuadamente los iconos, LIDA conserva una sintaxis bien definida por un conjunto de iconos de forma singular; no obstante se diseña con características paramétricas, de tal suerte que por ejemplo, el icono ARCHIVO representa la relación que nombra su parámetro, esto lleva consigo que la sintaxis de LIDA, se divida también en sintaxis de sus parámetros y sintaxis de sus iconos. De manera que un icono instancia un objeto o un operación relacional, y éstos quedan bien declarados por los parámetros que se le definen.

2.4.1 Sintaxis paramétrica.

En seguida daremos a conocer ciertas características sintácticas que son válidas para los parámetros LIDA.

Cada icono, según lo requiera, puede contener parámetros como lo son: nombres de bases de datos, condiciones booleanas, operaciones aritméticas y símbolos especiales. Por ejemplo:

```
(SUELDO <= 230000) y (DEPTO = 'INFORMATICA')

(SUELDO > (2345/SALBASE*34) )

ISPT = SUELDO*SALARIO

NOMBRE,EDAD,RFC->CLAVE,DOMICILIO

EMPLEADOS
```

son parámetros válidos en LIDA.

LIDA subdivide los parámetros en dos tipos:

- *Nombres*
- *Parámetros operacionales*

Los nombres nominan generalmente las bases de datos en juego.

Los parámetros operacionales, pueden constituirse por cadenas que representan operaciones y que se componen de expresiones aritméticas y booleanas, constantes, y nombres de dominios de las bases de datos definidas; también lo pueden ser caracteres o símbolos paramétricos. A continuación daremos la relación de los iconos que utilizan parámetros y la sintaxis de estos últimos:

- El parámetro del icono de **definición de archivo** es una cadena alfanumérica

- Par efectuar la operación de **selección**, se requiere de una expresión booleana convencional que se construye por los siguientes operadores booleanos:

O ,o	"o" lógico
Y ,y	"y" lógico
<	Menor que
>	Mayor que
=	Igual
<=	Menor o Igual
>=	Mayor o Igual
<>	Diferente
( )	Paréntesis

y con operandos :

- Constantes numéricas
- Constantes alfanuméricas
- Expresiones aritméticas entre paréntesis
- Atributos
- Variables de memoria
- Expresiones booleanas

Los operadores booleanos guardan las siguientes precedencias de mayor a menor:

1. ( )
2. y
3. o
4. <, >, =, <=, >=, <>

Las constantes numéricas pueden ser cualquier número real, y las constantes alfanuméricas deben encerrarse entre apóstrofes ' '.

Los atributos son cadenas de caracteres alfanuméricos, y pertenecen al nombre del atributo de una relación existente.

Las variables de memoria se constituyen por una cadena alfanumérica, y deben ser creadas previamente por medio del icono de función.

Una expresión aritmética en LIDA se construye por los siguientes operadores aritméticos convencionales:

+	suma
-	resta
*	multiplicación
/	división
( )	paréntesis

y los operandos pueden ser:

- Constantes numéricas
- Atributos
- Variables de memoria
- Expresiones aritméticas

las siguientes expresiones aritméticas son válidas:

```
SALMINIMO * 1.34/SALBASE
(SUELEMPL * 123234.456 + (SALMINIMO/200))
CARGO * CREDITO + (345/45*34.556)
```

Los operadores aritméticos guardan las precedencias indicadas de mayor a menor:

1. ( )
2. * /
3. + -

De esta forma las siguientes expresiones booleanas son también válidas:

```
(SAL > SUELDOMINIMO) y (DESC < (345.8*89/900+(674-400)) )
(nombre = 'Héctor') o (nombre = 'Rubén')
(DIRECCION = '0906')
```

- La lista de atributos sobre los cuales se realiza la **proyección** debe seguir la siguiente sintaxis:

atr_1(->$ratr_1$), atr_2(->$ratr_2$), ...,atr_n(->$ratr_n$)

donde atr_1,..,atr_n son atributos de la relación y
$ratr_1$,..,$ratr_n$ son los atributos que renombran opcionalmente la primera lista de atributos. El símbolo -> indica renombra.

- El parámetro requerido para las **operaciones de ordenamiento** debe ser el nombre del atributo llave.

- Para el proceso de **actualización**, el primer parámetro es la ecuación de actualización y tiene la siguiente sintaxis:

atri=exparit

donde: atri es el nombre del atributo a actualizar y se constituye por una cadena alfanumérica.

exparit es una expresión aritmética, como la descrita en la operación de selección.

El segundo parámetro es el nombre del icono, y debe ser una cadena alfanumérica.

- El nombre del icono **función**, es el nombre de la función y es una cadena que puede ser cualquiera de los siguientes mnemónicos:

MAX, MIN, PROM, SUM, REGS, (funciones normales)
MAXA, MINA, PROMA, SUMA y **REGSA** (funciones agrupadas)

como parámetros del icono se requiere el nombre del atributo sobre el cual opera la función, que debe ser una cadena alfanumérica (las funciones REGS y RESGSA, no llevan este parámetro), y para las funciones agrupadas es requerido un segundo parámetro (separado por coma), que representa el nombre del atributo sobre el cual se agrupa la relación.

- El parámetro de la definición de variable, es el nombre de una variable de memoria numérica y debe ser una cadena alfanumérica.

- El parámetro para el icono de **impresión** puede ser:

CON o ningún caracter, para el monitor.
LPT1 para direccionar el reporte a la impresora.
El nombre de algún archivo estándar ASCII.

2.4.2 Sintaxis iconográfica

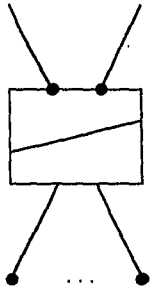
La sintaxis de cada icono se define por su misma estructura, a continuación mostraremos los iconos que conforman LIDA con cada una de sus líneas de flujo requeridas de entrada y salida:

ICONO ARCHIVO

ENTRADA: Una línea de flujo (opcional) que proviene de otro icono.

SALIDA: Una o varias líneas de flujo que se conectan hacia otros iconos.

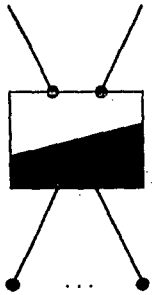
ICONO UNION



ENTRADA: Dos líneas de flujo, cada una proviene de diferentes iconos tomados como relaciones, para que se efectúe la operación de unión relacional.

SALIDA: Una o varias líneas de flujo.

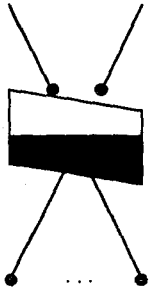
ICONO DIFERENCIA



ENTRADA: Dos líneas de flujo, cada una proviene de diferentes iconos tomados como relaciones, para que se efectúe la operación de diferencia relacional.

SALIDA: Una o varias líneas de flujo.

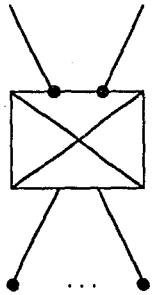
ICONO INTERSECCION



ENTRADA: Dos linas de flujo, cada una proviene de diferentes iconos tomados como relaciones, para que se efectúe la operación de intersección relacional.

SALIDA: Una o varias líneas de flujo.

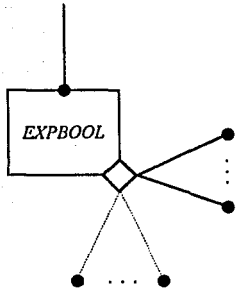
ICONO JUNTA



ENTRADA: Dos líneas de flujo, cada una proviene de diferentes iconos tomados como relaciones, para que se efectúe la operación de junta natural.

SALIDA: Una o varias líneas de flujo.

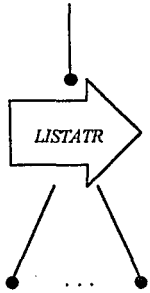
ICONO SELECCION



ENTRADA: Una línea de flujo, que proviene de un icono tomado como relación, al cual se le aplicará la operación de selección relacional.

SALIDA: Una o varias líneas de flujo. Estas pueden ser de dos tipos punteadas y continuas.

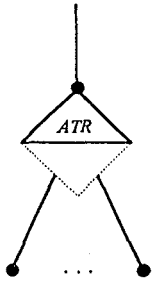
ICONO PROYECCION



ENTRADA: Una línea de flujo, que proviene de un icono tomado como relación, al cual se le aplicará la operación de proyección.

SALIDA: Una o varias líneas de flujo.

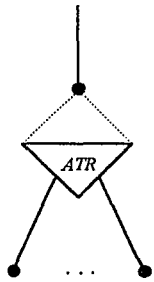
ICONO ORDENA ASCENDENTE



ENTRADA: Una línea de flujo, que proviene de un icono tomado como relación, al cual se le aplicará la operación de ordenación ascendente.

SALIDA: Una o varias líneas de flujo.

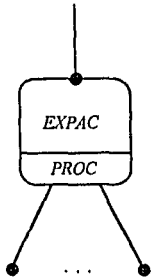
ICONO ORDENA DESCENDENTE



ENTRADA: Una línea de flujo, que proviene de un icono tomado como relación, al cual se le aplicará la operación de ordenación descendente.

SALIDA: Una o varias líneas de flujo.

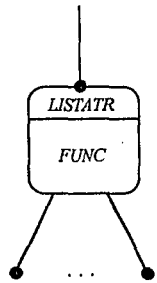
ICONO ACTUALIZA



ENTRADA: Una línea de flujo, que proviene de un icono tomado como relación, al cual se le aplicará un proceso determinado por una expresión de actualización.

SALIDA: Una o varias líneas de flujo.

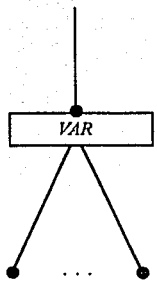
ICONO FUNCION



ENTRADA: Una línea de flujo, que proviene de un icono tomado como relación, al cual se le aplicará una función predefinida.

SALIDA: Una o varias líneas de flujo.

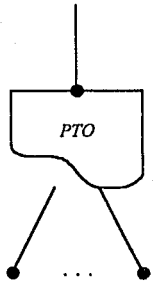
ICONO VARIABLE



ENTRADA: Una línea de flujo, que proviene de un ICONO FUNCION.

SALIDA: Una o varias líneas opcional(es) de flujo direccionadas hacia otro ICONO VARIABLE.

ICONO IMPRIME



ENTRADA: Una línea de flujo, que proviene de una relación, de la cual se generará un reporte.

SALIDA: Una o varias líneas opcional(es) de flujo que representan la relación que se imprimió.

CAPITULO 3: ESQUEMA DE DATOS

Cada uno de los niveles de un sistema de bases de datos cuenta con un esquema para ser descrito. Por lo tanto, tomando esta filosofía, hemos desarrollado una herramienta conocida como descriptor de archivos [CHAPA86] que sirve como esquema de datos dentro del sistema integrado SABRE. Además, hemos incluído un captador de descriptores y un captador de datos, utilizados para la implementación de dicha herramienta. Tanto el descriptor como los captadores serán analizados en este capítulo.

3.1 COMPOSICION DEL ESQUEMA DE DATOS

En todos los niveles de CASE existe un *diccionario* en el cual se registra la información de las características de cada nivel. En el nivel intermedio e inferior sirve como esquema para formar la estructura, documentación y programación de los sistemas y bases de datos respectivas. Existe una herramienta no tan amplia como lo es un diccionario, pero si tan efectiva para ser utilizada como esquema: nos referimos al *descriptor de archivos*. De hecho, el descriptor de archivos cuando es planteado como imagen de un captador, se considera como un diccionario debido a que sobre éste se define la semántica esencial de los atributos.

3.1.1 Descriptor de archivos

Un *Descriptor de Archivos* (DA), es un esquema que nos permite el acceso a los datos de un archivo en forma sencilla y eficiente.

Propiamente un DA es un registro D, cuyos campos relacionan la descripción de los registros de un archivo de datos, o dicho de otra manera, es un registro que describe las características físicas de un archivo, su estructura de datos, la identificación de cada uno de sus atributos (o campos) y la descripción de cada uno de ellos.

Como se menciona en el primer capítulo, la arquitectura de un sistema de bases de datos se compone de tres niveles: externo, conceptual e interno; y cada uno de ellos se puede contemplar como una vista y un esquema, donde la vista es lo que se puede percibir o ver del contenido de una base de datos, y el esquema es la definición de esa vista.

Pues bien, un conjunto de DAs puede tratarse como un esquema externo, y el conjunto de todos estos esquemas puede constituir el esquema conceptual. Además, el DA contempla las características de acceso a los datos, así como su estructura, por lo que también en su caso puede llegar a constituir en su conjunto el esquema interno.

Es posible conjuntar una serie de DAs en un archivo, al que se le conoce como *archivo de descriptores*, por lo cual, este archivo es un objeto más complejo constituido entonces por una secuencia de registros, en donde cada uno de ellos determina un descriptor para un archivo de datos. El archivo de descriptores tiene la misma configuración en sus registros, siendo uniforme en este aspecto. Por lo tanto, podemos considerar al DA propiamente un esquema de datos que conforma esquemas más completos.

Un DA puede constituirse por los siguientes campos agrupados:

- Nombre del archivo
- Tipo de organización
- Tipo de acceso
- Número de registros
- Tamaño del registro
- Número de campos llave
- Lista de campos llave
- Número de campos
- Lista de campos (LDC)

donde cada elemento de LDC contiene a su vez los campos:

- Nombre
- Apuntador inicial
- Apuntador final
- Tipo

Teniendo un registro como éste, es fácil lograr la recuperación de los datos de un archivo; para ello, el procedimiento se lleva a cabo por medio de una función conocida como **función VAL.** Esta función actúa sobre un archivo A, el cual tiene registros del tipo (v1,v2,v3,...,vn), donde cada vi representa un campo. De la misma manera, la parte LDC del registro descriptor puede verse como (d1,d2,d3,...,dn), donde cada di es la descripción de los datos que ocupan el campo i de cada registro del archivo A (Figura 3.1):

Nombre i	Corresponde al nombre del i-ésimo campo.
Apuntador inicial i	Es la columna del comienzo del i-ésimo campo.
Apuntador final i	Es la columna donde termina el i-ésimo campo.
Tipo i	Se refiere al tipo de dato que constituye el i-ésimo campo, generalmente se consideran tres tipos básicos: numérico, alfabético, y alfanumérico.

Nombre del archivo	Tipo de organización	Tipo de acceso	Número de registros	Tamaño del registro	Número de campos llave	Número de campos	Campo 1	Apuntador inicial 1	Apuntador final 1	Tipo de campo 1
							Campo i	Apuntador inicial i	Apuntador final i	Tipo de campo i

Figura 3.1 Estructura del Descriptor de Archivos

3.1.2 Acceso a los archivos

Como puede notarse el DA especifica las características fundamentales para dar acceso y tratar un archivo de datos cualquiera; así, el nombre del archivo servirá para reconocerlo en el directorio de archivos físicos; su organización y tipo de acceso indicarán a la función VAL la manera de entrar a los registros; el número de registros y el tamaño, podrán en algún momento ser utilizados para determinar la cantidad de memoria que puede ocupar el archivo; el número de campos llave y la lista de los mismos, serán requeridos dependiendo del acceso; y finalmente, el número de campos y la descripción de estos, representan la parte que se relaciona directamente con cada uno de los datos con fines de recuperación.

En términos generales, para lograr el acceso a un archivo se realiza lo siguiente (Figuras 3.2 y 3.3):

- Se efectúa una petición de acceso a un archivo, la cual será desprendida del procedimiento llevado a cabo en una aplicación.

- Se busca en el archivo de descriptores, el registro descriptor correspondiente al archivo que va a ser accesado.

- Se toman los parámetros correspondientes para abrir el archivo y accesar el campo correspondiente, tomando la posición y el tipo del campo.

- Se procede a accesar el archivo, recuperando la información del campo que se asocia con los parámetros que se obtuvieron del descriptor.

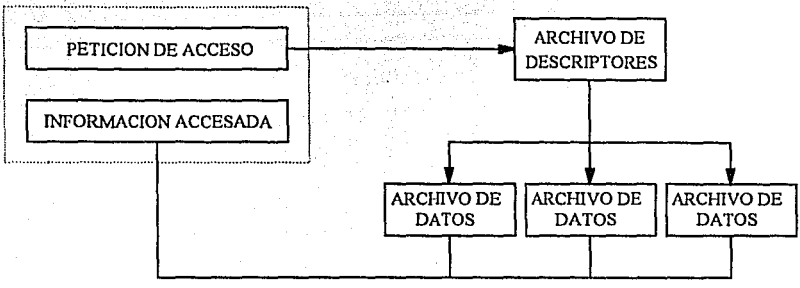


Figura 3.2 Esquema general de acceso

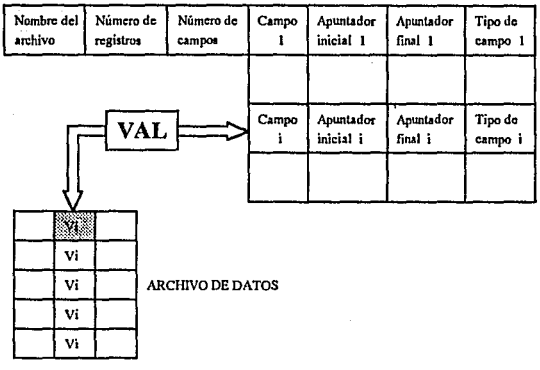


Figura 3.3 Proceso de recuperación de la función VAL

En estos términos, el acceso se hace independiente de sus propias características, es decir, la petición se realiza sobre un campo de un archivo determinado, no importando el método de acceso y la estructura de datos. Las aplicaciones pueden entonces trabajar con toda libertad para manejar cualquier objeto de un archivo, sin determinar la forma de como llegar a él. Por otro lado, se considera la función VAL como un procedimiento evaluador, el que recupera de forma automática los valores de todo un atributo en una relación, de tal suerte que se puede realizar una selección adecuada de alguno de estos valores para dar solución a una consulta.

3.1.3 El descriptor y SABRE

Recordemos que parte del presente trabajo lo hemos dedicado al desarrollo del sistema SABRE que soporta a LIDA como lenguaje para generar aplicaciones. La herramienta del descriptor se ha incluído en SABRE, para que por medio de ella LIDA maneje eficientemente las relaciones.

Las características propias de los archivos que dicta el enfoque relacional, nos llevaron a concluir que un archivo de organización secuencial, es adecuado para el manejo de bases de datos relacionales. Se puede hacer una analogía entre una tabla y un archivo donde sus columnas representan los campos, y las tuplas representan los registros; de esta forma las operaciones relacionales sobre dichos archivos se realizan de manera óptima.

Ya que el DA es una herramienta básica para el manejo de los archivos, y estos a su vez deben soportar las características relacionales, entonces se define un DA, el cual contemple lo más substancial para poder trabajar con archivos relacionales.

Se define el siguiente DA que se toma como base en el sistema SABRE (Figura 3.3):

Nombre del archivo
Número de registros
Número de campos
Lista de campos (LDC)

donde cada elemento de LDC contiene a su vez los campos:

Nombre
Apuntador inicial
Apuntador final
Tipo

El descriptor señalado no es tan amplio como el presentado con anterioridad, sin embargo, es suficiente para poder operar dentro del sistema. Cabe señalar que el descriptor carece de la lista donde se indican los campos llave, que en términos de las bases de datos relacionales son primordiales. No obstante, el presente trabajo tiene como objetivo mostrar las características de un lenguaje iconográfico que maneje bases de datos relacionales, no así un sistema íntegro de bases de datos. Además esta parte puede servir para el desarrollo de futuros algoritmos que permitan un adecuado manejo de archivos secuenciales indexados, e incluso la normalización de bases de datos. En lo sucesivo consideraremos bases de datos ya normalizadas [HERNANDEZ].

3.2 EL CAPTADOR

Así como el descriptor de archivos llega a ser una herramienta de software para la definición de los datos, un captador es una utilería que permite la captura de datos de manera eficiente y amigable para el usuario, el sistema integrado SABRE cuenta con un captador de descriptores y un captador de datos, herramientas que hacen las veces de un DML, dentro del sistema de bases de datos.

3.2.1 El captador de descriptores

El captador de descriptores es una herramienta que hace las veces de un DML de LIDA, ya que por medio de éste es posible definir y actualizar los descriptores de archivos. Cabe señalar que hasta el momento no existe un lenguaje gráfico formal estándar, con el cual se pueda programar gráficamente, de manera que LIDA no puede considerarse como un sublenguaje de datos, el cual pueda estar inmerso en un anfitrión. LIDA debe ser definido por el momento como un lenguaje gráfico que sirve para desarrollar aplicaciones sobre bases de datos relacionales.

SABRE por su parte interpreta cada flujograma de LIDA y genera código en lenguaje ISBL Extendido (que explicaremos en el siguiente capítulo) por lo cual, ISBL Extendido es visto como una librería interna de DML. Como complemento utiliza el captador de descriptores, donde cada descriptor es en sí el DDL de LIDA, ya que por medio de ellos se definen los datos o estructuras de cada base. Por lo anterior LIDA es un lenguaje, compuesto por un DML y un DDL, sin llegar a considerarse como un sublenguaje de datos formal. Más aún, como hemos mencionado, el lenguaje y el sistema SABRE debe considerarse como un producto de CASE intermedio e inferior.

El captador de descriptores, presenta dentro de una ventana virtual los atributos de una relación en forma de lista; de hecho, esta lista es la única que se puede modificar directamente, ya que en ella se definen las características de los atributos de cada descriptor. Sólo se requiere que sean capturados los nombres del archivo y sus campos, considerando para cada uno de estos últimos su tipo y longitud, con los que el captador arma el resto del descriptor.

3.2.2 El captador de datos

En los sistemas manejadores de bases de datos, se cuentan con diferentes herramientas que permiten actualizar los datos de forma directa, dichas herramientas son presentadas como utilerías del sistema, en la mayor parte de los sistemas integrados, las encontramos como módulos dentro del sistema. El captador es una herramienta de captura de datos que permite una interface cómoda y práctica para el usuario.

El captador de datos es una herramienta en la cual se representan los registros, y donde es posible actualizar los datos. Así el captador de datos permite realizar la actualización de los datos de manera simple. El captador utiliza el descriptor de archivos para crear una ventana virtual en la cual se muestran de manera columnar los atributos de una relación, presentándolos como tablas de registros que el usuario actualiza directamente.

Además el captador trabaja bajo la premisa de independencia de datos, ya que un cambio en la estructura de una base de datos no afecta los valores ya capturados, a menos que la parte modificada sea un atributo que previamente existe y que sus nuevos valores sean una incongruencia, o que los nuevos valores sean nulos.

El captador de datos desde el punto de vista de los sistemas de bases de datos es un DML, ya que por medio de él se manipulan los datos, es decir, se actualiza el contenido de las bases de datos en forma directa, y no por medio de instrucciones o comandos que requieren un procesamiento adicional.

Hasta el momento, existen diferentes dispositivos de hardware que agilizan la captura de datos, pero por ahora la captura de información por medio del teclado es la más común, pese a que existen sistemas de reconocimiento tipográfico y de barrido, así como de lectura óptica que de alguna forma pueden ser una opción de captura.

Dentro de un sistema de base de datos que utilice LIDA u otro lenguaje gráfico, como lenguaje principal para la manipulación de información, puede ser diseñado e implementado dentro de un sistema de ambiente gráfico, por lo cual, en la captura de datos se procurará que también sea desarrollada de esta manera. Un captador de datos es una buena opción para complementar un sistema de esta naturaleza.

Tanto el captador de descriptores, como el captador de datos, no pueden ser vistos como subsistemas o aplicaciones en línea, dispuestos para auxiliar al usuario; pero su fin es equiparado con lo que se conoce como DML de un sublenguaje de datos dentro de un sistema de bases de datos.

CAPITULO 4: LENGUAJE ISBL EXTENDIDO

En un sistema de bases de datos de alto nivel, cada aplicación es expresada por medio de un lenguaje sencillo para el usuario, y en el cual la representación y manipulación de los datos se logra de una manera igualmente sencilla [TODD].

La compañía IBM, en uno de sus centros de investigación, ha dedicado una gran parte de su tiempo al desarrollo de sistemas piloto del tipo mencionado. Uno de esos sistemas es el denominado *Peterlee Relational Test Vehicle* (PRTV).

PRTV [TODD] es un sistema interactivo de base de datos que puede usarse completamente como un sistema de mantenimiento de bases de datos o como un subsistema de datos para un sistema de aplicaciones. Sus objetivos principales son, además de ofrecer un sistema de alto nivel, prestar un soporte flexible para el mantenimiento de bases de datos y proveer una extensa funcionalidad. Para lograr ser un sistema de alto nivel, intenta ser capaz de soportar los conceptos basados en el modelo relacional; para ofrecer una máxima flexibilidad en el uso de bases de datos, los archivos son tratados como variables con un nombre específico, estos archivos son manejados a través de un lenguaje de usuario denominado *Information System Base Language* (ISBL).

4.1 EL LENGUAJE ISBL

Uno de los elementos más importantes del sistema PRTV, es ISBL, ya que, el usuario puede ver al sistema a través de él. ISBL es un lenguaje de cadena que provee variables, expresiones y asignaciones en la misma forma como lo pueden hacer los lenguajes de programación convecionales tales como COBOL. Todas las variables denotan relaciones, y las operaciones sobre éstas producen a su vez relaciones como resultado.

4.1.1 Características generales

Para las instrucciones de asignación, ISBL provee un control de sentencias para crear nuevos dominios, compartir relaciones entre usuarios, y funciones similares. ISBL no cuenta con instrucciones de control de flujo tales como DO WHILE o GO TO.

Para los casos en que las operaciones no pueden ser tomadas como una operación del álgebra relacional, ISBL permite hacer una liga con el lenguaje PL/1 para ser usada como una extensión. Existen rutinas estándar cerradas desde ISBL en PL/1 que permiten cargar y listar una relación, o bien realizar una operación aritmética. Estas rutinas pueden considerarse como sentencias o macroinstrucciones ISBL; incluso por medio de estas rutinas es posible accesar datos de una relación.

Por las características mencionadas [TODD], consideramos que ISBL es un lenguaje que puede ser utilizado como lenguaje intermedio para la interpretación y ejecución de programas en LIDA.

A continuación explicaremos la forma como ISBL trabaja dentro del sistema PRTV, y posteriormente analizaremos su funcionalidad dentro de LIDA.

4.1.2 ISBL dentro del sistema PRTV

La unidad básica en el contexto de PRTV es llamada objeto. Los objetos son agrupaciones dentro de un conjunto llamado dominio [TODD], donde cada uno de estos dominios cuenta con un nombre, por ejemplo la figura 4.1. En una aplicación para una librería, puede haber un dominio llamado *NOMBRE*, que consiste de títulos de libros y nombres de autores, y puede haber un segundo dominio que cubra las claves de adquisición *CLAVES*.

CLAVE	AUTOR	TITULO1	◄ Selector o atributos
CLAVE	NOMBRE	NOMBRE1	◄ Dominios
5 7	Austen Goethe	Persuación Fausto	◄ Objetos

Figura 4.1 Las relaciones son consideradas como tablas

Cada objeto es tomado por el sistema como un dato numérico o alfanumérico, y se dice que tiene tipos de datos N o C respectivamente. Todos los objetos en un dominio deben tener el mismo tipo de dato. Un dominio puede ser creado en cualquier momento; por ejemplo la instrucción ISBL:

```
CREATE DOMAIN NOMBRE,C
```

Puede ser usado para crear el dominio llamado NOMBRE con un tipo de dato C, (es decir de objetos alfanuméricos).

La información reunida por un conjunto de objetos es tomada por medio de relaciones, de tal suerte que una relación es una tabla similar a la presentada en la figura 4.1. Esta relación representa la información del libro número 5 Persuación de Austen, y el número 7 Fausto de Goethe. Cada renglón de la tabla es una tupla de la relación, y cada columna es una componente. Todos los elementos de una columna deben estar definidos dentro del mismo dominio. La lista de componentes de los cuales los objetos de una tupla o relación son dados, se denomina tipo de relación (o simplemente tipo).

Una relación puede ser asignada por un nombre de variable. Para hacer referencia a sus atributos, deberá aparecer el nombre de la relación a la izquierda de los mismos. Llamaremos LIBROS a la relación que hemos ejemplificado anteriormente.

Cada componente de una relación es asociada con un nombre llamado selector o atributo, y por lo regular, el nombre del dominio es el mismo que el selector. Cuando un dominio corresponde a varios componentes, entonces los selectores deben tener diferentes nombres. Usualmente los nombres de los dominios no son incluidos en la representación tabular de la figura 4.2, para evitar confusiones.

AUTOR	TITULO	CLAVE
Goethe	Fausto	7
Austen	Persuación	5

Figura 4.2 En una relación, el orden de las tuplas y componentes es arbitrario

Pese a las reglas de las relaciones, el orden de los componentes para PRTV no importa, de manera que las tablas de las figuras 4.1 y 4.2 representan la misma relación.

Para describir una relación, los nombres del dominio y del selector deben estar dados para cada componente. Cada par de nombres dominio/selector es separado por dos puntos, y el nombre del dominio es separado por una coma del nombre del selector. La relación representada en la primera tabla de nuestro ejemplo podría ser descrita por:

```
LIBROS (CLAVE, NOMBRE:AUTOR, NOMBRE:TITULO)
```

Como puede verse, cuando el nombre del dominio y del selector es el mismo, sólo se pone una vez.

Las tuplas no pueden ser repetidas en una relación, así que para una relación como la descrita, no hay forma de representar dos copias por ejemplo del libro Persuasión. Como se sabe, el número de componentes de una relación se le denomina grado, y al numero de tuplas cardinalidad.

4.1.3 Sintaxis y semántica de ISBL

ISBL contempla seis operaciones principales, que actúan sobre las relaciones para producir asimismo relaciones como resultado. Las operaciones son las siguientes [TODD]:

Operación	Operador
Selección	:
Proyección	%
Unión	+
Intersección	.
Diferencia	-
Junta	*

Estas son las operaciones que tradicionalmente se asocian con el álgebra relacional, pero sus acciones han sido adecuadas para trabajar dentro de ISBL. Esto simplifica la manera de programar con el lenguaje, así como la implementación de las aplicaciones realizadas en él. Otras operaciones que dan información acerca de las relaciones, como lo son el grado, selectores, dominio y cardinalidad, también producen relaciones. Esta característica simplifica aún más el lenguaje.

SELECCION (:)

Actúa sobre una relación y crea una nueva relación del mismo tipo. La nueva relación es un subconjunto de su antecesora, todas las tuplas de esta nueva relación satisfacen un criterio llamado filtro. Los filtros pueden contener comparaciones entre los objetos de dos componentes de una tupla, o entre un objeto de una tupla y una constante. Los objetos que se van a comparar deben ser del mismo dominio. Si el dominio es numérico, las comparaciones pueden ser: =, >, >=, <, <=, o <>. Para datos alfanuméricos sólo las comparaciones = y <> son permitidas. Los filtros también pueden contener cualquier operador booleano & (Y), | (O) y ! (NO); incluyendo los paréntesis ( ). Por ejemplo:

Seleccionar las tuplas de libros con la clave de adquisicion número 5.

```
LIBROS : CLAVE = 5
```

Seleccionar las tuplas de libros escritos por Austen con clave adquisición 5 o 7.

```
LIBROS: (CLAVE = 5 | CLAVE = 7) & (AUTOR = 'Austen')
```

PROYECCION (%)

También actúa sobre una relación para producir otra. Para cada tupla de la relación original, el resultado contendrá una tupla con las componentes renombradas, o solo algunos componentes según se hayan seleccionado. Una lista de selectores especifica las componentes y sus nuevos nombres.

Los selectores de la relación de entrada son cualificados por un nuevo selector y estos corresponderán a la relación resultante.

Para renombrar sólo algunos componentes y dejar el resto con sus nombres originales, la lista es dada con los nombres a cambiar seguida de tres puntos (...), por ejemplo:

```
LIBROS % AUTOR, TITULO
LIBROS % AUTOR->ESCRITOR,TITULO
LIBROS % AUTOR->ESCRITOR,...
```

La primera expresión crea una relación de grado dos, con los selectores *AUTOR* y *TITULO*. La relación denotada por la segunda expresión es similar, excepto que la componente *AUTOR* es renombrada por *ESCRITOR*. La tercera expresión da como resultado una relación similar a *LIBROS*, pero con la componente *AUTOR* renombrada por *ESCRITOR*.

Debido a que la proyección puede arrojar dos tuplas idénticas, posiblemente la cardinalidad del resultado sea menor que la cardinalidad de la relación origen.

UNION (+) e INTERSECCION (.)

Las operaciones unión, intersección y diferencia (que se verá adelante), tratan las relaciones como un conjunto de tuplas. Cada operación se ejecuta sobre dos relaciones y produce una tercera. El resultado de la unión es la relación que contiene todas las tuplas que aparecen en las dos relaciones operando. Una interseccion produce una relacion que contiene sólo las tuplas idénticas que aparecen en ambas relaciones operando. Para estas dos operaciones, las relaciones requieren ser compatibles, es decir, del mismo tipo; a su vez, la relación resultante también será del mismo tipo. Por ejemplo:

```
LIBROS + NUEVOS_LIBROS
```

Suponiendo que *NUEVOS_LIBROS* es la relación de las nuevas adquisiciones de la librería, entonces esta sentencia produciría la relación de todas las tuplas que contienen la información de todos los libros existentes en la librería.

```
LIBROS . NUEVOS_LIBROS
```

Produce sólo el conjunto de tuplas iguales que están contenidas en las relaciones *LIBROS* y *NUEVOS_LIBROS*.

DIFERENCIA (-)

Esta operación encuentra las tuplas de la primera relación para las cuales no existen tuplas duplicadas en la segunda relación. Para ello se consideran los selectores comunes, por lo que el resultado es llamado diferencia en los selectores comunes. En esta operación no es necesario que las relaciones sean compatibles, y cuando todos los componentes coinciden para ambas relaciones, esta operación se convierte en diferencia relacional convencional. Por ejemplo, si se tiene la relación

```
PRESTAMOS: (CLAVE, NOMBRE:LECTOR, FECHA:FECHA_SALIDA)
```

entonces la sentencia

```
LIBROS - PRESTAMOS
```

producirá una diferencia sobre *CLAVE*. Esto daría como resultado la relación que contenga *CLAVE*, *AUTOR* y *TITULOS* de aquellos libros que se encuentran disponibles.

