ABDO

ADMINISTRADOR

DE BASE DE DATOS

ORIENTADO A OBJETOS

** PROYECTO DE SOFTWARE **

Minetto Jorge
Sobredo Analia

Director: Lic. Gustavo Rossi

INDICE

INTRODUCCION

Este texto tiene como fin describir los pasos seguidos en nuestra investigación sobre el tema de BASE DE DATOS ORIENTADAS A OBJETOS. Nuestro objetivo final fue construir un prototipo con las características estudiadas.

El presente está organizado en cuatro capítulos. Luego enunciamos las conclusiones del proyecto realizado donde exponemos el estado actual del prototipo y qué aspectos se pueden mejorar. A continuación incluimos las referencias bibliográficas. Por último anexamos un apéndice con el manual del usuario donde instruimos sobre la utilización del prototipo. Dicho manual puede ser comprendido por lectores con conocimientos básicos de la filosofía de objetos, aún por aquellos que no hayan leído los capítulos previos. No obstante, se incluye un glosario de los términos más comunes en el paradigma de objetos.

En los siguientes párrafos describimos el contenido de cada capítulo.

En el CAPITULO I exponemos los temas investigados previamente a desarrollar nuestro prototipo. Investigamos las características de las bases de datos en general y de las orientadas a objetos en particular, las ventajas y desventajas de cada una. Estudiamos un modelo orientado a objeto óptimo, así como también algunos modelos de datos orientados a objetos existentes.

En el CAPITULO II definimos nuestro sistema, enunciando los fundamentos teóricos en que nos basamos para realizarlo. Describimos qué puede hacer un usuario que lo utilice. Introducimos conceptos generales a fin de que el

lector se familiarice con nuestro prototipo e incluimos consideraciones tenidas en cuenta al desarrarlo.

En el CAPITULO III explicamos cómo organizamos la información en el sistema (estructuras de datos utilizadas) y como está estructurado (modularización).

En el CAPITULO IV detallamos la implementación del sistema especificando las decisiones tomadas al diseñar cada operación. Completando el capitulo listamos un procedimiento para ejemplificar la etapa de implementación.

CAPITULO I

ESTUDIO PRELIMINAR Y ANALISIS DE MODELOS DE DATOS EXISTENTES

Sección I.1: Administradores de Base de Datos (DBMS)

I.1.1 - Conceptos generales

Un **sistema de información** es un sistema de computación para mantener y acceder un conjunto de información sobre algún aspecto del mundo real [ver Ref.21]. Tal sistema tiene una Base de Datos de hechos como núcleo, que es interrogada por usuarios a través de consultas y manipulada por programas de aplicación.

Los sistema de información son útiles, porque permiten a los usuarios responder preguntas sobre el mundo más convenientemente que realizando mediciones en el mundo real.

Un sistema de información puede ser visto como un modelo del mundo real.

Los objetivos mas sobresalientes de los lenguajes para describir sistemas de información son:

a) Hacer más fácil el diseño y mantenimiento de sistemas de información por medio de un vocabulario apropiado al dominio del problema.

b) Hacer más fácil el uso del sistema de información, ayudando al usuario a encontrar e interpretar el dato almacenado.

Un **modelo conceptual** es un sistema de información

en el cual se representa el mundo real en términos de entidades relacionadas mutuamente.

Para expresar la información en un modelo conceptual necesitamos un lenguaje. Tradicionalmente estos lenguajes se han concentrado en el modo en que los datos son almacenados y accedidos en la computadora mas que en los conceptos subordinados a ellos.

Los sistemas de manejo de bases de datos desarrollados en las últimas décadas, han enfatizado el procesamiento eficiente de grandes cantidades de datos. Basado en la estructuración de datos y en la facilidad de operación que ellos ofrecen, los sistemas manejadores de bases de datos se categorizan en uno de los tres modelos de datos clásicos: jerárquico, red o relacional. La mayoría de estos modelos están basados en registros; los dos últimos modelos mencionados además utilizan la noción de puntero.

Mucho se ha criticado la conveniencia de modelos de información basados en registros por las siguientes causas:

- Por la falta de estructuras básicas, la semántica resulta frágil al tratar de representar gran variedad de relaciones y entidades. Esto perjudica la interpretación de los datos.
- Si bien los registros, en un mundo particular representan claros modelos de información, resulta difícil su adaptación a consideraciones adicionales.
- Los registros normalmente requieren un identificador único, lo que suele ser problemático en la vida real, por lo que se establecen relaciones entre las claves de las entidades mas que entre las entidades mismas.

El complemento de estos sistemas de información basados en registros, son generalmente lenguajes standard de programación. Estos lenguajes han sido diseñados con el objetivo de expresar computaciones en general, y de ahí que provean pocos aspectos vinculados a la modelización de sistemas de información. Por otro lado estos lenguajes que sirven de soporte a los sistemas de información no cuentan con una estructura de datos integrada con la de los administradores de bases de datos; ésto fuerza una traducción que provoca ineficiencia e inconsistencia.

Algunos lenguajes experimentales como Pascal/R y Rigel han intentado integrar facilidades de bases de datos relacionales en lenguajes de programación tradicionales. No obstante están sujetos a los problemas mencionados en los sistemas basados en registro, fracasando al tratar de modelizar entidades conceptuales.

El diseño de los sistemas de base de datos tradicionales se ajusta a las necesidades de las aplicaciones comerciales. Con el surgimiento de otras áreas como la Inteligencia Artificial y la Ingeniería de Software se

necesitó extender las facilidades de las bases de datos para cubrir estos campos.

Uno de los problemas al desarrollar aplicaciones de base de datos es la impedance mismatch entre el lenguaje de manipulación de datos (DML) y el lenguaje de programación en el cual está escrita el resto de la aplicación. Ninguno de los dos lenguajes cubre los requerimientos de la aplicación completa: al DML le falta la parte computacional distinta del manejo de datos, mientras que en el lenguaje de programación no se pueden representar relaciones ni se provee persistencia adecuadamente.

I.1.2 - Rasgos de un manejador de base de datos (DBMS)

Lenguaje y modelo : un DBMS contiene estructuras de datos y provee un lenguaje para manipularlos. Los datos se crean, modifican y eliminan. Se proveen mecanismos de consulta.

Relaciones: un DBMS puede representar relaciones entre entidades; las relaciones pueden tener un nombre y se pueden consultar.

Persistencia: un DBMS provee almacenamiento estable y persistente. Persitente porque el dato es accesible después del fin del proceso que lo crea y estable porque el dato subsiste en caso de fallas de almacenamiento, del sistema o del proceso.

Sharing: un DBMS permite que los datos se compartan. Como distintos usuarios acceden a los mismos datos, existe asociado un mecanismo de control de concurrencia. Ante acciones inconsistentes, los datos son bloqueados.

Tamaño arbitrario : el espacio de direccionamiento de un DBMS no está restringido.

Restricciones de integridad: ayudan a asegurar la exactitud y consistencia de los datos. Por ejemplo, dominios para campos, claves para identificar items en una colección, restricciones para verificar que un item de dato referencia a un objeto existente.

Autorización: sirve para controlar el acceso a los datos. Generalmente los datos tienen un dueño, quien puede permitir el acceso a otros usuarios.

Consultas: provee un lenguaje de consulta declarativo y un soporte de acceso asociativo. Especialmente en los DBMS relacionales hay una facilidad de expresar las consultas o modificar la base de datos desentendiéndose de cómo se deriva la respuesta.

Esquema separado: la mayoría de los DBMS mantienen un esquema de información sobre la base de datos con los tipos definidos en la base de datos y objetos declarados con esos tipos. Este catálogo es compartido por todos los programas y usuarios.

Visiones: se usan para presentar los datos en forma más fácil de entender por el usuario. Se calculan a partir de datos almacenados y no se almacenan explícitamente.

Administración de la base de datos: se tiene una interfase especial para ejecutar funciones como reorganización de la estructura de los datos, cálculo de estadísticas sobre consultas más frecuentes, registro de información para auditoría, etc..

Sección I.2 : Administradores de Base de Datos Orientados a Objetos

I.2.1 - Terminología de Orientación a Objetos

Un **objeto** es un mecanismo abstracto que define un protocolo a través del cual los usuarios pueden interactuar. El objeto tiene un estado que es almacenado como una porción encapsulada de memoria.

El protocolo de un objeto se define por un conjunto de **mensajes**. Un mensaje puede enviarse a un objeto (receptor) para ejecutar alguna acción. El objeto interpreta el mensaje y lleva a cabo la acción correspondiente. El mensaje puede verse como una llamada a un procedimiento en el cual se distingue el receptor.

Un mensaje es interpretado por un **método** que es la parte codificable que lleva a cabo el efecto deseado. Así, un método sería el cuerpo del procedimiento enunciado en el

párrafo anterior. Un método es el único que puede acceder a la representación privada del objeto al cual el mensaje fue enviado. Un mensaje puede estar asociado a distintas implementaciones de métodos y el sistema selecciona, en tiempo de ejecución, el método correspondiente de acuerdo al tipo del receptor.

Una **clase**, a veces llamada **tipo**, es un molde para sus **instancias**. Generalmente, en los lenguajes donde se usan los dos términos, el tipo se refiere a la definición dada, mientras que la clase se refiere a todas las instancias del tipo correspondiente. Cada objeto es instancia de alguna clase. Esta clase define todos los mensajes a los cuales el objeto responde, como también la implementación del objeto.

Las clases están organizadas en un grafo dirigido conectando **superclases** a sus **subclases** mediante enlaces *is-a*. Estos expresan relaciones de compatibilidad entre las clases. Una clase **hereda** el comportamiento definido para sus superclases, además del que tiene como propio.

I.2.2 - Características de las Bases de Datos Orientadas a Objetos

Objetos Compartidos

El estado de un objeto está expresado por otros objetos. Un objeto puede referenciar a otro objeto. Es posible que un objeto sea referenciado por más de un objeto. Cualquier cambio en el objeto se refleja en los objetos que lo referencian. Una semántica adicional que puede ser agregada es la relación part - of, por la que operaciones sobre un objeto pueden tener efectos laterales en sus objetos componentes.

Tipos y Encapsulamiento

Un propósito de la base de datos orientada a objetos es proveer un sistema donde las primitivas básicas del modelo sean extensibles.

Un conjunto fijo de estructuras de datos primitivas no es suficiente para soportar nuevos diseños de aplicaciones. Se puede proveer la extensión a través de la creación de nuevos tipos [ver Ref. 13 y 14].

El principal uso de los tipos en los lenguajes de programación fue el chequeo de tipos, siendo discutido si dicho chequeo debía concretarse en tiempo de compilación o de ejecución.

El encapsulamiento es una técnica de estructuración en la cual un sistema está compuesto de módulos accesibles a

través de una interfase bién definida. En los tipos de datos abstractos dicha interfase está compuesta por un conjunto de operaciones fuertemente tipadas [ver Ref. 15]. Cada tipo define una representación para almacenar el estado del objeto, que es alocada con cada una de sus instancias. Dicho estado sólo puede ser modificado por los métodos, sin afectar al resto del sistema.

Algunos modelos orientados a objetos, no proveen una poderosa visión de encapsulamiento. En estos modelos, que han sido llamados Orientados a Objetos Estructuralmente, se pueden construír nuevos tipos pero la representación del tipo es visible.

Identidad

Los modelos de datos orientados a objetos se caracterizan en referenciar objetos a través de su identidad. Un objeto permanece invariante a través de todas las posibles modificaciones de su valor. Otros bases de datos anteriores y lenguajes de programación modernos, también se basan en esta noción. En cambio, en la corriente actual de bases de datos, los modelos son basados en valor, es decir, un objeto es identificado por un subconjunto de atributos (clave). Con el uso de esta última técnica, puede ser difícil expresar cierto tipo de información.

No hay operación que pueda cambiar la correspondencia entre un objeto y su identidad. La identidad persiste hasta tanto el objeto sea destruído.

La identidad permite expresar la noción de estructura compartida. Se distingue la idea de dos objetos idénticos de dos objetos que son, en efecto, el mismo.

El conjunto de referencias a objetos puede ser más extenso que el conjunto de objetos identificados.

Eliminar un objeto puede ocasionar referencias colgadas. Se podría evitar eliminar objetos si existen referencias desde otros objetos. Este tema es cuestionable en los modelos de bases de datos orientado a objetos.

Aunque las bases de datos orientadas a objetos se basan en la noción de identidad, la forma de relacionar objetos puede estar basada en valor.

La identidad de objetos es única en la base de datos, mientras que la clave es única en una relación simple. Por lo tanto, en el primer caso se pueden formar colecciones de objetos heterogéneos, lo que no sería posible en el caso de utilizar clave.

Para más detalles sobre este tema ver Ref. 16.

Herencia y Jerarquía de tipos

Los modelos Orientados a Objetos cuentan en general con un sistema de tipos, que incluye alguna forma de herencia.

La herencia es un mecanismo en el cual definiciones de tipo e implementaciones pueden ser relacionadas a otras a través de un orden parcial. Se pueden incrementar el conjunto

de tipos agregando tipos que sean subtipos de tipos existentes.

La literatura existente sobre tipos de herencia es muy variada, por lo que hay que tener sumo cuidado al comparar distintos modelos en este sentido.

Para más detalles ver Ref. 17.

Binding Dinámico

Un mensaje puede estar asociado a varias implementaciones de métodos en distintos tipos de la jerarquía. Cuando un mensaje es enviado a un objeto se selecciona, en tiempo de ejecución, el método correspondiente al tipo del receptor. En caso de no estar definido, se selecciona el de la superclase de menor nivel si la herencia es simple (la jerarquía de tipos es un árbol),o se selecciona mediante algún mecanismo de búsqueda, si la herencia es múltiple (la jerarquía de tipos es un DAG).

Otro tipo de enlaces, puede basarse en las distintas representaciones del tipo. Cuando un mensaje es enviado a un tipo, el sistema analizaría la representación y dispararía el método apropiado en tiempo de ejecución.

Existen otras formas de asociar métodos y mensajes, por ejemplo, asociarlos según el tipo de los argumentos del mensaje, pero es problemático debido al número total de casos que se tendrían que contemplar.

Polimorfismo

En los sistemas orientados a objetos se puede tener código que actúa sobre los objetos de distinta forma. Existe polimorfismo cuando un subtipo responde a un mensaje ejecutando un método definido en el supertipo, ignorando los campos propios del subtipo. Otra forma de polimorfismo existe cuando el mismo método puede ejecutarse basado en el tipo de sus argumentos.

I.2.3 - Modelo Optimo de un Administrador de Base de Datos Orientado a Objetos

Describiremos una serie de rasgos que deberán estar presentes en los DBMS Orientados a Objetos futuros.

a) proveer las funcionalidades de todo DBMS (ver I.1.2 en Sección 1).

b) soportar el concepto de identidad.

c) proveer alguna forma de encapsulamiento.

d) permitir objetos compartidos, es decir, permitir que un objeto sea referenciado por otros.

e) requiere que un objeto sea protegido del acceso directo, encapsulando las operaciones de los objetos. Un objeto puede ser una estructura de datos compuesta. Además se distinguen objetos de valores. Un objeto posee la capacidad de ser referenciado desde otros objetos mientras que un valor es propio de la representación del objeto.

f) la persistencia es alcanzada desde un objeto raíz persistente en la base de datos. Cualquier objeto que es alcanzado desde esta raíz es persistente. Los objetos de cualquier tipo pueden ser persistentes.
Para más detalles ver Ref. 18.

g) requiere que cada instancia de un objeto conozca su tipo. Cada variable y argumento en una definición de método tiene su tipo. La variable tiene un tipo declarado y un tipo inmediato (el tipo del objeto que es el valor corriente de la variable). El tipo inmediato es subtipo del tipo declarado (aquí tipo es subtipo de él mismo). Se requiere además que cada mensaje enviado a un objeto sea conocido por éste.

h) soporta tres agrupaciones de objetos distintas:
jerarquía de especificación: expresa consistencia entre especificaciones de tipos, es decir permite sustituir instancias de un subtipo en contextos donde se espera una instancia del supertipo.
- jerarquía de implementación de representaciones y métodos: permite compartir código entre tipos (reusabilidad).
- jerarquía de clasificación: decribe colecciones de objetos y las relaciones mantenidas entre esas colecciones. Los miembros de las colecciones pueden restringirse por un conjunto de predicados.

i) proveer polimorfismo a través de binding dinámico.

j) proveer constructores de tipos tales como sets, listas y arrays. Una colección es un modo de agregación de objetos relacionados. Las consultas siempre se realizan sobre algún tipo de colección. Para identificar objetos de una colección se utiliza el concepto de clave. El tipo de una colección está relacionado con el tipo de los objetos que contiene. Si dos colecciones comparten el mismo conjunto de mensajes pero los elementos que contienen son de distinto tipo, puede usarse la noción de parametrización de tipos.

k) permitir que los nombres de variables persistentes estén disponibles en un espacio de nombres estructurado. Esos nombres se usarán en expresiones de consulta. Una variable puede ser declarada de cualquier tipo y se puede tener cualquier número de variables del mismo

tipo. Los valores de una variable persistente perduran de una sesión a otra en la base de datos.

l) proveer un lenguaje de consulta de alto nivel. Las consultas pueden ser pensadas como expresiones sobre métodos especiales definidos sobre los tipos, que son colecciones de objetos. El encapsulamiento de la representación de un objeto sería un obstáculo para las consultas, pero existe un mecanismo, que forma parte del sistema, que puede acceder a la implementación del objeto, violando el encapsulamiento. Una forma de optimizar las consultas sería incorporando caminos de acceso auxiliares como índices. El índice es asociado a un conjunto dado y puede ser construido por cualquier mensaje que retorne un valor. Debe actualizarse cada vez que el conjunto sea modificado por una operación.

m) proveer relaciones. Una relación es una correspondencia entre objetos. El tipo de relación soportada es binaria y permite navegar en cualquiera de las dos direcciones posibles. La relación es un objeto privado ya que no se puede referenciar desde otro objeto que no sean los intervinientes en la relación.

n) proveer distintas versiones del estado de un objeto, registrando de esta forma la historia de este último. Un objeto es un conjunto parcialmente ordenado de versiones. De este conjunto se selecciona una única versión, que puede tener sucesores y predecesores. Estas versiones pueden ser alcanzadas mediante referencias estáticas o dinámicas. La primera apunta a la misma versión dentro del conjunto de versión en cualquier momento, mientras que la dinámica selecciona distintos objetos dependiendo del tiempo en que es accedido, de acuerdo a alguna estrategia (por ej. la mas reciente). Al agregar una nueva versión en un objeto dependiente de un objeto compuesto, se genera una nueva versión en este último.

Para más detalles ver Ref.19.

I.2.4 - Sistemas Orientados a Objetos VS Sistemas de Bases de Datos Relacionales

a) Vamos a ver que conceptos necesarios en una base de datos están en estudio en los sistemas orientados a objetos:

- Los modelos orientados a objetos no cuentan con la noción de conjunto como los modelos relacionales, ni proveen el álgebra basada en conjuntos (unión, selección).

La mayoría de los sistemas orientados a objetos no proveen persistencia. La única forma de mantener los datos de una sesión a otra, es usando archivos, lo que requiere una operación explícita de almacenamiento por parte del programador, complicando el código y degradando la performance.

Tampoco provee protección contra fallas de soft o hard, ni provee mecanismos para asegurar integridad de transacciones.

- No se permiten accesos concurrentes a los mismos datos, son sistemas monousuarios.
- Los sistemas orientados a objetos estan limitados por el espacio de direccionamiento. La manipulación de gran cantidad de datos degrada la performance.

b) Ahora vemos que ventajas tiene el paradigma de objetos sobre los sistemas relacionales.

- Se pueden almacenar y manejar objetos complejos, mientras que los sistemas relacionales para poder representarlos tendrían que contar con distintos niveles de estructura.
- Mediante la identidad de objetos se pueden modelizar objetos compartidos, sin necesidad de agregar una capa extra sobre el sistema.
- Agregando nuevos tipos o clases en el sistema, se puede extender su capacidad. Esto es importante para adaptar al sistema nuevas aplicaciones.
- Programas y datos se almacenan en el mismo sistema, a diferencia de los sistemas relacionales.
- En los sistemas relacionales no existe la noción de tipo. No se puede definir un tipo y afirmar que algunas relaciones son de ese tipo. Esto es posible en los sistemas orientados a objetos y junto con los conceptos de herencia, polimorfismo y binding dinámico representan una ventaja para los programadores.

c) Hay características de los sistemas relacionales que no deben modificarse:

- El conocimiento es más fácil de expresar pues existen pocos conceptos básicos con respecto al mundo de los objetos, donde además no existe un formalismo unificado para modelizar.
- Los lenguajes de consulta que favorecen el acceso a bases de datos provistos por los sistemas relacionales, no existen en los orientados a objetos pues las estructuras de datos son complejas, falta de un modelo común y porque se contraponen con el concepto de encapsulamiento.

- La declaratividad de las consultas de los sistemas relacionales se pierde en los orientados a objetos.

- La interface relacional, que no se cuenta en los sistemas orientados a objetos, sería bueno incorporarla.

La velocidad con que se tratan objetos en los sistemas orientados a objetos deberían poder competir con la del manejo de tuplas en los sistemas relacionales.

Para más detalles ver Ref. 20.

Sección I.3 : Modelos de Datos para Aplicaciones Orientados a Objetos

I.3.1 - Un Modelo de Datos para Aplicaciones Orientado a Objetos

Este modelo de datos fue pensado para el sistema de base de datos **ORION**. Trata de resolver temas como deleteo de objetos persistentes, cambio dinámico del esquema, objetos compuestos, versiones, que no fueron considerados por los sistemas orientados a objetos existentes.

Las clases están organizadas según un esquema reticulado: la herencia es múltiple. Hay conflicto de herencia con los nombres de variables de instancia y los métodos que se resuelve de la siguiente manera:

- tiene precedencia la clase sobre la superclase;
- tiene precedencia la primer superclase de la lista de superclases de la clase. Se puede cambiar explícitamente.

Orion tiene una clase Object como raíz de la jerarquía. Existe una clase Class a la que pertenecen todas las clases. Existe una clase Ptype que tiene las clases primitivas que pueden ser usadas como dominio. Existe una clase Setof para cada clase definida por el usuario o primitiva, que es el conjunto de todas las instancias de la clase. Enviando mensajes a esos conjuntos se obtiene información sobre los objetos.

Evolución del esquema

Se pueden realizar cambios de esquema dinámicamente

manteniendo los siguientes Invariantes:

i) Invariante del esquema reticulado: no hay nodos aislados; es un grafo acíclico dirigido, con una sóla raíz.

ii) Invariante de nombres distintos: las variables de instancia y métodos de una clase deben tener nombres distintos.

iii) Invariante de la Identidad: todas las variables de instancia y métodos tienen distinto origen.

iv) Invariante de la herencia completa: se heredan todas las variables y métodos de cada una de sus superclases, a menos que no respeten los invariantes ii) y iii).

v) Invariante de compatibilidad de dominio: el dominio de una variable de instancia de una clase debe ser el mismo o superclase del dominio de la variable con el mismo nombre en las superclases.

