CORPORACIÓN SANTO TOMAS TALCA.

Carrera: Tec. Sistema informático.

Nivel : II

Índice

Introducción

Un archivo es un elemento de información conformado por un conjunto de registros. Estos registros a su vez están compuestos por una serie de caracteres o bytes. Los archivos, alojados en dispositivos de almacenamiento conocidos como memoria secundaria, pueden almacenarse de dos formas diferentes: archivos convencionales o bases de datos.

Los archivos convencionales, pueden organizarse como archivos secuénciales o archivos directos. Sin embargo, el almacenamiento de información a través de archivos convencionales presenta una serie de limitaciones que restringen de manera importante la versatilidad de los programas de aplicación que se desarrollan.

Definición de Base de Datos

Se define una base de datos como una serie de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los sistemas de información de una empresa o negocio en particular.

Las bases de datos proporcionan la infraestructura requerida para los sistemas de apoyo a la toma de decisiones y para los sistemas de información estratégicos, ya que estos sistemas explotan la información contenida en las bases de datos de la organización para apoyar el proceso de toma de decisiones o para lograr ventajas competitivas. Por este motivo es importante conocer la forma en que están estructurados las bases de datos y su manejo.

Los modelos de datos

En el proceso de abstracción que conduce a la creación de una base de datos desempeña una función prioritaria el modelo de datos. El modelo de datos, como abstracción del universo de discurso, es el enfoque utilizado para la representación de las entidades y sus características dentro de la base de datos, y puede ser

dividido en tres grandes tipos (KORTH y SILBERSCHATZ, 1993: 6−11):

1. **Modelos lógicos basados en objetos**: los dos más extendidos son el modelo entidad−relación y el orientado a objetos. El modelo entidad−relación (E−R) se basa en una percepción del mundo compuesta por objetos, llamados entidades, y relaciones entre ellos. Las entidades se diferencian unas de otras a través de atributos. El orientado a objetos también se basa en objetos, los cuales contienen valores y métodos, entendidos como órdenes que actúan sobre los valores, en niveles de anidamiento. Los objetos se agrupan en clases, relacionándose mediante el envío de mensajes. Algunos autores definen estos modelos como "modelos semánticos".

2. **Modelos lógicos basados en registros**: el más extendido es el relacional, mientras que los otros dos existentes, jerárquico y de red, se encuentran en retroceso. Estos modelos se usan para especificar la estructura lógica global de la base de datos, estructurada en registros de formato fijo de varios tipos. El modelo relacional representa los datos y sus relaciones mediante tablas bidimensionales, que contienen datos tomados de los dominios correspondientes. El modelo de red está formado por colecciones de registros, relacionados mediante punteros o ligas en grafos arbitrarios. El modelo jerárquico es similar al de red, pero los registros se organizan como colecciones de árboles. Algunos autores definen estos modelos como "modelos de datos clásicos".

3. **Modelos físicos de datos**: muy poco usados, son el modelo unificador y el de memoria de elementos. Algunos autores definen estos modelos como "modelos de datos primitivos".

Los objetivos del modelo de datos son dos:

1. **Formalización**: definir formalmente las estructuras permitidas y las restricciones a fin de representar los datos de un SI.

2. **Diseño**: el modelo resultante es un elemento básico para el desarrollo de la metodología de diseño de la base de datos.

Los diferentes modelos de datos comparten, aunque con diferentes nombres y notaciones, unos elementos comunes, componentes básicos de la representación de la realidad que realizan. Estos componentes se identifican gracias a la clasificación, y pueden identificarse conceptos estáticos y conceptos dinámicos.

Los conceptos estáticos corresponden a:

1. **Objeto**: cualquier entidad con existencia independiente sobre el que almacenan datos. Puede ser simple o compuesto.

2. **Relación**: asociación entre objetos.

3. **Restricción estática**: propiedad estática del mundo real que no puede expresarse con los anteriores, ya que sólo se da en la base de datos; suele corresponder a valores u ocurrencias, y puede ser sobre atributos, entidades y relaciones.