JUNTA (*)

También produce una nueva relación de dos relaciones operando. En el caso más extremo, cada tupla de la primera relación es adherida con todas las tuplas de la segunda relación. Por cada par de tuplas es creada una nueva tupla con todos los componentes de ambas tuplas, y la unión de todas ellas son el resultado de la junta. Para esos casos, la cardinalidad del resultado es el producto de las cardinalidades de las relaciones operando, y el grado de la relación resultante es la suma de sus grados. Esta forma de junta es denominada como *junta completamente cuadrática*.

La junta más común es la *junta natural* o *equijunta*. Si los selectores de dos relaciones son iguales, entonces las tuplas son puestas dentro del resultado sólo si los valores de las tuplas en dichos selectores son iguales. La tupla concatenada aloja sólo una ocurrencia de los dos selectores, los cuales contienen el valor común. El resultado es denominado *junta sobre los selectores comunes*. Su cardinalidad puede ser cualquier valor, desde cero, hasta el producto de las cardinalidades de las relaciones de entrada, y su grado es igual a la suma de los grados de los operandos menos el número de selectores iguales. Cuando dos tipos de relación son iguales, la equijunta se convierte en una operación de intersección. Por ejemplo:

```
LIBROS * PRESTAMOS
```

Produce una junta sobre *CLAVE*, combinando la información de ambas relaciones. El resultado es una relación de grado 5, con los selectores *CLAVE*, *AUTOR*, *TITULO*, *LECTOR* y *FECHA_SALIDA*.

La comparación automática de selectores en las operaciones de junta y diferencia, algunas veces asocian selectores que no son requeridos, pero que sin embargo son comunes. La operación de proyección resuelve este problema, ya que puede renombrar selectores. Por ejemplo, para encontrar pares de libros del mismo autor, la parte *AUTOR/TITULO* de *LIBROS*, es juntado con el mismo sobre el selector *AUTOR*; para realizar la junta y no incluir a los selectores *TITULO*, estos son renombrados por *TITULO1* y *TITULO2*. Esta operación nos dará tuplas con el autor, y dos títulos de libros, además para las cuales se seleccionarán los títulos que son diferentes. En ISBL, la operación aparecería como sigue:

```
(LIBROS%AUTOR,TITULO->TITULO1) * (LIBROS%AUTOR,TITULO->TITULO2):
TITULO1 != TITULO2
```

4.2 EXTENSIONES AL LENGUAJE ISBL

Como mencionamos al principio del capítulo, las características de ISBL son apropiadas para la construcción de un lenguaje que permita portar la interpretación de LIDA sobre un lenguaje de cadena. ISBL Extendido es el nombre que hemos adjudicado a este lenguaje, diseñado para que se tome como lenguaje intermedio en la ejecución de los programas escritos en LIDA. En la mayoría de sus instrucciones y comandos ejecutables, ISBL Extendido conserva muchas de las características de su lenguaje de origen ISBL, por lo que su estructura es también muy similar. A continuación, mostramos la sintaxis y semántica del nuevo lenguaje.

4.2.1 Nomenclatura

Para entender la sintaxis considere la siguiente, nomenclatura:

- Todos los nombres de las relaciones (archivos) en juego dentro de una operación aparecerán con mayúsculas, por ejemplo *RELACION1*, *RELACION2* etc. Recordemos que las operaciones relacionales dan como resultado otra relación, por lo que también son representadas como nombres de relación.

- Todos los atributos y parámetros en juego aparecerán en minúsculas

- Las partes y parámetros opcionales de una instrucción aparecerán entre llaves { }

- Las constantes numéricas y alfanuméricas pueden almacenarse dentro de variables creadas en memoria, las cuales hemos denominado *Variables de Memoria.*

Las constantes numéricas pueden ser cualquier número real, y las constantes alfanuméricas pueden constituirse por todos los caracteres ASCII.

4.2.2 Expresiones y operandos matemáticos

- Una *Expresión Aritmética* puede construirse a partir de los operadores:

+	suma
-	resta
*	multiplicación
/	división
( )	paréntesis

y los operandos:

- Constantes numéricas
- Atributos
- Variables de memoria

Por ejemplo:

```
23244*909/8988+sueldo
3445+89*(prestacion/sueldo)
78.90/56
```

Como puede verse, una expresión aritmética puede contener a su vez otras expresiones aritméticas.

Los operadores aritméticos guardan las precedencias convencionales, de mayor a menor son:

1. ( )
2. * /
3. + -

- Una Expresión Booleana se construye a partir de los operadores:

O	o lógico
Y	y lógico
<	Menor que
>	Mayor que
=	Igual
<=	Menor o Igual
>=	Mayor o Igual
< >	Diferente
( )	Paréntesis

y los operandos:

- Constantes numéricas
- Constantes alfanuméricas
- Expresiones aritméticas
- Atributos
- Variables de memoria

Por ejemplo:

```
sueldo <= (prestaciones + 30000)
(sueldo = sueldobase) y (3459088 >= impuesto)
nombre = 'HECTOR SANCHEZ PEREZ' o CLVE = 'HRS'
```

Como puede verse, una expresión booleana puede contener expresiones booleanas, de manera similar a las aritméticas.

Los operadores booleanos guardan las precedencias convencionales:

1. ( )
2. Y
3. O
4. <, >, =, <=, >=, < >

4.2.3 Sintaxis y semántica de ISBL Extendido

Copia

Sintaxis: *RELACION1* = *RELACION2*

Semántica: El contenido de *RELACION2* es asignado a la *RELACION1*, en caso de que el archivo *RELACION1* no exista, se crea.

Unión

Sintaxis: *RELACION1* + *RELACION2*

Semántica: Se realiza la operación de unión relacional entre *RELACION1* y *RELACION2*, y el resultado es almacenado en una relación. Las dos relaciones deben ser compatibles bajo la unión.

Intersección

Sintaxis: *RELACION1* . *RELACION2*

Semántica: Se realiza la operación de intersección relacional entre *RELACION1* y *RELACION2*, y el resultado es nuevamente una relación. Las dos relaciones deben ser compatibles bajo la unión.

Diferencia

Sintaxis: *RELACION1 - RELACION2*

Semántica: Se realiza la operación de diferencia relacional, entre *RELACION1* y *RELACION2*, y el resultado es nuevamente una relación. Las dos relaciones deben ser compatibles bajo la unión.

Junta Natural

Sintaxis: *RELACION1 * RELACION2*

Semántica: Se realiza la operación de Junta Natural, entre *RELACION1* y *RELACION2*, y el resultado es nuevamente una relación. Las dos relaciones deben ser compatibles en al menos un atributo.

Selección

Sintaxis: *RELACION1* : (*expbool*) { , *RFALSA* }

Semántica: La operación de selección relacional se efectúa de acuerdo a la expresión booleana expbool, produciendo una relación con las tuplas que la cumplen; las tuplas que no la cumplen pueden ser alojadas dentro de la relación *RFALSA*, si ésta es definida.

Proyección

Sintaxis: *RELACION* % at_1{->rat_1} {,at_2{->rat_2}, ... ,at_n{->rat_n} }

Semántica: Se realiza la operación de Proyección sobre la lista de atributos at_1,at_2, ... ,at_n, con los atributos renombrados por la lista de los nuevos atributos opcionalesrat_1, rat_2, ... ,rat_n.

Actualización

Sintaxis: *RELACION* [*atr* = *exparit*]

Semántica: Actualiza todas las tuplas de *RELACION* en el atributo atr, con el nuevo valor dado por la expresión aritmética exparit. En caso de que atr no exista, éste será creado para cada tupla.

Ordena

Sintaxis: *RELACION* # *atr* , *ordenpor*

Semántica: Ordena las tuplas de *RELACION* sobre el atributo *atr*, en el orden dado por ordenpor como sigue: si *ordenpor* es *A*, entonces las tuplas se ordenan de forma ascendente; si ordenpor es *D*, entonces el ordenamiento es descendente.

Permuta Llave

Sintaxis: *RELACION* ~ *atr* , *ctenum*

Semántica: Cambia todas las tuplas de *RELACION* sobre el atributo *atr*, moviendo el caracter que se encuentra en la posición *ctenum* de atr a la primera posición, y los demás caracteres a su nueva posición relativa. Como restricción *atr* debe ser un atributo alfanumérico. Por ejemplo, la expresión `EMPLEADO~RFC,3` cambiaría el valor del atributo *RFC* de `SAPH650726` a la cadena `PH650726SA`.

Crea Variable

Sintaxis: *@var* = *conexfun*

Semántica: Crea o asigna a la variable de memoria *var* el valor de *conexfun*, donde *conexfun* puede ser:

- Una constante numérica
- Una constante alfanumérica
- Una expresión aritmética
- Una función
- El valor de una variable de memoria

Valor Variable

Sintaxis: { & } *var*

Semántica: Obtiene el valor de la variable de memoria *var* cuando ésta es utilizada en alguna expresión.

Función Máximo

Sintaxis: max (*RELACION, atmum*)

Semántica: Obtiene el valor máximo presente del atributo *atmum* en *RELACION. atmum* debe ser un atributo numérico.

Función Mínimo

Sintaxis: min (*RELACION, atmum*)

Semántica: Obtiene el valor mínimo presente del atributo *atmum* en *RELACION.* atmum debe ser un atributo numérico.

Función Promedio

Sintaxis: prom (*RELACION, atmum*)

Semántica: La función calcula el promedio sobre todos los valores del atributo *atmum* en *RELACION. atmum* debe ser un atributo numérico.

Función Suma

Sintaxis: sum (*RELACION, atmum*)

Semántica: Se obtiene la suma de los valores de todas las tuplas sobre el atributo *atmum* en *RELACION. atmum* debe ser un atributo numérico.

Función Registros

Sintaxis: regs (*RELACION*)

Semántica: Esta función calcula el número de registros (cardinalidad) de *RELACION*.

Función Máximo sobre Grupos

Sintaxis: maxa (*RELACION*, *atmum*, *atrgrup*)

Semántica: *RELACION* es agrupada con respecto al atributo *atrgrup*, y posteriormente de cada grupo se selecciona aquella tupla cuyo atributo *atmum* sea el máximo para cada grupo (*atmum* debe ser numérico). El resultado de esta operación es una relación que contiene todas las tuplas seleccionadas.

Función Mínimo sobre Grupos

Sintaxis: mina (*RELACION*, *atmum*, *atrgrup*)

Semántica: Obtiene un conjunto de tuplas de *RELACION* cuyo valor del atributo *atmum* es el mínimo de un grupo. *RELACION* es agrupada con respecto al atributo *atrgrup*. El atributo *atmum* debe ser numérico.

Función Promedio sobre Grupos

Sintaxis: proma (*RELACION*, *atmum*, *atrgrup*)

Semántica: Evalua el promedio de cada grupo de tuplas de *RELACION*. La operación promedio se calcula con respecto al campo *atmum* que debe ser numérico, y la agrupación de *RELACION* se realiza con respecto al atributo *atrgrup*.

Función Suma sobre Grupos

Sintaxis: suma (*RELACION*, *atmum*, *atrgrup*)

Semántica: Se efectúa la suma de los valores del atributo *atmum* de cada grupo de tuplas de *RELACION*. El atributo *atmum* debe ser numérico. La agrupación de *RELACION* se efectúa

con respecto al atributo *atrgrup*. El resultado de esta función, al igual que todas las involucradas con grupos, es nuevamente una relación.

Función Registros sobre Grupos

Sintaxis: regsa (*RELACION*, *atrgrup*)

Semántica: Obtiene el número de tuplas de cada grupo de *RELACION*, la agrupación se realiza con respecto al atributo *atrgrup*. El resultado de esta operación es una relación con dos atributos, el primero será el atributo *atrgrup*, y el segundo el atributo denominado *registros*, el cual se destina para alojar la cardinalidad de cada grupo de tuplas.

4.3 ISBL EXTENDIDO COMO LENGUAJE INTERMEDIO

Como se puede observar, LIDA es un lenguaje cuya semántica y sintaxis se definen de acuerdo a su estructura gráfica. No obstante, su implementación sobre una computadora lleva consigo el desarrollo de herramientas de software que permitan el buen desempeño y manejo del propio lenguaje; más aún, LIDA sobre una computadora sufre de limitantes al igual que cualquier aplicación computarizada, como lo son la velocidad de acceso de los datos, la capacidad de almacenamiento, el hardware de video que lo soporta, etc.

LIDA puede tomarse como una solución al problema de diseño de un lenguaje visual e interface hombre-máquina; lo que resta desde un punto de vista sistemático es la liga del lenguaje "hacia arriba" y "hacia abajo", es decir, por un lado el desarrollo de un sistema que soporte un lenguaje visual, y por el otro la forma de interpretación interna del mismo lenguaje. Para responder a la primer cuestión, parte de nuestro proyecto de tesis lo hemos dedicado al desarrollo de un sistema integrado que soporte el lenguaje, no obstante éste debe tomarse como un prototipo, y del cual hablaremos más tarde; la segunda cuestión es el objetivo que cubre este apartado.

Como se sabe, cualquier lenguaje de aplicación requiere de una herramienta de software denominada *compilador* para interpretar el *programa fuente* en otro, conocido como *programa objeto*, y posteriormente un *ligador* para conseguir el producto final nombrado *programa ejecutable*, que en otras palabras, sería el software que interpreta para la computadora lo que el desarrollador escribe. Existe también el concepto de *intérprete*, el cual hace las veces de compilador y ligador a la vez: cada que el programador escribe una instrucción el intérprete la reconoce y la ejecuta.

La idea de desarrollar un compilador para LIDA fue descartada debido a que su desarrollo saldría de los límites del presente trabajo; por lo que se optó en desarrollar un intérprete del lenguaje visual; de hecho, el sistema SABRE sirve de intérprete del lenguaje, pero aún el problema no queda resuelto hasta aquí, ya que se trata en este caso de un lenguaje visual, por lo que su tratamiento es un poco diferente a cualquier otro lenguaje común. Para poder transformar un programa LIDA, o bien, un flujograma, en lenguaje entendible para la computadora, fue necesario acudir a un lenguaje lo suficientemente formal como para adecuarse a las necesidades de LIDA, y que éste sirviera de lenguaje intermedio entre LIDA y la máquina.

La utilización de un lenguaje de cadena parecía ser la solución. ISBL como tal ofrece ciertas características requeridas por LIDA, además de ser un lenguaje prototipo utilizado para el desarrollo de sistemas que manejan bases de datos relacionales. Empero, ISBL no cumple totalmente con la funcionalidad de LIDA, ya que éste cuenta con ciertas instrucciones no consideradas por LIDA. Por tal motivo ISBL fue transformado, tomándose de ISBL sólo las instrucciones requeridas y añadiendo las no existentes; de tal forma que se crea el lenguaje ISBL Extendido, como lenguaje intermedio para LIDA.

De esta forma se desarrollan dos herramientas, primero un intérprete de LIDA a ISBL Extendido, y luego un procesador para decifrar las cadenas generadas por el intérprete de flujogramas. De hecho, ISBL Extendido

es la plataforma sobre la que descansa LIDA, pero con esto no quiere decir que LIDA es ISBL Extendido desde un punto de vista gráfico: LIDA es un lenguaje lo suficientemente formal para considerarse como tal, e ISBL Extendido es simplemente un lenguaje intermedio para LIDA dentro del sistema SABRE en específico. Podemos afirmar entonces, que LIDA puede implementarse con alguna otra herramienta como manejador de bases de datos, y no necesariamente con ISBL Extendido.

Posteriormente, en el apartado 5.2.3 se analiza la forma detallada en la que ISBL Extendido interviene para la ejecución de un programa escrito en lenguaje LIDA.

CAPITULO 5: EL SISTEMA INTEGRADO SABRE

Realmente las ciencias computacionales, y en particular la Ingeniería de Software que es la disiplina que se ocupa de elevar la productividad del desarrollo de programación y considera un enfoque sistemático para el desarrollo, la operación y el mantenimiento del software, tiene como uno de sus principales objetivos el de mejorar la interface de comunicación hombre-máquina. Este mejoramiento se reduce a una palabra: *simplificación.*

Como contribución a este mejoramiento, nuestro proyecto incluye el desarrollo de un sistema que soporte LIDA como lenguaje principal, y sobre del que se pueda implementar un sistema de información con bases de datos relacionales: con esta idea surge el *Sistema para el Desarrollo de Aplicaciones con Bases de Datos Relacionales* que hemos nombrado en su forma abreviada **SABRE.**

5.1 CONSIDERACIONES DE DISEÑO

SABRE viene a ser un sistema integrado para manejar bases de datos a través del lenguaje LIDA, y con ello generar aplicaciones. A lo largo de los capítulos anteriores, hemos tratado de dar un marco de referencia para el sistema SABRE, sin que esto signifique que este sistema sea el tema principal de nuestro trabajo. SABRE es sólo uno de los posibles resultados en la aplicación de LIDA.

5.1.1 Análisis y planteamiento

En estos últimos años, la ingeniería de software se orienta cada vez más al desarrollo de productos con características de:

- *Facilidad de operación*
- *Macro operaciones*
- *Aumento de la velocidad de procesamiento*
- *Procesos distribuidos*
- *Integración de procedimientos*

Al parecer, los sistemas con interfaces gráficas **GUI** (*Graphical User Interface*), los lenguajes generadores de aplicaciones, los métodos de optimización de algoritmos y estructuras de datos y las herramientas para redes de computadoras, aumentan enormemente dichas características.
El sistema SABRE presenta parte de los conceptos que se manejan en soluciones de la actualidad, y se define como un sistema integrado de propósito general para el manejo de información sobre una estructura de bases de datos relacionales, con un lenguaje iconográfico para el desarrollo de aplicaciones que le dió parte del nombre al sistema, y un lenguaje de cadena incluído. LIDA cuenta con cinco módulos de operación:

1. *Descriptor*
2. *Captador*
3. *ISBL*
4. *LIDA*
5. *Configuración*

El módulo Descriptor es utilizado para el mantenimiento de descriptores de archivos: en él es posible declarar los descriptores para definir la estructura de las relaciones manipuladas con LIDA, tal como hemos visto en el apartado 3.1.3 El descriptor y SABRE. El Captador es precisamente la herramienta de software conocida como captador de datos, y utiliza el descriptor de archivos para generar una ventana virtual de captura para los archivos definidos en los descriptores. El módulo ISBL es todo un subsistema que ofrece un ambiente adecuado para trabajar con el lenguaje ISBL Extendido: es un intérprete de cadenas que permite al usuario armar procedimientos con las bases de datos existentes. Este módulo también adiciona a ISBL Extendido una serie de comandos de sistema, para complementar su funcionamiento. LIDA es el módulo en el que se opera el lenguaje iconográfico, e incluye un editor y un menú de comandos adicionales para la manipulación sencilla de los flujogramas. El último módulo denominado Configuración se incluye para darle flexibilidad al sistema, ya que en éste permite declarar el tipo de monitor con que cuenta el equipo de cómputo, así como la definición del directorio en donde se encuentran los archivos de trabajo.

SABRE cuenta con las siguientes características reelevantes:

- Todo el sistema se presenta por medio de la técnica de ventaneo, que visualiza mejor los datos.
- Cada opción se realiza por medio de menús de fácil acceso.
- El procedimiento de captura de información es sencilla y amigable, además cuenta con teclas programadas con funciones específicas para asistir la captura.
- Integra desde la definición de la estructura de una base de datos relacional, hasta su operación y mantenimiento.
- Permite la emisión de resultados de los procedimientos hacia la impresora, el monitor o un archivo.
- Es posible generar código ISBL Extendido en archivos de procedimientos, para utilizarse por otras herramientas de software creadas específicamente para tratar código ISBL Extendido.

- El tratamiento de la misma información puede realizarse vía LIDA o ISBL Extendido, ya que estos dos lenguajes se basan en la herramienta del descriptor de archivos. Además, existe una interface de comunicación de LIDA con ISBL.
- SABRE es un sistema generador de aplicaciones, cuyo nivel de encapsulamiento puede aumentar, dadas las características de ejecución de procedimientos.
- El mantenimiento de las bases de datos a nivel físico, es complementado con una utilería de compresión de archivos.
- La comunicación con el usuario no sólo es gráfica, ya que el sistema ofrece toda una serie de mensajes que lo advierten de posibles errores, procedimientos en ejecución y estado actual del sistema.

Como se ha podido observar, los productos de CASE se convierten cada vez más populares entre los programadores, y en algunas ocasiones rebasan los límites de su área de trabajo, es decir, la programación tiende a ser una actividad cada día más simple.

Ciertamente, esto lleva un análisis mayor que puede salir del objetivo de este trabajo, pero cabe mencionar que el principal problema observable en los lenguajes de cuarta generación que se basan en el modelo relacional, y que son un producto del concepto CASE, es la dificultad para los usuarios sin conocimientos de computación de comprender los conceptos más básicos del álgebra y el cálculo relacional, que sin lugar a duda no son conceptos tan sencillos para cualquier usuario, lo que obliga a que se deba tener un poco de más análisis al respecto. Tal vez este problema de la programación se llegue a resolver en futuras generaciones, cuando los sistemas lleguen a un grado tal de desempeño que lo permita.

El sistema SABRE conserva la problemática de los lenguajes de cuarta generación, pero también su diseño incluye ciertas características de simplificación, tanto operativa como de comprensión; y esto es algo intrínseco del propio lenguaje, ya que es un lenguaje visual. Cada icono que representa una operación, debe ser comprendido por el usuario, pero además de ello, el propio icono se ha diseñado con una estructura lo más ordenada posible; por lo tanto, aunque el propio lenguaje requiera de algunos conocimientos del álgebra relacional, su manejo y comprensión no deja de ser simple.

Se ha tratado de cubrir toda la definición del LIDA dentro de este sistema, y más aún, se han adecuado ciertas especificaciones al LIDA del Sistema Integrado: para aprovechar las ventajas del intérprete del lenguaje ISBL Extendido y siguiendo las reglas de un lenguaje iconográfico, se ha añadido el ICONO COMANDO que acepta como parámetro cualquier cadena ISBL Extendido, e inclusive comandos del sistema manejados en el intérprete y que analizaremos más adelante. Esto implica que toda una instrucción ISBL puede ser comprendida dentro de un icono. El icono comando permite mayor nivel de encapsulamiento.

ICONO COMANDO

Los arcos que entran y salen de este icono no tienen ningún efecto, más sin embargo se utilizan para no perder el sentido semántico del flujograma.

Este icono tiene como parámetro cualquier instrucción o comando del sistema (que veremos más adelante) aceptado en el intérprete de ISBL Extendido, a excepción del comandos *$CLS*, que no tiene ningún efecto.

SABRE cuenta con una opción para poder generar código ISBL Extendido. Así SABRE se vale del lenguaje ISBL como intermediario entre las llamadas a la biblioteca de procedimientos y el flujograma en LIDA. Es posible generar este código intermedio, tal y como se realiza al momento de la ejecución. Dicho código, que se almacena dentro de un archivo ASCII con extensión *.LIP*, es realmente un archivo de procedimientos ISBL Extendido, y éste puede ser ejecutado posteriormente por medio del comando *$EJEC* del intérprete.

Para facilitar la operación del intérprete, se diseñaron ciertos comandos como los son *$CLS*, que limpia la pantalla de despliege, *FIN* que finaliza la sesión, *$EJEC* del cual ya hablamos, y el símbolo *?* para editar el valor de una variable de memoria.

A continuación, describiremos la estructura interna del sistema LIDA para comprenderlo desde el punto de vista de programación, y posteriormente mostraremos la forma conveniente de operar LIDA, como parte de la presentación de la interface hombre-máquina.

5.1.2 Características de un sistema de bases de datos relacional

En los últimos años la palabra relacional, parece que ha venido a ser la panacea para muchos sistemas de bases de datos, hasta llegar al grado de considerar, que por el sólo hecho de llamar relacional a un sistema, ya cumple con las características del un Sistema Relacional. El doctor Codd, declaró las siguientes reglas para decidir si un sistema es relacional o no [VAUGHAN]:

Regla 1: Cualquier Sistema de Bases de Datos Relacional debe manipular las bases de datos por medio de sus capacidades relacionales. Si un sistema de bases de datos utiliza herramientas de software cuyo manejo dependa de un procesamiento del tipo registro por registro, entonces no será completamente relacional.

Regla 2: Todos los datos en una base de datos relacional son explícitamente representados (en el nivel lógico), como valores en tablas. Los datos no pueden ser almacenados en otra forma.

Regla 3: Cada dato debe ser accesado a través del uso de la combinación de la llave primaria, el nombre de la tabla (relación), y el nombre de la columna (atributo).

Regla 4: Los valores nulos son explícitamente soportados. Un nulo representa un valor esperado, o bien información inaplicable.

Regla 5: La descripción de una base de datos, es también almacenada en un nivel lógico y como valor tabular.

Regla 6: Un Sistema de Bases de Datos Relacional debe soportar una definición clara de la manipulación de datos, por medio de un lenguaje que a su vez soporte la definición y manipulación de datos, definición de vistas, integridad de los datos, y control de acceso y privilegios.

Regla 7: Todas las vistas que pueden actualizarse, deben ser actualizadas por medio del sistema.

Regla 8: En un sistema de bases de datos relacionales, además de hacer una recuperación de información, se debe permitir insertar, actualizar, y eliminar datos vistos como un conjunto relacional.

Regla 9: Los datos deben ser físicamemte independientes de las aplicaciónes. Por ejemplo, un programa de aplicación no debe ser cambiado cuando un índice es anexado a una tabla, o bien cuando los valores de una tabla son actualizados.

Regla 10: Siempre que sea posible, las aplicaciones de software, deben ser independientes de los cambios hechos en las tablas básicas (estructuras de datos). Es decir, los programas desarrollados por medio del sistema, no deben modificarse para reflejar un cambio en las bases de datos principales. Por ejemplo, el código no debe ser reescrito cuando las tablas son combinadas en una vista.

Regla 11: La integridad de los datos debe ser definible a través de un lenguaje relacional, y debe almacenarse en un catálogo.

Regla 12: Un sistema de bases de datos relacional debe tener características de procesamiento distribuido.

Regla 13: Si un sistema de bases de datos relacionales cuenta con lenguaje que maneje el proceso de los datos a nivel registro, este lenguaje no puede ser usado, para aprobar las reglas de integridad y las características de un lenguaje relacional. Un sistema de bases de datos relacionales no solo debe estar gobernado por reglas relacionales, sino que estas reglas deben ser sus bases fundamentales. El hecho que un programa de bases de datos cuente con un lenguaje como SQL, no lo convierte en un sistema de base de datos relacional.

Codd estableció en un principio que para que un sistema de base de datos se considere relacional, debería cumplir sólo siete de estas reglas. Pero dado que algunos sistemas las han cumplido en su totalidad, desde entonces Codd ha declarado que para que un sistema se considere perfectamente relacional debe cumplir todas estas reglas.

Es un hecho que el sistema SABRE, no es un sistema completamente relacional (como veremos más adelante), pero esto no significa que el lenguaje LIDA no pueda formar parte de un sistema que sí lo sea.

5.2 ESTRUCTURA INTERNA DE UN FLUJOGRAMA

Como hemos descrito anteriormente, LIDA utiliza el flujograma para representar gráficamente aplicaciones con bases de datos del tipo relacional. En este apartado trataremos de describir la manera en que un flujograma, como herramienta de software, es utilizado para resolver una aplicación en LIDA. La utilización del lenguaje Pascal para la implementación del lenguaje iconográfico en una computadora fue de utilidad, debido a la flexibilidad en cuanto al manejo de memoria por medio de apuntadores.

5.2.1 Estructura de datos

Internamente, un flujograma es tratado como una matriz de apuntadores, cada uno de los cuales puede alojar un nodo. En lo subsecuente, al hablar de un nodo o elemento de flujograma nos estaremos refiriendo a una de sus operaciones, que visualmente está representada como un icono con sus argumentos o parámetros.

Cuando se genera un nuevo elemento en el diagrama, este es creado de forma dinámica en la memoria de la computadora. La estructura de datos que determina cada elemento, contiene los campos necesarios para almacenar la información que indican, tanto sus características visuales, como aquellas que determinan su comportamiento dentro del flujograma y su relación con los demás elementos.

Dado que el sistema fue concebido y desarrollado en el lenguaje de programación Pascal, aunado a la sencillez de su comprensión, nos permitiremos indicar las estructuras de datos utilizadas en su forma nativa.

Los elementos básicos que conforman un flujograma son:

- La matriz de apuntadores *DIBUJO*.

  Como mencionamos inicialmente, esta matriz aloja cada uno de los nodos que se van generando: cuando se inserta un nuevo elemento al flujograma, el apuntador correspondiente a su posición cartesiana, es orientado hacia la dirección de memoria donde se reserva el espacio necesario para almacenar los campos correspondientes a su estructura de datos. La definición de la matriz de apuntadores está dada de la siguiente manera:

```
DIBUJO : array [1..MAXR , 1..MAXC] of ^ELEMENTO;
```

  donde *ELEMENTO* es el nombre de la estructura de datos de un nodo, y *MAXR* y *MAXC* son el número de renglones y columnas que componen la matriz respectivamente.

- Las estructuras que determinan las características visuales y funcionales de cada operación del lenguaje LIDA.

Estas estructuras son generadas automaticamente al ejecutarse el sistema, y residen en memoria durante toda la sesión de trabajo, hasta que el sistema completo es desalojado de la memoria de la computadora. Cuando se inserta un nuevo elemento en el flujograma, éste se relaciona con una de tales estructuras, donde se define el tipo de operación a la que el elemento representa. La estructura lógica de cada una de las operaciones del lenguaje es:

```
UNAFIGU = record
  NUMLINP,NUMLINS    : byte;
  LINEASS,LINEASP    : pointer;
  ANCHO,ALTURA       : byte;
  RELLENO            : boolean;
  XV,YV,XF,YF,PESO   : byte;
  HAYPAR             : array [1..MAXPAR] of char;
  COORPAR            : array [1..MAXPAR] of integer
end;
```

La información que contiene cada uno de los campos de una estructura de este tipo, es la siguiente:

NUMLINS Indica cuantos puntos conforman el polígono que debe dibujarse con línea sólida, como parte del icono asociado a la operación.

NUMLINP Indica cuantos puntos conforman el polígono que debe dibujarse con línea punteada, o el polígono que debe rellenarse de color, como parte del icono asociado a la operación.

LINEASS Es un apuntador que indica la dirección de memoria donde se encuentra el arreglo de los pares ordenados correspondientes a cada uno de los puntos que conforman el polígono que debe dibujarse con línea sólida.

LINEASP Es un apuntador que indica la dirección de memoria donde se encuentra el arreglo de los pares ordenados correspondientes a cada uno de los puntos que conforman el polígono que debe dibujarse con línea punteada, o que debe rellenarse de color.

ANCHO Indica la anchura máxima del icono en unidades de dibujo, desde su centro hasta uno de sus extremos laterales.

ALTURA Indica la altura máxima del icono, en unidades de dibujo, desde su centro hasta su extremo superior o inferior.

RELLENO Este componente booleano indica (verdadero) si los campos *LINEASP* y *NUMLINP* se refieren a un polígono relleno, o (falso) si se refieren a un polígono cuyo contorno debe ser dibujado con línea punteada.

XV Es la coordenada x, en unidades de dibujo con respecto al centro del icono, desde donde debe salir una flecha sólida. Como vimos con anterioridad, existen flechas sólidas y punteadas, las cuales representan la salida de diferentes tipos de datos, específicamente, datos que cumplen (en el caso de línea sólida) una condición de selección, y los que no la cumplen (en el caso de línea punteada).

YV Es la coordenada y en unidades de dibujo con respecto al centro del icono, desde donde debe salir una flecha sólida.

XF Es la coordenada x en unidades de dibujo con respecto al centro del icono, desde donde debe salir una flecha punteada.

YF Es la coordenada y en unidades de dibujo con respecto al centro del icono, desde donde debe salir una flecha punteada.

PESO Contendrá un número que representa la prioridad de la operación. Entre mayor sea este número, mayor será la prioridad que la operación tendrá para ejecutarse en caso de que esta se encuentre en un nodo terminal del flujograma.

HAYPAR Es un arreglo de caracteres que define el tipo de argumentos (parámetros) que debe contener la operación. Si el i-ésimo componente de este arreglo contiene un caracter 'N', indicará que el i-ésimo argumento de la operación contendrá un nombre, que puede ser, por ejemplo, el nombre de un archivo. Si por el contrario, contiene una 'P', indicará que se trata de un parámetro, como lo puede ser la condición en una operación de selección. El arreglo va de 1 a *MAXPAR* (constante que indica el máximo número de argumentos).

COORPAR Es un arreglo que contiene las coordenadas y en unidades de dibujo con respecto al centro del icono, donde deben ser visualizados cada uno de los *MAXPAR* argumentos de la operación.

Cada una de estas estructuras están agrupadas en un arreglo llamado *ARREFIG*, que se define como:

```
ARREFIG : array [1..MAXFIG] of UNAFIGU;
```

donde *MAXFIG* es el número total de operaciones definidas en el lenguaje LIDA.

- Los nodos o elementos que componen un flujograma.

Los nodos son insertados directamente por el usuario para simbolizar una acción u operación bien definida. Estos nodos, además de estar asociados con los elementos del arreglo *ARREFIG*, poseen información relevante que define su posición visual en el flujograma, y su relación con otros nodos del mismo. Podríamos decir, que las ligas o líneas que indican gráficamente el flujo de datos, y la manera y orden en que el flujograma será ejecutado, están contenidas en estas estructuras que presentan la definición:

```
ELEMENTO = record
  NUMFIGURA      : byte;
  CADPAR         : array [1..MAXPAR] of ^string;
  OPERANDO       : aray [1..MAXOPDO] of APUCOMP;
  POSX,POSY      : integer;
  YACALC         : boolean;
  ESULTI         : byte;
  ARCHIT,ARCHIF : ^string
end;
```

donde:

```
APUCOMP  = record
  CX,CY : byte;
  TIPOL : (SOLIDA,PUNTEADA)
end;
```

NUMFIGURA Es el número de la posición (índice) que ocupa la operación asociada al nodo, dentro del arreglo *ARREFIG*.

CADPAR Es un arreglo de apuntadores, cada uno de los cuales señalarán las cadenas que se generan dinámicamente, y que conformarán los argumentos o parámetros de la operación asociada al nodo. Estas cadenas serán capturadas directamente por el usuario del sistema.