Los cambios permitidos son:

1 - Cambios al contenido de una clase.
- 1.1 - Cambios a una variable de instancia.
  - 1.1.1 Agregar
  - 1.1.2 Eliminar
  - 1.1.3 Cambiar nombre
  - 1.1.4 Cambiar dominio
  - 1.1.5 Cambiar herencia
  - 1.1.6 Cambiar valor por defecto
  - 1.1.7 Manipular valor compartido
    - 1.1.7.1 Agregar
    - 1.1.7.2 Cambiar
    - 1.1.7.3 Eliminar
- 1.2 - Cambios a un método
  - 1.2.1 Agregar
  - 1.2.2 Eliminar
  - 1.2.3 Cambiar nombre
  - 1.2.4 Cambiar código
  - 1.2.5 Cambiar herencia

2 - Cambios del esquema.
- 2.1 - Hacer una clase superclase de otra
- 2.2 - Eliminar una superclase de una clase
- 2.3 - Cambiar el orden de superclases de una clase

3 - Cambios de un nodo.
- 3.1 - Agregar una clase
- 3.2 - Eliminar una clase
- 3.3 - Cambiar nombre a una clase

Veremos algunas consideraciones tenidas en cuenta ante una modificación en el esquema.

- Si no se especifica una superclase al crear una

clase, se asigna Object como tal; luego pueden agregarse superclases, con lo cual Object dejará de serlo.

- Si se agrega una variable de instancia que existía como heredada, la nueva definición la anulará.

- Si se elimina una clase, sus instancias se perderán y sus subclases heredarán variables y métodos de las superclases de la clase eliminada. Pueden quedar referencias colgadas. Si la clase eliminada era dominio de alguna variable de instancia de otra clase, ésta tomará como dominio la primer superclase de la lista de superclases de la clase eliminada.

Si se elimina una variable de instancia, podrá heredarse desde alguna superclase. Puede haber métodos que referencien variables inválidas si la variable eliminada no se hereda, por lo que habrá que redefinirlos o eliminarlos.

- Se puede cambiar el dominio de una variable de instancia sólo para generalizarlo.

Para más detalles sobre la semántica de modificación del esquema ver Ref. 1.

Objetos compuestos

Orion provee el manejo de objetos compuestos para establecer la relacion is-part-of que no está presente en otros modelos de datos orientados a objetos. Un objeto compuesto es una coleccion de objetos (dependientes o independientes). Un objeto dependiente es aquel cuya existencia depende de la existencia de otro objeto, llamado dueño y es único. Un objeto componente puede a su vez, ser compuesto. El objeto compuesto tiene una raíz; accediendo a la raíz, se puede acceder a todos los objetos componentes (un objeto compuesto es una unidad de encapsulamiento para Orion). En un objeto compuesto hay variables de instancia que sirven como enlace con el objeto componente (enlaces compuestos). Dicho enlace es unico para cada objeto componente, es decir, cualquier otra referencia a ese objeto desde otro, no será mediante un enlace compuesto.

Una vez creado el objeto dependiente no puede tener existencia propia independiente. Si se elimina una instancia de un objeto compuesto, se eliminan todas las instancias de sus objetos dependientes recursivamente.

En Orion se aloca un segmento por cada clase donde se almacenan sus instancias. Para el caso de los objetos compuestos, es conveniente almacenar instancias de distintas clases en el mismo segmento.

Para profundizar el estudio de objetos compuestos ver Ref. 2, 3 y 4 .

Versiones

En algunas aplicaciones es útil experimentar con múltiples versiones de un objeto antes de seleccionar uno definitivamente.

Existen dos tipos de versiones:

- Transitoria: el usuario que la creó puede modificarla o deletearla. Además una nueva versión transitoria puede derivarse de otra transitoria que pasa a ser versión de trabajo.
- De trabajo: es estable y no puede ser modificada. Puede ser deleteada por su dueño. Puede ser promovida explícitamente por el usuario. Una versión transitoria puede derivarse a partir de ella.

Los objetos versionados se relacionan mediante binding estático o dinámico [ver Ref.2 y 6]. En el estático hace falta explicitar el número de versión del objeto referenciado, mientras que en dinámico el sistema lo asigna por defecto. En ambos casos se necesita el nombre del objeto y su identificador.

En un esquema de versiones lineal (de una versión sólo puede derivarse una versión [ver Ref.5]), el sistema elegiría siempre la más reciente (la última en crearse). Pero como de una versión pueden derivarse varias, es necesario que el usuario especifique el número de versión por defecto.

Para implementar versiones, cada vez que se crea una instancia de una clase versionable, se crea un objeto genérico con el identificador del objeto, número de versión por defecto, un contador de versión y un conjunto de descriptores que referencian las versiones existentes del objeto. Cada versión del objeto cuenta con información del objeto genérico, número de la versión y estado de la versión (transitoria o de trabajo). Con esta información alcanza para recorrer el esquema de versiones, generar nuevas y seleccionar la más conveniente (implícita o explícitamente).

Para más detalles sobre versiones, ver Ref. 2, 5 y 6. Para profundizar el estudio de este modelo ver Ref. 7.

I.3.2 - Modificación de Clases en GEMSTONE (Sistema Manejador de Base de Datos Orientado a Objetos)

El sistema GEMSTONE integra conceptos de lenguajes orientados a objetos con propiedades de los sistemas de bases de datos. El lenguaje utilizado es Opal, similar al

Smalltalk.

Hay dos metodologías a seguir para la modificación de clases:

1) Depuración: Los cambios en los objetos no se realizan hasta tanto éstos no sean referenciados. Las modificaciones, de esta manera, pueden diferirse indefinidamente.

2) Conversión: Todas las instancias de la clase se adaptan a la nueva definición de la clase en el momento de producirse la modificación.

Gemstone adopta la conversión para hacer frente a la modificación.

Soporta cuatro tipos de almacenamiento básicos: a) self-identifying (pór ej. character, boolean, etc), b) byte (por ej. string), c) pointer y d) non-sequenciable-collection (por ej. set).

Cada objeto pertenece a un segmento (unidad de autorización). Cada segmento tiene un propietario que puede leer y escribir sobre él, y puede permitir la lectura y escritura a otros usuarios. Tener habilitación para acceder a un objeto no implica tenerla sobre los objetos que éste referencia.

En cuanto a la concurrencia, el sistema Gemstone chequea los accesos conflictivos en su momento. En una sesión de usuario, durante una transacción, la base de datos se mantiene consistente hasta su finalización, es decir, cualquier cambio que afecte a la base de datos se reflejará en otra sesión de usuario recién cuando finaliza la transacción modificadora [ver Ref. 8].

Invariantes de la modificación de clases

i) **Invariante de representación:** La representación de un objeto es determinada por su clase.

ii) **Invariante de la jerarquía de clases:** La jerarquía de clases esta dada por un árbol. No existen componentes desconectados y cada clase tiene una única superclase, excepto la raíz (clase Object).

iii) **Invariante de la herencia completa:** La representación de instancias determinada por una clase, es heredada por todas sus subclases. Toda clase hereda toda variable de instancia definida en su superclase.

iv) **Invariante del dominio:** El dominio de una variable heredada debe ser el mismo o ser subclase del dominio en la superclase.

v) **Invariante de las referencias colgadas:** No hay referencias que apunten a objetos inexistentes. El objeto se mantiene mientras sea referenciado desde otro objeto.

vi) **Invariante de pérdida de información:** Sin autorización no se puede deletear un objeto. Como clases y objetos son compartidos por distintos usuarios, cada vez que se intenta deletear un objeto hay que enviar un reporte a

todos los objetos involucrados.

Operaciones de Modificación de Clases

La siguiente no es una lista completa de las operaciones que provee Gemstone.

Renombrar una variable de instancia

Se le puede cambiar el nombre a una variable de instancia, pero para preservar el invariante de representación el nuevo nombre no debe estar definido en la clase. Si la variable es heredada no se puede renombrar (invariante de herencia). El nuevo nombre se propaga a las subclases.

Agregar una variable de instancia

Se puede agregar una variable de instancia a una clase; en cada instancia de la clase la nueva variable tomará el valor nil por el invariante de representación. La nueva variable se propaga a las subclases. La operación no se permite si el nombre de la variable ya existía en la clase (invariante de representación), o si el nombre de la variable ya existía en alguna subclase de la clase donde fué agregada (violaría el invariante del dominio).

Eliminar una variable de instancia

Cualquier variable de instancia de la clase puede ser removida y todas las instancias de la clase deben ser adaptadas en ese momento. Esta modificación no se propaga a las subclases. Para preservar el invariante de herencia completa, una variable de instancia no puede eliminarse si es heredada de una superclase.

Modificar el dominio de una variable

Un dominio puede especializarse cuando el nuevo dominio es subclase del anterior; generalizarse cuando el nuevo dominio es superclase del anterior. En otro caso el dominio se puede modificar eliminando la variable e incorporándola con el nuevo dominio.

La modificacion no se propaga a las subclases de la clase modificada.

Un dominio no puede generalizarse si la variable es heredada y el nuevo dominio no es subclase del dominio heredado. Las instancias de la clase no necesitan ser modificadas. Asimismo un dominio no puede especializarse si en alguna subclase de la clase modificada, el dominio de la variable de instancia no es subclase del nuevo dominio. Para preservar el invariante de representación, aquellas instancias donde el valor de la variable no satisfaga el nuevo dominio serán reemplazadas por nil.

Eliminar una clase

No puede eliminarse una clase si tiene alguna

instancia. Para poder sacar una clase, habría que pasar todas sus instancias a una subclase de la misma. Las variables de instancias desconocidas en la subclase son eliminadas. Si la clase sirve como dominio en otras clases, el nuevo dominio pasará a ser la superclase de la clase eliminada (invariante de referencias colgadas).

Cuando se elimina la clase se marca como obsoleta, para prohibir la creación de nuevas instancias a partir de ella.

Agregar una clase

Una nueva clase puede ser agregada internamente en la jerarquía de árbol, indicando nombre de la clase y especificando la superclase y sus subclases (deben ser subclases inmediatas de la superclase). No se agregan variables ni se realizan cambios de dominio; este efecto puede lograrse explícitamente con las operaciones correspondientes.

Hay otros factores que influyen en la modificación de clases.

Sharing: El mecanismo de control de concurrencia de Gemstone no es adecuado para el manejo apropiado de modificación de clases.

Comportamiento: Cuando es enviado un mensaje a un objeto, el método a ejecutarse lo determina su clase o superclase.

Los invariantes vistos aseguran una estructura de objetos consistente con la jerarquía de clases, pero no con respecto a los métodos. Puede ser que al sacar una variable de instancia haya métodos que la usen o que en un comportamiento compartido modificar un método que es usado por otro método se tengan consecuencias no esperadas. Por otro lado si se cambiara un método de una clase, los emisores de mensajes que desencadenan el método podrían no estar al tanto de la modificación. [ver Ref. 9 y 10].

Enumerar todas las instancia de la clase: Para actualizar las instancias de una clase ante una modificación en esta última se puede mantener una lista de las instancias (ocupa tiempo y espacio adicional), o bien, recorrer la base completa para ubicar las instancias.

Agrupar modificaciones: Conviene agrupar modificaciones para evitar accesos redundantes a objetos. Por ej. para agregar varias variables de instancias, sería conveniente reescribir cada instancia de la clase una sola vez.

Binding: En Gemstone los mensajes son ligados a los métodos dinámicamente, por otro lado, las variables de instancia son ligadas estáticamente. Ante una modificación de clases el binding dinámico se adapta al nuevo ambiente, no así el binding estático, quien estará referenciando el entorno anterior. Detectar problemas con binding dinámico (como por

ej. el provocado al eliminar un método de la clase) es muy complicado y no puede ser estáticamente anticipado.

Comparación entre Gemstone y Orion

- El esquema de clases en Gemstone está representado por un árbol mientras que en Orion es reticulado, de aquí que la herencia sea múltiple a diferencia del primero (la herencia es simple).
- En Orion se necesitan más reglas para mantener los invariantes que hacen al sistema consistente, debido a la estructura de clases compleja.
- Al deletear una clase, se deletean todas las instancias de ella. Pueden existir referencias a éstas desde otros objetos de otras clases. En Orion, estas referencias quedan colgadas hasta tanto sean accedidas y corregidas. En Gemstone, se debe recorrer toda la base de datos para que esto no suceda.

Para más detalles sobre Gemstone ver Ref. 11.

I.3.3 - GALILEO: Un Lenguaje de Programación de Base de Datos Orientado a Objetos Fuertemente Tipado e Interactivo

Galileo es un lenguaje de programación de base de datos que cuenta con las características de la orientación a objetos,es fuertemente tipado y reúne también los siguientes rasgos:

Programación funcional: es un lenguaje de alcance estático. Las clases pueden ser pasadas como parámetros,retornadas como valores y formar parte de estructuras de datos.

- **Lenguaje de consulta:** es interactivo. No es necesario un lenguaje de consulta adicional.
- **Chequeo de tipos estático:** es fuertemente tipado.El sistema de tipos, garantiza que no se produzcan errores de tipos en tiempo de ejecución.

Sistema de tipos flexibles: Se proveen tipos de datos y constructores para definir nuevos tipos. Definiendo tipos abstractos de datos se puede restringir el conjunto de valores posibles y heredar las operaciones primitivas del tipo.

Jerarquía de tipos: existe el concepto de generalización. Las operaciones definidas para un cierto tipo pueden tener como argumentos a valores de un subtipo.

- **Modelización de base de datos:** soporta los

mecanismos de clasificación, agregación y generalización para modelizar objetos de estructura y complejidad arbitrarias. Los objetos pueden tener otros objetos como componentes.

- **Manejo de fallas:** provee manejo de excepciones.
- **Interfases gráficas:** mediante ventanas, proveen entrada y despliegan valores de datos abstractos. El sistema automáticamente genera las primitivas gráficas a partir de la definición de tipos de datos abstractos.
- **Persistencia de valores:** todos los valores son persistentes sin distinción de tipos.

Sistema de tipos

Los siguientes son los rasgos básicos del sistema de tipos:

- Hay tipos básicos y constructores de tipos, que se usan para definir tipos concretos, cuyos valores no pueden cambiarse. Los tipos concretos se usan para abreviar los tipos que representan.
- El usuario puede definir tipos de datos abstractos junto con las operaciones para crear y manipular sus valores. La equivalencia de tipos es por nombre.
- Los tipos de datos abstractos pueden tener definidas operaciones. Los subtipos pueden heredar operaciones, pero con sus propios valores. Esto es útil para no mezclar valores de distintos tipos en operaciones.

Los valores de tipos de datos abstractos pueden restringirse con aserciones. Las aseciones son mantenidas automáticamente por los operadores heredados desde la representación del tipo. Cuando una aserción es violada se bifurca a un manejador de excepciones.

- Hay noción de tipo y subtipo. La jerarquía es un grafo acíclico dirigido. La relación de subtipo para los tipos abstractos debe ser explícitamente definida por el usuario. Las aserciones se heredan a los subtipos.

Mecanismo de abstracción para objetos

Un lenguaje de programación orientado a objetos tiene las siguientes características:

i) Un programa es una colección de objetos, los cuales interactúan a través de la activación de métodos.

ii) Un programa está definido por un conjunto de clases, donde se especifican sus métodos de instancia, por la instanciación de uno o más objetos y por los mensajes.

iii) Las clases están agrupadas en jerarquías.

Dichas características proveen algunos beneficios:

a) Los programas se desarrollan primero para casos generales y luego se pueden agregar clases y refinar métodos.

b) Agregando subclases no siempre es necesario redefinir métodos (economía de software).

c) Se pueden agregar nuevos objetos con sus propios métodos sin tener necesidad de modificar el programa (reusabilidad).

Se pueden definir atributos especiales para especificar el estado interno de un objeto. Estos son: modificable (el valor inicial puede ser modificado); default (especifica un valor standar para todas las instancias del tipo); derivado (especifica un valor obtenido de una expresión evaluada cada vez que el atributo es seleccionado); oculto (no puede ser usado fuera del objeto).

Modelización de bases de datos

Una clase es una colección de todos los objetos de un tipo de datos abstracto. Un elemento de una clase es un objeto que es la representación computacional de ciertos hechos sobre la entidad del mundo que estamos modelizando. En esta visión orientada a objetos, la relación entre objetos de la base de datos y entidades modelizadas es uno a uno. Cuando se crea un elemento de un cierto tipo, automáticamente pasa a pertenecer a la clase correspondiente hasta tanto sea removido explícitamente.

Se pueden definir subclases, que son formas alternativas de ver las mismas entidades. Cuando se crea un objeto es automáticamente insertado en la clase y en todas sus superclases.

Los objetos modelizan entidades del mundo real; la relación entre entidades es a través de los atributos.

Cada clase tiene una primitiva gráfica que devuelve todos los elementos de una clase que satisfacen una cierta condición.

Metadatabase

Se tiene un diccionario donde el usuario puede consultar y recuperar información de los datos almacenados. La idea es que exista un conjunto predefinido de clases que describa la información principal de las clases, tipos y valores de la base de datos. Puede consultarse a través de interfases gráficas o mediante el lenguaje. Sólo pueden modificarse los comentarios.

Persistencia de valores

Las dos ideas que caracterizan la persistencia son que todo tipo de dato puede ser persistente y que el programador puede ver todo dato como persistente.

Los principales problemas para implementar persistencia en este lenguaje son:

— Problema del formato: se trata de optimizar el uso de los diferentes dispositivos de almacenamiento temporario y persistente. Si los datos tienen una estructura regular, se puede pensar alguna forma conveniente para representarlos, pero si son heterogéneos, no hay un esquema simple para representarlos. La mejor aproximación sería representar los datos temporaria, o persistentemente de forma similar, para optimizar al menos el tiempo de transferencia.

— Problemas de recuperación: cada vez que un grupo de objetos es modificado, un nuevo conjunto de páginas es alocado para contener su nueva versión una vez que la operación es completada exitosamente. Si se produce una falla antes de ésto, queda la versión anterior.

— Problema de liberación de espacio: para no recuperar objetos que no están accesibles, se debería buscarlos y liberarlos. Una buena estrategia sería almacenar datos estables en un espacio grande y pocas veces frecuentado y otro más pequeño con datos nuevos y dinámicos. Esto evitaría hacer búsquedas en espacios grandes.

Para más detalles sobre Galileo ver Ref. 12.

CAPITULO II

FUNDAMENTOS DE NUESTRO PROYECTO

Sección II.1 :Qué es el ABDO ?

El ABDO es un prototipo de un manejador de Base de Datos Orientado a Objetos. Con él intentamos cubrir algunos rasgos requeridos para los modelos de base de datos orientados a objetos, vistos en el capítulo anterior.

El objetivo de nuestro desarrollo fue crear un sistema que hiciera fácil el diseño, mantenimiento y recuperación de la información de la base de datos.

El usuario de este sistema podrá trabajar en una aplicación determinada: crear clases, instancias de esas clases (a las que llamamos objetos), realizar modificaciones sobre esas clases y objetos y efectuar consultas sobre éstas.

El énfasis de nuestra investigación estuvo puesto en mantener la consistencia del sistema, evitando que las numerosas operaciones provistas violaran los invariantes propuestos, que serán enunciados más adelante en esta sección.

Para cumplir con nuestro objetivo nos hemos encontrado con la falta de un modelo común, de fundamentos formales en los que respaldar nuestro trabajo, y de una actividad experimental importante.

Algunos rasgos que sostienen nuestro trabajo son los siguientes puntos:

- ***Los objetos en el modelo se corresponden con entidades del mundo real.***

La misma entidad del mundo real no es representada

por más de un objeto en el modelo. Recíprocamente, un objeto no puede representar a más de una entidad. Es un principio básico del diseño de base de datos.

Se establecen relaciones entre objetos.

Una consecuencia de ésto es prevenir que una entidad sea removida mientras participe de alguna relación, evitando así, tener referencias colgadas.

- ***Expresar las relaciones como propiedades de los objetos.***

Las propiedades (atributos de la clase) describen el objeto. Siguiendo cadenas de propiedades de un objeto se pueden obtener objetos relacionados.

- ***Los objetos están agrupados en clases.***

Todo objeto es instancia de una clase.

Las clases están organizadas en jerarquías.

Las clases están relacionadas unas con otras, formando una jerarquía reticulada. Esta organiza las descripciones de clases y frecuentemente elimina duplicaciones innecesarias. Esta herencia de atributos de una clase a sus especializaciones, es útil para abreviar descripciones.

- ***Se establecen restricciones.***

Es el caso de señalar un atributo como identificador único, restringiendo que sus valores sean todos distintos. Por otro lado, los atributos tienen clases definidas como sus tipos, así se restringen sus valores a objetos de esas clases.

Estos rasgos colaboraron para cumplir con el objetivo inicial, proveyendo las siguientes ventajas:

- **Tratamiento de objetos complejos.**

- **Extensibilidad:** agregando nuevas clases en el sistema, su capacidad se extiende fácilmente.

- **Disminución de la brecha entre los problemas reales y sus modelos.**

- **Reusabilidad de código.**

Para qué sirve el ABDO?

El prototipo desarrollado permite agregar clases como hojas en la jerarquía especializando clases existentes. Al crear una clase se definen sus componentes (atributos) y

sus superclases. La clase hereda los atributos definidos en sus superclases que no existan en su propia representación. La herencia es múltiple, pues un atributo puede heredarse de más de una superclase.

Cada clase puede verse como un molde para crear objetos. La representación del objeto se corresponde con la estructura definida para la clase. Cualquier cambio que afecte la estructura de la clase se reflejará en el estado de sus instancias.

Se proveen operaciones para agregar, eliminar atributos a una clase, cambiar algunas características del atributo, como valor por defecto o compartido (si posee), tipo, herencia. Se puede cambiar el atributo clave de una clase.

En la jerarquía reticulada se pueden agregar o eliminar clases, como también superclases a una clase. Se permite cambiar el nombre a una clase.

En cuanto a los objetos existentes de una clase, se pueden eliminar o modificar.

Toda la información almacenada en la base de datos (clases, subclases, objetos), puede ser consultada por el usuario de la aplicación.

Todas las operaciones que modifican el esquema deben mantener el sistema consistente, es decir se deben cumplir los siguientes invariantes, antes y después de ejecutada la operación:

1 - Toda clase tiene al menos una superclases. No hay clases aisladas.

2 - Todas las clases tienen nombres distintos.

3 Todos los atributos de la clase tienen nombres distintos.

4 - Un atributo se hereda en una clase, a menos que esté definido como propio en la misma.

5 - El tipo de un atributo en cualquier clase debe ser subclase del tipo de todo atributo con ese nombre en las superclases de la clase tratada o debe ser el mismo tipo.

6 - Toda clase debe tener un atributo especificado como clave.

7 - Los valores de los atributos claves son únicos (no se aceptan claves duplicadas).

8 No hay referencias que apunten a objetos inexistentes.

A raíz de ésto, ciertas operaciones provistas en el prototipo propagan sus efectos a las subclases de la clase involucrada en la operación. De esta forma, una operación podría acarrear consecuencias no esperadas por el usuario a los niveles inferiores de la jerarquía. Trataremos este tema en detalle al describir la implementación de las operaciones [Capítulo IV].

No fue el objetivo de nuestro desarrollo incluír la

implementación de métodos, por lo cual hemos obviado lo referente a ellos.

Sección II.2 :Conceptos Utilizados en el desarrollo del ABDO

En esta sección detallaremos algunos de los términos que se usaron en el desarrollo del prototipo:

* **Clase.** Es un conjunto de atributos a partir del cual pueden instanciarse los objetos.

* **Subclase** de una clase. Es una especialización de la clase; sólo se necesita especificar los atributos que especializan la definición de la clase. El resto se hereda en la subclase. La clase más general se llama **Superclase.**

* **Herencia.** Es la propiedad por la cual una clase puede utilizar atributos definidos en alguna superclase de ella. Como una clase puede tener más de una superclase, decimos que nuestra herencia es **múltiple.**

* **Conflicto de herencia.** Como la herencia es múltiple, un atributo puede ser heredado de más de una superclase, con lo cual se origina una situación que llamamos conflicto de herencia. Para resolverlo se pide la intervención del usuario.