4. **Objeto compuesto**: definidos como nuevos objetos dentro de la base de datos, tomando como punto de partida otros existentes, mediante mecanismos de agregación y asociación.

5. **Generalización**: se trata de relaciones de subclase entre objetos, es decir, parte de las características de diferentes entidades pueden resultar comunes entre ellas.

Por su parte, los conceptos dinámicos responden a:

1. **Operación**: acción básica sobre objetos o relaciones (crear, modificar, eliminar...).

2. **Transacción**: conjunto de operaciones que deben ejecutarse en su conjunto obligatoriamente.

3. **Restricción dinámica**: propiedades del mundo real que restringen la evolución en el tiempo de la base de datos.

Modelos en red

El modelo de datos en red general representa las entidades en forma de nodo de un grafo, y las asociaciones o interrelaciones entre éstas mediante los arcos que unen dichos nodos.

El modelo de red evita esta redundancia en la información, a través de la incorporación de un tipo de registro denominado el conector, que en este caso pueden ser las calificaciones que obtuvieron los alumnos de cada profesor.

La dificultad surge al manejar las conexiones o ligas entre los registros y sus correspondientes registros conectores.

Estructura de una base de Datos de Red.

Hay dos estructuras de datos básicos en el modelo debed: Registros y Conjuntos. Hablaremos de los registros y los tipos de registros, también hablaremos de los conjuntos y sus propiedades básicas, además de algunos tipos de especiales de conjuntos mostraremos como se representan e implementan los conjuntos por ultimo mostraremos como se representan los vínculos en el modelo de red.

Registros, tipos de registros y elementos de información

Los datos se almacenan en registros; cada registro consiste en un grupo de valores de datos relacionados entre sí. Los registros se clasifican en tipos de registros, cada uno de los cuales describe la estructura de un grupo de registros que almacenan el mismo tipo de información: Damos un nombre a cada tipo de registro y también damos un nombre y un formato (tipo de formato) a cada elemento de información (o atributo) del tipo de registros.

Con el modelo de red es posible definir elementos de información complejos. Un vector es un elemento de información que puede tener múltiples valores es un solo registro. Un grupo repetitivo permite incluir un conjunto de valores compuestos para un elemento de información en un solo registro. Por ejemplo, si queremos incluir la boleta de notas de cada estudiante dentro de cada registro de estudiante, podemos definir un grupo respectivo BOLETA para dicho registros; BOLETA consta de los cuatro elementos de información AÑO, CURSO, SEMESTRE Y NOTAS, como se demuestra en la figura. El grupo repetitivo no es esencial para la capacidad de modelado del modelo de red, ya que podemos representar la misma situación de los grupos repetitivos puede llegar a varios niveles de profundidad.

ESTUDIANTE				
NOMBRE	NSS	DIRECCIÓN	DEPTOCARRERA	FECHANACIM

Nombre de elemento de información **Formato**

Nombre Charácter 30

Nss Charácter 9

Dirección Charácter 40

Deptocarrera Charácter 10

Fechanacim Charácter 9

Todos los elementos de información que acabamos de mencionar se denominan elementos de información reales, por que sus valores se almacenan realmente en los registro. También es posible definir electos de información virtuales (o derivadas), cuyos valores no se almacenan realmente en los registros; en vez de ello, se derivan de los elementos de información reales mediante algún proceso definido específicamente para tal fin. Por ejemplo debemos aclarar un elemento de información virtual EDAD para el tipo de registros de la figura y escribir un procedimiento que calcule el valor de EDAD a partir del valor del elemento de información real FECHANACIM de cada registro.

Tipos de Conjuntos y sus propiedades Básicas

Un tipo de conjuntos es una descripción de un vínculo 1: N entre dos tipos de registros. La figura muestra como se representa un tipo de conjuntos en un diagrama mediante una flecha. Este tipo de representación diagramatica se llama Diagrama de Bachean. Cada definición de tipo de conjuntos consta de tres elementos básicos.