OPERANDO Es un arreglo que indica a qué otros nodos del flujograma está ligado uno en particular. El arreglo está formado por componentes de tipo *APUCOMP*, cuya estructura tiene el siguiente significado:

CX Es la coordenada x en unidades matriciales de un nodo al cual está ligado el nodo en cuestión.

CY Es la coordenada y en unidades matriciales de un nodo al cual está ligado el nodo en cuestión.

TIPOL Indica el tipo de liga que existe entre el nodo actual y el nodo con el cual se está relacionando. Este tipo de liga, además de tener una función operativa, también indica el tipo de línea con que se debe dibujar la flecha correspondiente entre ambos nodos. Los dos valores posibles que puede tomar el campo *TIPOL* son *SOLIDA* y *PUNTEADA*. Un nodo en particular, puede estar ligado hasta con *MAXOPDO* elementos del flujograma.

POSX Es la coordenada x en unidades de dibujo, correspondiente a la posición donde debe colocarse el centro del icono en el momento de visualizarlo.

POSY Es la coordenada y en unidades de dibujo, correspondiente a la posición donde debe colocarse el centro del icono al momento de visualizarlo en la pantalla de la computadora.

YACALC Es un campo booleano que indicará si el nodo ya ha sido calculado con anterioridad, dentro de una misma ejecución del flujograma al que pertenece.

ESULTI Dependiendo del valor de este campo, el sistema podrá determinar si se trata de un nodo terminal, es decir, un nodo del cuál no sale ningún flujo de datos (flechas).

ARCHIT Es un apuntador a una cadena de caracteres, la cual será generada dinámicamente para almacenar el nombre del archivo donde se arrojen los resultados de la operación asociada al nodo. Si la operación es un proceso de selección, entonces la cadena señalada por este apuntador indicará el nombre del archivo donde se almacenaron los registros que cumplieron la condición de filtrado.

ARCHIF Es un apuntador a una cadena de caracteres, la cual será generada dinámicamente para almacenar el nombre del archivo donde se arrojaron los

registros que no cumplieron la condición de filtrado en una operación de selección.

Los últimos cuatro campos de la estructura *ELEMENTO* están íntimamente relacionados con el método de ejecución del flujograma, por lo cual se explicará más tarde su funcionamiento detallado.

5.2.2 Relación interna de datos

Ahora procederemos a tratar con más detenimiento la manera han la cual se establecen las posiciones y tipos de cada nodo del flujograma, y de cómo se definen los vínculos que dan la conectividad y flujo de datos entre ellos.

En principio, se tiene la matriz DIBUJO con cada una de sus entradas vacías, o dicho de otra manera

```
DIBUJO[i,j] = NIL , 1 <= i <= MAXR, 1 <= j <= MAXC
```

Esto se podría representar esquemáticamente como lo muestra la figura 5.1.

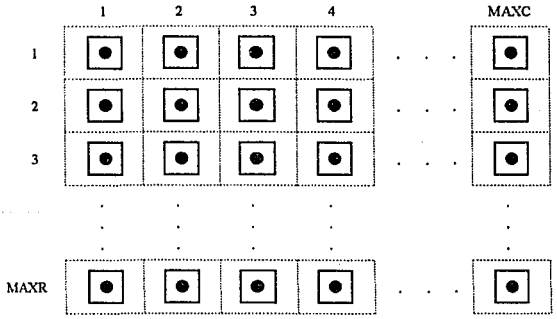


Figura 5.1 Representación esquemática de la matriz DIBUJO

En este momento, es importante formalizar lo que hemos llamado unidades de dibujo y unidades matriciales: las primeras se refieren a las que se utilizan para fines de visualización, considerando toda el área bidimensional reservada para fines del dibujo del flujograma. Propiamente, esta área de dibujo es en sí una matriz de puntos, en términos de los cuales se dan todas las dimensiones de los iconos, las

posiciones que estos ocupan dentro de tal área, y los lugares donde deben visualizarse las flechas y los parámetros de cada uno de los nodos. Por otro lado, las coordenadas matriciales se refieren específicamente a las entradas del arreglo bidimensional *DIBUJO*, y son utilizadas para accesar a cada uno de sus elementos. De hecho, las ligas entre los nodos del flujograma se establecen mediante este tipo de coordenadas.

Cada que se mueve el cursor de edición del flujograma, éste se desplaza una cantidad preestablecida de puntos de dibujo, tanto vertical como horizontalmente, pero cada posición que toma se refiere a una correspondiente dentro de la matriz *DIBUJO*, es decir, la primera posición que puede tomar el cursor (esquina superior izquieda del espacio de edición), corresponde al elemento (1,1) de la matriz; si posteriormente el cursor es movido una vez hacia la derecha, entonces la nueva posición se referirá al elemento (1,2), intependientemente del número de unidades de dibujo que se haya desplazado visualmente entre una posición y otra. De esta manera, cuando se decide el usuario a insertar un elemento al flujograma, son tomadas las coordenadas matriciales actuales del cursor para determinar qué apuntador es el que debe señalarlo.
En la figura 5.1, cada uno de los puntos representa un apuntador de la matriz. Cuando se inserta un nuevo elemento al flujograma, el apuntador correspondiente es dirigido hacia la localidad de memoria donde se genera dinámicamente tal elemento, bajo la estructura *ELEMENTO* anteriormente citada. Gráficamente, esta acción puede ejemplificarse con la figura 5.2.

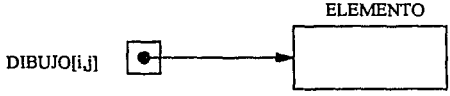


Figura 5.2 Generación dinámica de un nodo

Como se observó anteriormente, la estructura *ELEMENTO* tiene un componente llamado *NUMFIGURA*, el cual puede tomar valores entre 1 y *MAXFIG*. También se mencionó que existe un arreglo llamado *ARREFIG*, que contiene *MAXFIG* elementos, representando cada uno de ellos la descripción visual y operativa de una operación del lenguaje.

Antes de que se inserte un elemento al flujograma, el usuario del sistema debe indicar qué operación es la que precisamente quiere agregar, y al hacerlo, realmente está seleccionando un número, que no es sino la posición que ocupa la operación dentro del arreglo *ARREFIG*. Si n es el número de la operación seleccionada, entonces internamente se harán las asignaciones

```
new(DIBUJO[i,j]);
DIBUJO[i,j]^.NUMFIGURA := n;
```

Con ello, se podrá accesar toda la información necesaria para dibujar el icono correspondiente al elemento *(i,j)* del flujograma, así como aquélla que indicará el comportamiento de la operación.

Una vez insertado el elemento o nodo al flujograma, se le asignan las coordenadas de dibujo *(xd,yd)* de la posición visual del cursor:

```
DIBUJO[i,j]^.POSX := xd;
DIBUJO[i,j]^.POSY := yd;
```

con lo cual se sabrá en que lugar deberá aparecer el centro del icono de la operación.

Posteriormente, el usuario podrá introducir los argumentos o parámetros de la operación, según sea su sintaxis. Para ello se utiliza el arreglo de apuntadores llamado *CADPAR*. Si el k-ésimo elemento del arreglo *CADPAR* es editado, entonces se genera un espacio en la memoria donde será almacenado. Gráficamente, esta acción podría representarse como en la figura 5.3.

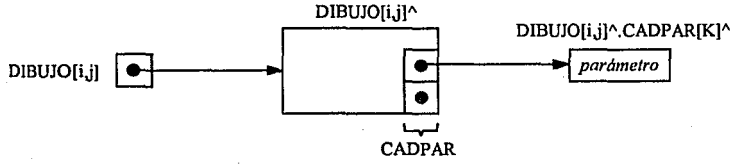


Figura 5.3 Asociación de parámetros

Los parámetros serán de vital importancia para el éxito en la ejecución del flujograma.

Por último, el usuario deberá indicar la relación existente entre los elementos del flujograma a través de flechas.

Si existe una flecha que parte del elemento *(i,j)* de la matriz, al elemento *(r,s)*, significará que hay un flujo de datos del elemento *(i,j)* hacia el elemento *(r,s)*. Los componentes de la estructura *ELEMENTO* que permiten definir internamente la relación anterior, son cada uno de los *MAXOPDO* integrantes del arreglo *OPERANDO*. Cada uno de esos integrantes poseen a su vez tres campos: los campos *CX* y *CY* indicarán las coordenadas matriciales del elemento hacia donde se dirige la flecha, y el campo *TIPOL* señalará el tipo de línea que se ha seleccionado para relacionar ambos elementos. A saber, están predefinidos dos tipos de línea: *SOLIDA* y *PUNTEADA*.

Si tomamos el ejemplo donde se ha decidido trazar una línea sólida desde el elemento *(i,j)* hasta el elemento *(r,s)*, internamente se harán las siguientes operaciones:

```
DIBUJO[r,s]^.OPERANDO[k].TIPOL := SOLIDA;
DIBUJO[r,s]^.OPERANDO[k].CX    := j;
DIBUJO[r,s]^.OPERANDO[k].CY    := i;
```

donde *k* es el número del primer integrante vacío encontrado en el arreglo *OPERANDO*. Dado que este último arreglo tiene un máximo de *MAXOPDO* integrantes, entonces el límite de flechas que pueden llegar a un nodo estará dado por tal número.

Como se puede observar, la asociación de los elementos se realiza "al revés" de como el usuario lo indica y lo visualiza, y esto se debe al método de ejecución de los flujogramas que hemos integrado en el sistema.

La figura 5.4 muestra ambas versiones de una relación existente entre dos elementos.

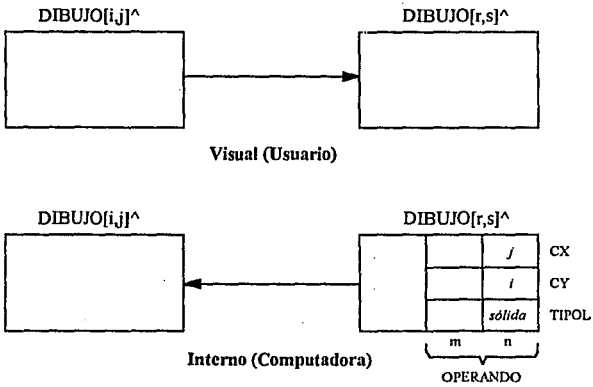


Figura 5.4 Versiones de vinculación de nodos

Si posteriormente se decidiera que del elemento (g,h) hay un flujo de datos hacia el (r,s), entonces se ocuparía otro de los integrantes del arreglo OPERANDO del elemento (r,s). De acuerdo a esta forma de vincular los elementos de un flujograma, desde el punto de vista del usuario del sistema, de un nodo podrán salir un sinfín de flechas.

5.2.3 Método de ejecución

El método utilizado para la ejecución de un flujograma, de acuerdo a su comportamiento, podríamos decir que es un método de *abajo hacia arriba*, que se basa en la premisa de que "para resolver esta operación, necesito conocer el resultado de las operaciones inmediatas anteriores". De esta manera, tenemos un algoritmo de resolución sencillo, basado en las técnicas de recursividad, que se aplica de la misma manera a cada uno de los nodos, salvo pequeñas variantes asociadas a la naturaleza misma de cada una de las operaciones involucradas.

Un ejemplo que muestra el funcionamiento básico del método, es dado por una operación aislada, como lo es la unión de dos relaciones que arrojan una tercera como resultado. Este ejemplo es mostrado en la figura número 5.5.

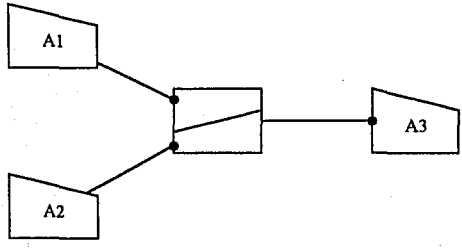


Figura 5.5 Unión de dos relaciones

En este caso, la ejecución iniciaría en el nodo terminal, es decir, el que indica la definición de la relación *A3*. Como *A3* está definida como resultado de una operación, entonces se procede a ejecutar el nodo inmediato anterior, que corresponde a una unión; pero éste no puede arrojar un resultado sino hasta conocer los datos provenientes de las operaciones anteriores, por lo cual nuevamente se procede a pasar "hacia arriba". Como existen en este caso dos nodos anteriores, entonces se ejecutará el correspondiente al señalado por el primer integrante del arreglo *OPERANDO*, es decir, *OPERANDO[1]*, siguiendo con el segundo (*OPERANDO[2]*), y así sucesivamente hasta completar los requerimientos de la operación asociada al nodo en cuestión; supongamos que para nuestro ejemplo, primero se ejecuta el nodo que indica la definición de la relación *A1*, como éste ya no tiene un nodo antecesor, entonces se procede a extraer la información previamente almacenada en ella, pasándola "hacia abajo" al nodo de la operación de unión, donde se procede a ejecutar la operación que define de igual forma la relación *A2*. Como ahora ya se tienen todos los elementos para realizar la unión, se procede a ejecutarla, completando la operación

del nodo y pasando la relación resultado hacia el nodo terminal, donde se procederá finalmente a copiar los resultados en la relación *A3*.

Por lo tanto, la manera de operar de un nodo se puede resumir en los tres pasos siguientes:

a) *Tomar los nombres de las relaciones arrojadas por las operaciones inmediatas anteriores,*

b) *Operar con tales relaciones, y*

c) *Pasar el nombre de la relación resultante hacia los nodos subsecuentes.*

Es pertinente aclarar que no todas las operaciones del lenguaje necesitan el mismo número de nodos antecesores, y que no todas las operaciones necesitan pasar los datos resultantes a un nodo posterior, pero en términos generales, los puntos anteriores engloban el funcionamiento de cada nodo en el flujograma.
Ahora es más claro visualizar el porqué de las referencias "a la inversa" entre los nodos: las ligas lógicas están realmente indicando el sentido en que se debe realizar la ejecución.

Es también en esta fase donde los campos *YACALC*, *ESULTI*, *ARCHIT* y *ARCHIF* de la estructura *ELEMENTO* toman participación: cuando una operación ha sido realizada en su totalidad, entonces en su nodo correspondiente se hará la asignación `DIBUJO[i,j]^.YACALC := TRUE`, que indicará que la operación ya ha sido calculada y ha producido una relación resultante, y si posteriormente se hace referencia al nodo en cuestión, se sabrá que no es necesario ejecutarlo; por otra parte, se generan dinámicamente las cadenas de caracteres apuntadas por los componentes *ARCHIT y ARCHIF*. Tales cadenas contendrán los nombres de las relaciones que se generaron como resultado de la operación, de tal manera que puede hacerse referencia a ellas en caso de ser necesario en una operación posterior. Recordemos que la única operación que puede arrojar dos relaciones resultantes es la selección, dado que en ella interviene una evaluación booleana de registros: a cada una de tales relaciones las hemos llamado salida verdadera y salida falsa, que en el diagrama se representan con una línea sólida y con una línea punteada respectivamente. Esto está íntimamente ligado con los campos *ARCHIT* y *ARCHIF*, puesto que dependiendo del tipo de salidas que presente un nodo, será la manera en qué tales campos se utilizarán.

Si tomamos el ejemplo de la figura 5.6, observaremos con claridad el funcionamiento de los campos anteriormente mencionados.

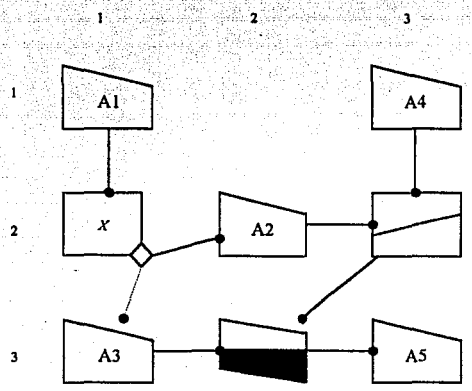


Figura 5.6 Flujograma y sus coordenadas matriciales

De acuerdo al método de ejecución, se empezará por el nodo *(3,3)*, pasando recursivamente "hacia arriba" por los nodos *(3,2)*, *(3,1)*, *(2,1)* y *(1,1)*, el cuál se resuelve al definir la relación *A1*. En este punto se hacen las asignaciones

```
DIBUJO[1,1]^.YACALC := TRUE;
new(DIBUJO[1,1]^.ARCHIT);      { dado que la línea existente entre los }
DIBUJO[1,1]^.ARCHIT^ := 'A1'; { nodos (2,1) y (1,1) es de tipo sólido }
```

y se procede a regresar "hacia abajo" el nombre de la relación resultante, que en este caso es el mismo nombre de la relación definida, o sea, *A1*. Así, el nodo *(2,1)* ya tiene el nombre de la única relación que necesita para operar, arrojando como resultado dos relaciones, de las cuales sólo puede determinar el nombre de la que contendrá los registros que no cumplan con la condición X, por lo que la relación correspondiente a los que sí la cumplan será generada con un nombre asignado por el sistema. De esta manera se procederá a realizar las asignaciones:

```
DIBUJO[2,1]^.YACALC := TRUE;
new(DIBUJO[2,1]^.ARCHIT);
new(DIBUJO[2,1]^.ARCHIF);
DIBUJO[2,1]^.ARCHIT^ := 'ZN!'; {    nombre asignado por el sistema }
DIBUJO[2,1]^.ARCHIF^ := 'A3';  {    nombre conocido por la visita
                                    previa al nodo (3,1) }
```

Realizado esto, es pasado el nombre de la relación A3 al nodo (3,1), con lo que se completa su operación de simplemente definir una relación, realizando las asignaciones

```
DIBUJO[3,1]^.YACALC := TRUE;
new(DIBUJO[3,1]^.ARCHIT);
DIBUJO[3,1]^.ARCHIT^ := 'A3';
```

y pasando el nombre *A3* al nodo *(3,2)*. Como este último requiere de dos relaciones para operar, se analiza "hacia arriba" el nodo *(2,3)* y el nodo *(2,2)*, que es la definición de la relación *A2* y que está sujeta al resultado del nodo *(2,1)*. Al visitar este último, se encontrará que `DIBUJO[2,1]^.YACALC = TRUE`, por lo que la operación no es necesario realizarla, y sólo se procederá a regresar al nodo *(2,2)* el nombre de la relación de la "salida verdadera" de la selección, cuyo nombre se localiza en `DIBUJO[2,1]^.ARCHIT^` (o sea 'ZN!'). Al reggresar al nodo *(2,2)*, se procederá a generar una relación con nombre *A2*, que contendrá la misma información que la *ZN!*, y se harán las asignaciones

```
DIBUJO[2,2]^.YACALC := TRUE;
new(DIBUJO[2,2]^.ARCHIT);
DIBUJO[2,2]^.ARCHIT^ := 'A2';
```

enviando su nombre hacia el nodo *(2,3)*. De éste se pasará nuevamente "hacia arriba" al nodo *(1,3)*, de donde se obtendrá el nombre de la relación *A4*, de tal manera que la operación del nodo *(2,3)* podrá realizarse. Como se puede observar, el nodo posterior (visualmente) al *(2,3)* corresponde a una operación con relaciones y no propiamente a la definición de una relación, por lo que al resultado de la operación en *(2,3)* se le asignará un nombre determinado por el sistema *(Z{N+1}!)*, asignando

```
DIBUJO[2,3]^.YACALC := TRUE;
new(DIBUJO[2,3]^.ARCHIT);
DIBUJO[2,3]^.ARCHIT^ := 'Z{N+1}!';
```

pasando este nombre hacia el nodo *(3,2)*, donde se realizará su operación, dándole el nombre *A5* a la relación resultante (nombre que conocía con la visita previa del nodo *(3,3)*) y asignando los valores correspondientes a los campos *ARCHIT* y *YACALC*. Finalmente, el nodo *(3,3)* recibirá el nombre *A5*, y como este nombre corresponde con el propio de su definición, no será necesario realizar la copia de datos y se finalizará el recorrido y la ejecución del flujograma.

De el recorrido detallado del ejemplo anterior, podemos determinar que los campos *ARCHIT*, *ARCHIF* y *YACALC* de la estructura *ELEMENTO* son indispensables para evitar la realización de operaciones que en un proceso anterior ya habían arrojado datos resultantes.

El campo *ESULTI* de un elemento indicará, de acuerdo a su valor, si el nodo en cuestión es terminal, y por lo tanto, un punto de referencia por el cual es posible iniciar la ejecución del flujograma.

En la figura 5.7 tenemos un ejemplo de un flujograma que cuenta con más de un nodo terminal.

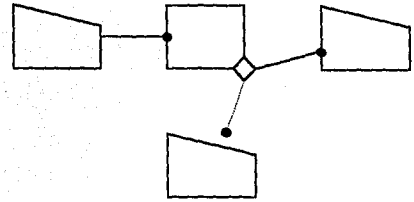


Figura 5.7 Flujograma con dos nodos terminales

Para iniciar la ejecución de un flujograma, debe seleccionarse el nodo terminal con mayor prioridad, y ejecutar toda la rama del flujograma involucrada. Finalizando esto, se procederá a seleccionar otro nodo terminal con menor o igual prioridad, aplicándosele el mismo proceso que al anterior. De esta manera se deberán cubrir totalmente los nodos terminales del flujograma para completar la ejecución del flujograma.

Las prioridades de los nodos, como recordaremos, están dadas por la naturaleza de su operación asociada, es decir, si se está tratando el nodo *(i,j)*, sabremos su prioridad al consultar el arreglo *ARREFIG* de la manera siguiente:

```
PRIORIDAD_I_J := ARREFIG[DIBUJO[i,j]^.NUMFIGURA].PESO;
```

Al momento de recibirse la orden de ejecutar un flujograma, primero se revisan todos los nodos que componen el flujograma, y cuando se encuentra uno que es terminal, se verifica que su operación asociada permita tal situación. De ser así, las coordenadas matriciales del nodo serán introducidas a un árbol binario ordenado de acuerdo al peso (prioridad) de los nodos. Una vez revisados todos los nodos del flujograma, el árbol binario contendrá las coordenadas matriciales de todos los nodos terminales, y sólo bastará recorrerlo en orden, ejecutando cada uno de los nodos señalados por las coordenadas en el árbol, para obtener los resultados adecuados, respetando las prioridades de las operaciones involucradas. Para el ejemplo de la figura 5.7 los dos nodos terminales tienen el mismo peso, por lo que el orden de ejecución depende de la forma en que fue capturado.

Hasta este momento hemos tratado el método de ejecución de un flujograma en la parte que corresponde al orden en el recorrido del mismo, y sólo falta aclarar la forma en que se ejecuta cada una de las operaciones en los nodos, que es prácticamente la única diferencia que existe en la manera de ejecutar un nodo con respecto de otro asociado a una operación distinta. Tal diferencia se da por la naturaleza de cada operación, que se hace notar mediante:

- *El número de operandos necesarios para realizar la operación*

- *El número de relaciones que genera la operación*

- *Un indicador que nos permite saber si la operación puede ser terminal*

- *El número y tipo de argumentos o parámetros que requiere la operación*

- *La sintaxis de la operación en el lenguaje intermedio, que para nuestro caso es el ISBL Extendido.*

Cuando la operación de un nodo ya posee todos los elementos con los cuales tabajar, se procede a armar una cadena en lenguaje ISBL Extendido que la represente, y entonces la pasa al módulo del intérprete de cadenas de comandos, en donde es ejecutada. Por ello llamamos al ISBL Extendido Lenguaje Intermedio del LIDA. El diagrama de la figura 5.8 muestra los pasos para ejecutar la operación de un nodo.

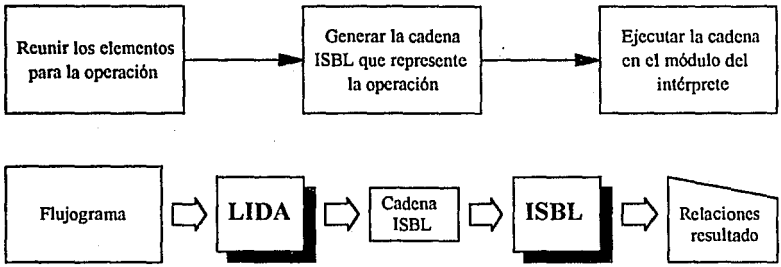


Figura 5.8 Algoritmo de resolución de un nodo

Retomando el ejemplo de la figura 5.6, tendremos que el intérprete de LIDA generará las cadenas de la manera siguiente:

NODO CADENA
(2,1) Z1I=A1:X,A3
(2,2) A2=Z1I
(2,3) Z2I=A2+A4
(3,2) A5=A3.Z2I

Es importante mencionar que las cadenas generadas a partir de dos flujogramas visualmnete iguales pueden diferir. En el caso del ejemplo anterior, las cadenas corresponden si se considera que el nodo

(2,1) primero se ligó al *(3,1)* y luego al *(2,2)*, que el nodo *(2,2)* se ligó primero al nodo *(2,3)* y posteriormente el nodo *(1,3)* al *(2,3)*, y que el nodo *(3,1)* fue ligado primero al *(3,2)* y luego el *(2,3)* al *(3,2)*. Si se hubiese ligado primero el nodo *(2,3)* al *(3,2)* y posteriormente el *(3,1)* al *(3,2)*, las cadenas generadas hubieran sido:

NODO	CADENA
(2,1)	A2=A1:X,Z1!
(2,3)	Z2!=A2+A4
(3,1)	A3=Z1!
(3,2)	A5=Z2!.A3

a consecuancia de un recorrido en diferente orden, el cual como puede observarse, no altera el resultado final de la ejecución del flujograma.

El orden de ejecución se dará de acuerdo al orden en que se establecen las ligas entre los nodos. Cuando se tira por primera vez una línea hacia el elemento *(i,j)*, digamos que desde el elemento *(r,s)*, el primer integrante del arreglo *DIBUJO[i,j]^.OPERANDO[1]* será ocupado con las coordenadas *(r,s)*; si posteriormente se dirige otra línea desde el elemento *(g,h)* hacia el mismo elemento *(i,j)*, entonces será ocupado el integrante *DIBUJO[i,j]^.OPERANDO[2]* con las coordenadas *(g,h)*. Al visitarse el nodo *(i,j)* para su ejecución, se hará la llamada recursiva hacia el elemento señalado por *OPERANDO[1]* primero, y posteriormente hacia el elemento señalado por *OPERANDO[2]*.

Para generar adecuadamente la cadena en ISBL Extendido de la operación de un nodo, se hace uso de los argumentos de la operación, almacenados en las localidades de memoria apuntadas por los integrantes del arreglo *CADPAR*.

El algoritmo completo de ejecución descrito de manera formal, es mostrado en las siguientes páginas:

PROCESO PRINCIPAL;

Se considera un árbol binario, cuyos elementos son del tipo NODARBOL:

```
NODOTREE = ^NODARBOL;

NODARBOL = record
             POSX,      { Columna en la matriz del nodo terminal }
             POSY,      { Renglón en la matriz del nodo terminal }
             PRIO       { Prioridad de ejecución del nodo }      : byte;
             IZQ,       { Apuntador a la rama izquierda }
             DER        { Apuntador a la rama derecha }          : NODOTREE
           end;
```

La variable ARBOL es del tipo NODOTREE y representa el apuntador al nodo padre del árbol binario.

En la rama izquierda de un elemento del árbol, se almacenarán las coordenadas de los nodos del flujograma que tengan prioridad de ejecución inferior o igual a él, y en la rama derecha, los que tengan prioridad superior.

Paso 1: Para toda i,j, 1 <= i <= MAXR y 1 <= j <= MAXC, se realizan lospasos 1.1 y 1.2;

> **Paso 1.1**: Si el nodo (i,j) del flujograma es terminal, entonces se verifica que la operación correspondiente permita tal situación. De ser así, se realiza el paso 1.2, de lo contrario, se envía un mensaje de ERROR y se finaliza la ejecución;
>
> **Paso 1.2**: Se procede a insertar las coordenadas matriciales (i,j) del nodo y su prioridad, señalada por:
> PRIORIDAD_I_J := ARREFIG[DIBUJO[i,j]^.NUMFIGURA].PESO;
>
> en un nuevo elemento de ARBOL:
>
> INSERTA_NODO_EJ (ARBOL,i,j, PRIORIDAD_I_J);

Paso 2: Se recorre el árbol binario, partiendo del apuntador ARBOL, con la llamada:

RECORRE_ARBOL_EJ (ARBOL);

FIN DEL PROCESO PRINCIPAL;

PROCESO INSERTA_NODO_EJ (PTR,REN,COL,PRIORIDAD);

PTR es un apuntador de tipo NODOTREE;

COL es la columna de la matriz DIBUJO donde se localiza el nodo a insertar en el árbol;

REN es el renglón de la matriz DIBUJO donde se localiza el nodo a insertar en el árbol;

PRIORIDAD indica la prioridad de ejecución de la operación asociada al nodo (REN,COL) de la matriz DIBUJO;

Paso 1: Si PTR no apunta a ningún elemento del árbol, entonces se realiza el paso 1.1, de lo contrario, se realiza el paso 1.2;

> **Paso 1.1**: Se genera un nuevo elemento en el árbol, que será señalado por PTR, y se asignan los valores:
>
> PTR^.POSX := COL;
> PTR^.POSY := REN;
> PTR^.PRIO := PRIORIDAD;
>
> dando por terminado el proceso INSERTA_NODO_EJ;
>
> **Paso 1.2**: Si PRIORIDAD <= PTR^.PRIO entonces el nuevo nodo debe generarse en la rama izquierda del elemento señalado por PTR:
>
> INSERTA_NODO_EJ (PTR^.IZQ, REN, COL, PRIORIDAD);
>
> de lo contrario, debe generarse en su rama derecha:
>
> INSERTA_NODO_EJ (PTR^.DER, REN, COL, PRIORIDAD);

FIN DEL PROCESO INSERTA_NODO_EJ;

PROCESO RECORRE_ARBOL_EJ (PTR);

PTR es un apuntador de tipo NODOTREE;

Paso 1: Si PTR apunta a algún elemento del árbol, entonces se proceden a realizar los pasos 1.1 al 1.3, de lo contrario, el proceso RECORRE_ARBOL_EJ se da por terminado;

Paso 1.1: Se recorre la rama del árbol correspondiente a los nodos con prioridad mayor de ejecución al indicado por PTR:

RECORRE_ARBOL_EJ (PTR^.DER);

Paso 1.2: Se procede a realizar el recorrido del flujograma, partiendo del nodo indicado por las coordenadas en el elemento del árbol apuntado por PTR:

TRADUCE_DIBUJO (PTR^.POSY , PTR^.POSX);

Paso 1.3: Se recorre la rama del árbol correspondiente a los nodos con prioridad igual o inferior de ejecución al del apuntado por PTR:

RECORRE_ARBOL_EJ (PTR^.DER);

FIN DEL PROCESO RECORRE_ARBOL_EJ;

FUNCION TRADUCE_DIBUJO (CY, CX);

CX es la columna de la matriz DIBUJO, correspondiente al nodo que va a ser ejecutado;

CY es el renglón de la matriz DIBUJO, correspondiente al nodo que va a ser ejecutado;

La función TRADUCE_DIBUJO regresa el nombre de la relación donde se almacene el resultado de la operación indicada por el nodo DIBUJO [CY,CX];

Paso 1: Si el nodo (CY,CX) ya ha sido calculado (DIBUJO[CY,CX]^.YACALC = TRUE) en un recorrido anterior, entonces se procede a verificar que exista el nombre de la relación donde se almacenó el resultado. Si no existe, significa que el flujograma es redundante y se procede a enviar un mensaje de *ERROR*, finalizando el proceso de ejecución; de lo contrario se realiza la asignación

TRADUCE_DIBUJO := DIBUJO[CY,CX]^.ARCHIT^; o
TRADUCE_DIBUJO := DIBUJO[CY,CX]^.ARCHIF^;

según sea el caso, y se finaliza la función. Si el nodo no ha sido calculado, se realiza el paso 2;

Paso 2: De acuerdo a la operación asociada al nodo (CY,CX) de la matriz, se determina el número de operandos NUMOPDS que se requieren para producir un resultado;

Paso 3: Se determinan los nombres de las NUMOPDS relaciones, con las cuales se debe operar para producir un resultado:

Para toda i, 1 <= i <= NUMOPDS, debe calcularse el nombre REL[i], donde

REL[i] := TRADUCE_DIBUJO(DIBUJO[CY,CX]^.OPERANDO[i].CY,
DIBUJO[CY,CX]^.OPERANDO[i].CX);

Paso 4: Se determina el número NUMPAR de argumentos que requiere la operación de acuerdo a su naturaleza.

Paso 5: Para toda i, 1 <= i <= NUMPAR, se definen el argumento PAR[i], donde:

PAR[i] := DIBUJO[CY,CX]^.CADPAR[i];

Paso 6: Con los nombres de las relaciones REL y los argumentos PAR, se forma una cadena de caracteres CAD_EJEC, según corresponda a la sintaxis de la operación dada en el lenguaje intermedio, que para nuestro caso es el ISBL Extendido.

Paso 7: La cadena CAD_EJEC es ejecutada en el módulo del lenguaje intermedio. De acuerdo a nuestro sistema, esto se hace siguiendo los mismos procedimientos aplicados a una cadena introducida por un usuario en el intérprete de comandos ISBL.

Paso 8: El nombre de la relación resultante de la ejecución de la cadena CAD_EJEC, es asignado a la función TRADUCE_DIBUJO y a la variable DIBUJO[CY,CX]^.ARCHIT^ o DIBUJO[CY,CX]^.ARCHIF^, según sea el caso.

FIN DEL PROCESO TRADUCE_DIBUJO;

5.3 ESTRUCTURA INTERNA DEL INTERPRETE

Una de las herramientas que fueron desarrolladas para lograr el sistema SABRE y la implementación de LIDA, fue el Intérprete de instrucciones ISBL Extendido. Adicionalmente, se desarrolló un intérprete de flujogramas LIDA que los transforma en instrucciones ISBL, lo cual fue explicado oportunamente en el apartado anterior. En seguida explicaremos la manera en que una cadena ISBL Extendido, es traducida en llamadas a subrutinas que representan las operaciones relacionales.

5.3.1 Intérprete de instrucciones ISBL

El intérprete de instrucciones ISBL posee tres módulos principales los cuales son utilizados secuencialmente para que se produzca un resultado (ver la figura 5.9).