* **Jerarquía de clases / Esquema reticulado.** Las clases se encuentran agrupadas en una jerarquía. Como las clases puede tener varias superclases, decimos que están organizadas en un esquema reticulado.

* **Clase RAIZ.** Es la clase que está en el nivel más alto de la jerarquía. Es superclase de todas las clases. Como en el sistema no se permiten clases aisladas, RAIZ se asigna como superclase inmediata a toda clase que no tenga superclases especificadas.

* **Superclase inmediata.** Una clase puede tener varias superclases. Llamamos superclases inmediatas a aquéllas que son superclases de la clase, sin clases intermedias. Ejemplo: la clase *Ayudante* es subclase de la clase *Estudiante*, y ésta es subclase de la clase *Persona*. Aunque *Estudiante* y

Persona son superclases de *Ayudante*, sólo *Estudiante* es superclase inmediata.

* **Tipo.** Todo atributo de una clase debe tener un tipo asignado, que coincide con una clase definida en el sistema. El valor de ese atributo deberá ser un objeto (o una referencia a un objeto) correspondiente a la clase que tiene como tipo.

El tipo puede ser una clase definida por el usuario o una de las clases provistas por el sistema (ver Sección II.3 Tipos Predefinidos). Estos dos grupos conforman el conjunto total de tipos existentes en la aplicación.

* **Restricción de tipos.** Llamamos así a la relación que existe entre el tipo de un atributo con el tipo del atributo con el mismo nombre en la superclase. En toda clase, el tipo de un atributo debe ser subclase del tipo del atributo con el mismo nombre (si existe) en las superclases.

* **Atributo propio.** Denominamos así a cada atributo que el usuario define cuando crea una clase.

* **Atributo heredado.** Todo atributo correspondiente a una superclase, que se hereda en una clase por no estar definido como propio en ella, lo denominamos atributo heredado.

* **Vía** de un atributo heredado. Identifica la clase de donde se hereda el atributo. No siempre coincide con la clase a la que pertenece el atributo, pues la clase vía también puede tener ese atributo como heredado.

* **Objeto.** Es una instancia de una clase, es decir un conjunto de valores asociados a cada atributo de la clase.

* **Objeto compuesto.** Es un objeto que referencia a otros objetos, que son instancias de clases definidas por el usuario.

* **Referencia.** Es el valor de un atributo dentro de un objeto, cuyo tipo es una clase definida por el usuario. Aquéllos objetos que tengan al menos un valor referencia son compuestos.

* **Valor nulo.** Es el valor que contiene un atributo al que no se le ha asignado uno explícitamente.

* **Valor por Defecto.** Es aquél que se asigna a un atributo en el momento de instanciar la clase si no se especifica otro valor explícitamente. Forma parte de la definición del atributo en la clase.

* **Valor Compartido.** Es aquél que se asigna a un

atributo en el momento de instanciar la clase. Todos los objetos de la clase tendrán ese valor para ese atributo. Forma parte de la definición del atributo en la clase.

* **Conversión.** Definimos así a la acción de adaptar instancias de una clase determinada a instancias de su superclase. El término se aplica sólo a clases definiidas por el usuario.

* **Propagación a subclases.** Es la acción de adaptar las subclases de una clase como consecuencia de las modificaciones sufridas por la clase. Los cambios se deben propagar a niveles inferiores del esquema reticulado para mantener la consistencia del sistema.

* **Clave.** Toda clase tiene un atributo que la identifica, y que luego de instanciado, identifica los objetos de la clase. Es el atributo clave. Mediante el valor de este atributo pueden referenciarse objetos desde objetos de otras clases.

* **Aplicación.** El usuario puede definir aplicaciones para trabajar independientemente de otras que se puedan definir. Cada aplicación tendrá asociada su propia jerarquía de clases, de modo que dos aplicaciones distintas podrían tener un mismo nombre de clase definida en ambas.

* **Sesión de trabajo.** Llamamos así al intervalo comprendido entre el ingreso a una aplicación y la salida de la misma.

Sección II.3 :Consideraciones de diseño del ABDO

En esta sección describimos someramente algunas decisiones que hemos tomado durante el desarrollo del ABDO, que completan la definición del prototipo.

* Contamos con un grupo de **tipos predefinidos** provistos por el sistema que están disponibles en cada aplicación.

Entre los tipos predefinidos existe una relación de *clase - subclase*:

* STRING INTEGER
* STRING - REAL
* REAL - INTEGER

El tipo BOOLEAN no es subclase de ningún otro.

* Permitiremos un sólo atributo **clave** por clase. Es obligatorio, pues se necesita un descriptor para los objetos; mediante el valor de la clave pueden referenciarse los objetos desde otros objetos.

* Las **superclases** asignadas a una clase no pueden guardar relación de subclase o superclase entre ellas, para evitar ciclos.

* El **tipo** de un atributo de una clase debe ser subclase del tipo del atributo con el mismo nombre en las superclases (inmediatas o de niveles superiores). Además debe ser superclase del tipo de los atributos propios que tengan el mismo nombre en cada una de las subclases.

* Una instancia de una clase no puede tener **referencias colgadas**, es decir cuando se asigna un valor a un atributo cuyo tipo es definido por el usuario, dicho valor debe hacer referencia a un objeto existente. Si el objeto no existiese, se permite asignarle un valor nulo hasta tanto se cree el objeto.

Al crear un objeto, puede asignarse valor nulo a cualquier atributo excepto a la clave.

* Como soportamos **herencia múltiple**, ante un conflicto de herencia, se brinda al usuario la posibilidad de elegir aquélla que le convenga.

* Los cambios que se produzcan en una clase, provocarán que se adapten las instancias existentes de las clases involucradas a la nueva definición.

Tratamos de no perder valores de atributos en los objetos siempre que sea posible. Por ésto, muchas veces debemos **convertir** valores de atributos de un cierto tipo a valores de atributos de otro tipo (que debe ser superclase del anterior).

Explicaremos la conversión basándonos en el siguiente **ejemplo**:

Supongamos que tenemos la clase *persona* y la clase *estudiante* como subclase de la anterior.

Si en una clase, tenemos un atributo del tipo

estudiante, en sus instancias los valores para ese atributo son referencias a objetos del tipo *estudiante*.

Si cambiamos el tipo de tal atributo por *persona*, tendremos que convertir sus valores, en todas las instancias, a referencias de objetos del tipo *persona*.

Para los tipos predefinidos, la conversión no es necesaria, ya que los valores de estos tipos se adaptan. Por ejemplo, un valor INTEGER sigue siendo un valor REAL o STRING.

Sección II.4 : Funcionalidades del ABDO

En esta sección enunciamos las funcionalidades provistas por el prototipo, que indican las acciones que se le permiten seguir al usuario que ingrese al sistema.

a) Crear una aplicación.
Es dar de alta una aplicación en el sistema.

b) Editar / Modificar una aplicación.
Es trabajar con una aplicación, es decir, crear y modificar clases, crear, editar y eliminar objetos, consultar clases y objetos.

c) Eliminar una aplicación.
Es dar de baja cada una de las clases, con sus objetos, de la aplicación.

d) Listar aplicaciones.
Es visualizar todas las aplicaciones que existen en el sistema.

e) Crear una clase.
Es dar de alta una clase en la aplicación. Aquí se definen las superclases y los atributos que le corresponden.

f) Modificar el contenido de una clase.
Este tipo de cambios afecta a los atributos que componen la definición de la clase.

- Alta: es definir un nuevo atributo para la clase.

- Baja: es eliminar un atributo de la clase.

  Cambiar tipo: se conserva el nombre del atributo pero se modifica su tipo.

- Cambiar herencia: es modificar la clase de donde se decide heredar el atributo.

  Cambiar el atributo clave: se permite especificar un nuevo atributo como clave de la clase.

- Cambiar valor por defecto: implica modificar el valor a un atributo definido con valor por defecto.

  Cambiar el valor compartido: implica modificar el valor a un atributo definido con valor compartido.

- Asignar valor compartido: es extender la definición del atributo, agregándole un valor que será compartido por todos los objetos.

  Desasignar valor compartido: es eliminar el valor compartido a un atributo, quién se comportará como cualquier otro atributo luego de esta operación.

g) Cambiar la estructura jerárquica de las clases.
Se permite agregar o eliminar una superclase a una clase determinada de la jerarquía.

h) Modificar una clase de la jerarquía.
Este tipo de modificación incluye dar de baja una clase del esquema o cambiar su nombre.

i) Crear un objeto.
Es instanciar una clase de la aplicación. Aquí se asignan valores a cada uno de los

atributos que componen la definición de la clase.

j) Editar objetos.
Se pueden editar los objetos permitiendo modificar los valores de sus atributos.

k) Eliminar objetos.
Se permite dar de baja a un objeto de una clase que no esté referenciado desde otro objeto en alguna otra clase.

l) Consultar clases.
Se permite visualizar por pantalla o imprimir información sobre las clases de una aplicación. Esto incluye:

Consultar todas las clases de la aplicación.

Consultar la estructura interna de una clase (atributos y superclases).

Consultar la relación clase subclase, entre las clases de la aplicación.

m) Consultar objetos.
Se pueden consultar los valores de los atributos que componen un objeto. Cuando el valor es una referencia a otro objeto (objeto compuesto), se puede visualizar el objeto componente.
Se pueden seleccionar los atributos a ser consultados y los objetos que le interesen (especificando condiciones).
Además los objetos pueden ser consultados en forma ordenada de acuerdo a un atributo.
Se pueden obtener cálculos de suma y promedio para ciertos atributos de los objetos seleccionados (según condiciones).
Asímismo, obtener el total de objetos seleccionados correspondiente a una clase.

CAPITULO III

ORGANIZACION DEL SISTEMA ABDO

Sección III.1 : Organización de la Información

En esta sección describimos como se estructura toda la información de nuestro sistema.

La estructura que describiremos corresponde a una aplicación. El usuario en una sesión de trabajo no tendrá activa más de una aplicación.

Una aplicación se representa por una serie de listas encadenadas relacionadas entre sí. Dichas listas están actualizadas en cualquier momento de una sesión de trabajo.

La estructura principal es la **lista de clases** donde figuran todas las clases definidas para la aplicación activa. Cada clase de la lista tiene asociadas las listas de atributos propios, atributos heredados, superclases e instancias.

La **lista de superclases** define las clases que son superclases inmediatas de la clase en cuestión. Siempre existe esta lista, pues cualquier clase tiene al menos como superclase a RAIZ.

La **lista de atributos propios** contiene información de cada atributo de la clase: nombre, tipo, tamaño, código, (si tiene valor por defecto o compartido), valor, clave. Tiene al menos al atributo clave.

En la lista de atributos propios el primer elemento es el atributo clave, independientemente del orden en que lo definió el usuario.

El tipo debe ser una clase existente en la aplicación. El tamaño debe especificarse sólo para el tipo STRING. El código indica si el atributo tiene valor por defecto o valor compartido; en tal caso se asocia un valor al atributo. Si el atributo tiene valor por defecto, en el momento de instanciar la clase, se asigna ese valor si el usuario no especifica otro explícitamente. Si el atributo tiene valor compartido, se asigna ese valor a todas las instancias de la clase.

El atributo seleccionado como clave no puede ser del tipo BOOLEAN ni tener valor por defecto o compartido, ya que el valor del atributo clave debe ser único.

La **lista de atributos heredados** contiene información de cada atributo que se hereda: nombre del atributo, clase de donde se obtiene la descripción del atributo (tipo, tamaño, etc.) y superclase inmediata de donde se hereda. Esta última es la que conoce el usuario, pero ambas clases son necesarias para mantener la consistencia del sistema luego de las operaciones de cambios de esquema provistas (ver Capítulo IV). Esta lista puede ser vacía.

La **lista de instancias** contiene todos los objetos de la clase . Cada objeto puede verse como un conjunto de valores, cada uno correspondiente a un atributo de la clase. De allí que una instancia de una clase se obtiene asignando valores a cada uno de los atributos que la componen. El valor del atributo clave para cada objeto de la clase es su identificador. Esta lista puede no contener elementos.

Existen otras estructuras, como la **lista de subclases** que contiene información de cuáles son las subclases inmediatas de cada clase. Esta lista, junto con la de superclases, nos permite navegar por todo el esquema reticulado de clases de la aplicación.

Para atacar el problema de **persistencia de los datos**, definimos un grupo de archivos (lo que es transparente para el usuario) que sirve como soporte físico cuando se abandona el sistema.

La decisión de *salvar* una aplicación cuando se abandona el sistema, depende del usuario. La aplicación queda en su estado anterior si no se salvan los cambios. Al salvar la aplicación, la información almacenada en la base de datos se vuelca a los archivos.

Cuando se quiere reingresar a una aplicación, se *recupera* la información de dichos archivos, generándose la aplicación en las estructuras descriptas anteriormente.

Sección III.2 : Estructura del Sistema

Para realizar nuestro proyecto, integrando tecnologías de las bases de datos con la filosofía Orientada a Objetos, seguimos los siguientes pasos:

- especificamos un modelo de datos orientado a objetos y los aspectos dinámicos del sistema;
- desarrollamos la teoría para mejor entendimiento del modelo;
- desarrollamos los algoritmos necesarios para implementar el sistema.

El desarrollo de nuestro sistema fue realizado en un lenguaje de alto nivel tradicional (TURBO PASCAL).

El sistema está estructurado en módulos (*units* de PASCAL). Describiremos brevemente su organización indicando qué algoritmos contiene y qué módulos referencia:

* PRINC: es el módulo principal del sistema. Es invocado por el programa de arranque del sistema.
  Algoritmos: crear, listar, eliminar y modificar aplicaciones.
  Módulos: CLASOBJ y CONSULTA.

* CLASOBJ
  Algoritmos: crear clase, crear objeto, modificación de clases, salvar aplicación en archivos y recuperar aplicación desde archivos.
  Módulos: OPERAC, OPERAC1 y OPERAC2.

* CONSULTA
  Algoritmos: consulta de clases, subclases, objetos, edición y eliminación de objetos.
  Módulos: CONSUL1.

* OPERAC, OPERAC1, OPERAC2
  Algoritmos: todos los programas de cambio de esquema que son invocados por el programa de modificación de clases de CLASOBJ.

* CONSUL1
  Algoritmos: los procedimientos que son invocados por los programas de consulta de clases, subclases y objetos de CONSULTA.
  Módulos: CONSULX.

* CONSULX
  Algoritmos: los procedimientos que son utilizados por los programas de consulta de CONSUL1.

* GENERAL: contiene todas las declaraciones de tipos utilizadas en el sistema y procedimientos de uso frecuente. Todas los módulos del sistema lo utilizan; incorporamos así estas definiciones a las provistas por el lenguaje .

El sistema resultante es el archivo *ABDO.EXE*, el cual debe instalarse junto con el archivo *APLIC.ARC*. Este último contiene las aplicaciones definidas en el sistema.

Culminamos el objetivo propuesto habiendo probado un volumen de aproximadamente 12000 líneas, organizadas en la forma previamente descripta.

CAPITULO IV

IMPLEMENTACION DEL SISTEMA

En este capítulo detallaremos cómo fue implementado este prototipo. Explicaremos las decisiones tomadas durante el desarrollo del sistema.

El usuario debe definir una aplicación para trabajar. Existen operaciones para tratar las aplicaciones.

Cuando se **crea una aplicación**, su nombre no se incorpora al archivo de aplicaciones hasta que la aplicación es salvada. Este archivo contiene los nombres de todas las aplicaciones definidas en el sistema. Asociado a cada aplicación, tenemos un conjunto de archivos con la información concerniente a ella (atributos, superclases, instancias).

Cuando de desea **trabajar con una aplicación** existente, se *recupera* de los archivos toda la información sobre la aplicación y se almacena en memoria de acuerdo a la estructura definida en el Capítulo anterior, Sección III.1 .

Si la aplicación acaba de ser creada, no hay datos para recuperar, pues aún no existe información de ella almacenada en los archivos.

En este momento,el sistema genera la información concerniente a los tipos predefinidos, para que puedan usarse en la aplicación como tipos posibles.

En el momento de **crear una clase**, se debe especificar atributos y superclases, información que es almacenada en las estructuras de listas correspondientes. Se asigna RAIZ como superclase si el usuario no especificó una.

Se verifica que el tipo de cada atributo exista en la jerarquía de clases definida por el usuario o sea un tipo predefinido. Además, el tipo debe ser subclase del tipo del atributo con el mismo nombre (si existe) en las superclases de la clase que se está creando. Se puede ingresar un código para indicar valor compartido o por defecto para un atributo. En tal caso se debe ingresar un valor, que acompañará la definición del atributo y que debe ser compatible con ésta. En la lista de atributos se marca el atributo clave cuando el usuario así lo especifique y se reorganiza dicha lista ubicando el atributo clave en primer lugar.

La asignación de atributos heredados a la clase, la resuelve el sistema de la siguiente manera : se recorren las superclases inmediatas, seleccionando aquellos atributos que no figuren como propios en la clase, ya que los propios tienen precedencia sobre los atributos de las superclases. Si el mismo nombre de atributo, factible de ser heredado ,figura en más de una superclase, el usuario debe resolver el conflicto. Esta decisión se preservará siempre que sea posible. En la lista de atributos heredados se almacena junto con el nombre del atributo, el nombre de la clase de donde se obtiene su descripción (tipo, tamaño, etc.) y el nombre de la superclase inmediata de donde se hereda. A esta última la bautizamos *vía* del atributo heredado. Conservar esta información, ayuda a simplificar ciertas operaciones de cambio de esquema, ya que nos permite conocer que camino siguió un atributo que es heredado en una clase.

La clase nueva, con sus listas asociadas, pasa a formar parte de la lista de clases de la aplicación.

Al **crear un objeto de una clase**, debe darse un valor a cada atributo que la compone, ya sea propio o heredado. Cada valor ingresado debe ser compatible con la definición del atributo. En el caso de un tipo predefinido las restricciones se corresponden con las del lenguaje Pascal utilizado en el desarrollo, salvo en el tipo string que está acotado por el tamaño que especifica el usuario. En el caso de que el tipo del atributo sea definido por el usuario, es decir, sea una clase existente, el valor ingresado debe ser una referencia a un objeto existente de ella. De esta forma proveemos objetos compuestos, ya que desde un objeto se puede referenciar otro. El valor puede ser nulo siempre que no sea el atributo clave.

Para facilitar el ingreso de valores se provee ayuda al usuario sobre la definición del atributo.

Una vez ingresados los valores de todos los atributos de una clase, se incorporan como una nueva instancia en la lista correspondiente asociada a la clase (lista de instancias). No podrá crearse un objeto de una clase si ya existe otro con ese valor de clave (no se permiten objetos duplicados).

Al **modificar una clase**, los cambios se propagan a todas las clases involucradas (subclases de la clase en todos sus niveles). Dichos cambios pueden provocar que se adapten las instancias existentes de tales clases. Tratamos de no perder valores de atributos en los objetos siempre que sea posible. Por ésto, muchas veces debemos **convertir** valores de atributos de un cierto tipo a valores de atributos de otro tipo (que debe ser superclase del anterior).

Explicaremos la conversión basándonos en el siguiente **ejemplo:**

Supongamos que tenemos la clase *persona* y la clase *estudiante* como subclase de la anterior.

Si en una clase, tenemos un atributo del tipo *estudiante*, en sus instancias los valores para ese atributo son referencias a objetos del tipo *estudiante*.

Si cambiamos el tipo de tal atributo por *persona*, tendremos que convertir sus valores, en todas las instancias, a referencias de objetos del tipo *persona*.

Seguiremos los siguientes pasos:

1) Si la referencia a estudiante es nula, la nueva referencia se mantiene así.

2) Si en la clase *estudiante*, no se hereda de *persona*, el atributo clave de *persona*, la nueva referencia será nula, pues ella debería ser el valor del atributo clave de *persona* y no podría obtenerse de la información provista por el objeto *estudiante*.

3) Si la nueva referencia obtenida del objeto *estudiante* es nula, no se puede referenciar a ningún objeto persona. La referencia nula es permitida mientras el atributo en cuestión no sea la clave de la clase.

4) Si la nueva referencia (distinta de nula) no es *persona* existente, se crea ese objeto a partir de la información obtenida del objeto *estudiante*. Esto es posible pues un *estudiante* es una *persona* y, por lo tanto, hereda sus atributos.

Para los tipos predefinidos, la conversión no es necesaria, ya que los valores de estos tipos se adaptan. Por ejemplo, un valor INTEGER sigue siendo un valor REAL o STRING.

Al **agregar un atributo** en una clase, recorremos en la jerarquía de clases, sus superclases para verificar que el tipo del nuevo atributo sea subclase o el mismo tipo que cualquier otro con el mismo nombre de atributo en las superclases. De la misma manera recorremos las subclases para verificar que el tipo del nuevo atributo sea superclase de cualquier otro con el mismo nombre de atributo definido en las subclases.

Si el atributo existía como heredado, salvo que los tipos sean iguales, los valores para ese atributo en las instancias existentes se perderían por no poder adaptarse.

(Por ejemplo: si el tipo del atributo herado era REAL y ahora se lo agrega como INTEGER ,los valores reales no pueden *convertirse* a valores enteros). Como nuestro propósito es no perder información, avisamos al usuario para que él decida continuar la operación, con las consecuencias explicadas o abandonarla. Lo más factible es que él no se hubiera percatado antes de estos riesgos. Como consecuencia de la operación en las subclases se podrían perder instancias al no poder convertir los valores (las subclases pasan a heredar el atributo agregado). Por este motivo, antes de aceptar la operación, investigamos en el esquema reticulado de clases, aquéllas que heredaban el atributo en cuestión vía la clase modificada.

Tanto en el caso que el atributo sea nuevo en la clase y subclases como si se pierden valores (caso anterior), en las instancias existentes de dichas clases el atributo figurará con valor nulo.

Al **eliminar un atributo** de una clase, el atributo puede pasar a heredarse de una o mas superclases. Si se produce un conflicto de herencia, damos la opción que el usuario seleccione la que le convenga. En este caso se incorpora el atributo en la lista de atributos heredados.

Debido a las restricciones de tipo de nuestro sistema, el tipo del atributo eliminado es subclase del tipo del atributo heredado, si existiese. Entonces los valores del atributo en las instancias existentes de la clase se pueden *convertir*. Si la clave del nuevo tipo no se hereda en el tipo del atributo eliminado, se perderían las instancias ya que no habría forma de referenciar los nuevos objetos. Se da aviso al usuario de ésto para que acepte o no continuar con la operación. Para cumplir este objetivo se analizan tipo anterior y nuevo del atributo a eliminar, antes de efectuar la operación.

Si las subclases tienen definido el atributo o lo heredan, y no a través de la clase modificada, la operación no las afecta. En caso contrario, analizamos la nueva herencia del atributo, tratando de conservar cualquier decisión anterior del usuario.

Un atributo puede eliminarse en una subclase,lo que no significa que esto también ocurra en otra de nivel inferior, pues puede heredarse alguna otra clase. Esto obliga a analizar la herencia del atributo en cada nivel de subclases.

Si el atributo eliminado no existe en ninguna superclase, se elimina de la lista de atributos de la clase y los valores serán deleteados en las instancias existentes.