- Un nombre para el tipo de conjuntos
- Un tipo de registros propietario
- Un tipo de registros miembro

ESTUDIANTE						
NOMBRE		BOLETA				
			AÑO	CURSO	SEMESTRE	NOTAS

Silva 1984 CICO3320 Otoño A

1984 CICO3340 Otoño A

1984 MATE312 Otoño B

1985 CICO4310 Primavera C

1985 CICO4330 Primavera B

1.) Figura de Grupo repetitivo BOLETA.

DEPARTAMENTO	
NOMBRED	

ESTUDIANTE	
NOMBREE	

2.) Figura de el tipo de conjuntos DEPTO_CARRERA (que es MANUAL OPTIONAL)

El tipo de conjuntos de esta figura se llama DEPTO_CARRERA, DEPARTAMENTO es el tipo de registro propietario y ESTUDIANTE es el tipo de registros miembro. Esto representa el vínculo 1: N entre los departamentos académicos y los estudiantes que cursan una carrera en esos departamentos, en la base de datos

habrá muchas ocurrencias de conjunto (o ejemplares de conjunto) que corresponderán a un tipo de conjuntos. Cada ejemplar relaciona un registro del tipo de registros propietario− un registro DEPARTAMENTO en nuestro ejemplo − con el conjunto de registros del tipo de registro miembro relacionado con el: el conjunto de registro ESTUDIANTE de los estudiantes que cursan una carrera de ese departamento. Así, cada ocurrencia de conjunto se compone de

- Un registro propietario del tipo de registro propietario, y
- Varios registros miembro relacionado (cero o mas) del tipo de registro miembro.

Una ocurrencia del tipo de registro miembro no puede existir en mas de una ocurrencia de conjunto de un tipo de conjuntos particular. Esto mantiene la restricción de que un tipo de conjuntos representa un vinculo 1:n. en nuestro ejemplo, un registro ESTUDIANTE puede estar relacionado cuando mas de una ocurrencia de conjunto del tipo de conjunto DEPTO_CARRERA.

En el modelo, un ejemplar de conjuntos no es idéntico al concepto matemático de conjunto. Sus principales diferencias son dos:

- El ejemplar de conjunto tiene un elemento distinguido− el registro propietario− pero en un conjunto matemático no hay distinción alguna entre los elementos.
- En el modelo de red los registros miembros de un ejemplar de conjuntos están ordenados, mientras que un conjunto matemático el orden de los elementos carece de importancia. Así, podemos hablar del primero, segundo, i−ésimo y ultimo registros miembros de un ejemplar de conjunto.

Restricciones en el modelo de red

Cuando hablamos de restricción, hemos hablado de restricciones estructurales que gobiernan la forma como están estructurados los tipos de registros de conjuntos. En esta sección hablaremos de restricciones de comportamiento que se aplican a los miembros de los conjuntos, o bien al comportamiento de dichos miembros, cuando se realizan operaciones de inserción, eliminación y actualización con esos conjuntos. Podemos especificar varias restricciones sobre la pertenencia a un conjunto, y estas suelen dividirse en dos categorías principales, llamadas opciones de inserción y opciones de retención en la terminología CODASYL. Para determinas estas restricciones durante el diseño de la base de datos hay que conocer como deberá comportarse un conjunto cuando se inserten registros miembros o cuando se eliminen registros miembros o propietario. Las restricciones se especifican al SGBD cuando se declara la estructura de la base de datos empleando el lenguaje de definición de datos. No todas las combinaciones de las restricciones son posibles.

Opciones de inserción de conjuntos

Existen dos opciones de inserción:

- Automatic:

El nuevo registro miembro se conecta automáticamente a una ocurrencia de conjunto apropiada cuando se inserta en el conjunto.

- Manual:

El nuevo registro no se conecta a ninguna ocurrencia de conjuntos.

Si el programador lo desea, puede conectar después explícitamente (manualmente) el registro a una ocurrencia de conjunto, mediante la orden CONNECT.