Para trabajar, el intérprete requiere de una cadena de caracteres de entrada, la cual indicará las operaciones que se quieren ejecutar, dicha cadena de entrada puede provenir de tres fuentes: el tecleo en línea por parte del usuario, un archivo de comandos, o un flujograma.

La primera fase a la que se somete la cadena con las instrucciones, es al reconocimiento de los elementos básicos que las componen, tales como símbolos e identificadores. A esta fase se le conoce como *Análisis Lexicográfico*, en la cuál también se determinan las estructuras elementales con las cuales se tiene que operar. Asimismo, es en esta etapa donde se determina si en la cadena existen símbolos permitidos por el lenguaje.

La segunda fase consiste en el *Análisis Sintáctico*, es decir, la fase donde se verifica que el orden en que aparecen los símbolos y elementos básicos de las operaciones en la cadena, se apeguen a las normas establecidas para el lenguaje ISBL Extendido.

La tercera y última fase, consiste en la ejecución de las operaciones indicadas en la cadena, produciéndose los resultados de acuerdo a los datos de entrada indicados en la misma.

A continuación, daremos una explicación más detallada del funcionamiento específico de cada una de las tres fases que comprenden el módulo correspondiente al intérprete de instrucciones ISBL.

5.3.2 Análisis lexicográfico

Esta etapa tiene una doble función: además de determinar la validez de cada uno de los caracteres en la cadena de entrada, va formando un esquema donde se indiquen los componentes básicos de las operaciones involucradas. Tal esquema es sobre el cuál se trabajará en las dos fases posteriores, y su

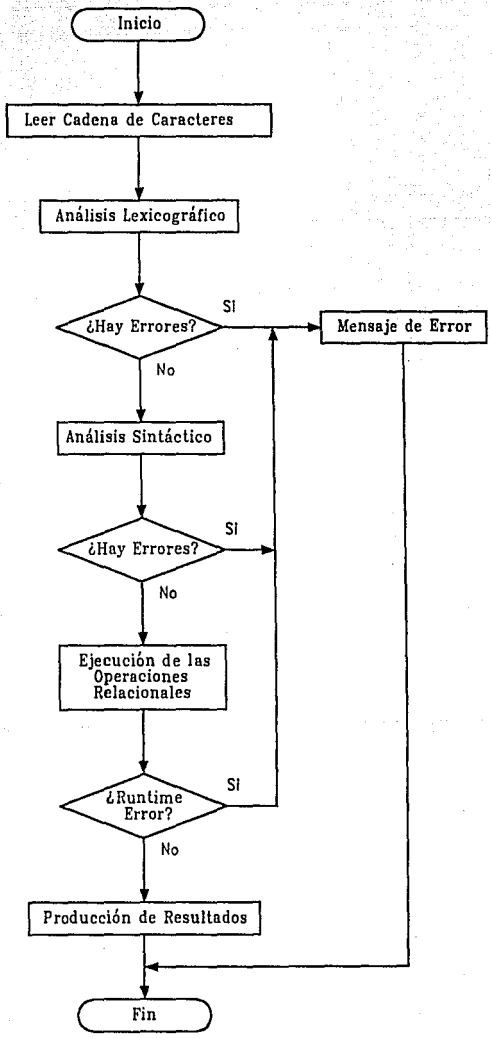


Figura 5.9 Etapas de interpretación de una instrucción ISBL Extendido

construcción es de vital importancia, dado que esto ayudará a una interpretación mas rápida y eficiente del problema que el usuario desea resolver.

El esquema del que estamos hablando está compuesto básicamente por dos estructuras: una que indica el tipo de cada uno de los componentes atómicos de la instrucción, y otra que contiene tanto los nombres de los identificadores involucrados, como las constantes numéricas y alfanuméricas citadas en la cadena. Llamemos a tales estructuras INSTR y ARRIDEN respectivamente, declaradas como:

```
INSTR : array [1..MAXINSTR] of
        record
           INS : símbolo_ISBL;
           NUM : integer;
        end;

ARRIDEN : array [1..MAXIDEN] of string;
```

Con el siguiente ejemplo, se muestra el funcionamiento de esta etapa.

Si tenemos la cadena de entrada

```
MAXSUEL = NOMINA : (SUELDO > 2000000) % NOMBRE , SUELDO
```

se formarán las estructuras con el contenido:

i	INSTR[i].INS	INSTR[i].NUM
1	Identificador	1
2	Igual	
3	Identificador	2
4	Selección	
5	Paréntesis izquierdo	
6	Identificador	3
7	Mayor que	
8	Constante numérica	4
9	Paréntesis derecho	
10	Proyección	
11	Identificador	5
12	Coma	
13	Identificador	6

i	ARRIDEN[i]
1	'MAXSUEL'
2	'NOMINA'
3	'SUELDO'
4	'2000000'
5	'NOMBRE'
6	'SUELDO'

Como puede observarse, el campo *NUM* de cada componente del arreglo *INSTR*, contiene el índice que señala en qué componente del arreglo *ARRIDEN* se encuentra el nombre del identificador o el valor de la constante, según sea el caso.

Una vez conformadas estas estructuras, y de no detectarse algún caracter no permitido por el lenguaje, se pasa a la siguiente fase del intérprete.

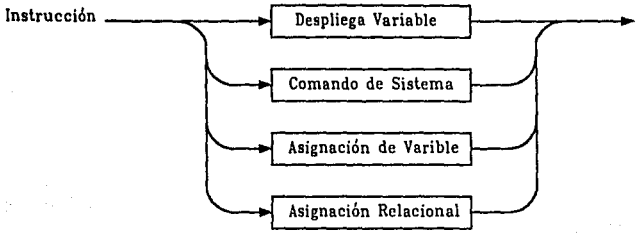
5.3.3 Análisis sintáctico

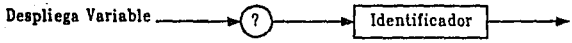
En esta etapa de interpretación, sólo se utiliza la estructura *INSTR* generada en la primera fase. Como es de comprenderse, para la revisión de la sintaxis es más facil y rápido analizar este esquema, que la cadena de caracteres original.

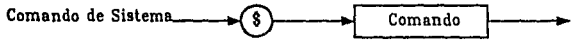
Todo lenguaje posee una sintaxis propia, es decir, un conjunto de reglas que indican la manera (el orden) en que deben aparecer sus elementos básicos, para formar un texto (instrucción) que sea entendido por el interlocutor (la computadora).

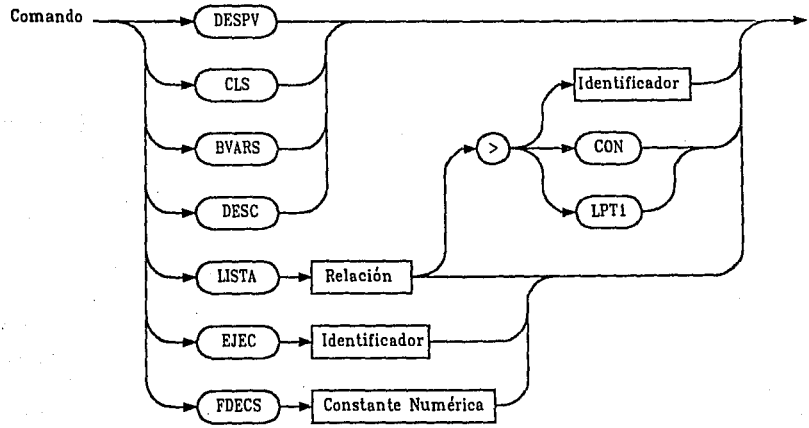
El objetivo de esta fase es verificar precisamente que cada uno de los elementos básicos que comprenden la instrucción ISBL, aparecen en el orden adecuado, apegandose a las reglas establecidas, para entender y ejecutar lo indicado. Si en determinado momento este orden no se respeta, el sistema envía un mensaje indicando la falla y se concluye con el análisis.

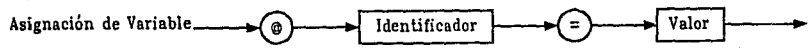
La sintaxis del lenguaje ISBL Extendido es mostrada en los diagramas siguientes:

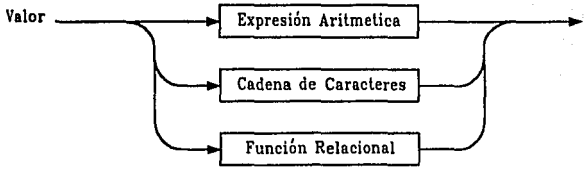

Instrucción
Despliega Variable
Comando de Sistema
Asignación de Varible
Asignación Relacional

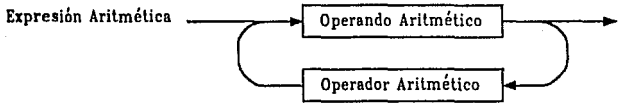

Despliega Variable
?
Identificador

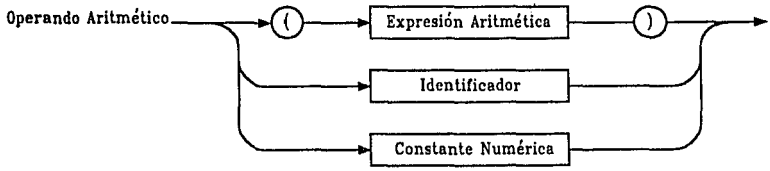

Comando de Sistema
$
Comando

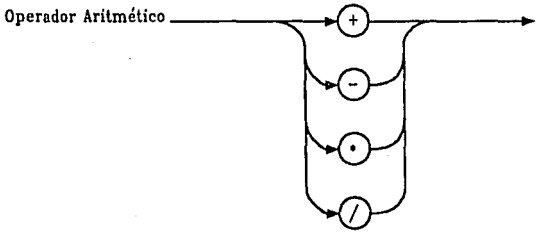

Comando
DESPV
CLS
BVARS
DESC
LISTA
Relación
EJEC
Identificador
FDECS
Constante Numérica
>
Identificador
CON
LPT1

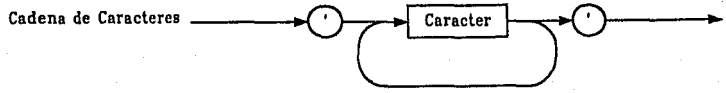

Asignación de Variable
@
Identificador
=
Valor

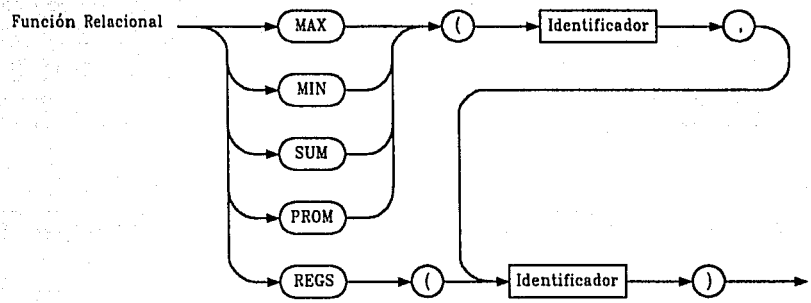

Valor
Expresión Aritmetica
Cadena de Caracteres
Función Relacional

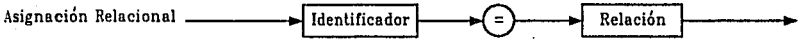

Expresión Aritmética
Operando Aritmético
Operador Aritmético

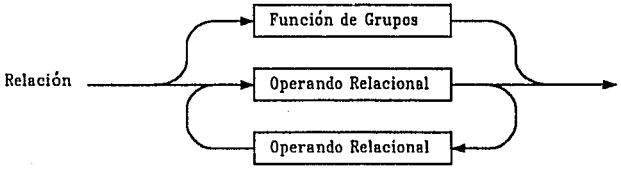

Operando Aritmético
(
Expresión Aritmética
)
Identificador
Constante Numérica

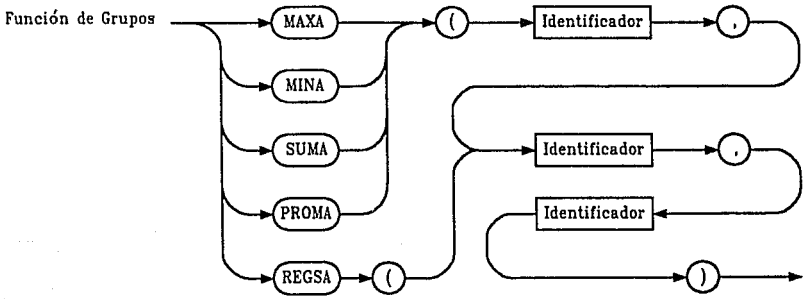

Operador Aritmético
+
-
*
/

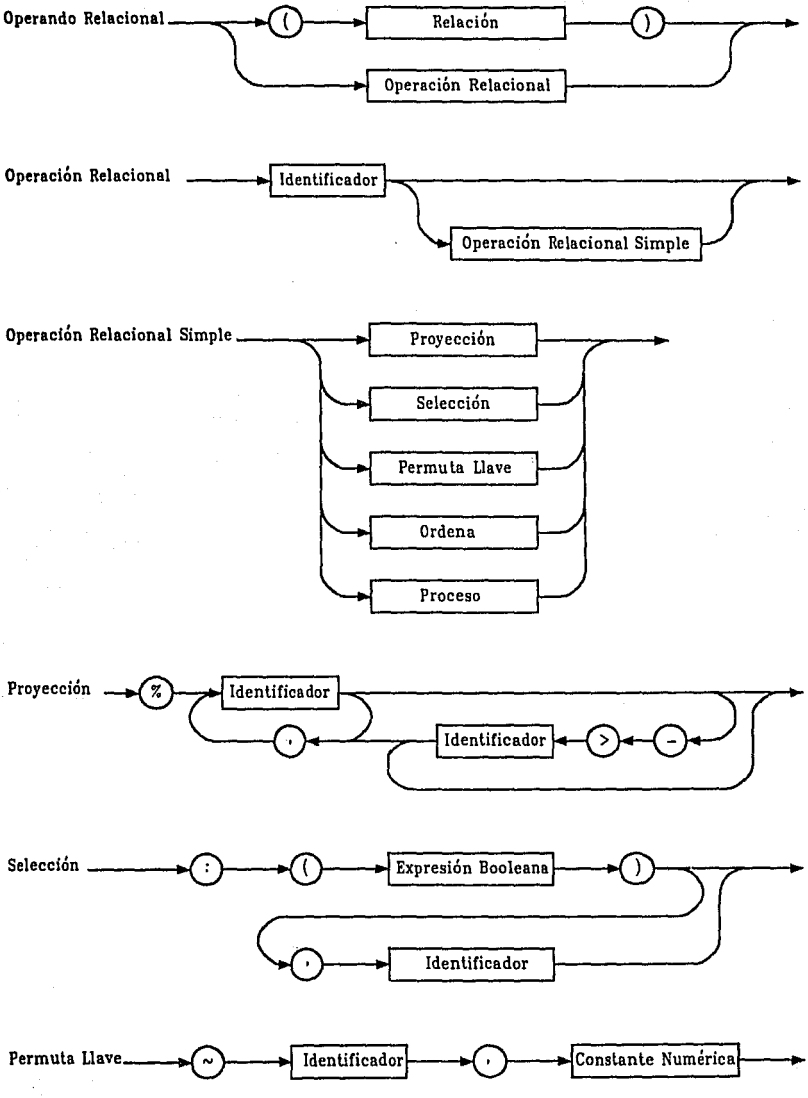

Cadena de Caracteres
'
Caracter
'

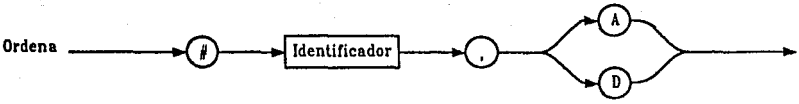

Función Relacional
MAX
MIN
SUM
PROM
REGS
(
Identificador
,
Identificador
)

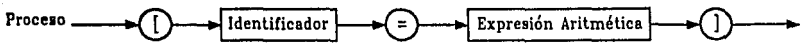

Asignación Relacional
Identificador
=
Relación

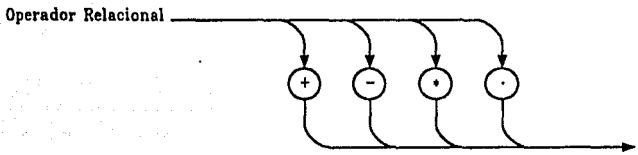

Relación
Función de Grupos
Operando Relacional
Operando Relacional

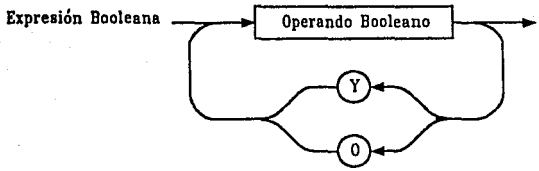

Función de Grupos
MAXA
MINA
SUMA
PROMA
REGSA
(
Identificador
,
Identificador
,
Identificador
)

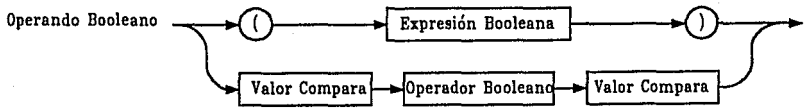

Operando Relacional
(
Relación
)
Operación Relacional
Operación Relacional
Identificador
Operación Relacional Simple
Operación Relacional Simple
Proyección
Selección
Permuta Llave
Ordena
Proceso
Proyección
%
Identificador
,
Identificador
>
-
Selección
:
(
Expresión Booleana
)
,
Identificador
Permuta Llave
~
Identificador
,
Constante Numérica

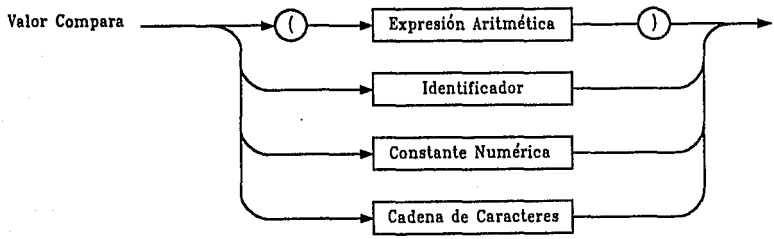

Ordena
#
Identificador
,
A
D


Proceso
[
Identificador
=
Expresión Aritmética
]

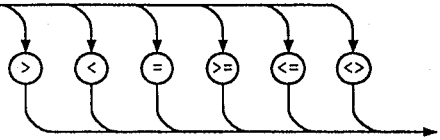

Operador Relacional
+
-
*
.


Expresión Booleana
Operando Booleano
Y
O

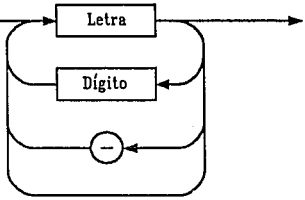

Operando Booleano
(
Expresión Booleana
)
Valor Compara
Operador Booleano
Valor Compara


Valor Compara
(
Expresión Aritmética
)
Identificador
Constante Numérica
Cadena de Caracteres

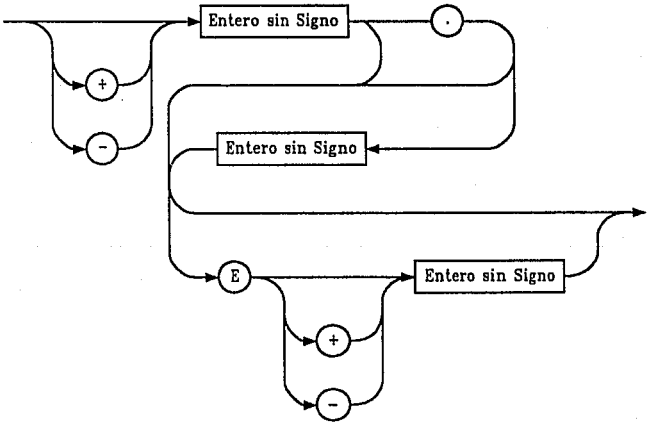
Operador Booleano



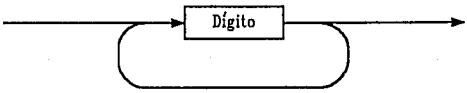
Identificador



Constante Numérica



Entero sin Signo



5.3.4 Ejecución de una cadena

Una vez completada y aprobada la etapa de sintaxis, se procede con le ejecución de las operaciones involucradas en la cadena de entrada. Para esta estapa se utilizan las estructuras INSTR y ARRIDEN generadas en el análisis lexicográfico. El orden de ejecución es el señalado por la sintaxis misma del lenguaje.

Durante la ejecución de las operaciones, es posible que sea detectado un error, de los comunmente conocidos como Runtime error. Estos tienen varias causas, entre las que se encuentran:

- *Nombres de relaciones inexistentes*

- *Nombres de atributos inexistentes en la relación indicada*

- *Comparación o mezcla de valores de diferente tipo*

Es importante aclarar que estos errores son detectados hasta la etapa de ejecución, debido a que no es sino hasta esta fase cuando las relaciones son abiertas y es posible determinar tanto su existencia, como la de los tipos y nombres de atributos que contiene cada una de ellas.

Cuando se recorre el esquema que contiene la estructura *INSTR* para su ejecución, y es necesario conocer el nombre de un identificador o el valor de una constante, entonces se recurre al arreglo *ARRIDEN*: Si el i-ésimo elemento del arreglo *INSTR* corresponde a un identificador o constante, entonces el nombre o valor correspondiente está contenido en *ARRIDEN[INSTR[i].NUM]*. Retomando el ejemplo anterior, si se va a realizar la ejecución de la operación de selección, entonces es necesario conocer el nombre de dos identificadores y de una constante: el nombre de la relación es *ARRIDEN[INSTR[3].NUM] = 'NOMINA'*, el nombre del atributo es *ARRIDEN[INSTR[6].NUM] = 'SUELDO'*, y el valor de la constante es *ARRIDEN[INSTR[8].NUM] = 2000000.*

Cuando se poseen todos los elementos para realizar una operación, entonces tales elementos son transferidos a la librería del lenguaje para que sea producido un resultado. Si la operación relacional que se va a resolver es intermedia, entonces la relación resultante será generada con un nombre asignado por el sistema, y este nombre será pasado a la operación subsecuente si es necesario.

De acuerdo al ejemplo y al orden de resolución indicado por la sintaxis del lenguaje, se resuelven las operaciones de izquierda a derecha, tal como se realiza en las operaciones aritméticas. Para el ejemplo, es necesario resolver primero la selección y posteriormente, al resultado arrojado por ella, debe sometersele a una operación de proyección. De acuerdo a lo citado con anterioridad, la selección produce una relación con un nombre *ZZn!* asignado por el sistema, donde n es un número natural consecutivo que

no se repite para todo el conjunto de operaciones incluidas en la cadena de caracteres. Si tal nombre de relación es *ZZ1!*, entonces la segunda operación resuelta será

```
ZZ1! % NOMBRE , SUELDO
```

dando una relación resultado *ZZ2!*; y por último, se hará la asignación *MAXSUEL = ZZ2!*

5.4 CONCEPTOS GENERALES DE OPERACION

Como se sabe, SABRE es un sistema que utiliza LIDA como lenguaje principal para generar aplicaciones, que pueden ir desde simples consultas a una base de datos, hasta un sistema de información completo. SABRE es una plataforma ejemplo, en la que descansa LIDA; su desarrollo y aplicación no debe considerarse la parte principal de este trabajo, sino el papel que juega LIDA dentro de un sistema como SABRE, ya que nuestra aportación principal, es definir un lenguaje iconográfico que pueda manipular bases de datos relacionales, y en un segundo plano la implementación del mismo, sin que esto reste importancia al sistema que lo soporta.

SABRE es el resumen de toda la teoría que hemos presentado hasta el momento, es el punto donde se reúnen conceptos como *Bases de Datos, CASE, Sistemas Gráficos, Algebrá Relacional, Flujograma, Descriptor de Archivos, Captador de Datos, Lenguaje Iconográfico* etc. A continuación describiremos la forma en que opera SABRE en cada uno de sus módulos y la manera en que se maneja LIDA por medio de un editor gráfico, e iremos abarcando desde la instalación del sistema hasta el manejo completo del mismo.

5.4.1 Nomenclatura

Este subcapítulo aborda sólo el manejo operativo del sistema, sin tomar en cuenta su propósito (un ejemplo de una aplicación con LIDA será analizada posteriormente). Esperamos que el lector tenga ciertos conocimientos sobre la operación de microcomputadoras, mas sin embargo esto no es una condición necesaria.

La implementación de SABRE se realizó sobre una microcomputadora con capacidad de 640Kb de memoria, monitor color del tipo VGA y disco duro de 80Mb. La programación se llevó a cabo bajo un ambiente operativo MS-DOS, y fue elaborado en lenguaje Pascal, para lo cual, se utilizó el compilador Turbo Pascal ver. 5.5 de Borland International [TURBO5].

En lo sucesivo, se considera la siguiente notación para fines prácticos de la explicación operativa de SABRE:

La mayoría de los teclados de las computadoras corresponden al estándar QWERTY, que se asemeja a las máquinas de escribir comunes, y cuenta con teclas especiales que generalmente sirven para mover el cursor, facilitar la captura de información numérica y alfanumérica, y otras más muy especializadas como la tecla conocida como retorno de carro y la tecla de escape, así como otras que generalmente son utilizadas por los programadores para realizar funciones específicas predefinidas. Cada que se tenga que oprimir una tecla especial como las mencionadas, nos referiremos a ella mostrándola entre símbolos "<" y ">" por ejemplo:

<ENTER>	Retorno de carro, a veces se presenta como <RETURN> o <INTRO> en algunas máquinas.
<ESC>	Tecla de escape
<BS>	Regreso de espacio (esta tecla generalmente es representada por una flecha apuntando hacia la izquierda)
<ARRIBA>	Flecha hacia arriba, generalmente tanto ésta tecla, como las siguientes, se encuentran dentro del teclado adicional numérico o por separado, y sirven para mover el cursor sobre la pantalla.
<ABAJO>	Flecha hacia abajo
<DER>	Flecha hacia la derecha
<IZQ>	Flecha hacia la izquierda
<PgDn>	Página hacia abajo
<PgUp>	Página hacia arriba
<Home>	Regreso al principio de línea
<End>	Fin de línea

En ocasiones utilizaremos abreviaciones del inglés, ya que los teclados estándar en su mayoría vienen con las mismas abreviaciones, por razones propias del mercado y del fabricante. Así encontraremos PgUp abreviación de Page Up ó Página Arriba, <BS> abreviación de Back Space o Regreso de Espacio.

Todas las instrucciones del sistema operativo aparecerán en mayúsculas.

Cada que se haga referencia a cierta unidad de disco, ya sea disco duro o flexible será mostrado el prompt respectivo (lo constituye la letra en mayúscula de la unidad que generalmente son: A y B para unidades de disco flexible y C para la unidad de disco duro; y el símbolo ">") de la unidad, así, si en cualquier momento se presenta C>, significa que nos encontramos en la unidad de disco duro.

5.4.2 Instalación

La instalación de SABRE en una computadora, se reduce a un simple algoritmo, ya que el tamaño de los archivos del sistema caben dentro de un disco flexible, por lo cual sólo es necesario realizar una copia de los archivos del sistema, o bien, introducir el disco en una unidad de lectura. De cualquier forma el procedimiento de instalación es el siguente:

- Se enciende la computadora y se carga el sistema operativo normalmente.

- Si la instalación se realiza sobre el disco duro, es necesario crear un subdirectorio donde sea alojado el sistema, para lo cual se teclea lo siguiente:

C>MD SABRE seguido de <ENTER>

luego

C>CD SABRE seguido de <ENTER>

- Se introduce el disco del sistema SABRE en la unidad de disco flexible de la computadora, y se copia el contenido de éste hacia el disco duro tecleando:

C>COPY A:*.* seguido de <ENTER>

- Si la instalación se realiza sobre una unidad de disco flexible, sólo se introduce el disco del sistema SABRE en la unidad de disco disponible.

5.4.3 Archivos de SABRE

El sistema se compone de ciertos archivos que se pueden dividir en tres tipos :

Archivos de Sistema:

SABRE.EXE Sistema SABRE ejecutable
SABRE.OVR Sistema SABRE overlay

SABRE.EXE corresponde al sistema en su parte residente en memoria, y SABRE.OVR es la parte no residente complementaria a SABRE.EXE denominada por Turbo Pascal como overlay [TURBO5].

Archivos de Utilería:

SABRE.CFG Configuración del sistema
DESCRIP.DES Archivo de Descriptores

Dentro de SABRE.CFG se almacena la configuración del sistema, que incluye el tipo de monitor y la ruta del directorio del archivo de descriptores. El archivo DESCRIP.DES contiene los descriptores de archivos.

Archivos Generados:

*.PRN	Impresiones en formato ASCII
*.LID	Flujogramas
*.LIP	Procedimientos
*!.	Archivos de Paso
*.	Bases de Datos

Cuando se realiza una impresión y ésta se envía hacia un archivo ASCII, el sistema nombra el archivo con extensión .PRN. Cuando se guarda un flujograma, el sistema nombra al archivo correspondiente con extensión .LID. Los archivos con extensión .LIP, son creados al momento de ejecutar la opción de Genera texto en el módulo LIDA; estos archivos contienen las líneas o cadenas en ISBL Extendido que representan un flujograma. Todos los archivos que contengan como sufijo un signo de admiración "!", son archivos creados por LIDA al momento de ejecutar alguna operación relacional, son sólo archivos de paso para realizar operaciones intermedias; se recomienda incluso eliminarlos periódicamente del sistema, vía mantenimiento de descriptores como se verá más adelante. Por último, las bases de datos relacionales se encuentran almacenadas en archivos sin extensión.

5.4.4 Acceso al sistema

- Estando en la unidad y directorio correspondiente se teclea:

C>SABRE seguido de <ENTER>, si su instalación se realizó en disco duro, o de otra forma

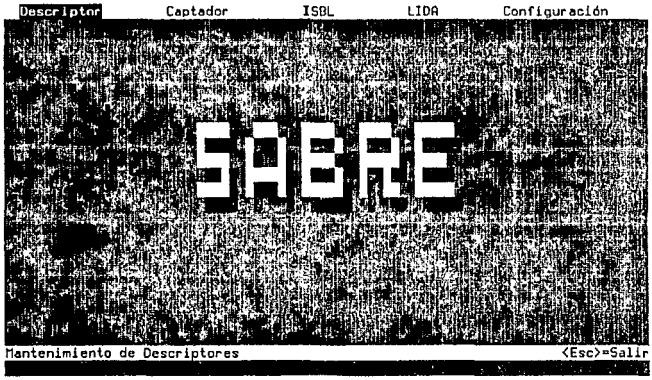
A>SABRE y <ENTER>

- Espere a que el sistema operativo cargue el LIDA en la memoria de su computadora, y aparezca la pantalla principal del sistema, mostrando cinco opciones que a saber son:

- *Descriptor*
- *Captador*
- *ISBL*
- *LIDA*
- *Configuración*

- La pantalla principal se divide en cuatro partes (vea la siguiente pantalla): una línea dedicada al menú principal, esta línea se encuentra en la parte superior de la pantalla; el cuerpo del sistema, donde se

dedica el espacio para las ventanas de trabajo y captura de datos; una línea para el despliegue de mensajes; y una última línea que muestra el estado del sistema, en la cual el sistema envía ciertas palabras cuando detecta que las teclas como el <INS>, <CAPS-LOCK> o el <NUM-LOCK> han sido activadas.



5.4.5 Procedimientos comunes

Antes de entrar de lleno hacia la explicación operativa del sistema, analizaremos los procedimientos que se han estandarizado sobre todo el sistema SABRE.

- Movimiento del cursor

En la mayor parte del sistema, el cursor es representado con una barra perfectamente distinguible que denominamos cursor-barra, el movimiento de este cursor se puede efectuar utilizando las teclas <ARRIBA>, <ABAJO>, <IZQUIERDA> y <DERECHA>.

- Modificación de datos capturados

Al momento de realizar la captura de los datos, el cursor-barra se convierte en un guión sobre la línea de captura, y es posible utilizar adicionalmente las teclas: <Home> y <END> para posicionar el cursor a la primera o última posición de la línea, <INS> para poner el sistema en modo inserción de caracteres (el sistema coloca las letras "INS" sobre la última línea de la pantalla) y poder capturar letras y números entre

los datos, <DEL> para borrar caracteres, también puede utilizarse la tecla de regreso de espacio o <BS> para el mismo fin, <PgUp> para posicionar el cursor-barra sobre el primer registro del archivo y <PgDn> para posicionarlo en el último.

- *Alta de registros*

Cuando se requiere realizar una alta de cualquier registro que contenga información, sólo debe oprimirse la tecla <INS>, y al instante dará comienzo el proceso de inserción.

- *Baja de registros*

Similarmente a la alta de cualquier registro, debe oprimirse la tecla <DEL>, previamente habiendo posicionado el cursor-barra sobre el registro a eliminar, el sistema envía un mensaje para confirmar la baja del registro: si se oprime la tecla <S> el registro será eliminado, de lo contrario se debe oprimir la tecla <N>.

- *Salida de un proceso*

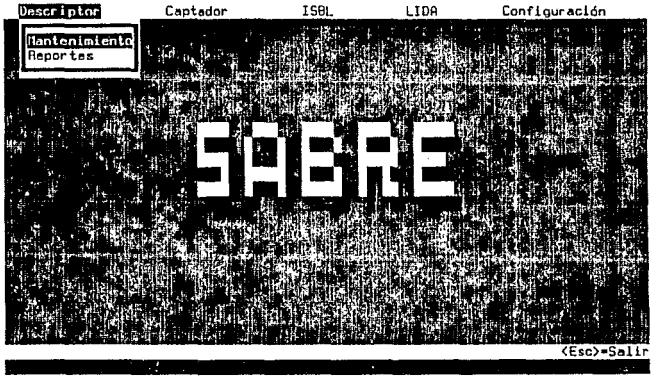
Para salir de cualquier proceso es necesario oprimir la tecla <ESC>. Para abandonar el sistema basta teclear <ESC> sobre el menú principal y el sistema envía un mensaje para confirmar la salida.

5.5 DESCRIPCION DEL SISTEMA

Modularmente describiremos el funcionamiento del sistema SABRE, para lo cual, es necesario que el lector tome en consideración el subcapítulo anterior, en donde se describen los procedimientos generales y la definición de teclas especiales.