Ante un **cambio de tipo** a un atributo verificamos que el nuevo tipo sea subclase de los tipos de atributos con el mismo nombre en las superclases, y que sea superclase del tipo

que se desea cambiar (el mismo tipo no tiene sentido). Por esta última razón es que no se compara con los tipos de atributos con el mismo nombre en las subclases, ya que si el sistema cumple la restricción de tipos con el tipo anterior, lo cumplirá también con el nuevo (que es superclase del anterior).

No permitimos cambiar el tipo de un atributo clave si el tipo es definido por el usuario, pues la *conversión* podría originar claves duplicadas en las instancias de nuestra clase. En los tipos predefinidos no ocurre ésto, pues los valores se reasignan sin *conversión* y por lo tanto siguen siendo únicas.

Al *convertir* los valores al tipo nuevo puede ocurrir que no puedan referenciarse objetos del nuevo tipo.
Ejemplo:

La clase *estudiante* hereda el atributo *documento* de su superclase *persona*. Supongamos que hacemos una operación cambio de tipo de *estudiante* a *persona*.

El valor del atributo en las instancias existentes tiene que pasar de ser una referencia a un objeto *estudiante* a ser una referencia a un objeto *persona*.

Si en algún objeto *estudiante* referenciado desde la clase modificada, el atributo *documento* tiene valor nulo, no podría obtenerse ninguna referencia correspondiente a un objeto *persona*. En este caso consultamos al usuario, ya que tratamos siempre de no perder valores en las instancias.

Por otro lado si los valores del atributo en las instancias existentes de la clase no pueden adaptarse a la nueva definición de tipo decidimos rechazar la operación (en el ejemplo anterior, si el atributo *documento* en la clase *estudiante* no se heredara de *persona* se perderían todas las instancias).

Al **cambiar la herencia** a un atributo, sus valores en las instancias existentes, se pierden (se les asigna valor nulo) ya que no tiene por qué haber relación entre el tipo anterior y el tipo de la nueva herencia. (Los tipos de atributos con el mismo nombre en las superclases de una clase no guardan relación subclase - superclase entre ellos). Igual tratamiento aplicamos a las subclases que son afectadas por la operación.

Consideramos como las nuevas herencias posibles, sólo a las superclases inmediatas de la clase, ya que si el atributo proviene de una superclase de nivel superior, siempre lo hereda a través de alguna superclase inmediata.

El cambio será asentado en la lista de atributos heredados de la clase modificada y de las subclases involucradas.

En el caso de querer **cambiar el atributo clave** de una clase, si el tipo del atributo elegido como nueva clave

es BOOLEAN o si el atributo tiene valor compartido o por defecto, rechazamos la operación pues no permitimos objetos duplicados. Asimismo testeamos que los valores del atributo elegido no sean nulos en ninguna instancia de la clase a modificar, pues así se perderían objetos existentes y además que no figuren duplicados en ninguna instancia de la clase a modificar. En ambos casos también se rechaza la operación.

El atributo elegido como clave pasa a ocupar el primer lugar en la lista de atributos de la clase, lo que nos obliga a reordenar los valores de los atributos en las instancias existentes de la clase. Se elimina la marca de clave del atributo clave anterior en la lista de atributos propios.

Pueden encontrarse objetos de otras clases que referencien instancias de la clase modificada (la referencia es el valor de la clave anterior). Esta operación ocasionaría la pérdida de los objetos referenciados. Para solucionar ésto, dichas referencias son cambiadas por los valores de la nueva clave (los objetos apuntados siguen siendo los mismos). Para llevar a cabo ésto, tenemos que recorrer todo el esquema reticulado de clases, investigando si el tipo de cada atributo de cada clase coincide con la clase modificada; de allí que no recomendemos esta operación.

Cuando se **cambia el valor por defecto a un atributo** en las instancias existentes de la clase (y de las subclases que hereden el atributo de ella), los valores del atributo no se modifican, pues, aunque el atributo tenga valor por defecto, puede haber instancias que tengan distintos valores asignados. El nuevo valor por defecto (una vez validada su compatibilidad con el tipo) pasa a formar parte de la definición del atributo en la lista de atributos propios.

Por el contrario, cuando se **cambia el valor compartido de un atributo**, en las instancias existentes de la clase modificada y de aquéllas que hereden el atributo en cuestión, se cambian los valores del atributo por el nuevo ingresado por el usuario. Esto es por definición de valor compartido (en todos los objetos, el atributo debe tomar el mismo valor). El nuevo valor (una vez validada su compatibilidad con el tipo) pasa a formar parte de la definición del atributo en la lista de atributos propios.

Brindamos la posibilidad de **asignarle valor compartido a un atributo** que no lo tuviera. No permitimos realizar esta operación con el atributo clave (se contrapone con el concepto de valor único de la clave).

En las instancias existentes de la clase modificada y de aquéllas que hereden el atributo en cuestión, se asigna el valor ingresado por el usuario a tal atributo.

Los valores anteriores al cambio se pierden y los de valor nulo tienen ahora ese valor asignado (después de validada su compatibilidad con el tipo). Esto es por definición de valor compartido, es decir, en todos los objetos el atributo debe tomar el mismo valor. En la lista de atributos propios de la clase modificada, se modifican código y valor al atributo especificado.

Permitimos **eliminar la propiedad de valor compartido** de un atributo, para lograr que en las futuras instancias de la clase se le pueda asignar cualquier valor. En las instancias existentes de la clase modificada y de aquéllas que hereden el atributo en cuestión, no se modifica el valor del atributo, pues después de efectuado el cambio, el atributo puede tomar cualquier valor y decidimos conservar los existentes. Si el usuario desea cambiarlos, puede hacerlo como en cualquier otro atributo, mediante la operación de edición provista.

En la lista de atributos propios se cambia el código y se elimina el valor asociado al atributo.

Al **asignar una clase como superclase de otra**, verificamos que el tipo de los atributos de la superclase agregada sea superclase del tipo de atributos con el mismo nombre, si existiesen, en la clase a la que se le agrega la nueva superclase. De no ser así, la operación se rechaza, por no cumplir las restricciones de tipos ya mencionadas. Para ello, comparamos los atributos de la superclase y de la clase, y en caso de estar redefinido el atributo en esta última, verificamos lo antepuesto.

Recorremos la jerarquía de clases para verificar que la superclase agregada no ocasione ciclos, para lo cual contamos con información de cuáles son las otras superclases de la clase, antes de permitir la operación.

En la lista de superclases de la clase se agrega la nueva. Las superclases no tienen precedencia unas sobre otras.

Los atributos de la superclase agregada que existían en la clase modificada, no se heredan, pues toda definición en una clase tiene precedencia sobre la definición del mismo atributo en una superclase. Igual tratamiento reciben las subclases involucradas.

En el caso de los atributos de la superclase agregada que ya se heredaban en la clase, la herencia no se cambia (pudo haber surgido de un conflicto de herencia que el usuario resolvió oportunamente y nosotros tratamos de preservar su decisión).

Los atributos que sí se heredan de la superclase, se agregan a la lista de atributos heredados de la clase y los valores correspondientes a dichos atributos en las instancias existentes, toman automáticamente el valor nulo. El usuario posteriormente, puede editar los objetos para asignarles un

valor.

Para mantener actualizada la información adicional, agregamos la clase modificada como subclase de la nueva superclase, en lo que llamamos lista de subclases.

Al **eliminar una clase como superclase de otra** se elimina ésta como superclase de la clase y se agregan como nuevas superclases en la lista de superclases de la clase afectada por la operación, aquéllas superclases de la superclase eliminada que no lo eran todavía.

Para cada atributo de la clase, heredado de la superclase eliminada, luego de la operación puede ocurrir:

a) Que el atributo ya no se herede.

En este caso, se elimina como atributo de la clase y sus valores en las instancias existentes de la clase.

b) Que el atributo se herede de alguna superclase de la superclase eliminada.

Por las restricciones de tipo, se pueden *convertir* los valores de los atributos. Como nuestro objetivo es no perder valores mientras sea posible, elegimos esta herencia.

c) Que el atributo se herede de una o más superclases que no verifican b).

En este caso puede llegar a surgir un nuevo conflicto de herencia, que el usuario debe resolver. En las instancias existentes de la clase, se asignan valores nulos al atributo por la nueva herencia.

En las subclases involucradas, se investiga también cada atributo heredado de la superclase eliminada a través de la clase modificada. Luego de la operación puede ocurrir:

a) Que el atributo ya no se herede.

En este caso, se elimina como atributo en la subclase y sus valores en las instancias existentes de la subclase.

b) Que el atributo se herede de la superclase pero a través de otra clase distinta de la clase modificada (cuando mencionamos *a través* de una clase, queremos significar que dicha clase se encuentra en el camino que une la clase donde se definió el atributo y la clase donde se hereda). En tal caso, elegimos esta herencia, ya que de esta manera las instancias de la subclase no se modifican.

c) Que el atributo se herede pero sin cumplir b).

En este caso, se trata de preservar cualquier resolución anterior del conflicto de herencia por parte del usuario en esta operación. En las instancias existentes de la subclase, los valores del atributo tratan de convertirse, en lo posible, para adaptarse a la nueva definición de tipo. En caso contrario, se les asigna valor nulo.

d) Que el atributo se herede de una o más superclases sin cumplir con b) ni c).

Puede pasar, pues un atributo que fue eliminado en un nivel superior de subclases, puede heredarse en un nivel inferior de profundidad a través de otras clases. En este

caso, el usuario podría tener que resolver un nuevo conflicto de herencia. En las instancias existentes de la subclase, se le asigna valor nulo al atributo por el cambio de herencia

Cada vez que el usuario resuelve un conflicto de herencia, su decisión trata de propagarse a todas las subclases de la clase afectada.

Al tratar una subclase de un nivel inferior, necesitamos que las superclases de ésta afectadas por la operación, estén todas actualizadas. Para solucionar ésto, realizamos un proceso que abandona momentáneamente a la subclase y modifica sus superclases afectadas por la operación, para luego, con esa información actualizada, poder continuar el tratamiento suspendido.

Para determinar el camino seguido por un atributo desde la clase donde fue definido hasta la clase donde se hereda, utilizamos la información reservada en la lista de atributos heredados.

Para mantener actualizada la información adicional, eliminamos la clase modificada como subclase de la superclase eliminada, en lo que llamamos lista de subclases.

Esta operación no implica eliminar la clase de la jerarquía.

Cuando se desea **eliminar una clase** verificamos que ninguna clase de la jerarquía la referencie, es decir, ningún atributo de otra clase tiene a esta clase como tipo. Si esto se detecta, rechazamos la operación y listamos las clases que la referencian.

Todas las clases que tenían a la eliminada como superclase, dejan de tenerla, al igual que todas las clases que la tenían como subclase. Esta información se modifica en las listas correspondientes.

Para cada subclase de la clase eliminada, realizamos una operación similar a **eliminar una superclase a una clase** (explicada previamente). Como vimos, pueden surgir conflictos de herencia que requieran la intervención del usuario. Para más detalles sobre la implementación, ver la operación antes mencionada.

Para eliminar toda información sobre la clase, basta con eliminarla en la lista de clases, pues a ésta se encuentran encadenadas las listas de atributos, superclases e instancias.

Cuando se **cambia el nombre de una clase**, se modifica la información de superclases para cada subclase inmediata de la clase modificada. En los atributos de cada clase del esquema que tengan como tipo a la clase modificada, se renombran tales tipos. Además, renombramos la clase cada vez que figure en la lista de atributos heredados de una clase y en la lista de subclases del sistema.

El resultado de esta operación no afecta las

instancias existentes de las clases involucradas.

Permitimos **editar objetos de una clase**, de esta forma cambiar sus valores. Se busca el objeto seleccionado en la lista de instancias de la clase, mediante el valor del atributo clave. Mostramos los valores que componen el objeto, permitiendo al usuario modificarlos, salvo que correspondan a la clave o sean valores compartidos. Proveemos una ayuda para el ingreso de valores (similar a la de creación de objetos). Verificamos que el valor ingresado sea compatible con la definición del atributo. Los cambios de valores se reflejan en la lista de instancias para el objeto correspondiente.

Permitimos **eliminar objetos de una clase**. Se busca el objeto seleccionado en la lista de instancias de la clase mediante el valor del atributo clave. Verificamos que el objeto seleccionado no esté referenciado desde otro objeto. Para ésto, recorremos la jerarquía de clases buscando atributos que tengan como tipo la clase a la que pertenece el objeto que se desea eliminar. Recorremos las listas de instancias correspondientes para ver si algún valor de dichos atributos coincide con la clave del objeto a eliminar. En caso afirmativo rechazamos la operación, para evitar referencias colgadas.

Antes de eliminar el objeto de la lista de instancias, lo mostramos al usuario para que confirme la operación.

Las clases y objetos del sistema puedeen accederse a través de **consultas**. Las consultas pueden desplegarse por pantalla o imprimirse.

Podremos visualizar la información contenida en la **clase:** superclases, atributos propios y atributos heredados. Dicha información es recuperada de las listas correspondientes asociadas a cada clase. En cuanto a los atributos propios de la clase se muestran sus características, es decir, nombre, tipo, tamaño, código, valor y una indicación de cual es el atributo clave. De los atributos heredados se muestra de donde se hereda el atributo.

Otro tipo de consulta es de **subclases**, donde se puede determinar cuales son las subclases de una clase determinada, recorriendo la lista de subclases.

Para consultar un **objeto**, se busca en la lista de instancias de la clase correspondiente, para obtener los valores de sus atributos. Algunos de dichos valores pueden ser referencias a otros objetos. El usuario puede seleccionar dichas referencias para visualizar el objeto correspondiente.

Para lograrlo es necesario ubicar en la jerarquía de clases, aquella clase que es tipo del atributo asociado a la referencia seleccionada; recorremos la lista de instancias de ella ubicando el objeto cuya clave coincide con la referencia. De allí obtenemos los valores que mostramos al usuario.

Además permitimos, que se consulte un grupo de objetos, según ciertas condiciones. Para ello proveemos una interfase donde se puedan ingresar condiciones (>, <, =, <=, >=, <>) que relacionen atributos y valores unidas por conectores AND y OR. Con esta información testeamos cada uno de los objetos de la lista de instancias de la clase y seleccionamos aquellos que verifican las condiciones.

Se pueden seleccionar por otro lado que atributos de la clase se desea visualizar. Se mantiene una lista con los atributos seleccionados y luego se recuperarán éstos de las instancias correspondientes.

La información que se visualiza en cualquier caso es mantenida en listas doblemente encadenadas, para permitir que el usuario navegue en cualquier dirección.

Puede hacerse una consulta ordenada de los objetos, simplemente indicando el atributo por cuyos valores se desea clasificar. Otra vez, pueden seleccionarse grupos de objetos y atributos para visualizar.

Otra facilidad provista es el cálculo de la suma y el promedio de valores, correspondientes a atributos cuyos tipos sean numéricos. Se puede restringir la cantidad de objetos que intervienen en la operación. Se visualiza por cada atributo seleccionado el resultado obtenido y el número de objetos intervinientes en el cálculo.

Asimismo puede obtenerse la cantidad total de objetos de una clase o sólo aquellos que cumplen determinada condición.

Al **abandonar una aplicación**, si desea *salvar* los cambios, se almacenará en un directorio con el nombre de la aplicación, los archivos con información sobre instancias, atributos propios, atributos heredados y superclases de cada clase de la aplicación. Si la aplicación es nueva, además se le da de alta en el archivo de aplicaciones. De estos archivos recuperamos la información cuando el usuario decide retomar la aplicación.

Se trata de liberar memoria que no se esté utilizando. Como parte de la información de la base se encuentra organizada en listas, tratamos de liberar de la stack aquéllas listas auxiliares que se usan en algunas operaciones, como así también información que comienza a generarse, por ejemplo al crear una clase, y que luego, es innecesaria por haber cancelado el usuario la operación.

A modo de ejemplo de cómo implementamos el prototipo incluímos a continuación el procedimiento *sacarsup* utilizado para eliminar una superclase a una clase. Elegimos incluir este procedimiento porque contiene procesos de manejo de estructuras de listas, que han sido típicos durante el desarrollo de nuestro sistema. Cabe acotar que el presente fue uno de los procedimientos implementados más complejos.

Este procedimiento primero solicita ingresar la clase que se desea modificar y la superclase de ella que se quiere eliminar. Una vez validadas ambas clases, se verifica mediante el procedimiento recursivo *testsup* si alguna de las superclases de la superclase eliminada es ya superclase de la clase modificada (como ya explicamos, de ser posible las superclases de la superclase eliminada pasan a serlo de la clase modificada).

Se analizan los atributos que se heredaban de la superclase (pueden perderse, heredarse de alguna otra - u otras - superclases). Los atributos afectados por la operación se mantienen en una lista para propagar los cambios en las subclases de la clase involucrada.

Para propagar los cambios en las subclases se utiliza el procedimiento *subcla*. Este, con el procedimiento *atrisub* trata aquéllos atributos de la subclase que podrían resultar afectados (eliminados o con nueva herencia). Dichos atributos se rescatan de la lista mencionada en el párrafo anterior. Luego de finalizado *atrisub*, se llama recursivamente *subcla* para analizar las subclases de la subclase corriente.