Opciones de retención en conjuntos

Existen tres opciones de retención:

- Optional (opcional): Un registro miembro puede existir por si solo sin ser miembro de ninguna ocurrencia del conjunto. Se le puede conectar y desconectar de las ocurrencias de conjunto a voluntad con las ordenes CONNECT y DISCONNECT del DML de red
- Mandatory (obligatoria): Ningún miembro puede existir por si solo; siempre debe ser miembro de una ocurrencia de conjuntos. Se le puede reconectar en una sola operación de una ocurrencia de conjunto a otra mediante la orden RECONNECT del DML de red
- FIXED (fija): Al igual que en MANDATORY, ningún registro miembro puede existir por si solo, por añadidura, una vez insertado en una ocurrencia de conjunto, queda fijo; no se le puede reconectar a otra ocurrencia de conjunto.

Definición de datos en el modelo de red

Declaraciones de tipo de registros y de elementos de información.

Las declaraciones del DDL de red para los tipos de registros del esquema. Cada tipo de registros recibe un nombre a través de la cláusula RECORD NAME IS (nombre de registro es). Se especifica un formato (tipo de area) para cada uno de sus elementos de información (campos), juntos con cualquier restricción que se apliquen a los elementos. Los tipos de datos que normalmente están disponibles dependen de los tipos que se pueden definir en el lenguaje de programación anfitrión.

Modelo red

El esquema representa los aspectos estáticos, la estructura de los datos (tipos de entidades, tipos de interrelaciones, etc...), mientras que una ocurrencia del esquema (base de datos) son los M valores que toman los elementos del esquema en un determinado momento, los cuales irán variando a lo largo del tiempo por el efecto de aplicar los operadores de manipulación de datos a una ocurrencia del esquema.

El modelo en red general es muy flexible debido a la inexistencia de restricciones inherentes, pero también por esta misma razón su instrumentación física resulta difícil y poco eficiente. Esta es la causa de que se le suela introducir restricciones al llevarlo a la práctica. El modelo jerárquico y el modelo CODASYL son modelos que responden a estructuras del tipo de red pero con restricciones bastante fuertes.

Los diferentes tipos de interrelaciones que se pueden representar en un modelo en red son:

Tipo de interrelación N:M y ocurrencia de la misma

Tipo de interrelación 1: N y ocurrencias de la misma

Interrelación reflexiva 1:N y ocurrencias de la misma

Interrelación reflexiva N: M y ocurrencias de la misma

Tipo de interrelación entra más de dos tipos de entidad y ocurrencias de la misma

Tipos especiales de conjuntos

CODASYL

Es un modelo de datos de tipo red que introduce restricciones inherentes. este modelo constituye una simplificación del modelo en red general, en la que se admiten solo determinados tipos de interrelaciones y se incluyen algunas restricciones adicionales, que, sin embargo, no limitan excesivamente la flexibilidad que proporciona el modelo en red, pero si que facilita una instrumentación eficiente.

Objetivos

A) Flexibilidad para los usuarios

Permitir la estructuración de los datos de la forma mas adaptada a cada aplicación, independientemente del hecho de que todos o parte de dichos datos pudiesen utilizarse en otras aplicaciones, una flexibilidad que debe conseguirse evitando las redundancias. Este es un objetivo esencial en un sistema de base de datos, que permite diferenciarlo de los sistemas clásicos de ficheros.

B) Uso concurrente

Facilitar a varias aplicaciones recuperar o actualizar concurrentemente los datos de la base. Este ha sido uno de los puntos más controvertidos y criticados, a pesar de ser una necesidad reconocida, la verdad es que las especificaciones, ni la de 1973 ni la de 1978, proporcionaban las facilidades necesarias para obtener un verdadero acceso concurrente. De hecho, algunos de los sistemas basados en este modelo no se comportaban nada bien en este aspecto.