5.5.1 Descriptor

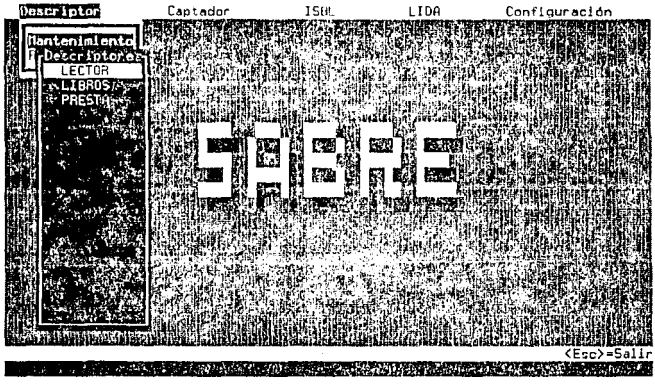
Al inicio el cursor-barra se coloca en la primera opción del menú horizontal, para accesar ésta, presione <ENTER>, al momento se abre un submenú que muestra dos opciones:



La primera opción permite realizar el mantenimiento de los descriptores de archivos, y la segunda permite generar los reportes correspondientes al contenido de los mismos.

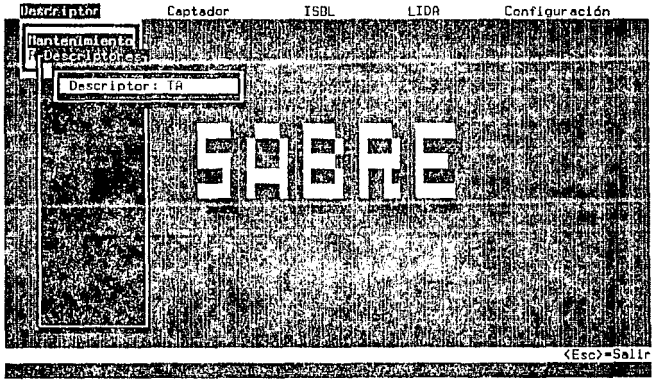
MANTENIMIENTO

Oprimiendo <ENTER> sobre la primera opción, aparecerá una ventana en la que se muestran los nombres de los archivos contenidos en el archivo de descriptores.

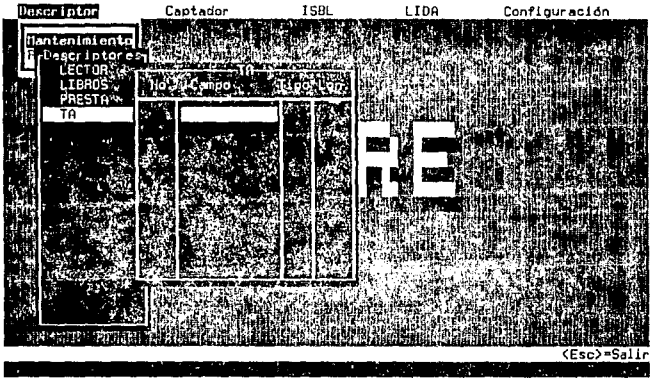


- Alta, baja y modifcación de los descriptores de archivos

Para generar un nuevo descriptor, sóto se necesita pulsar <INS> sobre la ventana de los archivos del descriptor; a continuación será requerido el nombre del archivo:



En seguida se deberá definir la descripción de la estructura del archivo, para ello se coloca el cursor-barra sobre el nombre del descriptor que se acaba de dar de alta y se teclea <ENTER>, en ese momento se abre una nueva ventana mostrando el espacio para poder definir la estructura:

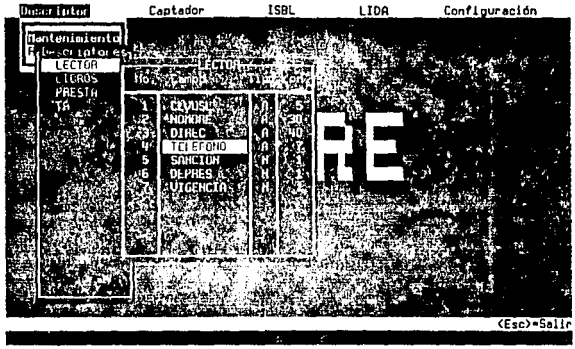


esta ventana incluye de forma columnar los siguientes campos :

No.	Número consecutivo del campo.
Campo	Nombre del campo
Tipo	Tipo (A= Alfanumérico, N= Numérico)
Lon	Longitud del campo

En ese momento se puede comenzar a capturar los atributos. Tecleando <INS> o <ENTER>, el sistema abre el espacio para la captura del nombre del campo. El número consecutivo del campo es asignado automáticamente.

El atributo queda insertado y los campos de tipo y longitud tienen valores iniciales, para cambiarlos, se debe posicionar el cursor-barra sobre la columna respectiva y pulsar <ENTER>, el sistema permite la captura de dichos valores, el tipo sólo puede ser "A" (alfanumérico) o "N" (numérico), y la longitud debe encontrarse dentro del rango 1 a 78; también se puede cambiar el nombre del campo. Para terminar debe oprimirse <ESC> y responder a la pregunta que hace el sistema para abandonar.



Para eliminar un archivo conjuntamente con su descriptor, es necesario posicionar el cursor-barra sobre el archivo a eliminar y teclear <DEL>, el sistema hará una pregunta para confirmar si el archivo señalado se da de baja.

Para modificar el contenido de cualquiera de los descriptores, es necesario posicionar el cursor-barra sobre el nombre del archivo y teclear <ENTER>; y de la misma forma como se cambian los valores iniciales, se procede a efectuar el procedimiento de modificación. El único campo que no es posible modificar es el No. consecutivo. Cabe señalar que si se cambia el nombre de cualquiera de los campos, y el archivo ya contiene datos o registros, entonces se perderá la información de ese campo.

También es posible añadir más campos, e insertar campos entre dos consecutivos, el sistema reordena su número de campo.

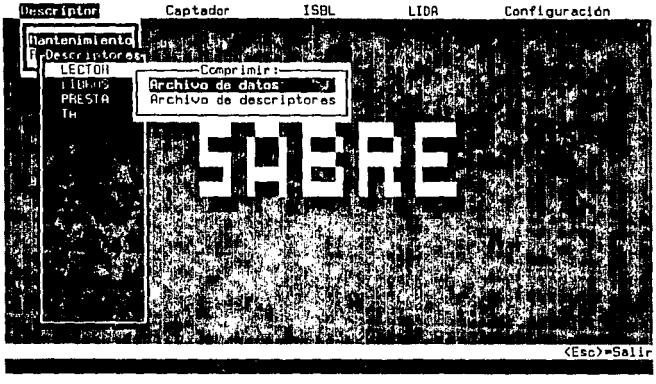
- Captura de datos de archivos

Si al finalizar la descripción del archivo, se requiere realizar la captura de los datos, es necesario teclear <F2>, en ese momento será abierta una ventana de captura.

Refiérase al siguiente apartado 5.5.2. Captador, ya que la operación es la misma.

- Compresión de datos

Al momento de borrar registros, el sistema marca los registros borrados, por lo tanto en los archivos de datos como en el propio archivo de descriptores se realiza una baja lógica de registros, este tipo de borrado tiene la ventaja de realizar las bajas de los registros a mayor velocidad. Lo anterior implica que un archivo muy utilizado pudiera contener registros borrados sin que estos hayan desaparecido físicamente, lo que hace pensar en la forma para dar de baja física estos registros por medio del sistema. Al oprimir las teclas <ALT><C> simultáneamente aparecerá un submenú como se muestra en seguida:



La primera opción comprime es el archivo en el que se encuentra posicionado el cursor-barra.

Para poder determinar en que momento se deben de comprimir los datos, debe de considerarse lo siguiente:

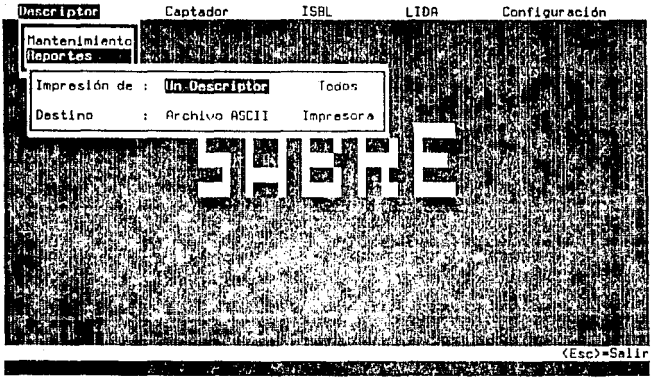
- El espacio libre en disco duro
- La velocidad de cálculo del sistema
- El borrado frecuente de registros de un archivo
- El borrado frecuente de archivos

Es lógico pensar que al comprimir un archivo de datos, éste reduce su tamaño, puesto que se realiza la baja física de los registros; además, cuando LIDA efectúa una operación relacional, siempre se recorren todos los registros de los archivos, ya que su almacenamiento es secuencial y también su acceso, por lo que posiblemente esto retarde un poco las operaciones.

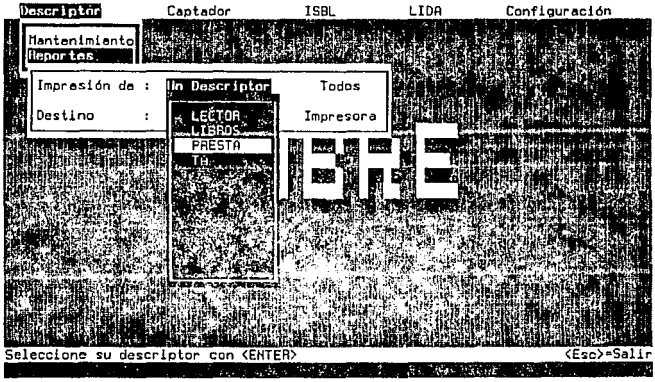
La depuración de los archivos del sistema debe hacerse con cierta frecuencia, eliminando aquellos archivos que no son utilizados, y aquellos generados como archivos de paso por el propio sistema, estos últimos son fácilmente reconocidos puesto que terminan con un símbolo !. Al finalizar la depuración debe utilizarse la opción de compresión sobre el descriptor.

REPORTES

Al accesar esta opción, es abierta una ventana la cual contiene dos menús horizontales como se muestra en la pantalla siguiente:



Los reportes se realizan tomando la información almacenada por el archivo de descriptores, se puede seleccionar si así se requiere el reporte de un descriptor o de todos, únicamente si se opta por la impresión de un descriptor, entonces, otra ventana se abrirá mostrando todos los descriptores que existen, para lo cual se debe seleccionar el descriptor a imprimir.



El segundo submenú determina el destino de la impresión; es posible generar el reporte hacia la impresora o hacia un archivo en formato ASCII. Si se desea emitir un reporte por la impresora, el sistema envía un mensaje para asegurar que la impresora está en línea. Si el reporte se envía hacia un archivo,

entonces el sistema requerirá que se capture el nombre de dicho archivo sin extensión, si éste ya existiera el sistema envía un mensaje para confirmar su sobreescritura, los archivos generados por medio de esta opción siempre tendrán extensión .PRN, por lo que no es requerido que se capture. Un reporte en un archivo ASCII, puede ser procesado por otro sistema o programa, por ejemplo su información puede modificarse por medio de un procesador de palabras que soporte el estándar ASCII.

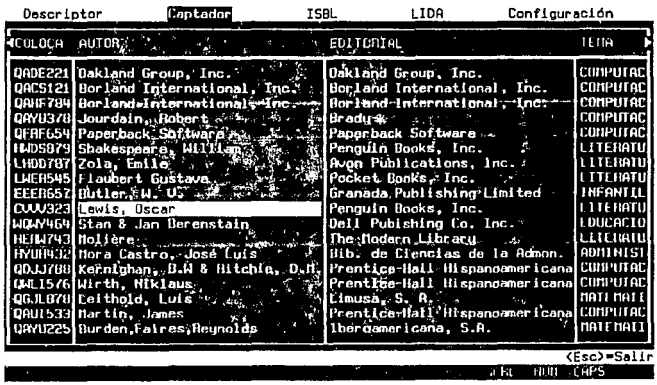
5.5.2 Captador

De alguna manera el captador de archivos representa todo un sistema de captura de datos, en donde se pueden realizar altas, bajas y cambios; a los registros de las bases de datos definidas, su manejo es sumamente sencillo. Una vez que se han definido los campos o atributos de una base de datos específica, por medio del subsistema de mantenimiento de descriptores, es posible proceder a la captura de los datos.

Al accesar esta opción vía opción Captador desde el menú general, o vía <F2> sobre la ventana del descriptor de archivos, se abre otra ventana, la cual presenta los atributos de forma columnar; el sistema estará listo para que se comience con la captura de datos.

- Captura de datos

La ventana de captura de datos, se constituye por varias columnas, cada una de ellas tiene como encabezado, el nombre del atributo que se definió en el descriptor de la base de datos.



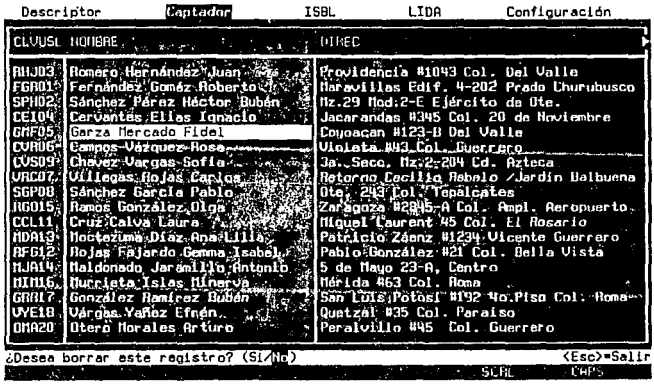
El movimiento del cursor-barra es libre a través de la ventana; con ayuda de las flechas del teclado, se puede posicionar el cursor hacia donde se requiera, o también utilizar la tecla <PgUp> para llevar el cursor hacia el primer registro de la base de datos, o bien, <PgDn> para ir al fin del archivo; incluso puede

teclear <Home> para posicionar el cursor hacia la primera columna, o <END> para posicionarlo sobre la última. Si el tamaño de las columnas de la relación, es mayor al tamaño de la ventana, entonces el captador simulará una ventana virtual; con ayuda de las flechas <IZQ> y <DER> es posible continuar moviéndose sobre todas las columnas la relación. Las flechas que aparecen en los bordes superiores de la ventana, indican que hay más información hacia la derecha o hacia la izquierda.

Si el cursor se encuentra sobre alguna columna y se oprime <INS>, el sistema abre el espacio para comenzar a capturar los datos del nuevo registro. El primer dato que se requiere capturar es aquel en el que se halla el cursor-barra al momento de oprimir <INS> (los demás campos deberán capturarse posteriormente).

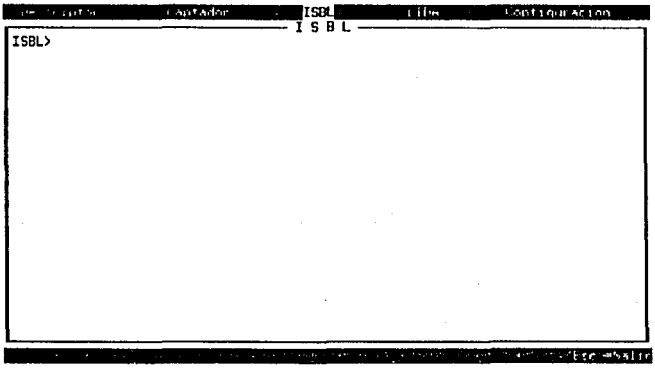
Al momento de capturar un dato, se pueden utilizar las teclas <BS> para borrar caracteres, o <INS> para insertarlos, así como las flechas <DER> e <IZQ> para mover el cursor en ambas direcciones, o bien, <HOME> para posicionar el cursor al principio del campo, o <END> para posicionarlo al final; para terminar con la captura sólo se oprime <ENTER> y la actualización habrá concluído; el proceso es el mismo para los siguientes campos. Si es necesario recapturar algún campo, debe hacerse de la misma manera.

Para dar de baja algún registro de la base de datos, se requiere posicionar el cursor-barra sobre dicho registro y teclear <DEL>, el sistema envía un mensaje para confirmar si el registro se desea eliminar.



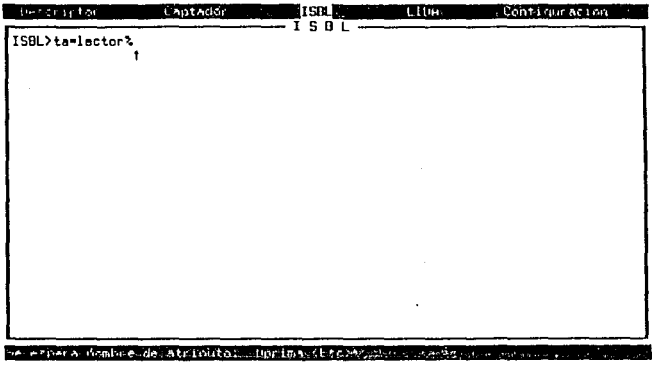
5.5.3 ISBL

Esta opción del menú permite manejar el lenguaje de cadena ISBL Extendido para trabajar con las bases de datos relacionales declaradas con el descriptor. Al accesar el intérprete, en seguida se muestra una ventana en la cual es posible comenzar a teclear los commandos ISBL. El prompt ISBL> que se envía sobre la pantalla, indica que el sistema está listo para aceptar cualquier comando.

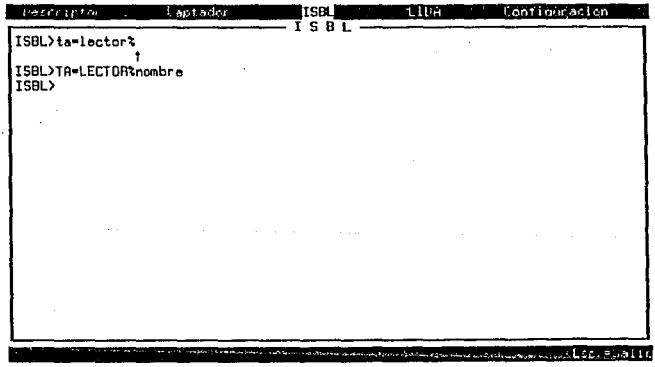


El itérprete adiciona ciertos comandos que fueron diseñados como parte del mismo intérprete, no son propiamente instrucciones ISBL Extendido, mas sin embargo, son consideradas instrucciones propias para un manejo adecuado del sistema.

El editor del intérprete cuenta con las facilidades de las teclas especiales para realizar la captura de los comandos e instrucciones ISBL, no obstante la tecla <ARRIBA>, se ha redefinido con el fin de volver a editar la instrucción anterior. Esto es, en el momento que se oprime <ENTER> al finalizar una línea ISBL, el sistema interpreta la cadena y la ejecuta; cualquier error de sintaxis detectado por el intérprete, es indicado por medio de un mensaje sobre la pantalla, y una flecha apuntando en la posición del error sobre la cadena.



Si en ese momento se requiere corregir la instrucción, sólo se debe teclear <ARRIBA>, y la línea volverá a aparecer para proceder a su corrección.



No necesariamente, esto se puede hacer después de haber cometido un error, también es posible valerse de esta tecla, con fines de reedición de la línea.

A continuación se muestra una lista de instrucciones aceptadas por el intérprete:

OPERACION	SIMBOLO	EJEMPLO	SIGNIFICADO DEL EJEMPLO
Copia	=	EMPLEADO=OBREROS	Copia la relación OBREROS en la relación EMPLEADO
Unión	+	A+B	Realiza la operación unión entre las relaciones A y B
Intersección	.	A.B	Efectúa la intersección de las relaciones A y B
Diferencia	-	A-B	Resta la relación A de la relación B
Junta	*	A*B	Realiza la Junta Natural de las relaciones A y B
Selección	:	EMPLEADO:(SUELDO>30000)	Selecciona las tuplas de la relación EMPLEADO cuyo atributo SUELDO sea mayor a 300000
		EMPLEADO:(EDAD>60),JOVENES	Selecciona las tuplas de la relación EMPLEADO cuyo atributo EDAD sea mayor a 60. El resultado que no cumple esta condición se almacena dentro de la relación JOVENES
Proyección	%	EMPLEADO%NOMBRE,SUELDO	Obtiene la relación resultado de proyectar EMPLEADO, sobre sus atributos NOMBRE y SUELDO
		EMPLEADO%NOMBRE->NOMEMP	Obtiene la relación resultante de proyectar la relación EMPLEADO sobre su atributo NOMBRE, renombrándolo en la relación resultado como NOMEMP

OPERACION	SIMBOLO	EJEMPLO	SIGNIFICADO DEL EJEMPLO
Actualización	[]	EMPLEADO[SUELDO=SUELDO/12]	Obtiene una relación similar a EMPLEADO pero con los valores del atributo SUELDO divididos entre 12
Ordena	#	EMPLEADO#SUELDO,A	Se genera una relación con las mismas tuplas que EMPLEADO, pero ordenadas ascendentemente con respecto al atributo SUELDO
		EMPLEADO#SUELDO,D	La relación resultado, será una relación con las mismas tuplas que empleado, pero ordenadas descendentemente, con respecto al atributo SUELDO
Permuta Llave	~	EMPLEADO~LLAVE,5	Se obtiene una relación parecida a EMPLEADO, pero con los valores del atributo LLAVE permutado en 5 posiciones a la derecha.
Máximo	MAX()	MAX(EMPLEADO,SUELDO)	Regresa el valor numérico máximo del atributo SUELDO en la relación EMPLEADO
Mínimo	MIN()	MIN(EMPLEADO,SUELDO)	Calcula el valor numérico mínimo del atributo SUELDO en la relación EMPLEADO
Promedio	PROM()	PROM(EMPLEADO,EDAD)	calcula el promedio de los valores del atributo EDAD en la relación EMPLEADO. El resultado es un valor numérico.
Suma	SUM()	SUM(EMPLEADO,SUELDO)	Obtiene el valor numérico de la suma de todos los valores del atributo SUELDO en la relación EMPLEADO
Número de registros	REGS()	REGS(EMPLEADO)	Obtiene la cardinalidad de la relación EMPLEADO. Al igual que las funciones anteriores, regresa un valor numérico que debe asignarse a una variable

OPERACION	SIMBOLO	EJEMPLO	SIGNIFICADO DEL EJEMPLO
Máximo agrupado	MAXA( )	MAXA(EMPLEADO, SUELDO, DEPTO)	Agrupa la relación EMPLEADO por elatributo DEPTO, y selecciona cada valor máximo del atributo SUELDO de cada grupo, la relación resultado contendrá sólo esos dos atributos cuyos valores fueron máximos.
Mínimo agrupado	MINA( )	MINA(EMPLEADO, SUELDO, DEPTO)	Agrupa la relación EMPLEADO por elatributo DEPTO, y selecciona cada valor mínimo del atributo SUELDO de cada grupo, la relación resultado contendrá sólo esos dos atributos cuyos valores fueron mínimos.
Promedio agrupado	PROMA( )	PROMA(EMPLEADO, EDAD, ZONA)	Agrupa la relación EMPLEADO por el atributo ZONA, y calcula el promedio de los valores del atributo EDAD de cada grupo. La relación resultado contendrá sólo esos dos atributos, el valor de cada grupo y su promedio
Suma agrupado	SUMA( )	SUMA(EMPLEADO, PROD, TIPO)	Agrupa la relación EMPLEADO por el atributo TIPO, y efectúa la suma de los valores del atributo PROD de cada grupo. La relación resultado contendrá sólo esos dos atributos, el valor de cada grupo y su suma
Número de registros agrupado	REGSA( )	REGAS(EMPLEADO, DEPTO)	Agrupa la relación EMPLEADO por el atributo DEPTO, y calcula el número de tuplas de cada grupo. La relación resultado contendrá el valor de cada grupo y la suma de las tuplas de cada grupo

Recuerde que las operaciones relacionales dan como resultado otra relación (exceptuando las funciones MAX, MIN, PROM, REGS que regresan un valor), de tal suerte, que se puede realizar un nivel de anidamiento de operaciones según se requiera. Por ejemplo:

```
OBR=((EMPLEADOS:(SUEL<400000))+(EMPLEADAS:(SUEL<400000)))%NOMBRE
```

Esta cadena nos podría dar como resultado la relación OBR conteniendo los nombres de los empleados hombres y mujeres que perciben un sueldo menor a $400,000 , suponiendo de antemano que los empleados se encuentran separados en dos relaciones para distinguir los empleados de las empleadas.

Como parte de las características del Intérprete del lenguaje ISBL Extendido, se crearon los siguientes comandos denominados comandos de sistema, para ofrecer un mayor y mejor control en la ejecución de cadenas ISBL Extendido:

Comandos de sistema:

`$LISTA` Genera un reporte hacia algún dispositivo (Consola, Impresora o Archivo ASCII), con un formato previo, por ejemplo :

`$LISTA EMPLEADOS`	Envía relación `EMPLEADOS` hacia la consola (monitor).
`$LISTA EMPLEADOS > CON`	Tiene el mismo efecto que el ejemplo anterior.
`$LISTA EMPLEADOS > LPT1`	Envía un reporte de la relación `EMPLEADOS` hacia la impresora.
`$LISTA EMPLEADOS > REMP`	Envía un reporte de la relación `EMPLEADOS` hacia el archivo `REMP.PRN`.

Cuando se envía una relación hacia la consola, el *Intérprete* se vale de la erramienta del *Captador de datos* para mostrar la relación resultante sobre la pantalla. El procedimiento es el mismo que describimos en el subcapítulo anterior, por lo que el usuario además de poder visualizar el resultado, también puede modificar los datos.

$DESC Abre menú de Mantenimiento de Descriptores

$CLS Limpia la pantalla de edición

$DESPV Despliega las variables de memoria

$BVARS	Borra las variables de memoria que se han generado
$FDECS	Define número de decimales a usar ejem. `$FDECS 2`
$EJEC	Ejecuta un programa .LIP, ejem. `$EJEC NOMINA`
FIN	Sale del ISBL (antes de abandonar la opción el sistema envía un mensaje para confirmar la salida)

Manejo de variables de memoria:

`@X=30.78` Asigna 30.78 a la variable de memoria X, y la crea

`@X='HOLA'` Asigna la cadena 'HOLA' a la variable de memoria X

`?X` Permite editar el contenido de la variable X.

`@A=&B` Asigna el contenido de B en A, y crea la variable A

`@A=B` Asigna el contenido de B en A, y crea la variable A

La asignación de valores numéricos o cadenas de caracteres es inmediata. Las varibles de memoria son creadas o redefinidas al momento de la asignación.

Observaciones:

- El comando ? origina que la variable a editar se muestre de la siguiente forma: suponga que el valor de la variable Y es 30, entonces si se teclea `?Y`, el sistema genera la cadena `@Y=30.00` en el siguiente renglón y coloca el cursor al final de la línea, para que el valor 30.00 sea reeditado.

- Sólo la SELECCION puede utilizar expresiones con los siguientes operadores:

O	"o" lógico
Y	"y" lógico
<	Menor que
>	Mayor que
=	Igual
<=	Menor o Igual
>=	Mayor o Igual

< > Diferente
( ) Paréntesis

- Las expresiones aritméticas dentro de las operaciones de actualización, selección, y asignación de variables de memoria pueden utilizar los siguientes operadores:

\+ Suma
\- Resta
* Multiplicación
/ División
( ) Paréntesis

Nota: No es posible realizar operaciones aritméticas con cadenas.

Un ejemplo serían las dos líneas siguientes:

```
EMPLEADO=EMPLEADO[SUELDO=SUELDO*34.56+PRESTA/6*(INTERES+3)]
@X = 3455.78*6789/88979*(900-45667)
```

- En la operación de SELECCION, las expresiones aritméticas dentro de una expresión booleana, siempre deben encerrarse entre paréntesis, por ejemplo:

```
IMPUESF=EMPLEADO:(SEXO='F' Y (SUELDO+PRESTA*1.5) > 3000000)
```

- No hay límite en el número de paréntesis anidados.

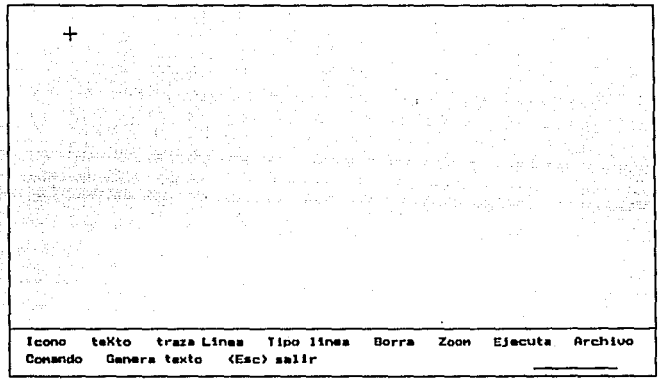
- Una instrucción ISBL no puede tener una longitud mayor a los 72 caracteres (que es la longitud de la ventana del intérprete, dispuesta para la captura de instrucciones), pero sin embargo, es posible expander una instrucción hasta los 256 caracteres haciendo uso de las variables de memoria; por ejemplo:

```
@COMPLETA='34.56+PRESTA/6*(INTERES+3)'
EMPLEADO=EMPLEADO[SUELDO=SUELDO*&COMPLETA]
```

- El tamaño del nombre de una variable, no puede exceder de 70 caracteres.

5.5.4 LIDA

Este módulo permite trabajar con el lenguaje LIDA siguendo su sintaxis y semántica que ya hemos descrito anteriormente. Al accesar esta opción se presenta la siguiente pantalla del editor gráfico:



Su manejo es sumamente simple, la pantalla es dividida en dos ventanas: una ventana superior que se destina para la representación del los flujogramas, y una ventana inferior para mostrar el menú de comandos, los mensajes de error y para realizar la captura de datos.

Al comenzar, en la primera ventana se muestra un cursor en forma de cruz (cursor-cruz), que indica la posición donde tendrá efecto cualquiera de los comandos mostrados en la segunda ventana denominada ventana de comandos, estos últimos se encuentran formando un menú (que explicaremos adelante).

Cuando se edita una carta gráfica, el movimiento del cursor-cruz se puede realizar con las flechas del teclado, similarmente como se hace en la captura de datos:

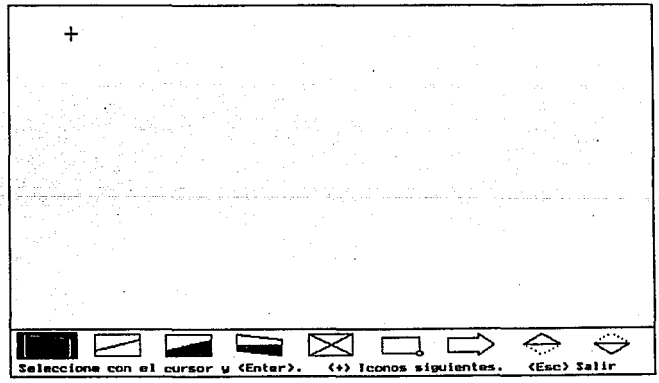
<IZQ>	Mueve el cursor hacia la izquierda
<DER>	Mueve el cursor hacia la derecha
<ABAJO>	Mueve el cursor hacia abajo
<ARRIBA>	Mueve el cursor hacia arriba
<CTRL><IZQ>	Mueve el cursor hacia la pantalla izquierda
<CTRL><DER>	Mueve el cursor hacia la pantalla derecha
<PGDN>	Mueve el cursor hacia la pantalla superior
<PGUP>	Mueve el cursor haciaJa pantalla inferior

La ventana superior muestra una parte del espacio total para la captura del flujograma. Utilizando las teclas <CTRL><IZQ> y <CTRL><DER> es posible mover la ventana hacia la izquierda o derecha respectivamente, o bien <PGDN> o <PGUP> para mover la ventana hacia abajo o hacia arriba. El espacio total de edición puede verse por medio del comando **Zoom** (cuya aplicación describiremos después).

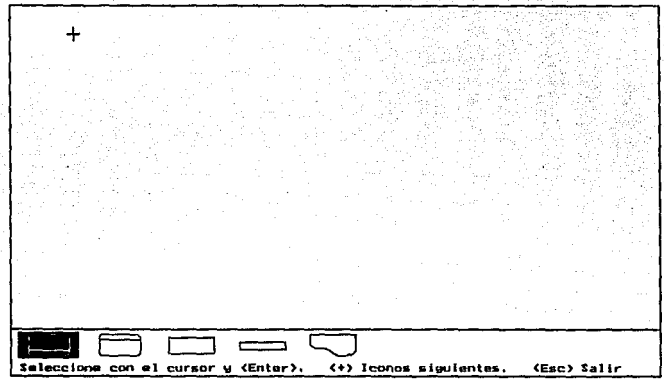
Por medio de los comandos de edición, es posible comenzar a armar el flujograma de forma conveniente. En seguida explicaremos la función de cada uno de éstos comandos.

Para tener acceso a cualquiera de los comandos, basta con teclear la letra correspondiente del comando que se encuentra en mayúscula, por ejemplo el comando Icono se accesa tecleando <I>. Para abandonar la operación como lo hemos venido explicando a lo largo del sistema, es necesario teclear <ESC>, incluso para abandonar la opción de LIDA.

Icono. Esta opción permite accesar una lista de iconos de LIDA en forma de menú; con las flechas <IZQ> y <DER> es posible colocar el cursor sobre el icono deseado y teclear <ENTER> para confirmar la selección. El icono se dibuja sobre la ventana superior, exáctamente en el lugar que marca el cursor-cruz.



Para tener acceso al resto de los iconos, es necesario pulsar la tecla <+>.

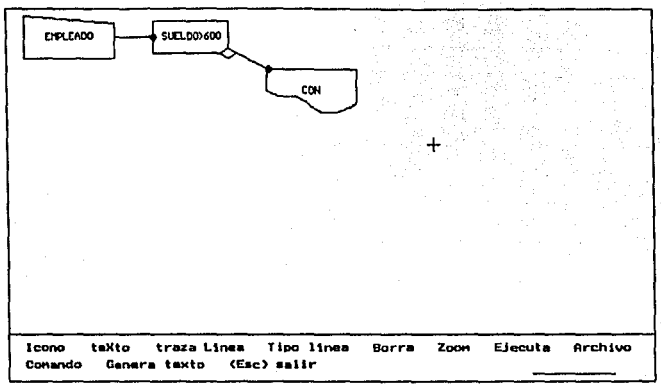


Una vez dibujado el icono, el sistema regresa nuevamente al menú de comandos para esperar la nueva orden.