Atrisub utiliza los procedimientos *buscavia* y *buscamino*. El primero es un procedimiento recursivo que sirve para analizar si la clase modificada se encuentra en el camino que une la clase donde está definido el atributo y la clase (la subclase que se está analizando) que lo heredá. Si es así, debemos propagar los cambios para este atributo en la subclase. El procedimiento recursivo *buscamino* se usa cuando tratamos de propagar los cambios en una subclase y no todas sus superclases han sido actualizadas como consecuencia de esta operación. Devuelve una lista con las clases no actualizadas y *atrisub* se llama recursivamente para propagar los cambios a dichas clases. Luego se retorna a la subclase que se estaba tratando.

```
procedure SACARSUP(var pisub:pgclasub; pigcl:pgcls; pigpred:pgpred; nomap:str8);
type  plis=^lista;
      lista=record
            pslis : plis;
            nom   : str15;
            end;
var tvie,tnue,cla,clasup,super :str8;
    p,pc,pp,pcl                :pgcls;
    vp :array[1..5] of pgathe;
    st,msg           :string;
    pii              :pinss;
    pini,pinia       :pin;
    psupa,psup,psupx :pgsup;
    psub,psuba       :pgclasub;
    ppsub,ppsuba,ppsubx:pgsub;
    ph,pha,piah,pah,paha,pahx  :pgathe;
    pat,patx              :pgatpr;
    cc,cont,i,h,j,k         :integer;
    ppred                 :pgpred;
    atri                  :str15;
    pl,pla,pil            :plis;


procedure testsup (psup:pgsup;nom:str8;var msg:string);
var psup1 : pgsup;
      pax : pgcls;
begin
while (psup<>nil) and (msg=' ') and (psup^.nomsup <>'RAIZ') do
      begin
      if psup^.nomsup = nom  then msg:=nom+' existe como superclase de
                             else begin
                             pax:=pigcl;
                             while (pax <> nil) and (pax^.nomcl<>psup^.nomsup) do
                                   pax:=pax^.psgcl;
                             psup1:=pax^.pigsup;
                             testsup(psup1,nom,msg);
                             if msg = ' ' then psup:=psup^.psgsu
                                  end;
       end;
 end;   {testsup}

 procedure subcla(clase:str8; atri:str15);
  var clhe,via     :str8;
      psub         :pgclasub;
      ppsub        :pgsub;

  procedure atrisub(sub:str8; var clhe,via:str8);
    type  pcam = ^camino;
          camino = record
                   pscla :pcam;
                   pacla :pcam;
                   nombre :str8;
                   end;
    var p,pcl          :pgcls;
        psup           :pgsup;
        super          :str8;
        viac,conver    :char;
        piah,ph,pahx,pah,paha,pha :pgathe;
        patx,pat          :pgatpr;
        pcami,pca,pcaa    :pcam;
        i,k:integer;

  procedure buscavia(clas:str8);
  var pa:  pgcls;
    paha:pgathe;
  begin
  { buscavia }
  pa:=pigcl;
  while(pa^.nomcl<>clas) do
```

```
     pa:=pa^.psgcl;
  paha:=pa^.pigath;
  while (paha<>nil) and (paha^.nomah<>atri) do
       paha:=paha^.psgath;
  if paha = nil then  viac:='s'
            else if paha^.via = cla then viac:='s'
                                    else  if paha^.nomclhe<>paha^.via then
                                                          buscavia(paha^.via);
  end;

  procedure buscamino(clas:str8);
      var pa    :pgcls;
          paha  :pgathe;
  begin
      {buscamino }
  pa:=pigcl;
  while(pa^.nomcl<>clas) do
     pa:=pa^.psgcl;
  paha:=pa^.pigath;
  while (paha<>nil) and (paha^.nomah<>atri) do
       paha:=paha^.psgath;
  if paha <> nil then begin
                 new(pca);
                 pca^.nombre:=clas;
                 pca^.pscla:=nil;
                 if pcami=nil then begin
                              pca^.pacla:=nil;
                              pcami:=pca;
                              end
                              else begin
                              pcaa^.pscla:=pca;
                              pca^.pacla:=pcaa;
                              end;
                 pcaa:=pca;
                 if paha^.via<>cla then buscamino(paha^.via)
                 end;
    end;

begin
      { atrisub }
p:=pigcl;
while p^.nomcl<>sub do p:=p^.psgcl;
cont:=0;
pat:=p^.pigatp;
while pat<>nil do begin
                  cont:=cont+1;
                  pat:=pat^.psgatp;
                  end;
ph:=p^.pigath;
pha:=ph;
while (ph<>nil) and (ph^.nomah<>atri) do
    begin
    cont:=cont+1;
    pha:=ph;
    ph:=ph^.psgath;
    end;
if ph<>nil then
           if ph^.nomclhe=clasup then begin
                              piah:=nil;
                              psup:=p^.pigsup;
                              if psup^.nomsup <> 'RAIZ' then
                                   while (psup<>nil) do begin
                                    super:=psup^.nomsup;
                                    pcl:=pigcl;
                                    while (pcl^.nomcl<>super) do
                                       pcl:=pcl^.psgcl;
                                    patx:=pcl^.pigatp;
                                    while (patx<>nil) and (patx^.nomat<>atri) do
                                           patx:=patx^.psgatp;
                                    if patx<>nil then begin
                                                      new(pah);
                                                      pah^.nomah:=atri;
```

```
                                    pah^.nomclhe:=super;
                                    pah^.via:=super;
                                    pah^.psgath:=nil;
                                    if piah = nil then piah:=pah
                                                  else paha^.psgath:=pah;
                                    paha:=pah;
                                          end
                                       else begin
                                        pahx:=pcl^.pigath;
                                    while (pahx<>nil) and (pahx^.nomah<>atri) do
                                                pahx:=pahx^.psgath;
                                            if pahx <> nil then begin
                                               new(pah);
                                               pah^.nomah:=atri;
                                            pah^.nomclhe:=pahx^.nomclhe;
                                            pah^.via:=super;
                                            pah^.psgath:=nil;
                                            if piah = nil then piah:=pah
                                                          else paha^.psgath:=pah;
                                            paha:=pah;
                                                 end;
                                end;
                                psup:=psup^.psgsu;
                     end; {psup<>nil}
               if piah=nil then begin {atri no se hereda de ninguna superclase}
                           clhe:=' ';
                           pii:=p^.pigin;
                           while pii<>nil do begin
                            pini:=pii^.pins;
                            pinia:=pini;
                            for i:=1 to cont do begin
                            pinia:=pini;
                            pini:=pini^.psregin;
                                                 end;
                           pinia^.psregin:=pini^.psregin;
                           pii:= pii^.psinss;
                                                end;
                          if pha<> ph then  pha^.psgath:=ph^.psgath
                                      else begin
                                      p^.pigath:=ph^.psgath;
                                      pha:=p^.pigath;
                                      end;
                          cont:=cont-1;
                          end


        else begin    { el atributo se hereda }
        viac:='s';
        pah:=piah;
        while pah<>nil do
           begin
           if pah^.nomclhe=clasup then
                if pah^.via<>cla then begin
                                  viac:='n';
                                  buscavia(pah^.via);
                                  end;
           if viac='s' then pah:=pah^.psgath
                       else begin
                       paha:=pah;
                       pah:=nil;
                       end;
           end;
        if viac='n' then begin
                    clhe:=paha^.nomclhe;
                    ph^.via:=paha^.via;
            { ph^.nomclhe ya está bien y  las  Inst. quedan = }
                    end
                    else begin
                    pah:=piah;
                    while pah<>nil do
                      begin
                      while (pah<>nil) and (pah^.nomclhe<>clasup) do
```

```
                    begin
                    paha:=pah;
                    pah:=pah^.psgath;
                    end;
                  if pah<>nil then begin
                              pcami:=nil;
                              buscamino(pah^.via);
                              pca:=pcami;
                              while pca^.pscla<>nil do pca:=pca^.pscla;
                              pcaa:=pca;
                              while pcaa<>nil do
                                begin
                                atrisub(pcaa^.nombre,clhe,via);
                                pcaa:=pcaa^.pacla;
                                end;
                              if clhe=' ' then begin
                               if (piah=pah) then
                                if  piah^.psgath=nil
                                            then piah:=nil
                                            else piah:=pah^.psgath
                                       else paha^.psgath:=pah^.psgath;
                                     end
                                     else begin
                                     pah^.nomclhe:=clhe;
                                     pah^.via:=via;
                                     paha:=pah;
                                     end;
                            end;  {if pah<>nil}
                  if pah<>nil then pah:=pah^.psgath;
                  end;  {while pah<>nil}
                if  piah=nil then begin {atri no se hereda}
                       clhe:=' ';
                       pii:=p^.pigin;
                       while pii<> nil do begin
                        pini:=pii^.pins;
                        pinia:=pini;
                        for i:=1 to cont do begin
                        pinia:=pini;
                        pini:=pini^.psregin;
                                                end;
                       pinia^.psregin:=pini^.psregin;
                       pii:= pii^.psinss;
                                              end;
                      if pha<> ph then  pha^.psgath:=ph^.psgath
                                  else begin
                                  p^.pigath:=ph^.psgath;
                                  pha:=p^.pigath;
                                  end;
                      cont:=cont-1;
                            end
                            else begin    {atri se hereda}
                       if piah^.psgath<>nil then begin
                          pah:=piah;
                          while (pah<>nil) and (pah^.nomclhe<>clasup) do
                              pah:=pah^.psgath;
                          if pah<>nil then begin
                                      piah:=pah;
                                      piah^.psgath:=nil;
                           conver:='i';
                                      end
                                      else begin
                          pah:=piah;
                          while (pah<>nil) and (pah^.via<>ph^.via) do
                              pah:=pah^.psgath;
                          if pah<>nil then begin
                                      piah:=pah;
                                      piah^.psgath:=nil;
                                               conver:='s';
                                      end
                                      else begin
borralin(8,24);
gotoxy(1,23); for k:=1 to 80 do write(#196);
```

```
                gotoxy(1,8);
                write('El atributo ', atri,' cambia de herencia en la clase ',sub);
                gotoxy(1,9);
                write('(',sub,' es subclase de ',cla,')');
                gotoxy(1,11);
                write('De dónde desea HEREDAR el Atributo  ',atri,' ? ');
                k:=13;
                       j:=1;
                       while(piah<>nil) do begin
                           vp[j]:=piah;
                           gotoxy(5,k);   k:=k+1;
                           write(j,'- ',piah^.via);
                           j:=j+1;
                           piah:=piah^.psgath;
                                       end;
                       gotoxy(6,k+1);
                       write('Ingrese una opción ==> ');
                       repeat
                       gotoxy(30,k+1);
                       readln(st);
                       val(st,h,cc);
                       if (cc<>0) or (h<=0) or (h>=j) then   begin
                                gotoxy(1,22);
                                write('Opción no válida...Reingrese');
                                end;
                       until (h>0) and (h<j);
                       piah:=vp[h];
                       conver:='n';
                                              end;
                                                   end;
                                                    end {varias her.}
                                                    else begin  {una her. posible}
                                                if piah^.nomclhe=clasup then conver:='i'
                              else   if piah^.via=ph^.via then  conver:='s'
                                                      else conver:='n';
                                                    end;
                             pcl:=pigcl;
                             while (pcl^.nomcl<>piah^.nomclhe) do
                                 pcl:=pcl^.psgcl;
                             pat:=pcl^.pigatp;
                             while (pat^.nomat<>atri) do
                                 pat:=pat^.psgatp;
                             tnue:=pat^.tipat;

                              ph^.nomclhe:=piah^.nomclhe;
                              ph^.via:=piah^.via;
                              clhe:=piah^.nomclhe;

if conver='s' then
    if tvie=tnue then conver:='i'
                else
                begin
                ppred:=pigpred;
                while (ppred<>nil) and (ppred^.nompred<>tvie) do
                    ppred:=ppred^.pspred;
                if ppred<>nil then
                    if ((tvie = 'INTEGER') and (tnue = 'REAL')) or
                       ((tvie = 'INTEGER') and (tnue = 'STRING')) or
                       ((tvie = 'REAL') and (tnue = 'STRING')) then conver:='i'
                                                               else conver:='n'
                              else begin
                                   pcl:=pigcl;
                                   while (pcl^.nomcl<>tvie) do
                                     pcl:=pcl^.psgcl;
                                   msg:=' ';
                                   testsup(pcl^.pigsup,tnue,msg);
                                  if msg <> ' ' then convertir(sub,atri,tvie,tnue,pigcl)
                                                else conver:='n';
                                   end;
                end;
                if conver='n' then begin
                                     pii:=p^.pigin;
```

```
                                    while pii<>nil do begin
                                          pini:=pii^.pins;
                                          for i:=1 to cont do pini:=pini^.psregin;
                                          pini^.campo:=' ';
                                          pii:=pii^.psinss;
                                          end;

                                    end;

                                          end;  .{atri se hereda}
                                end;  {else viac='n'}
                          end;   {atri se hereda}
                                end;  {ph^.nomclhe=clsup}
via:=sub;
end;    {atrisub}


begin
   {  subcla  }
psub:=pisub;
while (psub<>nil) and (psub^.clasu<>clase) do
   psub:=psub^.psclasu;
if psub<>nil then begin
             ppsub:=psub^.pigsub;
             while ppsub<>nil do
                begin
                atrisub(ppsub^.nomsub,clhe,via);
                subcla(ppsub^.nomsub,atri);
                ppsub:=ppsub^.psgsub;
                end;
             end;
end;  {subcla}


begin           {sacarsup}
clrscr;
gotoxy(1,1); write(nomap);
gotoxy(1,3);
textcolor(0);  textbackground(15);
write('                       ELIMINAR UNA SUPERCLASE A UNA CLASE
textcolor(15); textbackground(0); lowvideo;
gotoxy(1,4);  for i:= 1 to 80 do write(#196);
gotoxy(1,23);  for i:= 1 to 80 do write(#196);
repeat
gotoxy(60,24); highvideo; write('Q'); normvideo;
write(' -> Salir');
gotoxy(5,8); write ('Ingrese Nombre de CLASE a modificar ==>');
gotoxy(52,8); clreol;
for i:=1 to 8 do write('_');gotoxy(52,8);
readln(st);
cla:=mayu(st);
if cla<>'Q' then begin
             gotoxy(1,22);
             clreol;
             p:=pigcl;
             while  (p<>nil) and (p^.nomcl<>cla) do
                  p:=p^.psgcl;
             if p=nil then begin
                    gotoxy(1,22);
                    textcolor(0);  textbackground(15);
                    write (  ',cla,' no existente (3) ');
                    textcolor(15);  textbackground(0);
                    normvideo;
                          end
                else begin   {p=nil}
                   gotoxy(5,11);
                   write('Ingrese nombre de SUPERCLASE a eliminar  ==>');
                  repeat
                   msg:=' ';
                   gotoxy(52,11); clreol;
                   for i:=1 to 8 do write('_'); gotoxy(52,11);
                   readln(st);
```

```
        gotoxy(1,22);
        clreol;
        clasup:=mayu(st);
        if clasup <> '0' then begin
        if cla=clasup then begin
        msg:='error';
      gotoxy(1,22);
      textcolor(0);  textbackground(15);
      write (' ',Clasup,' es la misma (37) ');
      textcolor(15);  textbackground(0);
      normvideo;
                           end
                     else begin
        pc:=pigcl;
        while (pc<>nil) and (pc^.nomcl<>clasup) do
              pc:=pc^.psgcl;
        if pc=nil  then begin
        msg:='errror';
      gotoxy(1,22);
      textcolor(0);  textbackground(15);
      write (' ',clasup,' no existente (3) ');
      textcolor(15);  textbackground(0);
      normvideo;
                       end
                   else begin
psup:=p^.pigsup;
while (psup<>nil) and (psup^.nomsup<>'RAIZ') and (psup^.nomsup<>clasup) do
      begin
      psupa:=psup;
      psup:=psup^.psgsu;
      end;
if (psup=nil) or (psup^.nomsup='RAIZ') then begin
        msg:='error';
      gotoxy(1,22);
      textcolor(0);  textbackground(15);
      write (' ',clasup,' no es Superclase de ',cla,' (41) ');
      textcolor(15);  textbackground(0);
      normvideo;
                                          end
                                      else begin
          if (psup=p^.pigsup) and (psup^.psgsu=nil) then psup^.nomsup:='RAIZ'
                                                    else
          if psup=p^.pigsup then p^.pigsup:=psup^.psgsu
                            else psupa^.psgsu:=psup^.psgsu;
          psupa:=p^.pigsup;
          while psupa^.psgsu<>nil do psupa:=psupa^.psgsu;
          psup:=pc^.pigsup;
          while(psup<>nil) and (psup^.nomsup<>'RAIZ') do
              begin
              msg:=' ';
              testsup(p^.pigsup,psup^.nomsup,msg);
              if  msg=' ' then begin
                       if  psupa^.nomsup='RAIZ' then psupa^.nomsup:=psup^.nomsu
                                                 else begin
                                                 new(psupx);
                                                 psupx^.nomsup:=psup^.nomsup;
                                                 psupx^.psgsu:=nil;
                                                 psupa^.psgsu:=psupx;
                                                 psupa:=psupx;
                                                 end;
                          psub:=pisub;
                          while psub^.clasu<>psup^.nomsup do
                            psub:=psub^.psclasu;
                          ppsub:=psub^.pigsub;
                          while (ppsub^.psgsub<>nil) do
                             ppsub:=ppsub^.psgsub;
                          new(ppsubx);
                          ppsubx^.nomsub:=cla;
                          ppsubx^.psgsub:=nil;
                          ppsub^.psgsub:=ppsubx;

                          end; {msg=' '}
```

```
     psup:=psup^.psgsu;
     end;

psub:=pisub;
while psub^.clasu<>clasup do
   begin
   psuba:=psub;
   psub:=psub^.psclasu;
   end;
ppsub:=psub^.pigsub;
while(ppsub^.nomsub<>cla) do
   begin
   ppsuba:=ppsub;
   ppsub:=ppsub^.psgsub;
   end;
if (ppsub=psub^.pigsub) and (ppsub^.psgsub=nil) then
   begin
   if (psub=pisub) and (psub^.psclasu=nil) then pisub:=nil
                                           else
               if (psub=pisub) then pisub:=psub^.psclasu
                               else psuba^.psclasu:=psub^.psclasu;
   end
                                                 else
   if  ppsub=psub^.pigsub then psub^.pigsub:=ppsub^.psgsub
                          else ppsuba^.psgsub:=ppsub^.psgsub;


  cont:=0;
  pat:=p^.pigatp;
  while pat<>nil do begin
                    cont:=cont+1;
                    pat:=pat^.psgatp;
                    end;
  pil:=nil;
  ph:=p^.pigath;
  pha:=ph;
  while ph<>nil do
     begin
     cont:=cont+1;
     if ph^.via=clasup then begin
            atri:=ph^.nomah;
            new(pl);
            pl^.nom:=atri;
            pl^.pslis:=nil;
            if pil=nil then pil:=pl
                       else pla^.pslis:=pl;
            pla:=pl;
            piah:=nil;
            psup:=p^.pigsup;
            if psup^.nomsup <> 'RAIZ' then
               while (psup<>nil) do begin
                   super:=psup^.nomsup;
                   pcl:=pigcl;
                   while (pcl^.nomcl<>super) do
                      pcl:=pcl^.psgcl;
                   patx:=pcl^.pigatp;
                   while (patx<>nil) and (patx^.nomat<>atri) do
                         patx:=patx^.psgatp;
                   if patx<>nil then begin
                                     new(pah);
                                     pah^.nomah:=atri;
                                     pah^.nomclhe:=super;
                                     pah^.via:=super;
                                     pah^.psgath:=nil;
                                     if piah = nil then piah:=pah
                                                   else paha^.psgath:=pah;
                                     paha:=pah;
                                     end
                                else begin
                                      pahx:=pcl^.pigath;
                                while (pahx<>nil) and (pahx^.nomah<>atri) do
                                           pahx:=pahx^.psgath;
```

```
                                                    if pahx <> nil then begin
                                                       new(pah);
                                                       pah^.nomah:=atri;
                                                       pah^.nomclhe:=pahx^.nomclhe;
                                                       pah^.via:=super;
                                                       pah^.psgath:=nil;
                                                       if piah = nil then piah:=pah
                                                             else paha^.psgath:=pah;
                                                       paha:=pah;
                                                      end;
                                        end;
                                        psup:=psup^.psgsu;
                    end;  {psup<>nil}
if piah=nil then begin {atri no se hereda de ninguna superclase}
          pii:=p^.pigin;
          while pii<> nil do begin
                 pini:=pii^.pins;
                 pinia:=pini;
                 for i:=1 to cont-1 do begin
                        pinia:=pini;
                        pini:=pini^.psregin;
                                end;
                 pinia^.psregin:=pini^.psregin;
                 pii:= pii^.psinss;
                 end;
          if pha<> ph then  pha^.psgath:=ph^.psgath
                      else begin
                           p^.pigath:=ph^.psgath;
                           pha:=p^.pigath;
                           end;
          cont:=cont-1;
                  end


             else begin    { el atributo se hereda }
             pha:=ph;
             if ph^.nomclhe=clasup then begin
                    msg:=' ';
                    pahx:=piah;
                    while (pahx<>nil) and (msg=' ') do
                        begin
                        testsup(pc^.pigsup,pahx^.nomclhe,msg);
                        if msg = ' ' then  pahx:=pahx^.psgath;
                        end;
                    if msg<>' ' then begin {atri se her. de alguna supercl. de clasup}
                                pcl:=pigcl;
                                while pcl^.nomcl<>ph^.nomclhe do
                                      pcl:=pcl^.psgcl;
                                patx:=pcl^.pigatp;
                                while patx^.nomat<>atri do
                                      patx:=patx^.psgatp;
                                tvie:=patx^.tipat;
                                pcl:=pigcl;
                                while pcl^.nomcl<>pahx^.nomclhe do
                                      pcl:=pcl^.psgcl;
                                patx:=pcl^.pigatp;
                                while patx^.nomat<>atri do
                                      patx:=patx^.psgatp;
                                tnue:=patx^.tipat;
                                ppred:=pigpred;
                                while (ppred<>nil) and (ppred^.nompred<>tvie) do
                                      ppred:=ppred^.pspred;
                                if (ppred=nil) and (tvie<>tnue) then
                                    convertir(cla,atri,tvie,tnue,pigcl);
                                ph^.nomclhe:=pahx^.nomclhe;
                                ph^.via:=pahx^.via;
                                end
                                else begin  {msg=' '}
                                if piah^.psgath <> nil then
                        begin
                borralin(8,24);
                gotoxy(1,23); for j:=1 to 80 do write(#196);
```

```
gotoxy(1,8);
write('El atributo ', atri,' cambia de herencia en la clase ',cla);
gotoxy(1,10);
write('De dónde desea HEREDAR el Atributo  ',atri,' ? ');
k:=12;
       j:=1;
       while(piah<>nil) do begin
           vp[j]:=piah;
           gotoxy(5,k);  k:=k+1;
           write(j,'- ',piah^.via);
           j:=j+1;
           piah:=piah^.psgath;
                           end;
       gotoxy(6,k+1);
       write('Ingrese una opción ==> ');
       repeat
       gotoxy(30,k+1);
       readln(st);
       val(st,h,cc);
       if (cc<>0) or (h<=0) or (h>=j) then begin
                gotoxy(1,22);
                write('Opción no válida...reingrese');
                end;
       until (h>0) and (h<j);
       piah:=vp[h];
       end;
                ph^.nomclhe:=piah^.nomclhe;
                ph^.via:=piah^.via;
                pii:=p^.pigin;
                while pii<>nil do begin
                      pini:=pii^.pins;
                      for i:=1 to cont-1 do pini:=pini^.psregin;
                      pini^.campo:=' ';
                      pii:=pii^.psinss;
                      end;
                end;
                        end
                else begin   {ph^.nomclhe<>clasup}
        pahx:=piah;
        while (pahx<>nil) and (pahx^.nomclhe<>ph^.nomclhe) do
              pahx:=pahx^.psgath;
        if pahx=nil then begin
                if piah^.psgath <> nil then
       begin
borralin(8,24);
gotoxy(1,23); for j:=1 to 80 do write(#196);
gotoxy(1,8);
write('El atributo ', atri,' cambia de herencia en la clase ',cla);
gotoxy(1,10);
write('De dónde desea HEREDAR el Atributo  ',atri,' ? ');
k:=12;
       j:=1;
       while(piah<>nil) do begin
           vp[j]:=piah;
           gotoxy(5,k);  k:=k+1;
           write(j,'- ',piah^.via);
           j:=j+1;
           piah:=piah^.psgath;
                           end;
       gotoxy(6,k+1);
       write('Ingrese una opción ==> ');
       repeat
       gotoxy(30,k+1);
       readln(st);
       val(st,h,cc);
       if (cc<>0) or (h<=0) or (h>=j) then begin
                gotoxy(1,22);
                write('Opción no válida...reingrese');
                                        end;
       until (h>0) and (h<j);
       piah:=vp[h];
       end;
```

```
                                    ph^.nomcine:=piah^.nomcine;
                                    ph^.via:=piah^.via;
                                    pii:=p^.pigin;
                                    while pii<>nil do begin
                                          pini:=pii^.pins;
                                          for i:=1 to cont-1 do pini:=pini^.psregin;
                                          pini^.campo:=' ';
                                          pii:=pii^.psinss;
                                          end;
                                    end
                               else   {pahX<>nil}
                               ph^.via:=pahx^.via;
                                        end;    {ph^.nomclhe<>clasup}
                                    end;  {piah<>nil}
                                end  {ph^.via<>clasup}
                                else pha:=ph;
                           ph:=ph^.psgath;
                           end;    {while ph<>nil}

                      { s u  b  c  l  a  s  e  s }
                 pl:=pil;
                 while pl<>nil do
                    begin
                    pat:=pc^.pigatp;
                    while (pat<>nil) and (pat^.nomat<>pl^.nom) do
                        pat:=pat^.psgatp;
                    if pat<>nil then begin
                    tvie:=pat^.tipat;
                    subcla(cla,pl^.nom);
                                  end;
                    pl:=pl^.pslis;
                    end;
                  gotoxy(1,22);  clreol;
                  textcolor(0);  textbackground(15);
                  write (' Modificación realizada ');
                  textcolor(15);  textbackground(0);
                  normvideo;
                               end; {psup=nil....}
                          end; {pc<>nil}
                            end; {cla=clasup}
                             end;{clasup<>0}
            until (msg=' ') or (clasup='0');
                         end;{p=nil}
   end; {cla <> '0'}
 borralin(8,21);
 until  (cla='0');

end;   {sacarsup}
```

CONCLUSIONES

Comenzamos nuestro proyecto investigando aspectos vinculados a modelos de datos orientados a objetos y proyectos sobre modelos y administradores de base de datos existentes, cumpliendo así, con la primera etapa del Plan de Trabajo.

En una segunda etapa, especificamos las características básicas de nuestro sistema: tipo de herencia, sistemas de tipos, existencia de clases, objetos compuestos, etc..

El proyecto realizado obtuvo como resultado un prototipo de base de datos con orientación a objetos, donde se implementó un sistema de definición de tipos y clases, operaciones de cambio de esquema y un sistema de consultas a la base de datos. Nuestro esfuerzo estuvo puesto en el análisis e implementación de los cambios de esquema, manteniendo en todo momento el sistema en un estado consistente. No encontramos bibliografía suficiente como para lograr un consenso general de cómo resolver las distintas situaciones originadas por las operaciones; debido a ésto, las consideraciones de cada caso fueron propias (sostenemos que otras desiciones pudieron haber sido tomadas en tales casos). Durante el desarrollo del sistema tuvimos siempre presente los invariantes propuestos en el sistema, por lo cual alguna operación pudo haber perdido eficiencia al tratar de mantener alguno de ellos.

Este prototipo puede utilizarse como lo indica el manual del usuario que se incluye seguidamente en el apéndice de este texto.

Cabe acotar que en el momento en que el proyecto fue iniciado, el campo de los sistema orientados a objetos comenzaba a investigarse en nuestro medio. Aspectos que se encontraban en estudio en ese momento hoy se encuentran incorporados a los sistema orientados a objetos, debido al constante crecimiento de esta área.

Consideramos que hay aspectos del prototipo que se pueden mejorar, como el subsistema de consultas, al cual se le podrían incorporar nuevos tipos de accesos a la base de datos.

Además existen rasgos que podrían agregarse al sistema realizado, como la inclusión de métodos (y mensajes), mediante los cuales, los objetos puedan acceder o modificar su representación interna. Proveer métodos, no formó parte de nuestro plan de trabajo; nuestro modelo es estructural. También podrían querer manejarse distintas versiones de un objeto.

Por último podría incorporarse un soporte para realizar transacciones, es decir, un lenguaje de programación para hacer uso de la base de datos.