C) Estrategias de búsqueda diversas

Suministrar y permitir el uso de varias estrategias de búsqueda, tanto sobre el conjunto de la base como sobre una parte de ella. El lenguaje de definición de datos CODASYL facilita la consecución de este objetivo mediante diferentes opciones para la elección de la forma de ubicación de cada tipo de registro. Estas opciones se encontraban en la especificaron de 1973 en el lenguaje de definición de datos del esquema, lo que fue muy criticado por su implementaciones físicas, pero en 1978 paso al lenguaje de definición del almacenamiento de datos.

D) Seguridad

Proteger la base de datos de accesos no autorizados y de interacciones indeseable de los programas. Este objetivo es de gran importancia ya que asegura la confidenciabilidad y la integridad de la base de datos, para esto se establecieron distintas cláusulas de control de acceso y asignación de contraseñas (PRIVACY LOCK...)

E) Gestión centralizada del almacenamiento físico

Hacer los programas independientes del almacenamiento físico de los ficheros

F) Independencia del almacenamiento físico

Hacer los programas independientes del almacenamiento físico de los ficheros

G) Flexibilidad en el modelo de datos

Permitir la especificación de diversas estructuras de datos, desde entidades aisladas hasta redes. El modelo CODASYL no es excesivamente restrictivo en cuanto a las características de las entidades e interrelaciones que pueden ser representadas. Sin embargo, se ha de tener en cuenta que no permitía en la especificaciones de 1973 la existencia de SET reflexivos, aunque en 1978 se modifico este punto, admitiéndose las interrelaciones

reflexivas, siendo posible incluso que una ocurrencia de registro sea propietaria de si misma.

H) facilidad para el usuario

Permitir la interacción del usuario con los datos, pero descargándolo de la operaciones de mantenimiento de las estructuras que han sido declaradas. De hecho, una vez realizada la definición de la base de datos, compilada esta e introducidos los datos, el programados utiliza el lenguaje de manipulación sin tener que ocuparse del mantenimiento del esquema, solo ha de conocer la estructura de los datos y manejarla.

I) Independencia de los programas respecto a los datos

Conseguir programas tan independientes de los datos como sea posible con las técnicas existentes.

J) Descripción de datos independiente

Separar la descripción de la base de datos en su conjunto de la de cada programa.

K) Independencia respecto a los lenguajes

Dotar de medios de descripción de la base de datos que no estén restringidos a un determinado lenguaje. Aunque CODASYL señala este objetivo y le concede bastante importancia, la verdad es que las especificaciones de los diferentes lenguajes de definición y manipulación están claramente orientadas a COBOL.

L) Interfaces con múltiples lenguajes

Proporcionar una arquitectura que permita que la descripción de la base de datos, así como la manipulación de la misma, pueda tener interfaces con múltiples lenguajes.

Elementos del modelo y definiciones

Los elementos básicos de la estructura de datos propia del modelo CODASYL son:

- *Elemento de datos (Data ítem)*: Es la unidad de datos más pequeña a la que se puede hacer referencia en el modelo CODASYL. Un elemento de datos a de tener un nombre, y una ocurrencia del mismo contiene un valor que puede ser de distintos tipos (booleano, numérico, tira de caracteres,...) además, un elemento de datos puede definirse como dependiente de los valores de otros elementos (datos derivados).
- *Agregado de datos (data aggregate)*: puede ser un vector, con un número fijo de elementos, o un grupo repetitivo. El elemento y le agregado de datos se corresponden con los campos de los ficheros clásicos o con los atributos de otros modelos, aunque a diferencia de algunos modelos, como el relacional, aquí si que se admiten estructuras no planas como son los agregados.
- *Registro (record)*: colección nominada de elementos de datos. Es la unidad básica de acceso y manipulación de la base de datos y se corresponde con el concepto de registro (en los ficheros) y de entidad (en otros modelos como el E/R).
- *Conjunto (SET o COSET)*: es una colección nominada de dos o más tipos de registros que establece una vinculación entre ellos. Constituye el elemento clave y distintivo de este modelo de datos, siendo el origen de muchas de sus restricciones, tanto inherentes como opcionales. Las interrelaciones 1: N de otros modelos se representan en CODASYL como SET.
- *Área (área o realm)*: es una subdivisión nominada del espacio de almacenamiento direccionable de la base de datos que contiene ocurrencias de registros. En un área puede haber ocurrencias de más de un tipo de registro y las ocurrencias de un mismo tipo de registro pueden estar contenidas en distintas