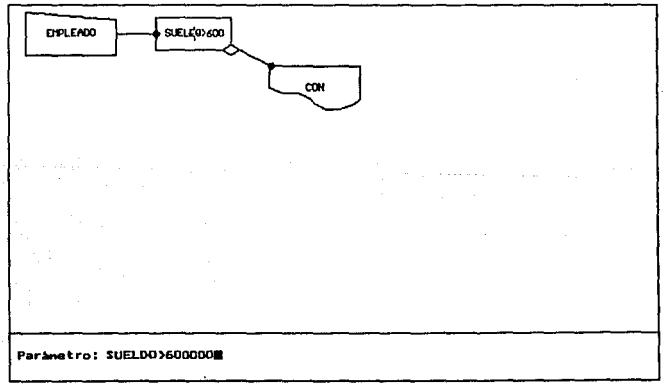
Como mencionamos al principio, en SABRE se ha diseñado un icono que aparece dentro de la lista de iconos de LIDA denominado ICONO COMANDO, este icono no es parte del lenguaje, sin embargo es un procedimiento que representa una operación en ISBL Extendido.

teXto. Por medio de este comando, se pueden asignar o modificar los parámetros y nombres respectivos de cada icono. Al seleccionar el comando, el sistema muestra dos opciones correspondientes para asignar el nombre o el parámetro, para ello es necesario oprimir la tecla <N> o <P> según sea el caso e iniciar la captura de inmediato.

Esta opción es muy importante, ya que junto con la opción anterior y la siguiente conforman los procedimientos mínimos para armar un flujograma. Antes de proseguir el lector debe tomar en cuenta que los parámetros de los iconos corresponden a lo que definimos en los subcapítulos 2.3 Semántica y 2.4 Sintaxis, de tal forma que por ejemplo si tuviéramos una relación llamada EMPLEADO, y quisiéramos seleccionar al conjunto de empleados cuyo sueldo fuera mayor a $600,000, entonces tendríamos tres iconos en la pantalla el icono ARCHIVO con nombre EMPLEADO, el icono SELECCIONA con parámetro SUELDO > 600000 y finalmente un icono IMPRIME con parámetro CON, y la pantalla se vería similar a la siguiente:



El parámetro del icono SELECCIONA no puede verse completo, ya que por demás es obvio que si la cadena del parámetro es grande no podrá visualizarse completa, sin embargo esto no es un obstáculo como para no distinguir el parámetro dentro del icono. Además si se requiere ver la cadena completa, entonces es posible acudir a esta opción, y teclear <N> o <P>, lo cual origina que el sistema coloque la cadena completa para recapturarla o simplemente analizarla.



traza Línea. Para poder dibujar las las líneas de flujo de datos entre dos iconos, es necesario colocar el cursor-cruz sobre el icono de donde partirá la línea, después oprimir la tecla <L> para accesar el comando, y por último, colocar el cursor-cruz sobre el icono destino y confirmar con <ENTER>.

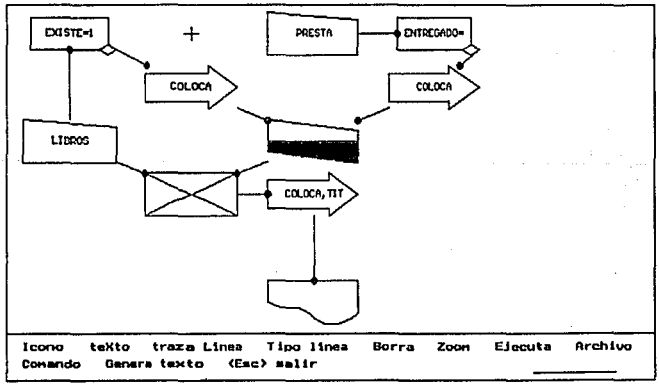
Tipo línea. El comando descrito anteriormente, tiene una relación directa con éste. Cada vez que se dibuja una línea de flujo de datos, ésta se traza con el tipo actual de línea. Existen sólo dos tipos de línea: continua y punteada; para poder cambiar el tipo actual sólo se oprime la tecla <T>.

El único icono sobre el cual tiene efecto el tipo de línea es el icono SELECCIONA, ya que como recordaremos, su salida puede dividirse en parte verdadera y parte falsa, dependiendo si se cumple o no su parámetro.

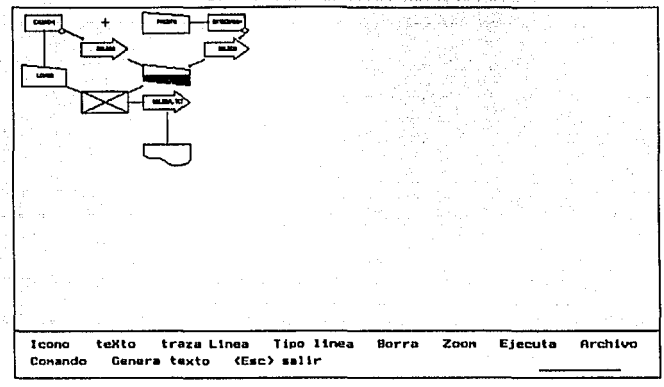
Borra. Para eliminar algún elemento del flujograma, e incluso el flujograma mismo, debe accesarse este comando. Borra incluye tres opciones borra Icono, borra Línea y borra Flujograma. La primera opción elimina tanto líneas como iconos.

Zoom. El comando Zoom permite ver una parte del puerto o el puerto completo dentro de la ventana superior, este comando puede visualizar el flujograma en tres tamaños, para ello es necesario teclear <Z>, sucesivamente:

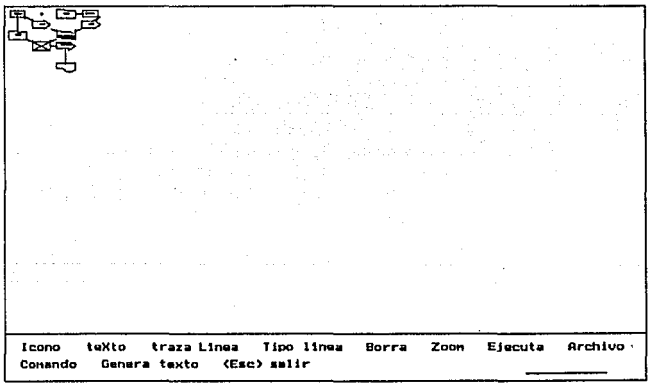
Primera vista:



Segunda vista:



Tercera vista:



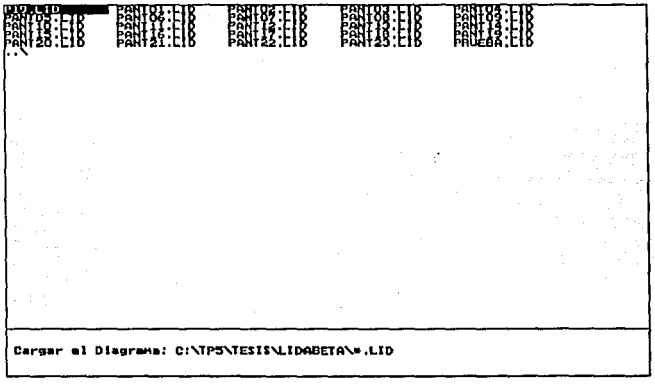
Ejecuta. Es posible ejecutar el programa definido dentro del flujograma. En el momento en que el sistema detecta un error de sintaxis o semántica el proceso termina, al momento que se envía un mensaje de error y se coloca el cursor-cruz en el icono donde se detectó el error.

Archivo. Este comando le permite desplegar el contenido de un directorio, cargar un flujograma o bien grabarlo en disco, para lo cual cuenta con las siguientes opciones respectivamente: Directorio, Carga archivo y Salva archivo. Para accesar cualquier opción se requiere pulsar la letra <D>, <C> o <S> según

sea el caso. Cada opción permite capturar la ruta completa y la máscara de los archivos de forma similar como se realiza en el sistema operativo, es decir se pueden utilizar los caracteres * y ? para simplificar la selección. Supóngase que se requiere cargar un flujograma, del cual no se recuerda el nombre, pero si su directorio y sea éste C:\TP5\TESIS\LIDABETA entonces proceda seleccionando la opción Carga archivo, y capture la ruta:

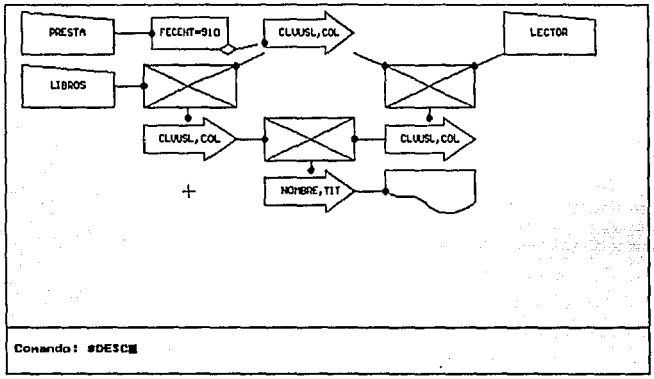
```
C:\TP5\TESIS\LIDABETA\*.LID
```

Al instante se desplegarán todos los archivos con terminación .LID (que corresponden a los flujogramas). Por medio de las flechas es posible recorrer la lista de los archivos y seleccciонar el flujograma correspondiente.



Al salvar flujogramas, cualquier condición de sobreescritura de archivos, puede originar que el sistema envíe un mensaje para confirmar su sobreescritura, y depende del usuario que el proceso se realice o no.

Comando. Todos los comandos e instrucciones que acepta el intérprete del lenguaje de cadena ISBL Extendido, también pueden ejecutarse desde LIDA. Para ello accese esta opción y proceda a capturar la cadena ISBL Extendio, de la misma forma como se realiza en el intérprete.



Genera código. Existe una manera de generar un archivo de procedimientos con cadenas ISBL Extendido a partir de un flujograma. Este archivo se puede tomar como un procedimiento ejecutable por medio del comando $EJEC o vía ICONO COMANDO. Al accesar esta opción, es necesario capturar el nombre del archivo para alojar el procedimiento.

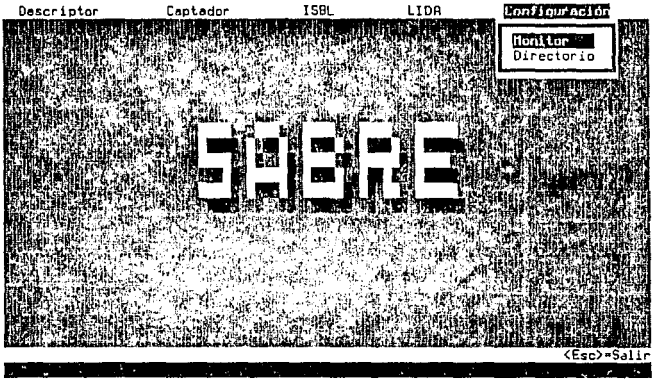
Los archivos generados por esta opción llevan al extensión .LIP, y son de tipo ASCII estándar, lo que implica que pueden ser modificados y creados por medio de un procesador de palabras u otra aplicación que pueda manejar archivos de este tipo. El contenido del procedimiento generado por el flujograma ejemplo de la opción teXto, se muestra como sigue:

```
ZZZ2!=EMPLEADO:(SUELDO>600000),ZZZ1!
$LISTA ZZZ2!>CON
```

Esto representa una ventaja para poder programar en ISBL Extendido escribiendo las operaciones dentro de un archivo de procedimientos, y posteriormente ejecutarlo por medio de SABRE, tal como si fuera un procedimiento fuera de línea.

5.5.5 Configuración

Al accesar la última opción del menú principal, de inmediato será mostrado un submenú con dos opciones:



La opción Monitor permite configurar el tipo de monitor con que cuenta la computadora seleccionando una de las dos opciones que aparecen Monocromático o Color.

La opción Directorio sirve para que el archivo que contiene los descriptores denominado DESCRIP.DES pueda ser direccionado hacia otro subdirectorio, esto con el fin de que el sistema cuente con más de un archivo de descriptores. Es necesario que tanto las bases de datos como el archivo de descriptores se encuentren en un sólo subdirectorio, ya que el sistema utiliza la misma ruta del directorio del archivo de descriptores para hacer referencia a las bases de datos; si fuera necesario o si así se requiriera se puede crear otro subdirectorio.

Si al entrar a esta opción y capturar la ruta completa del nuevo subdirectorio éste no existe, entonces el sistema envía un mensaje para confirmar si el subdirectorio se crea, de ser así, el sistema creará el subdirectorio y el nuevo archivo de descriptores, finalmente el sistema estará listo para trabajar con el nuevo conjunto de bases de datos generadas en ese subdirectorio.

CAPITULO 6: APLICACIONES

El desarrollo de software nos deja pensar en las múltiples aplicaciones que una herramienta nos puede proporcionar. LIDA es parte de estas herramientas y como tal, contribuye con toda una gama de posibilidades de aplicación, y no sólo en el ámbito práctico, ya que también contribuye con la teoría computacional que abre caminos aún más grandes para el desarrollo de software.

6.1 SISTEMAS DE INFORMACION

El procesesamiento de grandes volúmenes de información en un menor tiempo, fue uno de los motivos principales del avance de la computación, los sistemas de información como lo hemos analizado desde el principio de este trabajo, son la fuerza motriz que incrementa el interés por crear nuevas técnicas de programación, así como nuevos lenguajes de nivel superior, las aplicaciones que puede tener LIDA, repercuten directamente en el diseño de sistemas de información que marcan una filosofía de operación diferente a la convencional, y que por tanto es necesario analizar el campo en el que se puede ubicar LIDA.

6.1.1 Herramientas para LIDA

Como ya lo hemos expuesto en repetidas ocasiones, el objetivo de nuestro proyecto se enfoca plenamente en el diseño de un lenguaje iconográfico que permite procesar bases de datos del tipo relacional; no obstante, también desarrollamos herramientas de software que permitieron implementar el lenguaje dentro de un sistema integrado para generar aplicaciones; esto nos permitió considerar en la posibilidad de continuar el desarrollo de software para agilizar y mejorar el desempeño del lenguaje dentro del campo de los sistemas de información, ya sea en la construcción de herramientas basadas en LIDA, o posiblemente en herramientas gráficas que permitan generar aplicaciones más complejas como:

- *Captura de descriptores, con más campos de descripción de archivos*
- *Procedimientos de mantenimiento de archivos*
- *Un captador de datos*
- *Otras operaciones relacionales*
- *Un asistente de operaciones*
- *Un ambiente completamente gráfico*
- *Herramientas que incluyan color y sonido*
- *Un intérprete mejorado*
- *Traductores de lenguaje hacia otros como SQL o SQUARE*
- *Integridad de información*

entre otras, y en las que LIDA debe ser tratado como la parte principal.

6.1.2 LIDA dentro de un sistema de información

Es preciso ubicar el papel que juegan los sistemas de información, en el ámbito de cualquier empresa. Como sabemos la administración se vale de la informática para apoyar sus funciones, y la informática se vale de los sistemas de información para administrar el procesamiento de datos.

La mayor parte de la carga del procesamiento y flujo de datos, lo llevan los sistemas de información, no obstante, muchos procedimientos manuales los complementan.

El producto final que desarrolla un centro de cómputo, son los sistemas de información; mientras mejor se diseñe un sistema, mejor será su funcionalidad y calidad. Además los procedimientos manuales deben también planearse adecuadamente; ya que son complemento de un sistema de información, por lo que entre mejor sea su planeación, dichos procedimientos serán más óptimos.

Las ventajas que ofrece un sistema de información de características visuales, cumple con los lineamientos de un CASE de calidad, de manera que los sistemas de información desarrollados de ésta forma, mantendrán a su vez el buen desempeño de sus objetivos.

La construcción de sistemas de información basados en LIDA, deben ser sistemas desarrollados por medio de un CASE de nivel intermedio o inferior que soporte el lenguaje. Por lo que los sistemas basados en LIDA pueden llegar a ser lo suficientemente versátiles y poderosos, como para apoyar las actividades administrativas de cualquier empresa.

Con el auge que comienza de la *programación orientada a objetos*, *sistemas expertos*, y *herramientas CASE*, es posible desarrollar software que permita la creación de nuevos sistemas basados en LIDA. Este software puede constituir sistemas gráficos mucho más complejos, simplificando cada vez mejor el grado de las instrucciónes y tal vez macroinstrucciones completas.

6.1.3 Sistemas distribuidos y LIDA

El avance tecnológico ha logrado que el *proceso distribuido* tenga cada vez más alcance, así los sistemas de redes de computadoras se han venido incrementando enormemente. Mientras se desarrolla hardware para que las máquinas procesen información a distancia, también se desarrolla el software para que explote esta capacidad. LIDA no ha sido diseñado para ser un lenguaje que trabaje dentro de una red de computadoras. Pero es un lenguaje que puede ser implementado en un sistema de que opere dentro de una Red de computadoras.

Cuando se genera una aplicación en LIDA, primero se crea el flujograma y éste se transforma por medio de un intérprete hacia cadenas ISBL Extendido, para posteriormente ser ejecutadas; el orden en el que se ejecuta un flujograma, depende del peso asignado a cada icono, y como sabemos, los iconos representan bases de datos u operaciones relacionales, si a esto le añadimos un peso adicional que es característico de los sistemas distribuidos, entonces sería posible utilizar tanto operaciónes como bases datos bloqueadas o desbloqueadas, dependiendo de la asignación que el elemento servidor de la Red le diera a cada usuario.

La implementación de LIDA en un sistema como el mencionado, podría agilizar aún más los procedimientos de gestión de una empresa, inclusive LIDA podría producir procesos en paralelo, utilizando los recursos de hardware de la Red, aumentando considerablemente el desempeño de la empresa en cuestión.

LIDA como aplicación en un sistema distribuído, sólo requerirá entonces del software que sirva de plataforma sobre la Red. Gran parte del trabajo que desempeña una Red lo realiza el sistema operativo de la misma, el software que se desarrolla para trabajar en Red complementa el trabajo a través de herramientas que manipulan archivos en proceso distribuído. Con lo anterior pensamos que llevar LIDA a un sistema en Red, es una aplicación del mismo lenguaje que puede ser considerada.

6.1.4 Un sistema manejador de bases de datos con LIDA

El sistema integrado LIDA que hemos dado a conocer, dista de ser un verdadero sistema de bases de datos que cumpla con todos sus niveles de operación. El lenguaje LIDA puede ser parte de un *sistema manejador de bases de datos relacionales* completo, incluyendo el lenguaje para que sirva como un DML, ya que está diseñado para serlo. Como aplicación nos referimos a que el lenguaje LIDA puede llegar a ser parte de un sistema de bases de datos, así que el resto del trabajo queda a los ingenieros de software para desarrollarlo.

Debe definirse el lugar que ocupa LIDA en el área del procesamiento de datos. Primero, desde el punto de vista de CASE, LIDA representa un lenguaje para el desarrollo de aplicaciones basado en la teoría de las relaciones, y desde el punto de vista de los sistemas de bases de datos es un DML. Así LIDA sólo es el lenguaje, los sistemas que se desarrollen en base a éste son considerados ya, como aplicaciones del mismo lenguaje.

Como nos hemos referido con anterioridad, los sistemas CASE vienen a proporcionar un apoyo tanto para los administradores de una empresa, como para los diseñadores de software. Toda la teoría del CASE gira entorno de un sólo tema: *El procesamiento de datos*. Realmente todos los sistemas CASE son una herramienta para el diseño y construcción de sistemas de información. Es decir son un conjunto de sistemas para el desarrollo de sistemas.

LIDA es una herramienta más de CASE, su utilización se orienta principalmente al diseño de sistemas de información gráficos. El sistema SABRE es un ejemplo de la aplicación del propio lenguaje, es decir, LIDA

como tal es la proposición de un lenguaje gráfico lo suficientemente formal, con una sintaxis y semántica bien definidas, y su utilización puede llegar a ser parte de cualquier sistema de información diseñado para explotar el poderío de bases de datos relacionales.

6.2 PERSPECTIVAS DE PROGRAMACION

LIDA contribuye a los nuevos estilos de programación, posiblemente en unos años la *programación gráfica* de computadoras sea una actividad común, al menos así se observa el panorama, cada vez se incrementan los sistemas de CASE y cada vez se muestra una tendencia a desarrollar sistemas sobre ambientes gráficos, de tal suerte que el concepto CASE es considerado como software para hacer software, por lo tanto todo indica que en poco tiempo habrá software gráfico para desarrollar software gráfico. LIDA es un prototipo de lo que en un futuro no muy lejano, quizá se convierta en una forma de programar computadoras.

6.2.1 Programación automática

Dado que LIDA es un lenguaje gráfico su programación es sencilla, y al parecer la parte un tanto compleja es: primero seleccionar el lenguaje intérprete, y después desarrollar el software que pueda interpretar un flujograma para transformarlo en términos del intérprete. Ya que sería un tanto más complejo desarrollar un compilador para LIDA. Exáctamente en este punto entra el concepto de *Programación Automática* (PA).

La PA es un concepto de la ingeniería de software que se utiliza como una herramienta para generar código apartir de parámetros, así una carta gráfica es decifrada dentro de un procedimiento que analiza su sintaxis y semántica, para después generar el código correspondiente a un lenguaje que cuenta ya con un compilador, o bien, que es intérprete.

De hecho el intérprete de LIDA genera el código correspondiente por medio de la PA al lenguaje ISBL Extendido, después se hace pasar por otro proceso de interpretación pero ahora de ISBL Extendido a simples llamadas a una librería de operaciones relacionales.

La ventaja de contar con un intérprete que genera código, es que ese código es portable. De esta forma también LIDA puede ganar algo de independencia en cuanto a la generación de aplicaciones, debido a que basta contar con un compilador de ISBL Extendido que utilice programas fuente dentro de archivos en código ASCII y entonces lo único que se requerirá es diseñar el flujograma en LIDA generar el código sobre un archivo fuente y procesarla por medio del compilador de código ISBL Extendido.

La cuestión no queda ahí, ya que este concepto da la opción a que se pueda desarrollar un procesador que interprete la carta gráfica en LIDA y que genere código en otro lenguaje, por ejemplo COBOL. Cabe señalar

que además de que se genere el código, debe contarse con la librería de operaciones relacionales en COBOL, ya que el código interpretado quedará en términos de llamadas a procedimientos o subrutinas en el mismo lenguaje.

6.2.2 Encapsulamiento

Hemos podido observar que un lenguaje iconográfico, debe tener ciertas características para serlo, entre una de ellas es que cada icono representa una operación o una función bien definida. El mismo proceso evolutivo de los lenguajes de programación ha incrementado el nivel de encapsulamiento de los procedimientos, es decir, cada nueva generación de lenguajes ha intentado definir nuevos comandos o sentencias de nivel superior, de tal suerte que cada vez se escriben menos líneas de programación, y se realizan procesos más complejos.

LIDA contribuye al desarrollo de lenguajes de un grado de encapsulamiento superior. En el diseño de LIDA se consideró este punto, primero LIDA tenía que generar código por medio de un proceso de PA, después una instrucción en LIDA debería ejecutar ese código, de tal suerte que el código se pudiera contener en algún lugar, para después interpretarlo, ahora bien, una instrucción en LIDA debe corresponder a un icono; así nace el icono COMANDO, y el comando como parámetro $EJEC, que ejecuta un archivo tipo batch con instrucciones ISBL Extendido, cuyo funcionamiento es el siguiente:

- Se arma el primer flujograma

- Se genera el código correspondiente, y se almacena en un archivo, sea este PROG

- Se arma un segundo flujograma y se añade el icono COMANDO con parámetro $EJEC PROG

El icono COMANDO en el segundo flujograma representa el primer flujograma, por lo que este comando tiene un grado de encapsulamiento superior. Suponga que este segundo flujograma se vuele a encapsular, y así sucesivamente. Esto indica que podemos aceptar el lenguaje LIDA como un generador de iconos de orden superior.

Supóngase que se desaarrolla un editor de flujogramas más poderoso, en el cual inclusive el usuario puede diseñar sus propios iconos. Si tomamos en cuenta que el resultado final de un flujograma siempre es una relación, entonces sin abandonar la semántica de LIDA, un flujograma encapsulado convertido en un nuevo icono para el lenguaje, sería como un icono que representa una relación, y con el cual se pueden realizar otras operaciones.

La forma de manejar el encapsulamiento en LIDA, puede servir para el desarrollo de procedimientos que sean considerados como macroinstruciones de nivel superior.

6.2.3 Generadores de aplicaciones con LIDA

En alguna forma, los sistemas generadores de aplicaciones, utilizan un lenguaje de cuarta generación para el desarrollo de sus programas, en algunas ocasiones se utilizan intérpretes de lenguaje natural como medio para generar aplicaciones.

Dadas las características de LIDA, puede ser parte de un sistema generador de aplicaciones, de hecho el sistema SABRE que hemos expuesto como parte de nuestro proyecto, es una muestra de la aplicación de LIDA dentro de un sistema generador de aplicaciones.

Como lo hemos expuesto anteriormente, LIDA puede ser el DML de un sistema de bases de datos, pero en un sentido estricto, LIDA es un lenguaje generador de aplicaciones, por lo que para un sistema generador de aplicaciones, LIDA es un leguaje idóneo.

6.3 El SISTEMA SABRE Y SU SOFTWARE

A lo largo de nuestro trabajo, hemos expuesto el lenguaje LIDA como la columna vertebral del mismo. No obstante, al implementar LIDA en un sistema, hechamos mano de herramientas de software que sirvieron para implementar el lenguaje, esto sirvió también para la aplicación de conceptos como: *descriptor de archivos, lenguaje intérprete, flujograma, editor gráfico*, etc. y en los cuales se invirtió una gran parte del tiempo para completar nuestro trabajo. Por ello decidimos dedicar un espacio para mencionar las posibles aplicaciones adicionales que pueden tener las herramientas de LIDA.

6.3.1 ISBL Extendido

Aunque el objetivo principal de nuestro proyecto, no es el desarrollo de un lenguaje de cadena, cabe mencionar que paralelamente a LIDA, y al sistema integrado, hemos contribuído también con la implementación de un lenguaje de cadena como lo es ISBL (ya que la compañía IBM desarrolló completamente este lenguaje), no obstante, ISBL como lenguaje cuenta con ciertos elementos que nos permiten opinar sobre los lenguajes de cadena como aplicación.

Como hemos explicado con anterioridad, el lenguaje de cadena ISBL es la interface entre un flujograma y los procedimientos relacionales a los que invoca. La semántica y sintaxis de un ISBL modificado (ISBL Extendido), da pauta para el desarrollo de una interface con el sistema LIDA, ya que por medio de este lenguaje, LIDA interpreta los procedimientos adecuados para el desarrollo de una aplicación, así que es

posible diseñar algunos otros sistemas CASE, para el desarrollo automático de procedimientos, y que sean escritos en lenguaje de cadena e interpretados de la misma forma, como lo hace el sistema LIDA.

El desarrollo de sistemas reconocedores de lenguaje natural, pueden ser un ejemplo para la utilización del ISBL como un lenguaje intermedio. El propio ISBL es suceptible de extenderse aún más, ya que es sólo una parte de lo que representa un sistema lo suficientemente formal; y no únicamente puede tomarse como parte de un sistema, sino pude ser prototipo para el desarrollo de nuevos sistemas de información como el PRTV de IBM. El lenguaje ISBL, no debe entenderse como un lenguaje de cadena para LIDA, sino una herramienta, que puede servir como lenguaje intermedio de otros sistemas de información o lenguajes, que se basen en el modelo relacional.

Propiamente como hemos descrito a ISBL, puede ser utilizado como un sublenguaje de datos (DSL), para lenguajes anfitriones como COBOL, C y Pascal; no obstante el lenguaje manejador de datos (DML) y el lenguaje de definición de datos (DDL), deberá incluirse posiblemente dentro de una librería como se hizo en LIDA.

6.3.2 Descriptor de Archivos

El *descriptor de archivos*, es una forma simplificada de lo que se conoce como *diccionario de datos*, en el que además de las descripciónes físicas de los archivos, se encuentran las descripciónes lógicas y operativas. Aún así, el descriptor de archivos sirve como esquema sobre el cual se pueden desarrollar sistemas de información completos. El descriptor por sí mismo es una herramienta diseñada para controlar el acceso de la información.

Un descriptor de archivos, puede utilizarse para generar la documentación física de un sistema de información. Además debido que en el descriptor se encuentra descrito el acceso a la información, entonces puede servir a cualquier herramienta de software que requiera cierta independencia de datos, por ejemplo un generador de pantallas, un reporteador, un formateador o un captador de datos, como se hizo en el sistema LIDA.

Para poder utilizar el descriptor de archivos, recuerde que debe también desarrollar la función VAL, que permita el acceso a los datos, la función VAL, es en sí una aplicación para el descriptor.

6.3.3 Captadores

Dentro del sistema LIDA se tienen dos captadores, un captador de descriptores, y un captador de datos. La idea de construír un captador, se obtiene de las funciones comunes de todo sistema de información, en cualquier sistema de este tipo, se efectúan procedimientos de alta, baja y modificación de datos, pero ello siempre obedece a la estructura misma de los datos, es decir el proceso de captura de datos, depende

directamente de las características del mismo dato, es decir un campo numérico sólo debe aceptar valores numéricos, y no otros, además si se trata de números reales con tres cifras decimales y dos enteros, entonces el procedimiento de captura debe validar que el dato capturado cumpla con estas características.
La captura de datos, siempre dependerá de la estructura de los mismos, así si se crea un procedimiento que permita la captura de valores, y que tome como parámetro la estructura de los datos, entonces estaremos armando un captador que es independiente de los mismos, es decir, no importa qué características tenga el campo de captura, siempre el procedimiento general será el mismo. Ahora bien, si a este procedimiento se le agregan diferentes funciones para la captura, como son movimiento del cursor, inserción, borrado y sobreescritura, entonces sólo quedaría considerar la correspondencia entre tipo de dato y procedimiento de captura, por lo que es necesario contar con una herramienta que permita realizar un mantenimiento a la estructura de los datos sin afectar el procedimiento de captura, esto se logra utilizando el descriptor de archivos.

El captador toma las características de un dato por medio del descriptor, esto corresponde a un proceso con cierta independencia de datos, ahora bien, si además el captador genera una pantalla de captura, entonces sin depender del tipo de datos siempre se podrá armar una pantalla de captura.

Un captador como el que desarrollamos para LIDA, actúa como un intérprete, armando en el vuelo la pantalla virtual de captura, algunos captadores además generán el código en algún lenguaje de alto nivel que corresponde a la pantalla de captura, y de esta forma la pantalla queda lista para compilarse y convertirse en una aplicación.

6.4 LIDA Y LA BIBLIOTECA (Ejemplo)

Dentro de las múltiples actividades que se realizan dentro de una biblioteca, quizás la de mayor reelevancia sea el procedimiento de préstamo y atención a usuarios. Los métodos comunes que en la mayoría de las veces se efectúan manualmente, repercuten en la pérdida de tiempo, tanto para los trabajadores que acuden a los anaqueles en la búsqueda de los libros, como de los usuarios en los ficheros. Es un hecho, que por mucho tiempo la gran mayoría de las bibliotecas en nuestro país trabajan con estos métodos manuales, apoyados por las fichas bibliográficas en grandes tarjeteros o microfilmadas; ya sea en su clasificación temática o por autor, y las técnicas de colocación convencionales para facilitar el acceso a las obras. En ocasiones las bibliotecas han implementado otros procedimientos de prestamo, como lo es el autoservicio, en donde el usuario después de una búsqueda en los ficheros, y haber encontrado el libro correcto y su colocación, acude a los anaqueles para su acceso.

Aunado a todos los procedimientos de servicio manuales, también encontramos los problemas que siempre tiene la biblioteca, como la pérdida o extravío de volúmenes, robo, mutilación y maltrato de obras; estos problemas también repercuten directamente en el funcionamiento de la biblioteca. Por otra parte, sabemos que un sistema de información no resolvería toda esta serie de cuestiones pero sí ayudaría a mantener un mejor control.

Parece lógico suponer que en nuestros días, con ayuda de la computadora podríamos agilizar los procedimientos de servicio de la biblioteca, automatizando el sistema que se lleva manualmente. Por experiencia propia, reconocemos que existen en nuestro país centros de información que utilizan grandes bases de datos para optimizar sus actividades; y que se valen de herramientas y sistemas de software comerciales, y otros desarrollados por las propias instituciones. LIDA realmente es una opción más, para que se desarrolle un sistema de información que permita realizar consultas sobre el acervo bibliográfico, y de esta manera poder mantener un mejor servicio y control de los usuarios, así como del material existente.

En este apartado trataremos de diseñar una base de datos sencilla basada en el procedimiento ficticio de préstamo que generalmente se efectúa en una biblioteca, y supondremos el funcionamiento convencional, para lo cual definimos las siguientes condiciones supuestas:

El usuario deberá presentar su credencial de préstamo vigente.

En caso de que el préstamo sea externo (préstamo a domicilio) el usuario deberá dejar su credencial a cambio de los libros.

Al efectuar el préstamo el usario entregará una ficha de registro con los datos del libros prestado, y el número de su credencial.

Cuando un libro se presta de forma externa, se puede entregar después de una semana, cuando es préstamo interno se entrega ese mismo día.

Al entregar los libros el usuario recibirá su credencial.

Un usuario amerita sanción si no entrega los libros a tiempo.

El entregar un libro en malas condiciones, se hace acreedor a una sanción.

Cuando el número de sanciones sea mayor a tres, el permiso de préstamo le es negado.

Suponemos también que la biblioteca cuenta con diferentes anaqueles los cuales se han distribuído las obras siguiendo el orden de una clave de colocación, así que para recuperar un libro o bien depositarlo, siempre se debe de realizar a través de su clave de colocación.