BIBLIOGRAFIA

ref. 1 Schema Evolution in Object Oriented Persistent Databases
Jay Banerjee, H. kim, Won Kim, H. F. Korth
Proceedings of the sixth advanced Database Symposium, 1986.

ref. 2 An Object - Oriented DBMS for Design Support Applications
T.M. Atwood
Proceedings IEEE COMPPINT'85.

ref. 3 Making Smalltalk a Database System
G. Copeland, D. Maier
Proceedings of ACM SIGMOD, 1984.

ref. 4 - Composite Object Support in an Object-Oriented Database System
Won Kim, Jay Banerjee, Hong-Tai Chou,
Jorge F. Garza, Darrel Woelk
OOSPLA'87 Proceedings.

ref. 5 - Integration of Time Versions into a relational Database System
P. Dadam, V. Lum, H. Werner
Proceedings of VLDB, 1984.

ref. 6 - Version Support for engineering Database Systems
K. Dittrich, R. Lorie
IBM Research, 1985.

ref. 7 - Data Model Issues for Object - Oriented Applications
Jay Banerjee, Hong-Tai-Chou, Jorge F. Garza, Darrel Woelk, Nat Ballou, Hyoung-Joo Kim
ACM Transactions on Office Informations Systems, 1987.

ref. 8 - Combining Optimistic and Pessimistic Concurrency Mechanisms in a Shared, Persistent Object Base
D.J. Penny, J. Stein, D. Maier
Workshop on Persistent Object Stores, 1987.

ref. 9 The Management of Changing Types in an Object Oriented Database
A. H. Skarra, S. B. Zdonik
OOSPLA'86.

ref. 10 Type Evolution in an Object oriented Database
A. H. Skarra, S. B. Zdonik
Research directions in Object Oriented Programming

ref. 11 - Class Modification in the Gemstone Object-Oriented DBMS
D. Jason Penney, Jacob Stein
OOSPLA'87.

ref. 12 - A Strongly typed,interactive Object-Oriented Databased Programming Language
A. Albano, G. Ghelii, M.E. Occhiuto, R. Orsini
IEEE SOFTWARE'86.

ref. 13 An extensible Database Management System
D. S. Batory, J.R. Barnett, J. F. Garza, K. P. Smith, K. Tsukuda, B. C. Twitchell, T. E. Wise
IEEE Transactions on Software Engineering, 1988

ref. 14 - An the EXODUS Extensible DBMS Proyect
M. J. Carey, D. J. Dewitt, G. Graefe, D. M. Haight, J.E. Richardson, D. T. Schuh, E. J. Shekita, S. L. Vandenberg.

ref. 15 - Abstraction Mechanisms in CLU
B. Liskov, A. Snyder, R. Atkinson, C. Schaffert.

ref. 16 Object Identidy
S. Khoshafian, G. P. Copeland.

ref. 17 - Encapsulation and Inheritance in Object Oriented Programming Languages
A. Snyder

ref. 18 – Persistent Object Management System
F.W. Cockshott, M. P. Atkinson, K. J. Chisholm, P. J. Bailey, R. Morrison.

ref. 19 – Database Support for Versions and Alternatives of Large Design Files
R. Katz, T. Lehman
IEEE Transactions on SE, 1984.

ref. 20 – Object-Oriented Database Systems
Francois Bancilhon
ACM'88.

ref. 21 – Features of Languages for the Development of Information Systems at the Conceptual Level
Alexander Borgida
IEEE SOFTWARE'85.

– Fundamentals of Object Oriented Databases
Dave Maier, Stan Zdonik
1990.

– TURBO PASCAL 4.0.

– DBASE III PLUS

APENDICE

MANUAL del USUARIO

INDICE

INTRODUCCION

Este manual está dirigido a aquellos lectores que posean una leve noción de **programación orientada a objetos.**

En la parte principal del presente, detallamos cómo puede utilizar este prototipo para crear y aprovechar su aplicación, explicando cada una de las operaciones provistas.

A continuación, se adjunta una lista de los códigos de error previstos mientras se encuentre operando dentro del sistema.

Por último, se mencionan algunos conceptos básicos del paradigma de objetos utilizados, en un glosario final.

MODO DE OPERACION DEL SISTEMA ABDO

1. Arranque del sistema

Desde el prompt donde tenga instalado el sistema, debe tipear la palabra **ABDO** y luego dar *ENTER*:

C:>ABDO

Inmediatamente se despliega la pantalla con el **menú inicial**, que nos introduce en el tratamiento de aplicaciones (figura 1). En la sección siguiente se desarrolla ésto detenidamente.

2. Aplicaciones

Una aplicación es el subsistema que el usuario debe definir para trabajar independientemente de otros subsistemas.

2.1. Crear una aplicación

Ud. debe ingresar un nombre de 8 caracteres a lo sumo, como nombre de la aplicación (figura 2). Dicho nombre debe comenzar con letra y no debe existir. Para verificar la existencia de la aplicación utilice la opción 4 del menú inicial.

Luego de creada la aplicación Ud. debe seleccionar

A B D O

MENU INICIAL

1 .- CREAR APLICACION

2 .- EDITAR / MODIFICAR APLICACION

3 .- ELIMINAR APLICACION

4 .- LISTAR APLICACIONES EXISTENTES

0 .- SALIR DEL SISTEMA

Seleccione Opción ===>

figura 1 : *Menú Inicial del Sistema*

CREAR UNA APLICACION

Ingrese nombre APLICACION ==> alumnos_

Q -> SALIR

figura 2 : *Crear Aplicación*

la opción 2 del menú inicial, para trabajar con ella.

2.2. Editar / Modificar una aplicación

Ud. debe ingresar el nombre de una aplicación existente (figura 3). Se selecciona esta opción para trabajar con una aplicación ya definida.

Si no encuentra un nombre de aplicación válido, puede tipear *Q* para abandonar la operación y retornar al menú inicial.

Nótese que la aplicación es la creada en el paso anterior (opción 1 del menú inicial) si Ud. acaba de generar la nueva aplicación.

Se despliega entonces el menú de Modificación de aplicaciones (figura 4), cuyas opciones serán ampliamente explicadas en las secciones siguientes.

2.3. Eliminar una aplicación

Ud. debe ingresar el nombre de una aplicación existente; el sistema le pide confirmar la operación (figura 5). Si Ud. la confirma, dicha aplicación es eliminada del sistema.

Si lo que desea eliminar es la aplicación recientemente creada, basta con abandonarla sin salvar cambios (ver sección 5).

2.4. Listar aplicaciones

Por esta opción del menú inicial se despliegan los nombres de todas las aplicaciones existentes en el sistema (figura 6).

Nótese que si la aplicación fue creada en esta sesión de trabajo, no figurará en esta lista.

MODIFICAR UNA APLICACION

Ingrese nombre APLICACION ==> alumnos_

Q -> SALIR

figura 3 : *Editar / Modificar una Aplicación*

ALUMNOS

MODIFICACIONES DE APLICACIONES

MENU PRINCIPAL

1 .- CREACION DE CLASES
2 .- MODIFICACION DE CLASES
3 .- CONSULTAS SOBRE CLASES
4 .- CREACION DE OBJETOS
5 .- EDICION DE OBJETOS
6 .- ELIMINACION DE OBJETOS
7 .- CONSULTAS SOBRE OBJETOS

0 .- SALIR DEL MENU DE MODIFICACIONES

Seleccione Opción ===>

figura 4 : *Menú Modificación de Aplicaciones*

```
                    ELIMINAR   UNA   APLICACION

Ingrese nombre APLICACION ==>       alumnos_



Confirma baja de aplicación?...(S/N)
```

figura 5 : *Eliminar una Aplicación*

```
                APLICACIONES   EXISTENTES   EN   EL   SISTEMA

AP
APLI
ALUMNOS

Presione cualquier tecla para continuar ...
```

figura 6 : *Listar Aplicaciones*

3. Clases

3.1. Creación de clases

Ud. debe ingresar un nombre de 8 caracteres a lo sumo, que debe comenzar con letra y no contener blancos intermedios. El nombre de la clase no debe existir en la aplicación.

Si no es posible encontrar un nombre de clase válido, puede tipear *Q* en su lugar, para abandonar la operación y retornar al menú de Modificación de aplicaciones.

Para completar la creación de la clase se deben definir sus superclases y atributos.

Definición de superclases

Ud. debe informar si desea o no especificar superclases (figura 7).

Si decide especificar superclases debe hacerlo una a una. Cada superclase debe existir en la aplicación.

Las superclases ingresadas no deben tener relación de subclase - superclase entre ellas.

Si no es posible ingresar una superclase válida puede abandonar la operación tipeando *Q* en lugar del nombre de superclase, recomenzando el alta de clases.

Para finalizar la lista de superclases, debe dar *ENTER* sin tipear ningún nombre de superclase.

A continuación se despliega la pantalla para ingresar los atributos que conformarán la descripción de clase.

Nota:
Si el usuario no especifica superclase, se le asigna la clase RAIZ. En nuestro sistema no existen clases aisladas.

Definición de atributos

Por cada atributo, Ud. debe ingresar el nombre del atributo además de otras características (figura 8):

- **nombre:** es una cadena de 15 caracteres, como máximo, sin blancos intermedios y que debe comenzar con letra. Debe ser único en la clase.

Si no es posible encontrar un nombre de atributo válido, puede tipear *Q* en su lugar para abandonar la operación y recomenzar el alta de clases.

tipo: puede ser un tipo predefinido o una clase definida por el usuario, existente. El tipo debe ser subclase

ALUMNOS

CREACION DE CLASES

INGRESE NOMBRE DE CLASE ==> estud___

Tiene superclases? (S/N)

figura 7 : *Definición de Superclases*

CREACION DE CLASES

CLASE: ESTUD

Ingrese ATRIBUTOS:

Atributo	Tipo	Tam	Cód	Valor
nro_alumno_____	integer_		_	

Confirma Alta de Atributo NRO_ALUMNO? (S/N)

figura 8 : *Definición de Atributos*

del tipo del atributo con este nombre en cada una de las superclases, si existiera tal atributo.

Si da *ENTER* sin entrar ningún tipo, puede recomenzar la definición del atributo.

- **tamaño:** es la longitud,de hasta 50 caracteres, que debe especificarse para el tipo STRING.

- **código:** debe ingresar *C* o *D* si el atributo tiene valor compartido o por defecto respectivamente y debe ingresar un **valor** de acuerdo al tipo y tamaño del atributo. En caso de no desear especificar un código al atributo, dar *ENTER*.

* valor compartido: significa que el atributo tendrá ese valor en todas las instancias de la clase.

* valor por defecto: significa que el atributo tendrá ese valor en todas las instancias de la clase a menos que Ud. especifique otro.

Para mejor información acerca del ingreso del valor, ver sección 4.1.

Una vez completada la definición del atributo, Ud. deberá confirmar el alta del atributo . Si no confirma, continúa con el ingreso del siguiente atributo ignorando la definición anterior.

Para finalizar la entrada de atributos, debe dar *ENTER* sin especificar nombre de atributo.

Luego se despliega una pantalla donde Ud. debe especificar un atributo como **clave** entre los atributos definidos (figura 9). Si no es posible puede abandonar la operación tipeando *Q* en lugar del nombre de atributo clave, recomenzando el alta de clases.

Posteriormente, debe confirmar el alta de la clase (figura 9). Si no confirma, continúa con el ingreso de una nueva clase ignorando la clase que acaba de definir.

Como consecuencia del alta de clase, pasan a heredarse atributos de las superclases. En caso de *conflicto de herencia* de un atributo, se le permite a Ud. decidir de qué clase desea heredar el atributo.

Concluída la operación Ud. recibirá un mensaje de *Creación de clase efectuada* y podrá continuar con la creación de nuevos objetos hasta tanto tipee *Q* en lugar del nombre de clase, para retornar al menú de modificaciones de aplicaciones.

CREACION DE CLASES

LASE: ESTUD

Superclases: PERSONA

Atributos definidos:

NRO_ALUMNO	FACULTAD	CARRERA	MATERIAS_APROB
PROMEDIO			

Seleccione un atributo como CLAVE ==> nro_alumno_____

onfirma Alta de la clase ESTUD? (S/N)

figura 9 : *Confirmación Alta de clases*

LUMNOS

MODIFICACIONES DE CLASES

MENU DE CAMBIOS DE ESQUEMA

1 .- CAMBIOS EN EL CONTENIDO DE UNA CLASE

2 .- CAMBIOS EN LA JERARQUIA DE CLASES

3 .- CAMBIOS EN UNA CLASE DEL ESQUEMA

0 .- SALIR DEL MENU

Seleccione Opción ===>

figura 10 : *Menú de Cambios de esquema*

3.2. Modificación de clases

En esta sección se describen las acciones a seguir para modificar el esquema de clases de nuestro sistema.

Como las instancias están ligadas a la definición de la clase, cualquier cambio en ésta provoca que se adapten las instancias existentes a la nueva definición.

Al modificar una clase, los cambios se propagan a todas las clases involucradas (subclases de la clase en todos sus niveles).

Hay tres grupos de cambios posibles (figura 10):

- modificaciones en el contenido de una clase,
- modificaciones en la estructura jerárquica de clases,
- modificaciones de una clase en la jeraquia.

Pasaremos a detallar cada uno de estos grupos de operaciones.

3.2.1. Modificaciones en el contenido de una clase

En esta sección describimos aquéllos cambios que afectan a los atributos de una clase (figura 11).

3.2.1.1. Agregar un atributo

Ud. debe ingresar el nombre de la clase a modificar. Si la clase existe, debe ingresar el nombre del atributo a agregar, que no debe estar definido en la clase. Debe informar las características del atributo (figura 12): tipo, tamaño y código, que deben ajustarse a las normas enunciadas en la sección 3.1.- definición de atributos - .

Además el tipo debe ser superclase del tipo del atributo propio con el mismo nombre en cada una de las subclases, si existiese.

Si no es posible ingresar un atributo válido, puede abandonar la operación tipeando @ en lugar del nombre de atributo, recomenzando la operación.

Si el atributo existía en la clase heredado de una superclase, como consecuencia de la operación pueden perderse valores en las instancias, tanto de la clase modificada como de las subclases que heredan este atributo por medio de esta clase. En este caso el sistema envía un mensaje informando la situación y Ud. decide si continúa o no con la operación, con las consecuencias mencionadas.

LUMNOS

MODIFICACIONES DE CLASES

CAMBIOS EN EL CONTENIDO DE UNA CLASE

1 .- AGREGAR UN ATRIBUTO

2 .- ELIMINAR UN ATRIBUTO

3 .- CAMBIAR TIPO DE UN ATRIBUTO

4 .- CAMBIAR ATRIBUTO CLAVE

5 .- CAMBIAR HERENCIA DE UN ATRIBUTO

6 .- CAMBIAR VALOR POR DEFECTO

7 .- CAMBIAR VALOR COMPARTIDO

8 .- CONVERTIR UN ATRIBUTO EN COMPARTIDO

9 .- CONVERTIR UN ATRIBUTO EN NO COMPARTIDO

0 .- SALIR DEL MENU

Seleccione Opción ===>

figura 11 : *Menú de Modificaciones de atributos*

ıLUMNOS

AGREGAR UN ATRIBUTO A UNA CLASE

Ingrese Nombre de CLASE ==> estud___

Ingrese nuevo ATRIBUTO:

ıtributo Tipo Tam Cód Valor

'echa_ingreso__ string__ 8_ _

Q -> Salir

figura 12 : *Agregar un atributo*

En los objetos de las clases involucradas en la operación, el atributo agregado contendrá valor nulo.

Si desea agregar un atributo clave debe realizar esta operación y seguidamente efectuar el cambio de clave (operación descripta en sección 3.2.1.5.).

Concluída la operación Ud. recibirá un mensaje de *modificación realizada* y podrá continuar realizando cambios de esta naturaleza hasta tanto tipee *Q* en lugar del nombre de clase a modificar.

3.2.1.2. Eliminar un atributo

Ud. debe ingresar el nombre de la clase a modificar. Si la clase existe, debe ingresar el nombre del atributo a eliminar,que debe estar definido en la clase y no ser el atributo clave.

Si no fuera posible ingresar un atributo válido puede abandonar la operación tipeando *Q* en lugar del nombre de atributo.

Si Ud. desea eliminar el atributo clave, primero debe hacer la operación cambio de clave (explicada en la sección 3.2.1.5.) y luego sí eliminarlo.

Si el atributo eliminado pasa a ser heredado,en caso de *conflicto de herencia* el sistema pide su intervención para resolverlo.

Puede ser que los valores del atributo eliminado, en las instancias existentes no puedan asignarse a la nueva definición del atributo, ahora heredado. En este caso se le avisa y Ud. debe decidir continuar o no con la operación, ateniéndose a las consecuencias de la decisión (figura 13).

Puede ser que las subclases que heredaban el atributo eliminado, lo hereden ahora de otra superclase. Entonces ante un *conflicto de herencia*, Ud. debe decidir.

Si las subclases tienen definido el atributo, o lo heredan, y no a través de la clase modificada, la operación no las afecta.

Concluída la operación Ud. recibirá un mensaje de *modificación realizada* y podrá continuar realizando cambios de esta naturaleza hasta tanto tipee *Q* en lugar del nombre de clase a modificar.

ALUMNOS

ELIMINAR UN ATRIBUTO DE UNA CLASE

Ingrese Nombre de CLASE ==> **catedra_**

Ingrese Nombre del ATRIBUTO a eliminar ==> titular_____

Se perderán referencias o objetos ... Confirma la operación? (S/N)

Q -> Salir

figura 13 : *Eliminación de atributos*

ALUMNOS

CAMBIAR TIPO A UN ATRIBUTO

Ingrese Nombre de CLASE a modificar ==> _____

No se permite perder objetos (39)

Q -> Salir

figura 14 : *Cambiar tipo de un atributo*

3.2.1.3. Cambiar tipo de un atributo

Ud. debe ingresar el nombre de la clase a modificar. Si la clase existe, debe ingresar el nombre del atributo a cambiar el tipo, que debe estar definido en la clase.

Si no fuera posible ingresar un nombre de atributo válido, puede abandonar la operación tipeando *Q* en lugar del nombre de atributo.

Seguidamente debe ingresar el nuevo tipo para el atributo, que debe ser una clase existente definida por el usuario o un tipo predefinido.

El nuevo tipo debe ser subclase de los tipos de atributos con ese mismo nombre en cada una de las superclases de la clase a modificar. Además debe ser superclase del tipo anterior del atributo.

Si el atributo es clave sólo puede cambiarse el tipo si es predefinido.

Si los valores del atributo en las instancias existentes de la clase no pueden adaptarse a la nueva definición de tipo, se rechaza la operación.

Si como consecuencia de la operación se pueden perder referencias a objetos del nuevo tipo, se le informa a Ud. para que decida continuar o no con la operación (figura 14). Damos un ejemplo de ello a continuación.

La clase *estudiante* hereda el atributo *documento* de su superclase *persona*. Supongamos que hacemos una operación cambio de tipo de *estudiante* a *persona*.

El valor del atributo en las instancias existentes tiene que pasar de ser una referencia a un objeto *estudiante* a ser una referencia a un objeto *persona*.

Si en algún objeto *estudiante* referenciado desde la clase modificada, el atributo *documento* tiene valor nulo, no podría obtenerse ninguna referencia correspondiente a un objeto *persona*. En este caso le consultamos, ya que se trata siempre de no perder valores en las instancias.

Si no fuera posible ingresar un tipo válido para el atributo, puede abandonar la operación tipeando *Q* en lugar del tipo.

Concluída la operación Ud. recibirá un mensaje de *modificación realizada* y podrá continuar realizando cambios de esta naturaleza hasta tanto tipee *Q* en lugar del nombre de clase a modificar.

3.2.1.4. Cambiar herencia de un atributo

Ud. debe ingresar el nombre de la clase a modificar. Si la clase existe, debe ingresar el nombre del atributo a cambiar la herencia, que debe estar heredado en la clase.

Si no fuera posible ingresar un nombre de atributo válido, puede abandonar la operación tipeando *Q* en lugar del nombre de atributo.

Seguidamente debe ingresar la nueva herencia para el atributo, que debe ser una clase existente y ser superclase inmediata de la clase a modificar.

Si no fuera posible ingresar una herencia válida, válido, puede abandonar la operación tipeando *Q* en lugar del nombre de la nueva herencia.

En las instancias de las clases involucradas en esta operación, al atributo en cuestión se le asigna valor nulo.

Concluída la operación Ud. recibirá un mensaje de *modificación realizada* y podrá continuar realizando cambios de esta naturaleza hasta tanto tipee *Q* en lugar del nombre de clase a modificar.

3.2.1.5. Cambiar atributo clave

Ud. debe ingresar el nombre de la clase a modificar. Si la clase existe, debe ingresar el nombre del atributo candidato a ser clave, que debe estar definido en la clase.
Tal atributo no puede ser de tipo BOOLEAN ni tener código D o C (atributo con valor por defecto o valor compartido, respectivamente).

Si en alguna instancia existente de la clase, el atributo tiene valor nulo o si el mismo valor para tal atributo figura en más de una instancia, se rechaza la operación (figura 15).

Si no fuera posible ingresar un nombre de atributo válido, puede abandonar la operación tipeando *Q* en lugar del nombre de atributo.

Concluída la operación Ud. recibirá un mensaje de *modificación realizada* y podrá continuar realizando cambios de esta naturaleza hasta tanto tipee *Q* en lugar del nombre de clase a modificar.

```
ALUMNOS

                        CAMBIAR  CLAVE  A  UNA  CLASE
-------------------------------------------------------------------------------

    Ingrese Nombre de CLASE a modificar ==>        estud___

    Ingrese ATRIBUTO candidato a ser Clave  ==>  _______________

 DNI origina instancias con Claves Duplicadas (35)
-------------------------------------------------------------------------------
                                                                Q -> Salir
```

figura 15 : *Cambiar atributo Clave*

```
ALUMNOS

                  CAMBIAR VALOR POR DEFECTO A UN ATRIBUTO
-------------------------------------------------------------------------------

    Ingrese Nombre de CLASE  ==>       persona_

    Ingrese ATRIBUTO por defecto a modificar ==> nacionalidad___

      <VALOR POR DEFECTO>    argentina

    Ingrese nuevo VALOR  ==> ____________________________________________

    ┌──────────────────────────┐
    │Tipo del Atributo: STRING │
    │Tamaño del Atributo: 15   │
    └──────────────────────────┘

-------------------------------------------------------------------------------
                                                                Q -> Salir
```

figura 16 : *Cambiar valor por defecto a un atributo*

Nota:
Se aconseja evitar el uso de esta operación, pues degrada la performance del sistema.

3.2.1.6. Cambiar valor por defecto de un atributo

Ud. debe ingresar el nombre de la clase a modificar. Si la clase existe, debe ingresar el nombre del atributo al que se le cambiará el valor por defecto. Tal atributo debe estar definido en la clase y como atributo con valor por defecto.

Si no fuera posible ingresar un nombre de atributo válido, puede abandonar la operación tipeando *@* en lugar del nombre de atributo.

Seguidamente debe ingresar el valor, que debe cumplir las restricciones que cumplía el valor anterior (figura 16).

El nuevo valor se asigna a cada futura instancia de la clase a no ser que Ud. asigne uno explícitamente. En las instancias existentes de la clase los valores del atributo no se modifican, pues, aunque el atributo tenga valor por defecto, puede haber instancias que tengan distintos valores asignados.

Concluída la operación Ud. recibirá un mensaje de *modificación realizada* y podrá continuar realizando cambios de esta naturaleza hasta tanto tipee *@* en lugar del nombre de clase a modificar.

3.2.1.7. Cambiar valor compartido de un atributo

Ud. debe ingresar el nombre de la clase a modificar. Si la clase existe, debe ingresar el nombre del atributo al que se le cambiará el valor compartido. Tal atributo debe estar definido en la clase y como atributo con valor compartido.

Si no fuera posible ingresar un nombre de atributo válido, puede abandonar la operación tipeando *@* en lugar del nombre de atributo.

Seguidamente debe ingresar el valor, que debe cumplir las restricciones que cumplía el valor anterior (figura 17).

El nuevo valor se asigna a cada futura instancia de

la clase y se modificará en todas las instancias existentes.

Concluída la operación Ud. recibirá un mensaje de *modificación realizada* y podrá continuar realizando cambios de esta naturaleza hasta tanto tipee *Q* en lugar del nombre de clase a modificar.

3.2.1.8. Convertir un atributo en compartido

Ud. debe ingresar el nombre de la clase a modificar. Si la clase existe, debe ingresar el nombre del atributo al que se convertirá en compartido. Tal atributo debe estar definido en la clase ,no ser atributo con valor compartido y no ser clave de la clase.