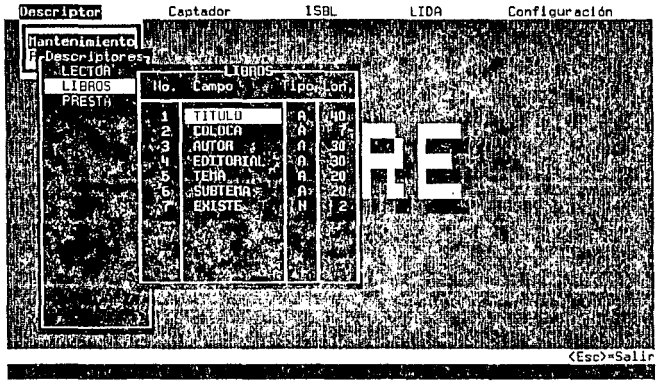
Los usuarios cuentan con credenciales, vigentes según el sello mostrado, la vigencia de una credencial corresponde al año corriente.

Cuando un usuario comete una sanción éste queda registrado en la base de datos, y se le marca la credencial, para que sea tomado en cuenta en la próxima falta.

Para poder operar este sistema de biblioteca simulada, se tienen los siguientes descriptores de archivos, su diseño dista mucho de comprender una base de datos real de esta magnitud, pero cabe mencionar que en este apartado tratamos en todo momento de ejemplificar la aplicación, y por lo tanto los datos como sus descriptores no son extensos:

En el descriptor de la tabla LIBROS, se almacena la información relacionada con los volúmenes de la biblioteca, tal como si fuera una ficha bibliográfica; como referencia se utiliza el campo COLOCA, en el que se almacena la clave de colocación o ubicación física del libro en los estantes. También cuenta con una clasificación por tema y subtema, para ello se utilizan los campos TEMA y SUBTEMA respectivamente. Cada libro tiene una condición escencial de préstamo, "si existe y si se tiene en existencia", es decir si se cuenta con él y si no están prestados todos los volúmenes.

La siguiente pantalla muestra el descriptor LIBROS, en el sistema SABRE:



Emitiendo un reporte de este descriptor se tiene:

```
Fecha: 24/5/1991    << DESCRIPTORES DE ARCHIVOS >>      Hoja  1
                <<< REPORTE DEL ARCHIVO DESCRIP.DES >>>
 ---------------------------------------------------------------------
 Archivo : LIBROS
  No. de Registros :      21
  No. de Atributos :       7
 =====================================================================
  Atributo  Nombre    Apuntador   Apuntador   Tamaño    Tipo
                         Inicial      Final
 =====================================================================

       1       TITULO        1          40         40     alfabético
       2       COLOCA       41          47          7     alfabético
       3       AUTOR        48          77         30     alfabético
       4       EDITORIAL    78         107         30     alfabético
       5       TEMA        108         127         20     alfabético
       6       SUBTEMA     128         147         20     alfabético
       7       EXISTE      148         149          2      numérico
```

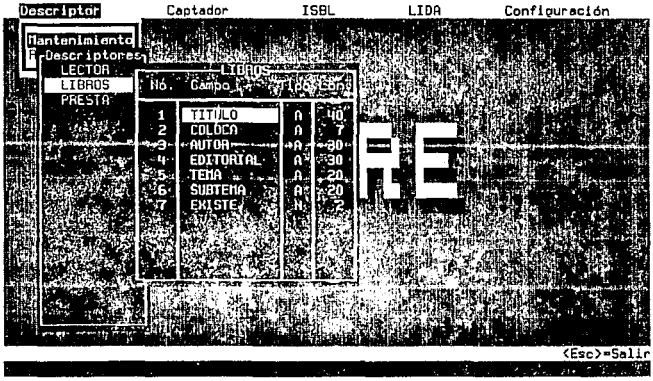
donde :

CAMPO	DESCRIPCION
TITULO	Título del libro
COLOCA	Ubicación física
AUTOR	Autor principal de la obra
EDITORIAL	Editorial del libro
TEMA	Tema central
SUBTEMA	Subtema
EXISTE	No. de volúmenes en existencia

El siguiente descriptor muestra la relación LECTOR, su contenido atiende a la identificación del lector para que se le pueda dar servicio como usuario.

En este caso cuenta con el campo CLVUSL que puede tomarse como llave primaria, también se dedica un campo para llevar el registro de las sanciones cometidas, la restricción para el préstamo a un usuario es que el campo SANCION, sea menor o igual a tres. Si el usuario no tiene derecho al servicio de préstamo, entonces sobre el campo DEPRES, aparecerá un cero, de lo contrario aparecerá un uno. Además cada período anual se renovará la vigencia de las credenciales, así que el campo VIGENCIA, contendrá un uno, si la credencial del usuario está actualizada y de lo contrario contendrá un cero.

A continuación se muestra el descriptor en el sistema SABRE:



Por medio de SABRE se obtuvo el reporte del descriptor:

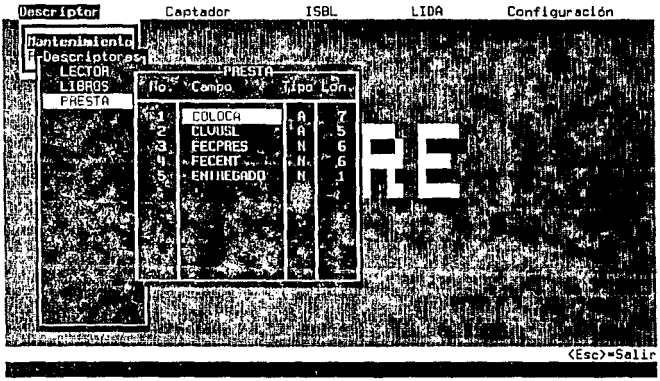
```
Fecha: 24/5/1991   << DESCRIPTORES DE ARCHIVOS >>      Hoja  1
              <<< REPORTE DEL ARCHIVO DESCRIP.DES >>>
 ---------------------------------------------------------------
 Archivo : LECTOR
   No. de Registros :      20
   No. de Atributos :       8
 ===============================================================
  Atributo  Nombre    Apuntador   Apuntador   Tamaño     Tipo
                         Inicial       Final
 ===============================================================

      1       CLVUSL        1           5         5     alfabético
      2       NOMBRE        6          35        30     alfabético
      3       DIREC        36          75        40     alfabético
      4       TELEFONO     76          82         7     alfabético
      6       SANCION      83          83         1      numérico
      7       DEPRES       84          84         1      numérico
      8       VIGENCIA     85          85         1      numérico
```

donde :

CAMPO	DESCRIPCION
CLVUSL	Clave del usuario
NOMBRE	Nombre
DIREC	Dirección
TELEFONO	Teléfono
TIPPRES	Tipo de préstamo (Int./Ext.)
SANCION	Número de Sanciones
DERPRES	Derecho de préstamo
VIGENCIA	Vigencia del año corriente

Para llevar un registro del servicio de préstamo, se tiene el siguiente descriptor:



el descriptor completo se muestra en el siguiente reporte:

```
Fecha: 24/5/1991   << DESCRIPTORES DE ARCHIVOS >>      Hoja  1
                <<< REPORTE DEL ARCHIVO DESCRIP.DES >>>
 ------------------------------------------------------------------
 Archivo : PRESTA
  No. de Registros :       20
  No. de Atributos :       5
 ==================================================================
  Atributo  Nombre    Apuntador   Apuntador   Tamaño     Tipo
                         Inicial       Final
 ==================================================================

    1       COLOCA         1            7          7    alfabético
    2       CLVUSL         8           12          5    alfabético
    3       FECPRES       13           18          6     numérico
    4       FECENT        19           24          6     numérico
    5       ENTREGADO     25           25          1     numérico
```

donde :

CAMPO	DESCRIPCION
COLOCA	Ubicación física
CLVUSL	Clave de usuario
FECPRES	Fecha de préstamo
FECENT	Fecha de entrega
ENTREGADO	Condición de entrega

La información contenida en la tabla PRESTA, contiene el registro de préstamos que se realizan diariamente. Entonces contiene tanto la clave del lector como la clave del libro, la fecha en la que se realiza el préstamo y la fecha cuando se entrega, para evitar ambigüedades, en la captura de las fechas su formato es:

AAMMDD

donde: *AA* Año
MM Mes
DD Día

Su condición de entrega es el siguiente: si ya entregó el libro el campo contendrá un uno, de lo contrario un cero.

Para llevar un control del tipo de préstamo que se realizó si la fecha de préstamo es la misma que la de entrega, entonces se considera que el préstamo es interno (para uso de los libros dentro de la biblioteca), de lo contrario el préstamo es externo (para uso de los libros dentro y fuera de la biblioteca).

A continuación veamos la forma en que se resuelven las siguientes consultas por medio de LIDA, hemos seleccionado una serie de preguntas que suponemos su respuesta podría ser de mucha utilidad al personal de una biblioteca. Primero observemos las tablas, por separado. Para no perder continuidad por razones de espacio hemos repetido (en algunas tablas) la clave en la primer columna para poder hacer referencia al mismo registro.

Suponemos entonces que la biblioteca citada, ha venido trabajando normalmente, y que cuenta con las siguientes bases de datos:

TABLA LIBROS:

TITULO	COLOCA	EXISTE	AUTOR
C-ESCAPE	QADE221	4	Oakland Group, Inc.
Turbo C++	QACS121	5	Borland International, Inc.
Turbo Pascal	QAHF784	10	Borland International, Inc.
Turbo Pascal Expess	QAYU378	3	Jourdain, Robert
VP Planner	QFRF654	5	Paperback Software
Othello	HWDS879	1	Shakespeare, William
The Gin Palace	LHDD787	2	Zola, Emile
Madame Bovary	LWER545	4	Flaubert Gustave
The Young Detective's Handbook	EEER657	1	Butler, W. V.
The Children of Sánchez	CVVV323	5	Lewis, Oscar
What Dr, Freud didn't tell you	WQWY464	3	Stan & Jan Berenstain
Plays	HERW743	1	Molière
Introducción a la Informática	HYUH432	6	Mora Castro, José Luis
El Lenguaje de Programación C	QDJJ788	15	Kernighan, B.W & Ritchie, D.M.
Algoritmos y estructuras de datos	QWL1576	10	Wirth, Niklaus
Cálculo con geometría analítica	QGJL878	15	Leithold, Luis
Organización de las bases de datos	QAUI533	8	Martin, James
Análisis numérico	QAYU225	10	Burden,Faires,Reynolds
Investigación de operaciones	QWWE123	4	Moskowitz
Probabilidad y estadística	QAIIH655	4	Canabos
Estadística y aplicaciones	QAWEY89	10	Mendenhalk

COLOCA	EDITORIAL	TEMA	SUBTEMA
QADE221	Oakland Group, Inc.	COMPUTACION	LENGUAJE C
QACS121	Borland International, Inc.	COMPUTACION	LENGUAJE C++
QAHF784	Borland International, Inc.	COMPUTACION	LENGUAJE PASCAL
QAYU378	Brady	COMPUTACION	LENGUAJE PASCAL
QFRF654	Paperback Software	COMPUTACION	HOJA DE CALCULO
HWDS879	Penguin Books, Inc.	LITERATURA	DRAMA
LHDD787	Avon Publications, Inc.	LITERATURA	NOVELA
LWER545	Pocket Books, Inc.	LITERATURA	NOVELA
EEER657	Granada Publishing Limited	INFANTIL	PASATIEMPOS
CVVV323	Penguin Books, Inc.	LITERATURA	NOVELA
WQWY464	Dell Pubishing Co. Inc.	EDUCACION	SEXO
HERW743	The Modern Library	LITERATURA	TEATRO
HYUH432	Bib. de Ciencias de la Admon.	ADMINISTRACION	INFORMATICA
QDJJ788	Printice-Hall Hispanoamericana	COMPUTACION	LENGUAJE C
QWL1576	Prentice-Hall Hispanoamericana	COMPUTACION	ESTRUCTURAS DE DATOS
QGJL878	Limusa, S. A.	MATEMATICAS	CALCULO
QAUI533	Prentice-Hall Hispanoamericana	COMPUTACION	BASES DE DATOS
QAYU225	Iberoamericana, S.A.	MATEMATICAS	ANALISIS NUMERICO
QWWE123	Prentice-Hall Hispanoamericana	MATEMATICAS	OPTIMIZACION
QAIIH655	McGraw-Hill	MATEMATICAS	PROBABILIDAD
QAWEY89	Iberoamericana, S.A.	MATEMATICAS	ESTADISTICA

TABLA LECTOR:

CLVUSL	NOMBRE	DIREC	TELEFONO
RHJ03	Romero Hernández Juan	Providencia #1043 Col. Del Valle	5758182
FGR01	Fernández Goméz Roberto	Maravillas Edif. 4-202 Prado Churubusco	*******
SPH02	Sánchez Pérez Héctor Rubén	Mz.29 Mod.2-E Ejército de Ote.	3455455
CEI04	Cervantes Elias Ignacio	Jacarandas #345 Col. 20 de Noviembre	7877888
GMF05	Garza Mercado Fidel	Coyoacan #123-B Del Valle	*******
CVR06	Campos Vázquez Rosa	Violeta #43 Col. Guerrero	5758181
CVS09	Chavez Vargas Sofía	3a. Secc. Mz.2-204 Cd. Azteca	5566776
VRC07	Villegas Rojas Carlos	Retorno Cecilio Rebelo /Jardín Balbuena	5789787
SGP08	Sánchez García Pablo	Ote. 243 Col. Tepalcates	5623434
RGO15	Ramos González Olga	Zaragoza #2345-A Col. Ampl. Aeropuerto	6556564
CCL11	Cruz Calva Laura	Miguel Laurent 45 Col. El Rosario	7898726
MDA13	Moctezuma Díaz Ana Lilia	Patricio Záenz #1234 Vicente Guerrero	*******
RFG12	Rojas Fajardo Gemma Isabel	Pablo González #21 Col. Bella Vista	2030507
MJA14	Maldonado Jaramillo Antonio	5 de Mayo 23-A, Centro	5782908
MIM16	Murrieta Islas Minerva	Mérida #63 Col. Roma	6789345
GRR17	González Ramírez Rubén	San Luis Potosí #192 4o.Piso Col. Roma	7893456
VYE18	Várgas Yañez Efrén	Quetzal #35 Col. Paraiso	*******
OMA20	Otero Morales Arturo	Peralvillo #45 Col. Guerrero	5239789
PCM19	Pérez Costa Magdalena	Ejército Nal. #234, Col. Polanco	5423236
MMA21	Medellín Moreno Adriana	Coyuya #110, Col. Santa Anita	6502326

CLVUSL	SANCION	DEPRES	VIGENCIA
RHJ03	1	1	1
FGR01	0	0	0
SPH02	0	1	1
CEI04	0	1	1
GMF05	0	1	1
CVR06	0	1	1
CVS09	4	0	1
VRC07	0	1	1
SGP08	0	1	1
RGO15	0	1	1
CCL11	2	0	0
MDA13	0	1	1
RFG12	0	1	1
MJA14	0	1	1
MIM16	0	1	1
GRR17	4	0	1
VYE18	0	1	1
OMA20	0	1	1
PCM19	3	1	1
MMA21	0	1	1

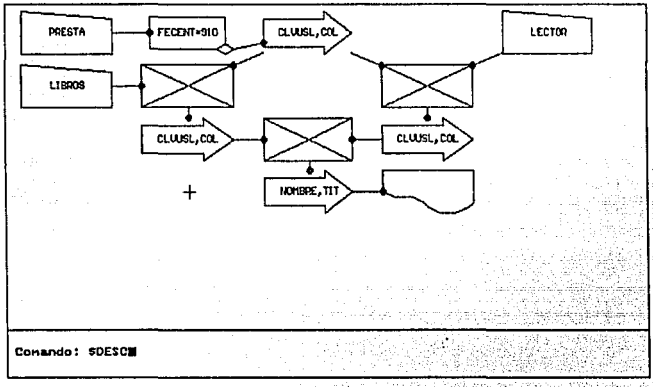
TABLA PRESTA:

COLOCA	CLVUSL	FECPRES	FECENT	ENTREGADO
QADE221	SPH02	910301	910308	0
HERW743	MMA21	910301	910308	1
QDJJ788	RFG12	910304	910311	0
QAWEY89	SGP08	910104	910111	0
QADE221	VRC07	910208	910215	1
QAYU225	CVS09	910610	910610	1
QAUI533	OMA20	910301	910308	0
QAHF784	MIM16	910620	910627	0
WQWY464	RHJ03	910610	910610	0
QAWEY89	GMF05	910611	910618	0
QAIHI655	MJA14	910605	910615	1
QFRF654	VYE18	910607	910614	0
HWDS879	PCM19	910608	910615	0
QDJJ788	MDA13	910401	910411	1
QWLI576	CVR06	910616	910623	0

En adelante presentamos diferentes consultas sobre las bases de datos anteriores, para las cuales serán resueltas por medio de un flujograma diseñado en LIDA. Los resultados mostrados fueron generados por el sistema integrado SABRE. Como parte adicional y como complemeto didáctico, se incluye el código generado por SABRE en ISBL extendido:

C1. *Se requiere determinar que libros se han prestado y conocer a los usuarios que les fue prestado un libro.*

LIDA:



Procedimiento ISBL Extendido:

```
ZZZ1!=LECTOR%CLVUSL,NOMBRE
ZZZ2!=PRESTA%CLVUSL,COLOCA
ZZZ3!=ZZZ1!*ZZZ2!
ZZZ4!=ZZZ3!*LIBROS
ZZZ5!=ZZZ4!%CLVUSL,NOMBRE,TITULO
$LISTA ZZZ5!>CON
```

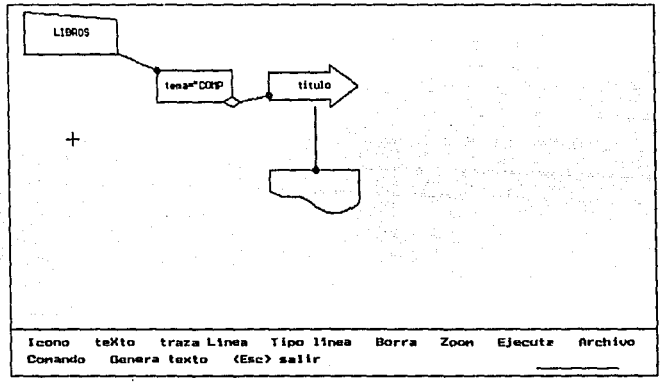
Resultado:

CLVUSL	NOMBRE	TITULO
RHJ03	Romero Hernández Juan	What Dr, Freud didn't tell you
SPH02	Sánchez Pérez Héctor Rubén	C-ESCAPE
GMF05	Garza Mercado Fidel	Estadística y aplicaciones
CVR06	Campos Vázquez Rosa	Algoritmos y estructuras de datos
CVS09	Chavez Vargas Sofía	Análisis numérico
VRC07	Villegas Rojas Carlos	C-ESCAPE
SGP08	Sánchez García Pablo	Estadística y aplicaciones
MDA13	Moctezuma Díaz Ana Lilia	El Lenguaje de Programación C
RFG12	Rojas Fajardo Gemma Isabel	El Lenguaje de Programación C
MJA14	Maldonado Jaramillo Antonio	Probabilidad y estadística

CLVUSL	NOMBRE	TITULO
MIM16	Murrieta Islas Minerva	Turbo Pascal
VYE18	Várgas Yañez Efrén	VP Planner
OMA20	Otero Morales Arturo	Organización de las bases de datos
PCM19	Pérez Costa Magdalena	Othello
MMA21	Medellín Moreno Adriana	Plays

C2. *Como se van a adquirir más libros de computación, se necesita saber que libros existen de este tema.*

LIDA:



Procedimiento ISBL Extendido:

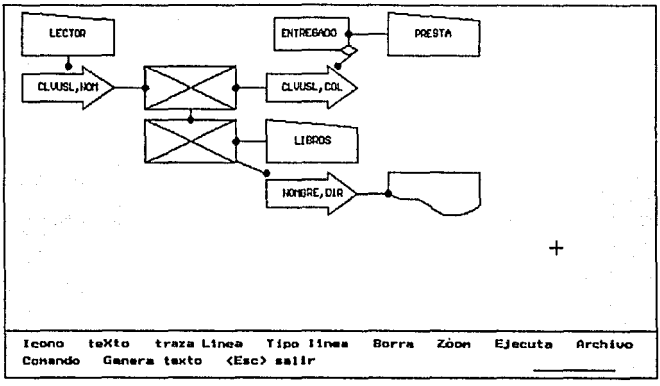
```
ZZZ2!=LIBROS:(tema="COMPUTACION"),ZZZ1!
ZZZ3!=ZZZ2!%titulo
$LISTA ZZZ3!>CON
```

Resultado:

TITULO
C-ESCAPE
Turbo C++
Turbo Pascal
Turbo Pascal Express
VP Planner
El Lenguaje de Programación C
Algoritmos y estructuras de datos
Organización de las bases de datos

C3. *Algunos usuarios no han entregado los libros que les fueron prestados, por lo que es necesario realizar una lista de ellos.*

LIDA:



Procedimiento ISBL Extendido:

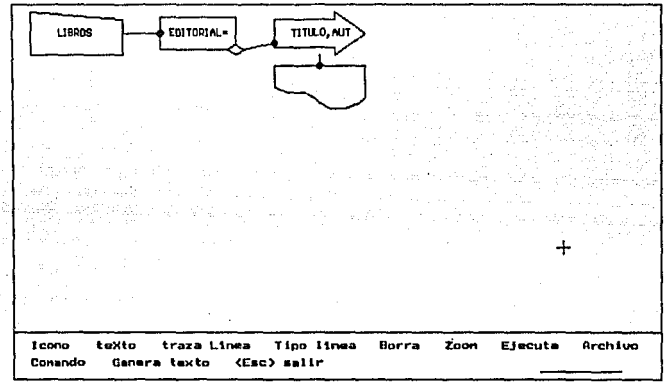
```
ZZZ1!=LECTOR%CLVUSL,NOMBRE,DIREC,TELEFONO
ZZZ3!=PRESTA:(ENTREGADO = 0),ZZZ2!
ZZZ4!=ZZZ3!%CLVUSL,COLOCA
ZZZ5!=ZZZ1!*ZZZ4!
ZZZ6!=LIBROS*ZZZ5!
ZZZ7!=ZZZ6!%NOMBRE,DIREC,TELEFONO
$LISTA ZZZ7!>CON
```

Resultado:

NOMBRE	DIREC	TELEFONO
Sánchez Pérez Héctor Rubén	Mz.29 Mod.2-E Ejército de Ote.	3455455
Murrieta Islas Minerva	Mérida #63 Col. Roma	6789345
Várgas Yañez Efrén	Quetzal #35 Col. Paraíso	*******
Pérez Costa Magdalena	Ejército Nal. #234, Col. Polanco	5423236
Romero Hernández Juan	Providencia #1043 Col. Del Valle	5758182
Rojas Fajardo Gemma Isabel	Pablo González #21 Col. Bella Vista	2030507
Campos Vázquez Rosa	Violeta #43 Col. Guerrero	5758181
Otero Morales Arturo	Peralvillo #45 Col. Guerrero	5239789
Garza Mercado Fidel	Coyoacan #123-B Del Valle	*******
Sánchez García Pablo	Ote. 243 Col. Tepalcates	5623434

C4. *Por ampliación de los libros de la editorial Prentice-All, es necesario determinar cuales son las obras y su existencia.*

LIDA:



Procedimiento ISBL Extendido:

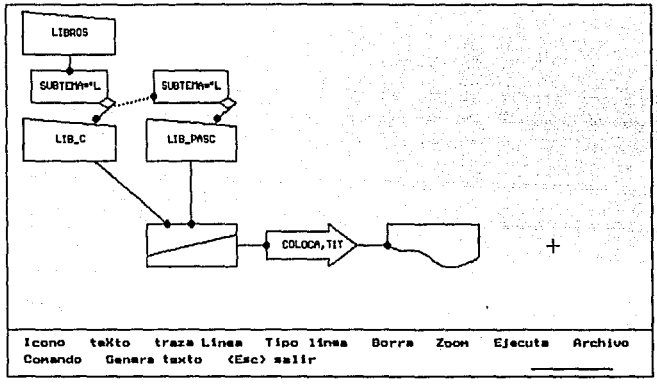
```
ZZZ2!=LIBROS:(EDITORIAL="Prentice-Hall Hispanoamericana"),ZZZ1!
ZZZ3!=ZZZ2!%TITULO,AUTOR,EXISTE
$LISTA ZZZ3!>CON
```

Resultado:

TITULO	AUTOR	EXISTE
El Lenguaje de Programación C	Kernighan, B.W & Ritchie, D.M.	15
Algoritmos y estructuras de datos	Wirth, Niklaus	10
Organización de las bases de datos	Martin, James	8
Investigación de operaciones	Moskowitz	4

C5. *Para realizar un prestamo, un usuario requiere conocer cuales son los libros que existen de lenguaje C, C++ y Pascal.*

LIDA :



Procedimiento ISBL Extendido:

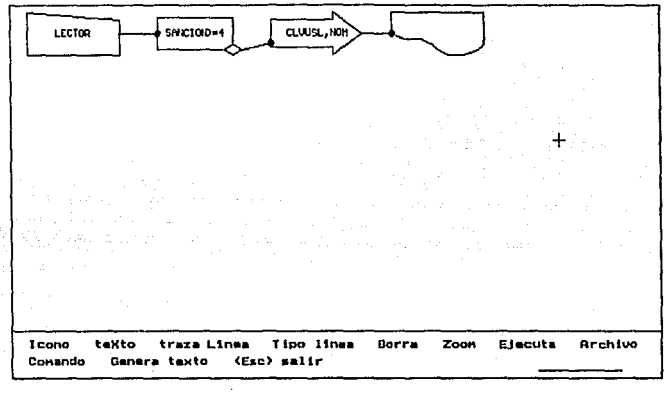
```
LIB_C=LIBROS:(SUBTEMA="LENGUAJE C" O SUBTEMA ="LENGUAJE C++"),ZZZ1!
LIB_PASC=ZZZ1!:(SUBTEMA="LENGUAJE PASCAL"),ZZZ2!
ZZZ3!=LIB_C+LIB_PASC
ZZZ4!=ZZZ3!%COLOCA,TITULO,AUTOR,EDITORIAL
$LISTA ZZZ4!>CON
```

Resultado:

COLOCA TITULO	AUTOR	EDITORIAL
QADE221 C-ESCAPE	Oakland Group, Inc.	Oakland Group, Inc.
QACS121 Turbo C++	Borland International, Inc.	Borland International, Inc.
QDJJ788 El Lenguaje de Programación C	Kernighan, B.W & Ritchie,D.M.	Prentice-Hall Hispanoamericana
QAHF784 Turbo Pascal	Borland International, Inc.	Borland International, Inc.
QAYU378 Turbo Pascal Expess	Jourdain, Robert	Brady

C6. *La biblioteca se dispone reconocer a los usuarios que se encuentran sancionados, para presentarlos en una lista y negarles los préstamos.*

LIDA:



Procedimiento ISBL Extendido:

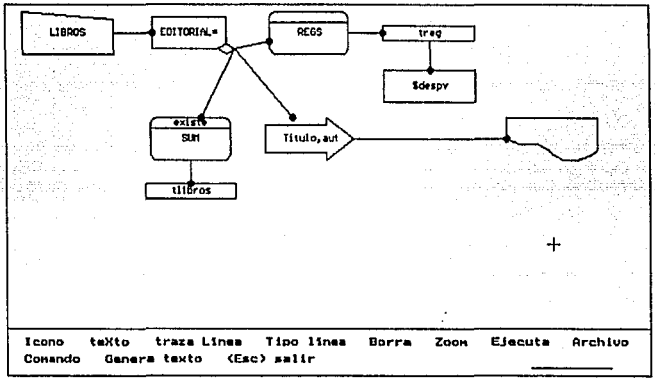
```
ZZZ2!=LECTOR:(SANCION>=4),ZZZ1!
ZZZ3!=ZZZ2!%CLVUSL,NOMBRE
$LISTA ZZZ3!>CON
```

Resultado:

CLVUSL	NOMBRE
CVS09	Chavez Vargas Sofía
GRR17	González Ramírez Rubén

C7. *Las obras de la editorial Borland International, Inc. han sido incluidas en un préstamo interbibliotecario, por lo que se requiere determinar cuales obras existen, así como la cantidad total de libros, y el total de obras.*

LIDA:



Procedimiento ISBL Extendido:

```
ZZZ2!=LIBROS:(EDITORIAL="Borland International, Inc."),ZZZ1!
@tlibros=SUM(ZZZ2!,existe)
@treg=REGS(ZZZ2!)
$despv
ZZZ3!=ZZZ2!%Titulo,autor,existe
$LISTA ZZZ3!>CON
```

Resultado:

```
1 TLIBROS   = 15
2 TREG      =  2
```

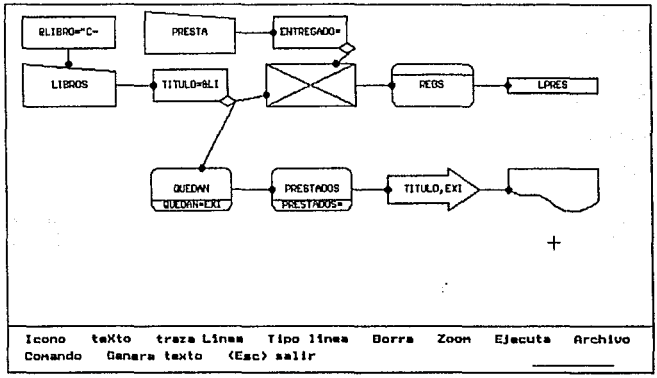
donde:

TLIBROS = Cantidad total de libros de la editorial Borland
TREG = Número de obras de la editorial Borland

TITULO	AUTOR	EXISTE
Turbo C++	Borland International, Inc.	5
Turbo Pascal	Borland International, Inc.	10

C8. *Al parecer el libro C-ESCAPE, es una obra muy solicitada, por lo cual es necesario conocer cuántos tomos existen en la biblioteca, así como con cuántos se cuenta en este momento, y en base a lo anterior cuántos se han prestado.*

LIDA:



Procedimiento ISBL Extendido:

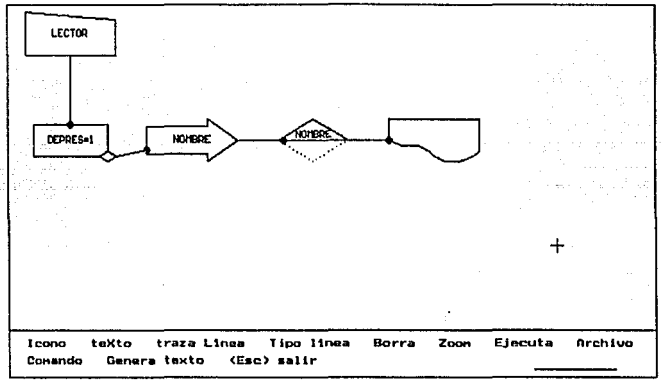
```
ZZZ2!=PRESTA:(ENTREGADO=0),ZZZ1!
@LIBRO="C-ESCAPE"
ZZZ4!=LIBROS:(TITULO=&LIBRO),ZZZ3!
ZZZ5!=ZZZ2!*ZZZ4!
@LPRES=REGS(ZZZ5!)
ZZZ6!=ZZZ4![QUEDAN=EXISTE-LPRES]
ZZZ7!=ZZZ6![PRESTADOS=LPRES]
ZZZ8!=ZZZ7!%TITULO,EXISTE,QUEDAN,PRESTADOS
$LISTA ZZZ8!>CON
```

Resultado:

TITULO	EXISTE	QUEDAN	PRESTADOS
C-ESCAPE	4	3	1

C9. Al iniciar el período de resello de credenciales es necesario que los usuarios que tienen derecho a préstamo se les debe dar prioridad, por lo que se requiere generar la lista de usuarios que tienen derecho a préstamo, mostrados de forma ascendente.

LIDA:



Procedimiento ISBL Extendido:

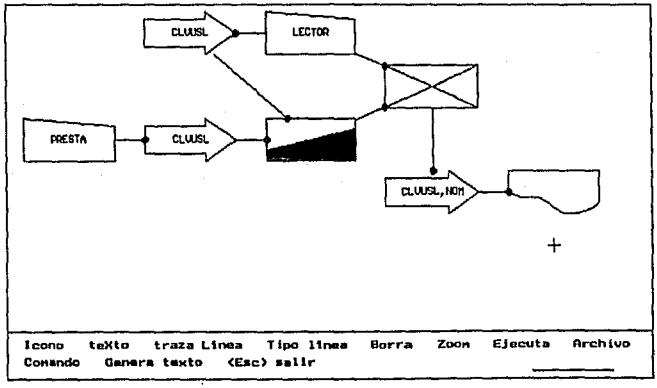
```
ZZZ2!=LECTOR:(DEPRES=1),ZZZ1!
ZZZ3!=ZZZ2!%NOMBRE
ZZZ4!=ZZZ3!#NOMBRE,A
$LISTA ZZZ4!>CON
```

Resultado:

NOMBRE
Campos Vázquez Rosa
Cervantes Elías Ignacio
Garza Mercado Fidel
Maldonado Jaramillo Antonio
Medellín Moreno Adriana
Moctezuma Díaz Ana Lilia
Murrieta Islas Minerva
Otero Morales Arturo
Pérez Costa Magdalena
Ramos González Olga
Rojas Fajardo Gemma Isabel
Romero Hernández Juan
Sánchez García Pablo
Sánchez Pérez Héctor Rubén
Villegas Rojas Carlos
Várgas Yañez Efrén

C10. *Para las nuevas políticas de servicio, el departamento de estadísticas ha solicitado la lista de usuarios que no han utilizado los servicios de la biblioteca.*

LIDA:



Procedimiento ISBL Extendido:

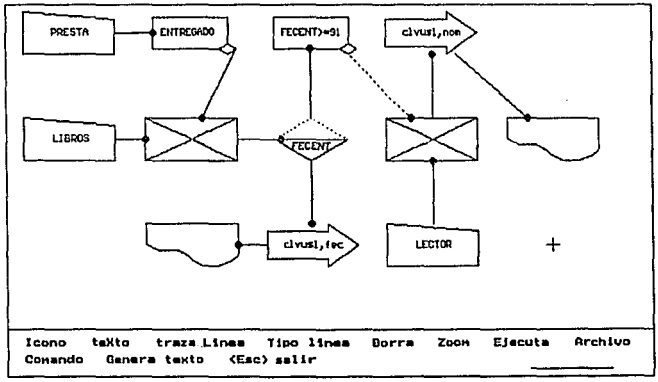
```
ZZZ1!=LECTOR%CLVUSL
ZZZ2!=PRESTA%CLVUSL
ZZZ3!=ZZZ1!-ZZZ2!
ZZZ4!=LECTOR*ZZZ3!
ZZZ5!=ZZZ4!%CLVUSL,NOMBRE
$LISTA ZZZ5!>CON
```

Resultado:

CLVUSL	NOMBRE
FGR01	Fernández Goméz Roberto
CEI04	Cervantes Elias Ignacio
RGO15	Ramos González Olga
CCL11	Cruz Calva Laura
GRR17	González Ramirez Rubén

C11. *Dentro de los procesos rutinarios de la biblioteca, es necesario obtener una relación de los usuarios que deben entregar libros, ordenados según su fecha de entrega, y adicionalmente una lista de los usuarios que debieron entregar sus libros antes del 01-Jun-91, los cuales ya son acreedores a una sanción.*

LIDA:



Procedimiento ISBL Extendido:

Usuarios que deben entregar libros.

```
ZZZ2!=PRESTA:(ENTREGADO = 0),ZZZ1!
ZZZ3!=ZZZ2!*LIBROS
ZZZ4!=ZZZ3!#FECENT,D
ZZZ5!=ZZZ4!%clvusl,fecent,titulo
$LISTA ZZZ5!>CON
```

Usuarios que deben entregar sus libros antes del 01-Jun-91.

```
ZZZ7!=ZZZ4!:(FECENT>=910601),ZZZ6!
ZZZ8!=ZZZ6!*LECTOR
ZZZ9!=ZZZ8!%clvusl,nombre,titulo
$LISTA ZZZ9!>CON
```

Resultado:

Usuarios que deben entregar libros.