Si no fuera posible ingresar un nombre de atributo válido, puede abandonar la operación tipeando *Q* en lugar del nombre de atributo.

Seguidamente debe ingresar el valor, que debe cumplir las restricciones que cumplía el valor anterior.

El nuevo valor se asigna a cada futura instancia de la clase y se modificará en todas las instancias existentes, perdiendo el anterior (por definición de atributo con valor compartido, en todos los objetos el atributo debe tomar el mismo valor).

Concluída la operación Ud. recibirá un mensaje de *modificación realizada* y podrá continuar realizando cambios de esta naturaleza hasta tanto tipee *Q* en lugar del nombre de clase a modificar.

3.2.1.9. Eliminar propiedad de valor compartido de un atributo

Ud. debe ingresar el nombre de la clase a modificar. Si la clase existe, debe ingresar el nombre del atributo que dejará de ser compartido. Tal atributo debe estar definido en la clase y como atributo con valor compartido.

Si no fuera posible ingresar un nombre de atributo válido, puede abandonar la operación tipeando *Q* en lugar del nombre de atributo.

En las futuras instancias de la clase, Ud. puede asignarle valores a este atributo como a cualquier otro de la clase. El valor del atributo en las instancias existentes permanecen con el valor anterior, pudiendo cambiarse con la operación de edición.

```
-UMNOS

                    CAMBIAR VALOR COMPARTIDO A UN ATRIBUTO

  Ingrese Nombre de CLASE  ==>       estud___

  Ingrese ATRIBUTO compartido a modificar ==>  universidad____

    <VALOR COMPARTIDO>     UNLP

  Ingrese nuevo VALOR  ==> ____________________________________________

    Tipo del Atributo: STRING
    Tamaño del Atributo: 20

                                                        Q -> Salir
```

figura 17 : *Cambiar valor compartido a un atributo*

```
LUMNOS

                    MODIFICACIONES DE CLASES

                CAMBIOS EN LA JERARQUIA DE CLASES

          1 .- ASIGNAR UNA CLASE COMO SUPERCLASE DE OTRA

          2 .- ELIMINAR UNA CLASE COMO SUPERCLASE DE OTRA


          0 .- SALIR DEL MENU



               Seleccione Opción ===>
```

figura 18 : *Menú de modificaciones en la jerarquía*

Concluída la operación Ud. recibirá un mensaje de *modificación realizada* y podrá continuar realizando cambios de esta naturaleza hasta tanto tipee @ en lugar del nombre de clase a modificar.

3.2.2. Modificaciones en la jerarquía de clases

En esta sección describimos aquéllos cambios que afectan la estructura reticulada del sistema y la relación entre las clases (clase superclase). (figura 18).

3.2.2.1. Asignar una clase como superclase de otra

Ud. debe ingresar el nombre de la clase a modificar. Si la clase existe, debe ingresar el nombre de la clase candidata a ser superclase de la primera. Tal clase debe existir y no ser subclase ni superclase (inmediata o de niveles superiores) de la clase a modificar.

El tipo de los atributos de la superclase agregada debe ser superclase del tipo de atributos con el mismo nombre, si existiesen, en las clase modificada. De no ser así, la operación se rechaza, por no cumplir las restricciones de tipos ya mencionadas (figura 19) .
Si no fuera posible ingresar un nombre de clase válido, puede abandonar la operación tipeando @ en lugar del nombre de superclase.

Esta operación causará que la clase modificada y las subclases involucradas hereden aquéllos atributos de la superclase que no existían hasta ese momento. Los atributos de la superclase agregada que existían en la clase modificada o en las subclases involucradas, no se heredan, pues toda definición en una clase tiene precedencia sobre la definición del mismo atributo en una superclase.

En las instancias existentes de las clases afectadas por la operación, se asignan valores nulos para los nuevos atributos.

Concluída la operación Ud. recibirá un mensaje de *modificación realizada* y podrá continuar realizando cambios de esta naturaleza hasta tanto tipee @ en lugar del nombre de clase a modificar.

3.2.2.2. Eliminar una clase como superclase de otra

Ud. debe ingresar el nombre de la clase a modificar. Si la clase existe, debe ingresar el nombre de la superclase de ésta que desea eliminar. Tal clase debe existir y ser superclase inmediata de la clase a modificar.

Si no fuera posible ingresar un nombre de superclase válido, puede abandonar la operación tipeando *Q* en lugar del nombre de superclase.

Esta operación no implica eliminar la superclase como una clase del esquema.

El efecto de esta operación es eliminar la superclase ingresada, como superclase de la clase a modificar y agregarle cómo nuevas superclases, a aquéllas superclases de la superclase eliminada que no lo eran todavía.

Los atributos heredados de la superclase eliminada pueden perderse, y por lo tanto, perderse sus valores en las instancias correspondientes, o heredarse de alguna otra clase.

Ud. puede tener que resolver un nuevo conflicto de herencia para los atributos afectados (figura 20). Cada vez que resuelve un conflicto de herencia, el sistema trata de propagar su decisión a las subclases. En las instancias existentes se *convierten* los valores o se asignan valores nulos, según la relación entre los tipos de los atributos. Similarmente ésto ocurre en las subclases involucradas.

La superclase eliminada ya no tendrá la la clase modificada como una de sus subclases.

Concluída la operación Ud. recibirá un mensaje de *modificación realizada* y podrá continuar realizando cambios de esta naturaleza hasta tanto tipee *Q* en lugar del nombre de clase a modificar.

3.2.3. Modificaciones en una clase de la jerarquía

En esta sección describimos aquéllos cambios que afectan a las clases como nodos de la jerarquía, es decir que involucran a una clase individualmente (figura 21).

ALUMNOS

ASIGNAR UNA SUPERCLASE A UNA CLASE

Ingrese Nombre de CLASE a modificar ==> estud___

Ingrese Nombre de SUPERCLASE a incorporar ==> _____

DNI no cumple restricción de tipo en la superclase PERSONA (44)

Q -> Salir

figura 19 : *Asignar Superclase a una clase*

ALUMNOS

ELIMINAR UNA SUPERCLASE A UNA CLASE

El atributo CODIGO cambia de herencia en la clase AYUD_REN

De dónde desea HEREDAR el Atributo CODIGO ?

1- AYUDANTE
2- DOCENTE

Ingrese una opción ==>

---

figura 20 : *Eliminar Superclase a una clase*

3.2.3.1. Eliminar una clase

Ud. debe ingresar el nombre de la clase a eliminar. Tal clase debe existir.

Si la clase que se desea eliminar es tipo de algún atributo en alguna clase del esquema, la operación se rechaza hasta tanto se solucione ese problema. Se despliega una lista con los nombres de todas las clases afectadas (figura 22).

Todas las clases que tenían a la eliminada como superclase, dejan de tenerla, al igual que todas las clases que la tenían como subclase.

Puede traer como consecuencia la pérdida de atributos o el cambio de herencia de alguno de ellos, lo que podría originar posiblemente la intervención del usuario para resolver el conflicto de herencia.

Se elimina toda la información sobre la clase.

Concluída la operación Ud. recibirá un mensaje de *modificación realizada* y podrá continuar realizando cambios de esta naturaleza hasta tanto tipee *Q* en lugar del nombre de clase a eliminar.

3.2.3.2. Cambiar el nombre de una clase

Ud. debe ingresar el nombre de la clase a modificar. Si existe, debe ingresar el nuevo nombre para tal clase, que debe comenzar con letra, no debe contener blancos intermedios y tener a lo sumo 8 caracteres de longitud. El nuevo nombre no debe existir en la aplicación.

Si no fuera posible ingresar un nombre de clase válido, puede abandonar la operación tipeando *Q* en lugar del nombre nuevo.

Como consecuencia de esta operación, se modifica la información de superclases para cada subclase inmediata de la clase modificada.

En los atributos de cada clase del esquema que tengan como tipo a la clase modificada, se renombran tales tipos.

El resultado de esta operación no afecta las instancias existentes de las clases involucradas.

Concluída la operación Ud. recibirá un mensaje de *modificación realizada* y podrá continuar realizando cambios de

ALUMNOS

MODIFICACIONES DE CLASES

CAMBIOS EN UNA CLASE DEL ESQUEMA

1 .- ELIMINAR UNA CLASE

2 .- CAMBIAR NOMBRE A UNA CLASE

0 .- SALIR DEL MENU

Seleccione Opción ===>

figura 21 : *Menú Cambios en una clase*

ALUMNOS

ELIMINAR UNA CLASE

En las siguientes clases existen atributos **con** tipo **= DOCENTE :**

AREA

DOCENTE es tipo **de** atributos en otras **clases (45)**

Presione **cualquier** tecla para continua**r...**

figura 22 : *Eliminar una clase*

esta naturaleza hasta tanto tipee *Q* en lugar del nombre de clase a renombrar.

3.2.3.3. Agregar una clase

Esta operación no forma parte del menú de operaciones del sistema. Su efecto se logra definiendo una clase nueva (sección 3.1).

La nueva clase se agrega sin subclases en el esquema, es decir, como una clase terminal de la estructura reticulada. Si Ud. quisiera asignarle subclases, debe usar la operación descripta en la sección 3.2.2.1., donde la superclase agregada para cada subclase es la clase que se agregó.

3.3. Consultas sobre clases

Esta sección se ocupa de describir el tratamiento a seguir para acceder a la información concerniente a las clases de una aplicación (figura 23). La consulta puede ser impresa o visualizarse por pantalla.

3.3.1. Listar todas las clases de la aplicación

Al seleccionar esta opción del menú de consultas sobre clases, se despliegan los nombres de todas las clases creadas en la aplicación (figura 24). Utilice las flechas *arriba* y *abajo* del cursor para navegar en la lista de clases. Usted puede seleccionar la clase resaltada dando *ENTER* para obtener por pantalla o impresora, según su elección, información completa sobre esa clase (superclases, atributos propios y heredados). Ver figura 25.

Para finalizar la consulta por pantalla debe presionar la tecla *ESC*. Asi retorna a la lista de clases existentes (figura 24) para seleccionar otra o bien retornar al menú de consultas presionando *ESC*.

3.3.2. Listar información de una clase

Por esta opción deberá ingresar el nombre de una clase existente. Si esto no es posible, ingrese *Q* en lugar de nombre de clase para abandonar la operación.

Debe responder si desea la salida impresa o por

ALUMNOS

CONSULTAS

CONSULTAS SOBRE CLASES

1 .- LISTAR TODAS LAS CLASES DE LA APLICACION

2 .- LISTAR INFORMACION SOBRE UNA CLASE

3 .- LISTAR SUBCLASES DE TODAS LAS CLASES

4 .- LISTAR SUBCLASES DE UNA CLASE

0 .- SALIR DEL MENU DE CONSULTAS

Seleccione Opción ===>

figura 23 : *Menú Consultas sobre clases*

ALUMNOS

CONSULTAS SOBRE CLASES

CLASES EXISTENTES:

PERSONA
DOCENTE
AYUDANTE
AYUD_REN
ESTUD
AREA
CATEDRA

ENTER-> Selec Clase | ↓ -> Clase Sgte | ↑ -> Clase Ant | ESC-> Salir

figura 24 : *Clases existentes en una aplicación*

pantalla. En cualquier caso obtendrá información completa (superclases, atributos propios y heredados) de la clase ingresada (figura 25).

Para terminar la consulta por pantalla debe presionar *ESC*. Luego puede seguir ingresando nombres de clase a consultar o bien tipear *Q* para retornar al menú de consultas.

3.3.3. Listar subclases de todas las clases

Por esta opción podrá visualizar las subclases de cada clase de la aplicación. En caso de optar por salida por pantalla, se irán desplegando las clases de a una por vez, con sus respectivas subclases (figura 26). Podrá presionar *ENTER* para visualizar la siguiente o bien *ESC* para retornar al menú de consultas.

3.3.4. Listar subclases de una clase

Se debe ingresar el nombre de una clase existente. De no ser posible ingrese *Q* en lugar del nombre de clase para abandonar la operación.

Debe responder si desea salida impresa o por pantalla. En cualquier caso, obtendrá información sobre cuáles son las subclases de la clase ingresada (figura 26). En caso de visualización por pantalla, debe terminar la consulta presionando *ESC*. Luego puede seguir consultando subclases o tipear *Q* para abandonar la opción.

CLASE: ESTUD

Superclases: RAIZ

Atributo	Tipo	Tam	Valor	Cód
NRO_ALUMNO	INTEGER			
FACULTAD	STRING	15		
CARRERA	STRING	20		
MATERIAS_APROB	INTEGER			
PROMEDIO	REAL			
FECHA_INGRESO	STRING	8		
CODIGO	STRING	2		
UNIVERSIDAD	STRING	20	UNLP	C
DNI	REAL			

ESC -> Retorna a pantalla anterior

figura 25 : *Listar Definición de una clase*

ALUMNOS

CONSULTAS SOBRE CLASES

SUBCLASES de la Clase -> PERSONA

DOCENTE
AYUDANTE

ENTER -> Clase Sgte ESC -> Salir

figura 26 : *Listar subclases de una clase*

4. Objetos

4.1 Creación de objetos

Crear un objeto es instanciar una clase, es decir, asignarle valores a cada atributo que la componen.

Ud. debe ingresar el nombre de una clase para instanciar. Dicho nombre debe existir en la aplicación.
Si no es posible encontrar un nombre de clase válido puede ingresar *@* en su lugar para abandonar la operación y retornar al menú de modificaciones de aplicaciones.

Se provee una pantalla por atributo para ingresar su valor (figura 27).
En el caso de ser un atributo con **tipo predefinido** se brinda una *ayuda* con tipo y tamaño del atributo.
El valor ingresado se restringe a los siguientes valores:

* para el tipo INTEGER: -21474836747<valor<2147483647
* para el tipo REAL: -9.9E-36<valor<9.9E+36
* para el tipo BOOLEAN: valor = T o F
* para el tipo STRING: longitud del valor debe ser <= tamaño especificado en la definición del atributo.

En el caso de ser un atributo con **tipo definido por el usuario** se brinda *ayuda* (figura 27) con tipo del atributo y además tipo y tamaño de la clave de la clase que tiene como tipo, pues el valor del atributo será una referencia a un objeto del tipo (ese valor es un valor de la clave).

Ejemplo:
Si un atributo *x* tiene como tipo a la clase *persona* cuya clave es *documento*, el valor ingresado corresponderá a un valor del atributo *documento*. La ayuda proveerá el tipo del atributo *documento*. Dicho valor hará referencia a un objeto de la clase *persona*.
La referencia debe apuntar a un objeto existente. Si Ud. desconoce el objeto puede dar *ENTER* y dejarle así un valor nulo. Luego de creado el objeto, puede editarlo y completar la referencia.

El valor del atributo que se está cargando puede ser nulo (dando *ENTER*) salvo que sea el atributo clave de la clase.
El valor asignado a un atributo clave no puede coincidir con el de otro objeto existente, pues no se permiten claves duplicadas.

Si no es posible asignarle un valor al atributo, se puede abandonar la operación tipeando *Q*, recomenzando la creación de objetos.

Si el atributo tiene valor por defecto, éste se muestra y Ud. puede cambiarlo tipeando uno nuevo. En caso contrario, dando *ENTER*, este valor se asigna por defecto al atributo.

Si el atributo tiene valor compartido, éste se asigna sin darle a Ud. posibilidad de modificarlo.

Una vez creado el objeto se despliegan todos los valores asignados a los atributos de la clase y se pide confirmación de la operación.

Concluida la operación Ud. recibirá un mensaje de *Creación de objeto efectuada* y podrá continuar con la creación de nuevos objetos hasta tanto tipee *Q* en lugar del nombre de clase, para retornar al menú de modificaciones de aplicaciones.

4.2. Edición de objetos

Por esta opción se podrá modificar los valores a los atributos de algún o algunos objetos de una clase determinada.

Deberá ingresar la clase a la que pertenece el objeto a modificar. De no ser posible encontrar un nombre válido de clase, podrá tipear *Q* para abandonar la operación.

Seguidamente aparecerá un menú de edición (figura 28) donde podrá escoger cambiar un objeto particular o ir visualizando todos los objetos para seleccionar aquéllos que desee editar.

Por la primera opción del menú de edición, debe ingresar el objeto que desee editar. Este objeto debe ser una instancia existente de la clase. De no ser posible encontrar una clave válida de objeto, podrá abandonar la opción tipeando *Q* en lugar del objeto.

Localizado el objeto, Ud. visualizará la pantalla de edición de tal objeto, donde se irán mostrando los atributos con su valor actual, que Ud. podrá modificar tipeando un nuevo valor, el cual será validado de acuerdo a su tipo y tamaño. Esta información se brinda como ayuda para cada atributo (figura 29).

No se permite modificar el valor del atributo clave ni un atributo con valor compartido.

ALUMNOS

CREACION DE OBJETOS

C L A S E : ESTUD

Ingrese valor para Atributo : CARRERA

==> informatica_____

Tipo: STRING Tamaño: 20

---

0 -> SALIR

figura 27 : *Creación de Objetos*

ALUMNOS

EDICION DE OBJETOS

Ingrese Nombre de CLASE ==> estud___

MENU DE EDICION
1.- EDICION DE UN OBJETO PARTICULAR
2.- EDICION DE TODOS LOS OBJETOS DE LA CLASE
0.- SALIR DE EDICION
Seleccione una opción ==>

figura 28 : *Menú de Edición de objetos*

Puede abandonar la edición del objeto tipeando *Q* en lugar del valor. Se mantienen los valores de los atributos previos al abandono de la operación.

Por la opción 2 del menú de edición (figura 28), obtendrá todos los objetos, uno a uno, de la clase especificada (figura 30). Podrá recorrer los objetos presionando la tecla *PgDn*; si posicionado en un determinado objeto, presiona la tecla *ENTER*, visualizará la pantalla de edición de un objeto, y podrá modificarlo como se mencionó en la opción 1 de este menú.

Luego de modificado un objeto, puede editar y modificar otros objetos de la misma clase hasta tanto presione *ESC*, con lo que podrá comenzar a editar objetos de otra clase.

Para finalizar la edición, no tiene más que tipear *Q* en lugar del nombre de clase.

4.3. Eliminación de objetos

Por esta opción se podrá eliminar algún o algunos objetos de una clase determinada.

Deberá ingresar la clase a la que pertenece el objeto a eliminar. De no ser posible encontrar un nombre válido de clase, podrá tipear *Q* para abandonar la operación.

Seguidamente aparecerá un menú de eliminación (figura 31) donde podrá escoger eliminar un objeto particular o ir visualizando todos los objetos para seleccionar aquéllos que desee eliminar.

Por la primera opción del menú de eliminación, debe ingresar el objeto que desee eliminar. Este objeto no debe ser referenciado desde un objeto de otra clase. De no ser posible encontrar una clave válida de objeto, podrá abandonar la opción tipeando *Q* en lugar del objeto.

Localizado el objeto, Ud. visualizará la pantalla de de tal objeto, donde se mostrarán los atributos con sus valores. Ud. debe confirmar o no la eliminación del objeto desplegado (figura 32).

Por la opción 2 del menú de eliminación (figura 31), obtendrá todos los objetos, uno a uno, de la clase especificada (figura 33). Podrá recorrer los objetos presionando la tecla *PgDn*; si posicionado en un determinado objeto, presiona la tecla *ENTER*, visualizará la pantalla de eliminación de un objeto, y podrá confirmar o no la operación

```
ALUMNOS
                                E D I C I O N
   C L A S E :    ESTUD
   NRO_ALUMNO : 100
.....................................................................
   FACULTAD : CS_EXACTAS
   CARRERA : INFORMATICA
   MATERIAS_APROB : 23
   PROMEDIO : 7.20
   FECHA_INGRESO :


Ingrese nuevo VALOR ==>  ____________________________________________

                                  ┌──────────────────┐
                                  │Tipo: STRING      │
                                  │Tamaño: 8         │
                                  │                  │
                                  └──────────────────┘
_____________________________________________________________________
ENTER -> Continuar                              Q -> Abandonar
```

figura 29 : *Edición de un objeto particular*

```
ALUMNOS
                          EDICION  DE  OBJETOS


          C L A S E :  ESTUD

     ┌──────────────────────────────────────────────────┐
     │                                                  │
     │   OBJETO:   100                                  │
     │                                                  │
     └──────────────────────────────────────────────────┘


_____________________________________________________________________
PgDn -> sgte objeto        Enter -> modificar        Esc -> Fin de edición
```

figura 30 : *Edición de todos los objetos de una clase*

```
ALUMNOS
-------------------------------------------------------------------------------
                          ELIMINACION   DE   OBJETOS
-------------------------------------------------------------------------------

    Ingrese Nombre de CLASE  ==>        ESTUD___

                  +-----------------------------------------+
                  |            MENU DE ELIMINACION          |
                  |                                         |
                  | 1.- ELIMINACION DE UN OBJETO PARTICULAR |
                  |                                         |
                  | 2.- ELIMINACION DE OBJETOS DE LA CLASE  |
                  |                                         |
                  |                                         |
                  | 0.- SALIR DE ELIMINACION                |
                  |                                         |
                  |                                         |
                  |     Seleccione una opción ==>           |
                  +-----------------------------------------+
```

figura 31 : *Menú de eliminación de un objeto*

```
ALUMNOS
                              E L I M I N A C I O N
   C L A S E :    ESTUD
.....................................................................................
   NRO_ALUMNO : 100
   FACULTAD : CS_EXACTAS
   CARRERA : INFORMATICA
   MATERIAS_APROB : 23
   PROMEDIO : 7.20
   FECHA_INGRESO :
   CODIGO :
   UNIVERSIDAD : UNLP
   DNI :




Está seguro que desea eliminar el objeto? (S/N)

-------------------------------------------------------------------------------
```

figura 32 : *Eliminación de un objeto particular*

como se mencionó en la opción 1 de este menú.

Luego de eliminado un objeto, puede continuar eliminando otros objetos de la misma clase hasta tanto presione *ESC*, con lo que podrá comenzar a eliminar objetos de otra clase.

Para finalizar la eliminación, no tiene más que tipear *Q* en lugar del nombre de clase.

4.4. Consultas sobre objetos

En esta sección describimos cómo consultar los objetos de una clase, permitiendo seleccionar atributos, objetos, clasificar por algún atributo, calcular sumas y promedios de atributos y contar objetos (figura 34). En todos los casos la consulta puede ser impresa o visualizarse por pantalla.

4.4.1 Listar

Ud. debe ingresar el nombre de una clase existente. De no ser posible, tipee *Q* para abandonar la operación.

Luego se desplegará el menú listar (figura 35), donde Ud. puede elegir si desea listar un objeto particular o varios objetos de la clase ingresada.

Si seleciona la opción 1, debe ingresar el objeto que desea consultar, el cual debe existir. De no ser posible, tipee *Q* para abandonar la opción.

Luego debe responder si desea la salida impresa o por pantalla. En cualquier caso, obtendrá información completa sobre el objeto requerido (nombre de cada uno de sus atributos con su valor correspondiente). Ver figura 36.

Si está consultando el objeto por pantalla, para visualizar un objeto referenciado por éste, utilice las flechas *arriba* y *abajo* para ubicar el atributo correspondiente, y presione *ENTER* para seleccionarlo (el atributo aparece resaltado). El atributo debe tener tipo definido por el usuario; su valor es una referencia a un objeto de su tipo. Para finalizar la visualización de este objeto, presione *ESC* y retornará al objeto principal (figura 37).

Para finalizar la consulta, presione *ESC*. Puede continuar consultando objetos individuales o tipear *Q* para comenzar a consultar objetos de otras clases.

ALUMNOS

ELIMINACION DE OBJETOS

C L A S E : ESTUD

OBJETO: 100

PgDn -> sgte objeto Enter -> eliminar Esc -> Fin de eliminación

figura 33 : *Selección de objetos a eliminar*

ALUMNOS

CONSULTAS

CONSULTAS SOBRE OBJETOS

1 .- LISTAR

2 .- CLASIFICAR

3 .- SUMAS

4 .- PROMEDIOS

5 .- TOTALES

0 .- SALIR DEL MENU DE CONSULTAS

Seleccione Opción ===>

figura 34 : *Menú de consultas sobre objetos*

ALUMNOS

CONSULTA SOBRE OBJETOS

Ingrese Nombre de CLASE ==> estud___

MENU LISTAR

1.- LISTAR UN OBJETO PARTICULAR

2.- LISTAR MAS DE UN OBJETO DE LA CLASE

0.- SALIR DE LISTAR

Seleccione una opción ==>

figura 35 : *Menú listar de objetos*

CLASE: ESTUD OBJETO: 100

NRO_ALUMNO:	100
FACULTAD:	CS_EXACTAS
CARRERA:	INFORMATICA
MATERIAS_APROB:	23
PROMEDIO:	7.20
FECHA_INGRESO:	1-1-86
CODIGO:	C1
UNIVERSIDAD:	UNLP
DNI:	17000000
PROFESOR:	11

ENTER-> ver Obj | ↓-> Atrib Sgte | ↑-> Atrib Ant | PgDn-> Obj Sgte | PgUp-> Obj Ant | ESC-> Fin Cons

figura 36 : *Listar un objeto particular*

Si seleccionó la opción 2 del menú listar, se presenta una pantalla para **seleccionar los atributos** de la clase que desea consultar (figura 38). Utilice las flechas *arriba* y *abajo* para navegar por la lista de atributos. Puede seleccionar uno presionando *ENTER* en el atributo resaltado. Aparece una X indicando que el atributo fue seleccionado. Si presiona *ENTER* sobre un atributo marcado, deja de estar seleccionado.

En este momento si lo desea puede abandonar la consulta presionando *ESC*.

Para finalizar la selección de atributos, presione la tecla *END*.

A continuación, debe responder si desea o no seleccionar objetos .

Si elige **seleccionar objetos**, aparecerán en un cuadro todos los atributos existentes en la clase para que Ud. ingrese uno de éstos. Posteriomente, se muestra una pantalla con el atributo seleccionado, donde debe ingresar una condición (comparación) permitida; las condiciones válidas figuran en un recuadro como ayuda (figura 39). Por último, debe ingresar un valor para la comparación, que debe ser compatible con la definición del atributo (se brinda como ayuda dicha definición).

En cualquiera de los tres puntos (ingreso de atributo, condición o valor), Ud. puede abandonar la opción tipeando *Q*.

Luego de ingresada la condición, Ud. debe responder si desea o no agregar otra. Si elige agregar más condiciones, debe tipear un conector *AND* u *OR* y repetir los pasos de la selección de objetos (figura 40).

En la parte inferior de la pantalla aparecen resaltadas las condiciones que se están ingresando (figura 40).

Cuando no desea agregar más condiciones o si Ud. había decidido no seleccionar objetos, se despliegan los objetos por impresora o pantalla, según su elección.

Si se están visualizando por pantalla, los objetos aparecen de a uno por vez; Ud. puede avanzar o retroceder en la lista de objetos seleccionados usando las teclas *PgUp* y *PgDn*. Como en la opción 1 del menú listar, se pueden consultar objetos referenciados por el que se está visualizando.

Para finalizar la operación presionar *ESC*.

4.4.2. Clasificar

Por esta opción del menú de consulta sobre objetos, se despliega una pantalla donde Ud. debe ingresar un nombre de

CLASE: DOCENTE OBJETO: 11
Referenciado desde la Clase: ESTUD

CODIGO: 11
FACULTAD: CS_EXACTAS
CARRERA: INFORMATICA
CURSO: 4
DNI: 166666666
NOMBRE: JUAN PERZ
DOMICILIO: CALLE 15
NACIONALIDAD:
TELEFONO:

ESC -> Retorna a pantalla anterior

figura 37 : *Visualización de un objeto referenciado*

CLASE: ESTUD

Seleccione los atributos que desee listar

X NRO_ALUMNO
. FACULTAD
X CARRERA
X MATERIAS_APROB
X PROMEDIO
. FECHA_INGRESO
. CODIGO
. UNIVERSIDAD
. DNI
X PROFESOR

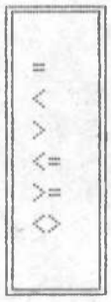
ENTER-> Selec Atrib ↓-> Atrib Sgte ↑-> Atrib Ant End-> Fin Selec ESC-> Anular Operación

figura 38 : ***Selección de** Atributos*

CLASE: **ESTUD**

SELECCIONAR OBJETOS:

Atributo	Cond	Valor
CARRERA	=	



Q -> Salir

figura 39 : *Selección de objetos*

CLASE: ESTUD

SELECCIONAR OBJETOS:

Atributo	Cond	Valor
CARRERA	=	informatica

Ingrese conector AND/OR ==> and

CARRERA = INFORMATICA

---

figura 40 : *Selección de Objetos*

clase existente. De no ser posible tipee *Q* en lugar del nombre de clase para abandonar la operación.

Luego se desplegarán en un cuadro los nombres de todos los atributos de la clase ingresada. Ud. debe ingresar uno de éstos para clasificar los objetos de la clase (figura 41). Si tipea *Q* en su lugar puede abandonar la operación.

A continuación, pasa a la pantalla de selección de atributos y de selección de objetos, descriptas en la sección 4.4.1 - opción 2 del menú listar - .

Como antes, se despliegan los objetos por pantalla o impresora, según su elección. Aparecen en el orden dado por el atributo seleccionado al principio de clasificar.

Si se están visualizando por pantalla, los objetos aparecen de a uno por vez; Ud. puede avanzar o retroceder en la lista de objetos seleccionados usando las teclas *PgUp* y *PgDn*. Como en la opción 1 del menú listar, sección 4.4.1, se pueden consultar objetos referenciados por el que se está visualizando.

Para finalizar la operación presionar *ESC*.

4.4.3 Sumas

Ud. debe ingresar el nombre de una clase existente. De no ser posible, tipee *Q* para abandonar la operación.

Luego se desplegarán en un cuadro, los nombres de todos los atributos de la clase ingresada, para que Ud. elija aquéllos que desea sean sumados. Debe ingresar nombres de atributo existentes y de tipo numérico (figura 42). Para finalizar la entrada de atributos presione la tecla *ENTER*. Si no es posible ingresar atributos válidos, tipear *Q* en su lugar para abandonar la operación.

A continuación, Ud. puede seleccionar o no objetos, como detallamos en la sección 4.4.1 - opción 2 del menú listar - .

En la salida, impresa o por pantalla, según su elección, se indican cuántas instancias de la clase intervinieron en las sumas y, para cada atributo seleccionado, se despliega el resultado de sumar los valores de dicho atributo en esas instancias. Ver figura 43.

En la salida por pantalla, para finalizar la consulta, se debe presionar cualquier tecla.

```
ALUMNOS

                        CLASIFICACION  DE  OBJETOS

 ┌──────────────────────────────────────────────────────────────────────────┐
 │ NRO_ALUMNO          FACULTAD          CARRERA          MATERIAS_APROB    │
 │ PROMEDIO            FECHA_INGRESO     CODIGO           UNIVERSIDAD       │
 │ DNI                 PROFESOR                                             │
 └──────────────────────────────────────────────────────────────────────────┘

Ingrese atributo ==>          fecha_ingreso

──────────────────────────────────────────────────────────────────────────────
                                                              Q -> Salir
```

figura 41 : *Selección de atributos para clasificar*

```
ALUMNOS
──────────────────────────────────────────────────────────────────────────────
                          SUMA DE ATRIBUTOS
──────────────────────────────────────────────────────────────────────────────

   Clase:  ESTUD

 ┌──────────────────────────────────────────────────────────────────────────┐
 │ NRO_ALUMNO          FACULTAD          CARRERA          MATERIAS_APROB    │
 │ PROMEDIO            FECHA_INGRESO     CODIGO           UNIVERSIDAD       │
 │ DNI                 PROFESOR                                             │
 └──────────────────────────────────────────────────────────────────────────┘

Ingrese atributo ==>          materias_aprob

──────────────────────────────────────────────────────────────────────────────
    ENTER -> Fin de Atributos                                 Q -> Salir
```

figura 42 : *Selección de atributos para Sumar*

4.4.4 Promedios

Ud. debe ingresar el nombre de una clase existente. De no ser posible, tipee *Q* para abandonar la operación.

Luego se desplegarán en un cuadro, los nombres de todos los atributos de la clase ingresada, para que Ud. elija aquéllos que desea sean promediados. Debe ingresar nombres de atributo existentes y de tipo numérico. Para finalizar la entrada de atributos presione la tecla *ENTER*. Si no es posible ingresar atributos válidos, tipear *Q* en su lugar, para abandonar la operación.

A continuación, Ud. puede seleccionar o no objetos, como detallamos en la sección 4.4.1 - opción 2 del menú listar - .

En la salida, impresa o por pantalla, según su elección, se indican cuántas instancias de la clase intervinieron en los promedios y, para cada atributo seleccionado, se despliega el resultado de promediar los valores de dicho atributo en esas instancias. Ver figura 44.

En la salida por pantalla, para finalizar la consulta, se debe presionar cualquier tecla.

4.4.5 Totales

Ud. debe ingresar el nombre de una clase existente. De no ser posible, tipee *Q* para abandonar la operación.

A continuación, Ud. puede seleccionar o no objetos, como detallamos en la sección 4.4.1 - opción 2 del menú listar - .

Una vez seleccionados los objetos que se desea contar, en la salida, impresa o por pantalla, según su elección, se despliega la cantidad de objetos seleccionados.
En la salida por pantalla, para finalizar la consulta, se debe presionar cualquier tecla. Ver figura 45.

```
Clase:  ESTUD
                        SUMA DE ATRIBUTOS
-------------------------------------------------------------------------

        Cantidad de Instancias Sumadas = 5


            total MATERIAS_APROB ---->   38

-------------------------------------------------------------------------
                        Presione cualquier tecla para continuar...
```

figura 43 : *Resultado Suma de Atributos*

```
Clase:  ESTUD
                        PROMEDIO  DE  ATRIBUTOS
-------------------------------------------------------------------------

        Cantidad de Instancias Promediadas = 5


            promedio MATERIAS_APROB ---->   13,6

            promedio PROMEDIO ---->   7,44

-------------------------------------------------------------------------
                        Presione cualquier tecla para continuar...
```

figura 44 : *Resultado Promedio de atributos*

5. Salida de la aplicación

Por esta opción del menú de Modificación de aplicaciones, Ud. abandona la aplicación.

Puede optar por *salvar* o no los cambios efectuados en esta sesión de trabajo. Ver figura 46.

Si decide no *salvar*, la aplicación queda en el estado anterior al inicio de la sesión. Si la aplicación fue creada en esta sesión, no se le da de alta.

6. Salida del sistema

Habiendo elegido esta opción del menú inicial del sistema, Ud. abandona el mismo.

Clase: ESTUD

TOTALES DE OBJETOS

Cantidad de Objetos Seleccionados --> 5

Presione cualquier tecla para continuar...

figura 45 : *Cantidad de objetos seleccionados*

Desea salvar la Aplicación ALUMNOS? (S/N)

figura 46 : *Salida de una aplicación*

CODIGOS Y MENSAJES DE ERROR

En esta parte del manual brindamos una breve explicación de los errores que pueden ocurrir mientras se encuentre operando dentro del prototipo.

Ud. puede buscar el código que se encuentra entre paréntesis en la última parte de los mensajes de error, en la lista que se brinda a continuación y de esta forma poder ubicar mas fácilmente la causa que lo ocasionó.

cód. 1 : X debe comenzar con letra

X debe tener un caracter alfabético en la primera posición.

cód. 2 : X existente

X no puede estar duplicado. Surge el error cuando:

- Se intenta crear una aplicación que existe en el sistema.
- Se intenta agregar una clase ya definida en la aplicación.

  Al crear una clase, se intenta agregar mas de una vez la misma superclase.
- Se intenta agregar un atributo en una clase donde ya está definido.

cód. 3 : X no existente

X no está definido. X puede ser una aplicación, una clase, una superclase, o un atributo.

cód. 4 : X no debe contener blancos

El error surge cuando X tiene blancos intermedios.

cód. 5 : X es subclase de una superclase de Y
Este error surge cuando se intenta agregar X como superclase de Y y X tiene relación de subclase con alguna superclase de Y.

cód. 6 : X existe como superclase de Y
El error surge cuando:
- Se intenta agregar X como superclase de Y, y X tiene relación de superclase con alguna superclase de Y.
  Se quiere agregar Y como superclase de X, y X es superclase de Y.

cód. 7 : Faltan superclases
El error surge cuando al crear una clase no se especifican superclases habiendo indicado que existía al menos una.

cód. 8 : X tipo no definido
X no es un tipo predefinido por el sistema ni una clase definida por el usuario.

cód. 9 : X no cumple restricción de tipos en la superclase Y
El tipo X no es subclase del tipo del atributo con el mismo nombre en la superclase Y.

cód. 10 : X debe ser entero
Surge cuando el tipo de un atributo es STRING y no se especifica un valor entero como tamaño.

cód. 11 : tamaño no puede exceder 50 caracteres
No se permite más de 50 caracteres como longitud de un atributo de tipo STRING.

cód. 12 : X valor incorrecto
X no es un valor compatible con el tipo del atributo o bien, está fuera de los rangos permitidos.

cód. 13 : X debe ser T o F
Surge cuando para un atributo de tipo BOOLEAN se ingresa un valor distinto de T (True) o F (False).

cód. 14 : X excede tamaño
El error surge cuando el valor X supera el tamaño especificado para un atributo de tipo STRING.

cód. 15 : X objeto inexistente

X es una referencia a una instancia inexistente de la clase. Surge cuando:

- al crear un objeto, se debe ingresar un valor para un atributo con tipo definido por el usuario. En este caso, se puede dar *ENTER* quedando asignado un valor nulo.
- se quiere editar, consultar, eliminar un objeto que no existe en la clase.
  .se asigna un valor a un atributo con valor por defecto o compartido y su tipo es definido por el usuario.

cód. 16 : X código no válido

Surge cuando se ingresa un código para un atributo distinto de D o C.

cód. 17 : atributo debe tener tipo

Es obligatorio que cada atributo tenga asociado un tipo. Cuando surge este error debe recomenzar la definición del atributo.

cód. 18 : Faltan atributos para la clase X

Es obligatorio definir al menos un atributo para la clase X.

cód. 19 : X tiene valor compartido o por defecto

Surge cuando se intenta seleccionar como clave un atributo cuyo código es C (Compartido) o D (por Defecto).

cód. 20 : X es de tipo BOOLEAN

Surge cuando se intenta seleccionar como clave un atributo cuyo tipo es BOOLEAN.

cód. 21 : Falta valor para el atributo clave

Surge cuando no se especifica el valor obligatorio correspondiente al atributo clave (la clave identifica al objeto).

cód. 22 : X objeto duplicado

Surge cuando se ingresó un valor como clave, al crear un objeto, que ya identificaba a otro objeto de la clase (no se permiten claves duplicadas).

cód. 23 : X referenciado en Y
El error surge cuando se quiere eliminar el objeto X, y X está referenciado desde objetos de la clase Y. (no se permiten referencias colgadas).

cód. 24 : X no tienes subclases
Surge cuando se intenta consultar las subclases de X, y X no tiene subclases.

cód. 25 : X ya seleccionado
Surge cuando se selecciona en las consultas de sumar o promediar valores de atributos, al atributo X más de una vez.

cód. 26 : X no tiene tipo numérico
Surge cuando se intenta sumar o promediar algún atributo cuyo tipo no es numérico.

cód. 27 : X condición inexistente
Surge cuando en una selección de objetos se ingresa una expresión de comparación no permitida.

cód. 28 : La referencia es nula
Surge cuando al consultar un objeto se desea visualizar un objeto componente cuya referencia tiene valor nulo.

cód. 29 : Tipo predefinido
Surge cuando al consultar un objeto se desea visualizar un objeto componente cuyo tipo es predefinido, es decir, su valor no es una referencia.

cód. 30 : X ya es compartido
Surge cuando se desea convertir un atributo en compartido y ya lo era.

cód. 31 : X es atributo clave
El error surge cuando se desea operar sobre un atributo que es la clave de la clase. No se permite en los siguientes casos:
- cuando se trata de transformar un atributo en compartido.
  cuando se trata de eliminar un atributo de una clase.
- se quiere transformar en clave un atributo que ya lo es.

cód. 32 : X no tiene valor compartido
Surge cuando:
- se quiere cambiar el valor a X y X no tiene valor compartido.
- se desea eliminar la propiedad de compartido a X, y X no es atributo con valor compartido.

cód. 33 : X no tiene valor por defecto
Surge cuando se quiere cambiar el valor a X y X no tiene valor por defecto.

cód. 34 : X no cumple restricción de tipos en la subclase Y
Surge cuando el tipo X no es superclase del tipo del atributo con el mismo nombre en la subclase Y.

cód. 35 : X origina instancias con claves duplicadas
Surge cuando se quiere especificar como nueva clave un atributo que tiene al menos un valor nulo, o dos valores idénticos en los objetos de la clase.

cód. 36 : X es atributo clave y de tipo no predefinido
El atributo al que se le intenta cambiar el tipo es clave. Si el tipo es definido por el usuario no puede realizarse la operación, pues los nuevos valores de la clave al convertir podrían originar claves duplicadas.

cód. 37 : X es el mismo
Surge cuando:
- al cambiar el tipo o la herencia X de un atributo de una clase, se especifica como nuevo tipo o herencia a X.
  se intenta agregar o eliminar una superclase X a una clase X.

cód. 38 : X no es superclase del tipo Y
El error se produce cuando se intenta cambiar el tipo X de un atributo por el tipo Y y X no es superclase de Y, por lo tanto no verifica las restricciones de tipos.

cód. 39 : No se permiten perder objetos
Surge cuando se intenta cambiar el tipo definido por el usuario de un atributo, y el tipo anterior no heredaba el atributo clave del tipo nuevo, pues:
- lo heredaba de otra de sus superclases
- o lo tenía definido como propio.

En cualquier caso, los objetos del tipo nuevo no pueden generarse a partir de los objetos del tipo viejo. Por lo tanto, las referencias correspondientes a ese atributo, en los objetos de la clase que se está modificando, no pueden pasar a ser referencias a objetos del nuevo tipo.

cód. 40 : X no se hereda
Surge cuando se intenta cambiar la herencia a un atributo X y X no es heredado.

cód. 41 : X no es superclase de Y
Surge cuando:
- se intenta cambiar la herencia de un atributo de una clase Y por X.
- se intenta eliminar X como superclase de una clase Y.

cód. 42 : X no tiene definido Y
El error surge cuando, al cambiar la herencia de un atributo Y, se intenta asignarle X como nueva herencia y en X no está definido el atributo Y.

cód. 43 : X es superclase de una subclase de Y
El error surge cuando se quiere agregar a X como superclase de Y y X ya es superclase de una subclase de Y.

cód. 44 : X no cumple la restricción de tipos en la superclase Y
Surge cuando intenta agregar una superclase Y a una clase y el tipo del atributo X no es subclase del tipo del atributo con el ese nombre (X) en la clase Y (o en alguna superclase de Y).

cód. 45 : X es tipo de atributos de otras clases
Surge cuando se intenta eliminar una clase X que se está usando como tipo de atributos de otras clases, por lo que X no se puede eliminar hasta tanto se resuelva qué hacer con esos tipos.

GLOSARIO

En esta anexo, definimos sintéticamente términos que se usaron a lo largo del manual.

aplicación:
subsistema donde el usuario define todo lo relacionado a una tarea.

atributo de una clase:
cada una de las características de un objeto, que en conjunto, describen a una clase.

atributo heredado de una clase:
atributo no definido en la clase sino en alguna de sus superclases.

atributo propio de una clase:
atributo definido en la clase.

clase:
conjunto de atributos que instanciados sirven de *molde* para crear objetos.

clave de una clase:
atributo cuyo valor en los objetos de la clase identifica dichos objetos.

código de un atributo:
indica si el atributo tiene valor compartido (C) o por defecto (D).

conflicto de herencia:
surge cuando un atributo se hereda de más de una superclase.

convertir:
adaptar los valores de atributos de un cierto tipo a valores correspondientes a otro tipo, superclase del primero.

definición de atributos:
información sobre los atributos propios (nombre, tipo, tamaño, código).

definición de una clase:
información sobre atributos propios y heredados, y superclases de la clase.

esquema de clases reticulado:
conjunto de clases definidas en la aplicación; es reticulado porque una clase puede tener varias superclases y subclases.

herencia:
propiedad por la que una clase puede utilizar todos los atributos definidos en sus superclases.

jerarquía de clases:
estructura en la que se organizan las clases de una aplicación; es jerárquica por la existencia de clases y superclases.

lista de superclases de una clase:
conjunto de clases que son superclases de la clase.

objeto / instancia de una clase:
resulta de asignar valores a cada atributo que compone la clase.

objeto referenciado:
objeto de una clase que es componente de un objeto de otra clase.

perder valores:
en un objeto de una clase, por un cambio en el esquema no se pueden conservar los valores de un atributo.

propagación a las subclases:
cambios y decisiones tomadas por el usuario en una clase, se propagan a sus subclases siempre que sea posible.

RAIZ:
superclase de todas las clases; superclase inmediata de todas las clases definidas por el usuario que no tengan definida ninguna explícitamente.

referencia a un objeto:
valor de un atributo en un objeto de una clase cuando el atributo tiene tipo definido por el usuario; la referencia corresponde al valor de la clave de un objeto de ese tipo.

restricción de tipos:
el tipo de un atributo debe ser subclase del tipo del atributo con el mismo nombre en cada una de sus superclases (si tal atributo existiera).

sesión de trabajo:
intervalo definido entre el ingreso a una aplicación y la salida de la misma.

subclase de una clase:
clase que especializa la definición de la clase (su superclase).

subclase inmediata de una clase:
subclase que se encuentra en el nivel inferior siguiente al de la clase en la jerarquía, sin clases intermedias.

superclase de una clase:
clase que generaliza la definición de la clase (su subclase).

superclase inmediata:
superclase que se encuentra en el nivel superior siguiente al de la clase en la jerarquía, sin clases intermedias.

tamaño de un atributo:
longitud máxima que puede tener un valor ingresado para para un atributo de tipo STRING.

tipo de un atributo:
clase a la que pertenece el valor del atributo.

tipo definido por el usuario:
clase definida por el usuario.

tipo predefinido:
tipo provisto por el sistema, disponible para todas las aplicaciones del usuario (INTEGER, REAL, STRING, BOOLEAN).

valor compartido de un atributo:
el atributo tiene ese valor en todos los objetos de la clase.

valor nulo de un atributo:
valor que toma un atributo en un objeto cuando *pierde el valor* como consecuencia de alguna operación, o cuando al crear un objeto no se le asigna un valor.

valor por defecto de un atributo:
el atributo tiene ese valor en todos los objetos de la clase donde no especifique otro valor explícitamente.