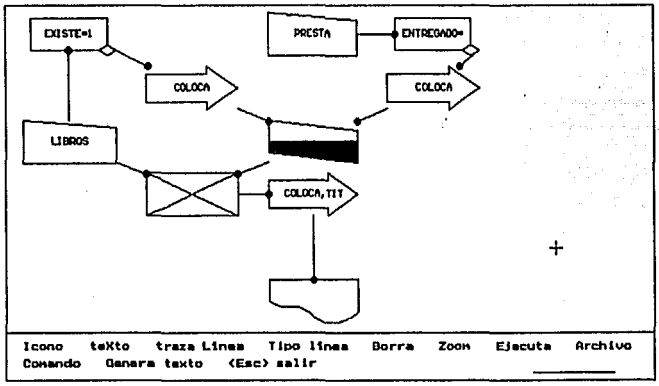
CLVUSL	FECENT	TITULO
MIM16	910627	Turbo Pascal
CVR06	910623	Algoritmos y estructuras de datos
GMF05	910618	Estadística y aplicaciones
PCM19	910615	Othello
VYE18	910614	VP Planner
RHJ03	910610	What Dr, Freud didn't tell you
RFG12	910311	El Lenguaje de Programación C
SPI02	910308	C-ESCAPE
OMA20	910308	Organización de las bases de datos
SGP08	910111	Estadística y aplicaciones

Usuarios que deben entregar sus libros antes del 01-Jun-91.

CLVUSL	NOMBRE	TULO
RFG12	Rojas Fajardo Gemma Isabel	El Lenguaje de Programación C
SPH02	Sánchez Pérez Héctor Rubén	C-ESCAPE
OMA20	Otero Morales Arturo	Organización de las bases de datos
SGP08	Sánchez García Pablo	Estadística y aplicaciones

C12. *La existencia de los libros es variable debido al uso y préstamo frecuente de los mismos, pero es necesario tomar atención en aquellos cuya existencia es mínima, y aún más aquellos que se encuentran prestados, y que debe suponerse son los de mayor solicitud.*

LIDA:



Procedimiento ISBL Extendido:

```
ZZZ2!=PRESTA:(ENTREGADO=0),ZZZ1!
ZZZ3!=ZZZ2!%COLOCA
ZZZ5!=LIBROS:(EXISTE=1),ZZZ4!
ZZZ6!=ZZZ5!%COLOCA
ZZZ7!=ZZZ3!.ZZZ6!
ZZZ8!=LIBROS*ZZZ7!
ZZZ9!=ZZZ8!%COLOCA,TITULO,AUTOR,EXISTE
$LISTA ZZZ9!>CON
```

Resultado:

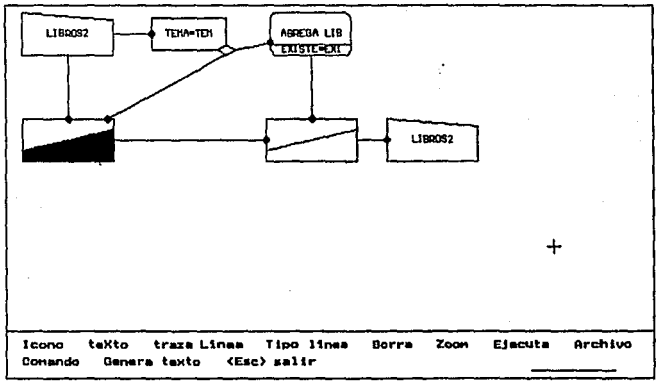
COLOCA	TITULO	AUTOR	EXISTE
IIWDS879	Othello	Shakespeare, William	1

C13. *Han llegado a la biblioteca un paquete de libros que se pidió el mes pasado, para lo cual es necesario actualizar la existencia de cada uno de los volúmenes de la siguiente forma:*

Todos los libros de computación con dos más para cada volumen, y los libros de literatura con uno más para cada volumen.

LIDA:

Genera procedimiento CRECE.LIP



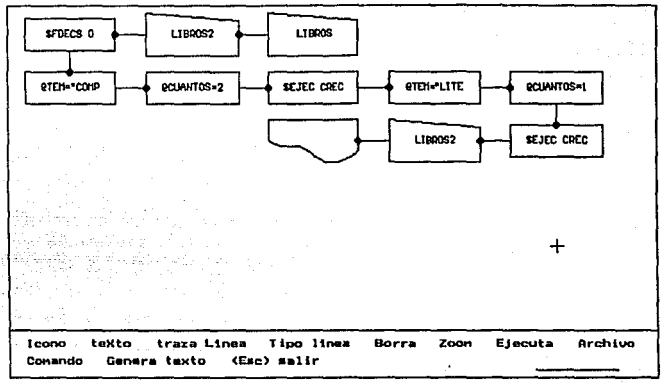
Procedimiento ISBL Extendido:

CRECE.LIP:

```
ZZZ2!=LIBROS2:(TEMA=TEM),ZZZ1!
ZZZ3!=ZZZ2![EXISTE=EXISTE+CUANTOS]
ZZZ4!=LIBROS2-ZZZ2!
LIBROS2=ZZZ3!+ZZZ4!
```

LIDA :

Utiliza el procedimiento CRECE.LIP para actualizar



Actualización:

```
LIBROS2=LIBROS
$FDECS 0
@TEM="COMPUTACION"
@CUANTOS=2
$EJEC CRECE
@TEM="LITERATURA"
@CUANTOS=1
$EJEC CRECE
$LISTA LIBROS2>CON
```

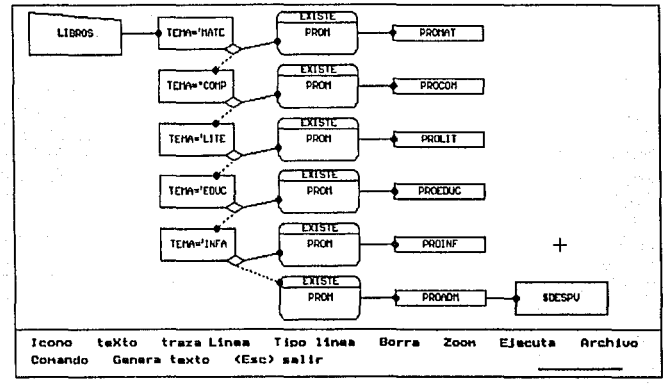
Resultado:

TITULO	COLOCA	AUTOR
Othello	HWDS879	Shakespeare, William
The Gin Palace	LHDD787	Zola, Emile
Madame Bovary	LWER545	Flaubert Gustave
The Children of Sánchez	CVVV323	Lewis, Oscar
Plays	HERW743	Molière
C-ESCAPE	QADE221	Oakland Group, Inc.
Turbo C++	QACS121	Borland International, Inc.
Turbo Pascal	QAIIF784	Borland International, Inc.
Turbo Pascal Expess	QAYU378	Jourdain, Robert
VP Planner	QFRF654	Paperback Software
El Lenguaje de Programación C	QDJJ788	Kernighan, B.W & Ritchie, D.M.
Algoritmos y estructuras de datos	QWLI576	Wirth, Niklaus
Organización de las bases de datos	QAUI533	Martin, James
The Young Detective's Handbook	EEER657	Butler, W. V.
What Dr, Freud didn't tell you	WQWY464	Stan & Jan Berenstain
Introducción a la Informática	HYUH432	Mora Castro, José Luis
Cálculo con geometría analítica	QGJL878	Leithold, Luis
Análisis numérico	QAYU225	Burden,Faires,Reynolds
Investigación de operaciones	QWWE123	Moskowitz
Probabilidad y estadística	QAHH655	Canabos
Estadística y aplicaciones	QAWEY89	Mendenhalk

COLOCA	EDITORIAL	TEMA	SUBTEMA	EXISTE
HWDS879	Penguin Books, Inc.	LITERATURA	DRAMA	2
LHDD787	Avon Publications, Inc.	LITERATURA	NOVELA	3
LWER545	Pocket Books, Inc.	LITERATURA	NOVELA	5
CVVV323	Penguin Books, Inc.	LITERATURA	NOVELA	6
HERW743	The Modern Library	LITERATURA	TEATRO	2
QADE221	Oakland Group, Inc.	COMPUTACION	LENGUAJE C	6
QACS121	Borland International, Inc.	COMPUTACION	LENGUAJE C++	7
QAIIF784	Borland International, Inc.	COMPUTACION	LENGUAJE PASCAL	12
QAYU378	Brady	COMPUTACION	LENGUAJE PASCAL	5
QFRF654	Paperback Software	COMPUTACION	HOJA DE CALCULO	7
QDJJ788	Prentice-Hall Hispanoamericana	COMPUTACION	LENGUAJE C	17
QWLI576	Prentice-Hall Hispanoamericana	COMPUTACION	ESTRUCTURAS DE DATOS	12
QAUI533	Prentice-Hall Hispanoamericana	COMPUTACION	BASES DE DATOS	10
EEER657	Granada Publishing Limited	INFANTIL	PASATIEMPOS	1
WQWY464	Dell Pubishing Co. Inc.	EDUCACION	SEXO	3
HYUH432	Bib. de Ciencias de la Admon.	ADMINISTRACION	INFORMATICA	6
QGJL878	Limusa, S. A.	MATEMATICAS	CALCULO	15
QAYU225	Iberoamericana, S.A.	MATEMATICAS	ANALISIS NUMERICO	10
QWWE123	Prentice-Hall Hispanoamericana	MATEMATICAS	OPTIMIZACION	4
QAHH655	McGraw-Hill	MATEMATICAS	PROBABILIDAD	4
QAWEY89	Iberoamericana, S.A.	MATEMATICAS	ESTADISTICA	10

C14. *Para complementar un estudio de mantenimiento y actualización de la biblioteca, se requiere determinar la existencia promedio de libros por tema.*

LIDA:



Procedimiento ISBL Extendido:

```
ZZZ2!=LIBROS:(TEMA='MATEMATICAS'),ZZZ1!
@PROMAT=PROM(ZZZ2!,EXISTE)
ZZZ4!=ZZZ1!:(TEMA="COMPUTACION"),ZZZ3!
@PROCOM=PROM(ZZZ4!,EXISTE)
ZZZ6!=ZZZ3!:(TEMA='LITERATURA'),ZZZ5!
@PROLIT=PROM(ZZZ6!,EXISTE)
ZZZ8!=ZZZ5!:(TEMA='EDUCACION'),ZZZ7!
@PROEDUC=PROM(ZZZ8!,EXISTE)
ZZZ10!=ZZZ7!:(TEMA='INFANTIL'),ZZZ9!
@PROINF=PROM(ZZZ10!,EXISTE)
@PROADM=PROM(ZZZ9!,EXISTE)
$DESPV
```

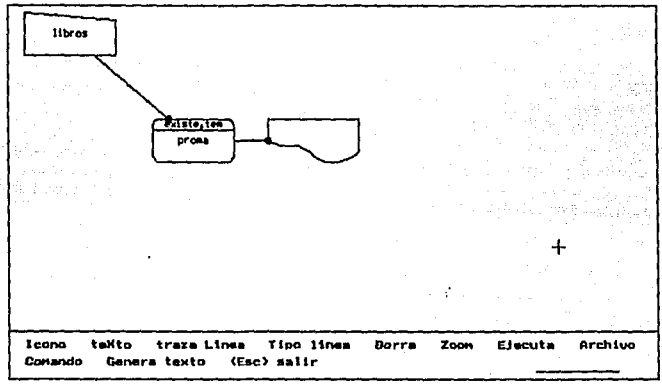
Resultado:

1 PROADM	= 6	Promedio de libros de Administración
2 PROCOM	= 7.5	Promedio de libros de Computación
3 PROEDUC	= 3	Promedio de libros de Educación
4 PROINF	= 1	Promedio de libros de Informática

5 PROLIT = 2.6 Promedio de libros de Literatura
6 PROMAT = 8.6 Promedio de libros de Matemáticas

Utilizando la función PROMA el procedimiento se simplificaría de la siguiente manera:

LIDA:



Procedimiento ISBL Extendido:

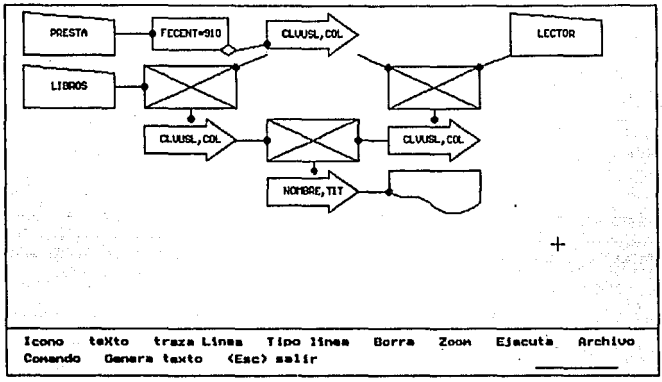
```
ZZZ1!=proma(libros,existe,tema)
$LISTA ZZZ1!>CON
```

Resultado:

TEMA	EXISTE
ADMINISTRACION	6.00000
COMPUTACION	7.50000
EDUCACION	3.00000
INFANTIL	1.00000
LITERATURA	2.60000
MATEMATICAS	8.60000

C15. *En el siguiente período de préstamos bibliotecarios, es necesario conocer la lista de usuarios que deberán entregar libros el día 18 de junio del 91.*

LIDA:



Procedimiento ISBL Extendido:

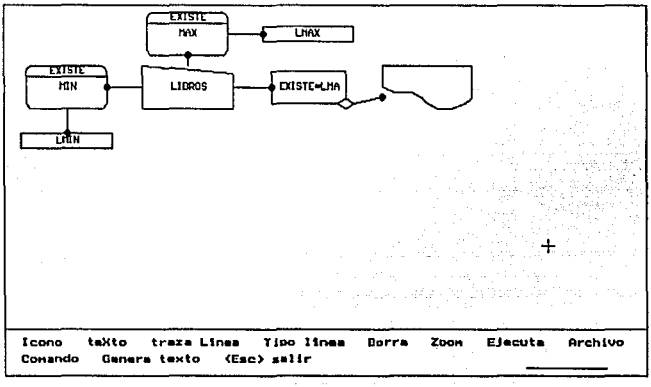
```
ZZZ2!=PRESTA:(FECENT=910618),ZZZ1!
ZZZ3!=ZZZ2!%CLVUSL,COLOCA
ZZZ4!=ZZZ3!*LIBROS
ZZZ5!=ZZZ4!%CLVUSL,COLOCA,TITULO
ZZZ6!=ZZZ3!*LECTOR
ZZZ7!=ZZZ6!%CLVUSL,COLOCA,NOMBRE
ZZZ8!=ZZZ5!*ZZZ7!
ZZZ9!=ZZZ8!%NOMBRE,TITULO
$LISTA ZZZ9!>CON
```

Resultado:

NOMBRE	TITULO
Garza Mercado Fidel	Estadística y aplicaciones

C16. *Antes de adquirir el siguiente pedido de libros, se desea saber cuales son los libros que tienen menor y mayor existencia.*

LIDA:



Procedimiento ISBL Extendido:

```
@LMIN=MIN(LIBROS,EXISTE)
@LMAX=MAX(LIBROS,EXISTE)
ZZZ2!=LIBROS:(EXISTE=LMAX O EXISTE=LMIN),ZZZ1!
$LISTA ZZZ2!>CON
```

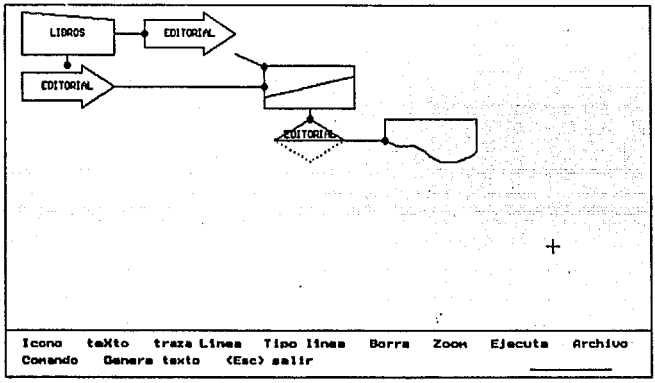
Resultado:

TITULO	COLOCA	AUTOR
Othello	HWDS879	Shakespeare, William
The Young Detective's Handbook	EEER657	Butler, W. V.
Plays	HERW743	Molière
El Lenguaje de Programación C	QDJJ788	Kernighan, B.W & Ritchie, D.M.
Cálculo con geometría analítica	QGJL878	Leithold, Luis

COLOCA	EDITORIAL	TEMA	SUBTEMA	EXISTE
HWDS879	Penguin Books, Inc.	LITERATURA	DRAMA	1
EEER657	Granada Publishing Limited	INFANTIL	PASATIEMPOS	1
HERW743	The Modern Library	LITERATURA	TEATRO	1
QDJJ788	Prentice-Hall Hispanoamericana	COMPUTACION	LENGUAJE C	15
QGJL878	Limusa, S. A.	MATEMATICAS	CALCULO	15

C17. *Uno de los servicios bibliotecarios consiste en no sólo brindarle al usuario los títulos y nombres de los autores de los libros, sino además, informar sobre las editoriales que se manejan; para ello se requiere generar una lista de las editoriales que cuentan con algunas obras en la biblioteca.*

LIDA:



Procedimiento ISBL Extendido:

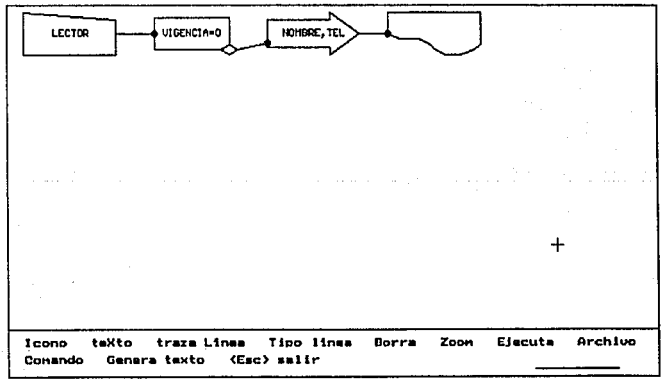
```
ZZZ1!=LIBROS%EDITORIAL
ZZZ2!=LIBROS%EDITORIAL
ZZZ3!=ZZZ1!+ZZZ2!
ZZZ4!=ZZZ3!#EDITORIAL,A
$LISTA ZZZ4!>CON
```

Resultado:

EDITORIAL
Avon Publications, Inc.
Bib. de Ciencias de la Admon.
Borland International, Inc.
Brady
Dell Pubishing Co. Inc.
Granada Publishing Limited
Iberoamericana, S.A.
Limusa, S. A.
McGraw-Hill
Oakland Group, Inc.
Paperback Software
Penguin Books, Inc.
Pocket Books, Inc.
Prentice-Hall Hispanoamericana
The Modern Library

C18. *Una vez iniciado el período de préstamos es necesario dar aviso a los usuarios que no han resellado su credencial, para lo cual, el servicio más rápido sería por vía telefónica, para ello, se debe generar una lista de los usuarios que no tienen credencial vigente y que cuentan con teléfono.*

LIDA:



Procedimiento ISBL Extendido:

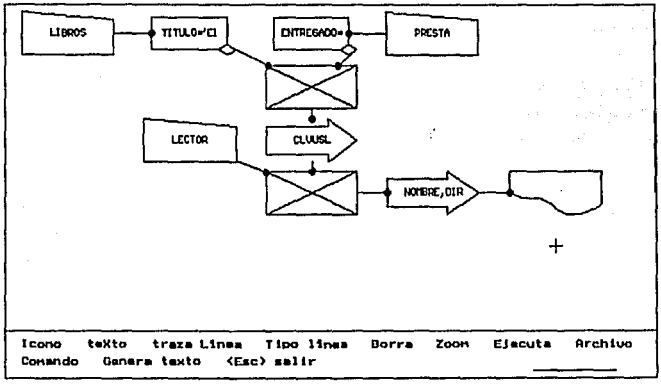
```
ZZZ2!=LECTOR:(VIGENCIA=0 Y TELEFONO <> "********"),ZZZ1!
ZZZ3!=ZZZ2!%NOMBRE,TELEFONO
$LISTA ZZZ3!>CON
```

Resultado:

NOMBRE	TELEFONO
Cruz Calva Laura	7898726

C19. *Un usuario desea saber si el libro "El lenguaje de Programación C", lo tiene otro usuario que él conoce y preferirla ya no pedirlo, pues estudiarán juntos.*

LIDA:



Procedimiento ISBL Extendido:

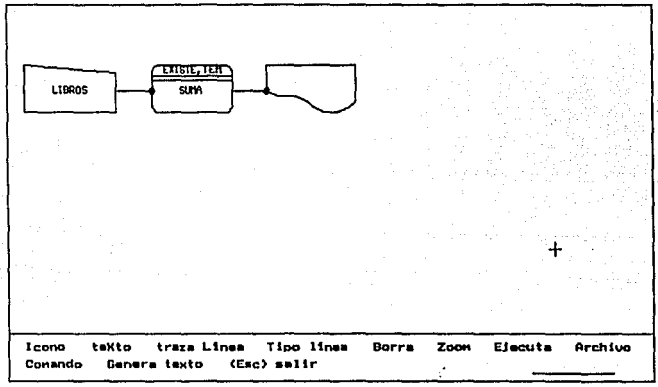
```
ZZZ2!=LIBROS:(TITULO='El lenguaje de programación C'),ZZZ1!
ZZZ4!=PRESTA:(ENTREGADO=0),ZZZ3!
ZZZ5!=ZZZ2!*ZZZ4!
ZZZ6!=ZZZ5!%CLVUSL
ZZZ7!=LECTOR*ZZZ6!
ZZZ8!=ZZZ7!%NOMBRE,DIREC,TELEFONO
$LISTA ZZZ8!>CON
```

Resultado:

NOMBRE	DIREC	TELEFONO
Rojas Fajardo Gemma Isabel	Pablo González #21 Col. Bella Vista	2030507

C20. *Con fines de actualizar el inventario se requiere determinar cuantas obras existen por tema.*

LIDA:



Procedimiento ISBL Extendido:

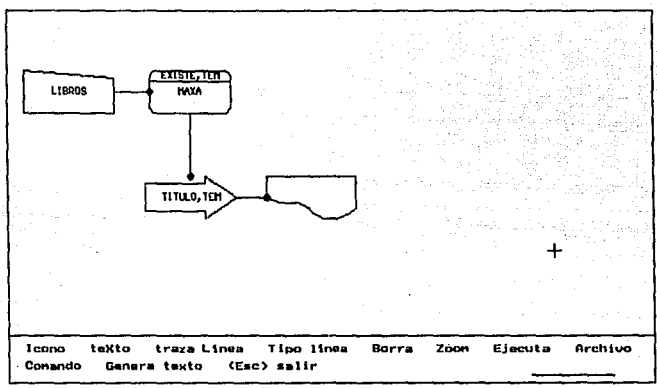
```
ZZZ1!=SUMA(LIBROS,EXISTE,TEMA)
$LISTA ZZZ1!>CON
```

Resultado:

TEMA	EXISTE
ADMINISTRACION	6.00000
COMPUTACION	60.00000
EDUCACION	3.00000
INFANTIL	1.00000
LITERATURA	13.00000
MATEMATICAS	43.00000

C21. *Para iniciar un nuevo sistema de ficheros el administrador de la biblioteca necesita una lista de los libros con mayor existencia por cada tema.*

LIDA:



Procedimiento ISBL Extendido:

```
ZZZ1!=MAXA(LIBROS,EXISTE,TEMA)
ZZZ2!=ZZZ1!%TITULO,TEMA,EXISTE
$LISTA ZZZ2!>CON
```

Resultado:

TITULO	TEMA	EXISTE
Introducción a la Informática	ADMINISTRACION	6
El Lenguaje de Programación C	COMPUTACION	15
What Dr, Freud didn't tell you	EDUCACION	3
The Young Detective's Handbook	INFANTIL	1
The Children of Sánchez	LITERATURA	5
Cálculo con geometría analítica	MATEMATICAS	15

CONCLUSIONES

Hemos logrado definir un lenguaje de características visuales que permite manipular bases de datos relacionales; LIDA es un prototipo para el desarrollo de software orientado tanto al manejo de bases de datos, como a las interfases gráficas. La creación de herramientas es sin lugar a duda una contribución a la Ingeniería de Software, y una oportunidad para alcanzar eficientemente un progreso tecnológico. En nuestro proyecto, se han conjugado varios aspectos y teorías computacionales que sirvieron no tan sólo para el desarrollo de LIDA, sino también para la implementación de SABRE y el desempeño de herramientas como el descriptor de archivos y los captadores, ideas propuestas primeramente por el Dr. Sergio Chapa quién con gran ímpetu dirigió nuestro trabajo.

Actualmente existe una tendencia por desarrollar sistemas con características visuales; la comunicación entre el usuario y la computadora intenta por lo general ser amigable, ello implica que los sistemas sean cada vez más sencillos de comprender y operar. Esto es uno de los objetivos que se ha fijado la propia Ingeniería de Software, la busqueda de metodologías de programación y desarrollo de sistemas, con la simplicidad de su manejo.

El sistema SABRE, es una muestra de lo que connstituye un sistema integrado utilizando el lenguaje LIDA como parte central. El lenguaje iconográfico ha quedado definido, y su semántica y sintaxis pueden ser aprovechadas para la implementación de un sistema de bases de datos completo. Nuevas herramientas pueden construirse a partir de LIDA:

- Un Adminnistrador de Bases de Datos Relacionales Gráfico.

- Un Manejador de Bases de Datos Relacionales, capaz de mantener la integridad de las bases de datos, y que permita trabajar con Indices.

- Software que genere código ISBL extendido, tal que pueda ser traducido y ejecutado por un intérprete ISBL Extendido.

- Un sistema gráfico que constituya utilerías de mantenimiento de bases de datos relacionales, así como respaldos y restauración de información.

- Disponibilidad de una plataforma para soportar LIDA en una red de microcomputadoras.

- Un sistema integrado SABRE para trabajar dentro de otros sistemas de computo, minicomputadoras, y macrocomputadoras.

- Otros lenguajes intermedios para LIDA

LIDA se encuentra enmarcado dentro del nivel más inferior del concepto CASE, nivel en el que la mayor parte de los sistemas no son gráficos, no obstante es un lenguaje que por sus características, podría ser objeto para desarrollar CASE de niveles superiores que se adecuen al lenguaje. Uno de los productos CASE son los lenguajes de cuarta generación, dentro del que se clasifica LIDA como un lenguaje visual para desarrollar aplicaciones dentro de un sistema de información, no obstante, LIDA también puede servir como contribución para el desarrollo de lenguajes visuales, y no precisamente de manejo de bases de datos, sino de lenguajes visuales de propósito más específico, como lo puede ser un lenguaje visual de procedimientos para un sistema operativo o bien, un lenguaje matemático visual para desarrollar procedimientos de cálculo, etc.

El sistema integrado SABRE que presentamos como parte del proyecto, sufre de las deficiencias de cualquier sistema que se implementa dentro de una computadora, así que es necesario advertir que su funcionamiento, se encuentra limitado por las características de hardware que lo puede soportar, de manera que es necesario considerar esta cuestión para la funcionalidad que puede tener LIDA dentro de SABRE. Indudablemente el aumento de la tecnología, puede mejorar en mucho la operación e implementación del lenguaje iconográfico; el refinamiento de los iconos ayuda notablemente su presentación, pensamos que junto con la estructura iconográfica ya definida, se le puede añadir color y posiblemente sonido a cada icono, para obtener un mejoramiento en el aprendizaje y desempeño del propio lenguaje.

Por último, deseamos que el lector interesado en continuar el desarrollo de herramientas como las presentadas aquí, tome en cuenta que los sistemas gráficos son simplemente una opción para un nuevo estilo de programación, y que aún falta mucho camino por recorrer. Lo importante es contribuir con ideas bien planteadas para elevar la calidad del Software, de nuestro gran espacio informático.

BIBLIOGRAFIA

[TURBO5] *Borland International Inc.*
Turbo Pascal ver 5.5.
2 tomos
Scotts Valley Ca., U.S.A.
Borland International Inc., 1989

[CHAPA85] *Chapa Vergara, Sergio V.*
Herramientas para consulta y captura basadas en el Descriptor de Archivos.
Centro de Investigación y de Estudios Avanzados,
I.P.N., Departamento de Ingeniería Eléctrica,
México, D.F.
Reporte técnico No. 15, 1985
Serie Amarilla, Investigación
60 pp.

[CHAPA86] *Chapa Vergara, Sergio V.*
Definición del Modelo de un lenguaje de Programación (Lenguaje de Flujograma).
Centro de Investigación y de Estudios Avanzados,
I.P.N.,Departamento de Ingeniería Eléctrica,
México, D.F.
Reporte técnico No. 51, 1986
Serie Amarilla, Investigación
57 pp.

[CHAPA87] *Chapa Vergara, Sergio V.*
Lenguaje de Flujograma para la Automatización de las Oficinas.
Memorias de las conferencias
de MEXICOM 87 IEEE, Ocutbre, 1987.
47-56 pp.

[CHAPA89] *Chapa Vergara, Sergio V.*
Lenguaje de Flujograma para la Consulta en Base de Datos (Un Estudio Comparativo).
Quinta Conferencia Internacional UNISYS-UNAM,
Las Computadoras en las Instituciones de Educación y de Investigación, Noviembre, 1989.
53-69 pp.

[DATE] *Date, C. J.*
Introducción a los Sistemas de Bases de Datos.
Tr. Jaime Malpica. 3a. ed. México,
Sistemas Técnicos de Edición, S.A. de C.V., 1986
648 pp.

[DREYFUSS] *Dreyfuss, Henry.*
An Authoritative Guide to International Graphic Symbols.
1a. ed., New York U.S.A.,
McGraw-Hill Company
297 pp.

[FINKELSTEIN] *Finkelstein, Richard.*
Fourth Generation Language Data Bases.
Software Reviews
Computer Language
San Francisco Ca., U.S.A.
Vol. 4, No. 5, May 1987
pp. 89 - 122

[HERNANDEZ] *Hernández Stefanoni, Raúl* y *Chapa Vergara, Sergio V.*
Normalización de Base de Datos en C
Centro de Investigación y de Estudios Avanzados,
I.P.N., Departamento de Ingeniería Eléctrica,
México, D.F.
Reporte técnico No. 107, 1985
Serie Amarilla, Investigación
70 pp.

[HOROWITS] *Horowitz, Ellis; Kemper, Alfons* y *Narasimhan, Balaji.*
A Suvery of Application Generators
Revista IEEE Software, Enero; E.U.A.
IEEE ,1985
pp. 40-53

[IORSYTHE] *Iorsythe, Alexandra.*
Lenguajes de Diagramas de Flujo
1a. ed., México, 1983
Ed. LIMUSA, S.A.
588 pp.

[KRUGLINSKI] *Kruglinski, David.*
Sistemas de Administración de Base de Datos.
Tr. Sebastian Dormido Becomo, 1a. ed., México,
Libros McGraw-Hill de México, S.A. de C.V., 1984
282 pp.

[LUCAS] *Lucas Gibson, Michael.*
The CASE Philosophy
Revista Byte, Volumen 14, Número 4, E.U.A.
McGraw-Hill Inc. Abril 1989
pp. 209-218

[MARTIN] *Martin, James.*
Organización de Bases de Datos.
Tr. Adolfo Di Marco, 2a. ed., México,
Printice-Hall Hispanoamericana, S.A., 1977
544 pp.

[NEWMAN] *Newman, William M.* y *Sproull, Robert F.*
Principles of Interactive Computer Graphics.
2a. ed, New York U.S.A.,
International Student Editions,
McGraw-Hill Book Co., 1981
541 pp.

[ROGER] *Roger, David F.*
Procedural Elements for Computer Graphics.
1a. ed., New York U.S.A.,
International Student Editions,
McGraw-Hill Book Co., 1981
433 pp.

[TODD] *Todd, S. J. P.*
The Peterlee Relational Test Vehicle (PRTV)
a system overview
Revista IBM Systems No. 4, E.U.A.
International Business Machines Corporation, 1976
pp. 285-307

[ULLMAN82] *Ullman, Jeffrey D.*
Database Systems.
2a. ed., U.S.A.
Computer Science Press, 1982
484 pp.

[ULLMAN83] *Ullman, Jeffrey D.; Aho, Alfred V.* y *Hopcroft, John E.*
Data Structures and Algorithms.
1a. ed., U.S.A.
Addison Wesley Publishing Company, 1983
427 pp.

[VAUGHAN] *Vaughan-Nichols, Steven J.*
Relational Databases: The Real Story
Revista Byte, Volumen 15, Número 13; E.U.A.
McGraw-Hill Inc. Diciembre 1990
pp. 321-325