áreas, aunque una ocurrencia determinada se almacena solo en un área. En 1981 paso a formar parte del esquema de almacenamiento.
- *Clave de base de datos (Database Key)*: identificador interno único para cada ocurrencia del registro, que proporciona su dirección en la base de datos. en principio, esta clave era permanente y se podía utilizar par acceder rápidamente a un registro de forma directa o para indicar donde almacenarlo, pero suponía un grave obstáculo para conseguir la independencia lógica / física por lo que las últimas versiones del modelo restringen mucho su uso, aunque se conservó.

En el modelo CODASYL, un mismo tipo de entidad puede ser propietarios de un tipo SET y miembro en otro distinto, lo que le da una gran flexibilidad al modelo. En otros modelos de datos de tipo red, como por ejemplo el instrumentado en el sistema total, no se admite tales interrelaciones.

También es posible en el modelo CODASYL considerar las ocurrencias de una entidad (registros) interrelacionadas entre si como si se tratara de un fichero clásico. Para ello habría que definir un SET especial, denominado singular, en el que el propietario sería ficticio (el sistema) y el miembro el tipo de entidad cuyas ocurrencias queremos que estén interrelacionadas con independencia de que dicho registro intervenga o no en otros SET; este tipo de SET singular tiene una única ocurrencia de SET y su forma de instrumentación a nivel interno suele ser de tipo INDEX.

Se puede resumir las características básicas del modelo en los siguientes puntos:

- Un SET es una colección nominada de dos o mas tipos de registros que representa un tipo de interrelación 1:N (también 1:1)
- Cada SET debe tener forzosamente un tipo de registro propietario y uno o más tipos de registros miembros
- No hay limitación en cuanto al número de SET que pueden declararse en el esquema
- Cualquier registro pude ser declarado propietario de uno o varios SET
- Cualquier registro puede ser declarado miembro de uno o varios SET
- Pueden existir SET singulares en los que el propietario es el sistema
- Aunque un tipo de registro se haya declarado miembro de un SET, existe la posibilidad de no encadenar en el SET ciertas ocurrencias del mismo, las cuales no pertenecerán a ningún propietario, es decir, quedarán "sueltas" respecto a ese SET.

Como consecuencia de estas restricciones se deduce que:

- No se pueden representar directamente interrelaciones entre ocurrencias distintas de una misma entidad (interrelaciones reflexivas), aunque las especificaciones de 1978 si que lo permiten, admitiendo incluso que una ocurrencia de registro pueda ser propietaria de si misma.
- No se pueden representar directamente interrelaciones de tipo N:1
- No se pueden representar directamente interrelaciones del tipo N:M

Por tanto, las estructuras admitidas por el modelo son, entre otras, las siguientes:

- *Red*: Un miembro con varios propietarios, pero en SET diferentes. De esta forma se podrán representar relaciones N:M sin que por ello se produzca redundancia en los datos.

- Propietario con miembros de distinto tipo (SET multimiembro).

- *Jerarquía a un nivel*: Un propietario y varios miembros, estas se pueden instrumentar de dos formas: un solo SET multimiembro, o tantos SET como registros miembros haya.

- *Jerarquía con múltiples niveles*: Donde los tipos de registro de los niveles intermedios son

miembros del SET del nivel superior u propietarios en el nivel inferior

- ♦ *Diferentes interrelaciones entre dos tipos de registro*: Esta estructura permite deshacer las interrelaciones reflexivas.

- ♦ *Registros aislados*: Pueden existir tipos de registros que no sean propietarios ni miembros de ningún SET.

- ♦ *Miembro opcional*: Existe la posibilidad de que un tipo de registro declarado como miembro de un SET tenga ocurrencias que no pertenezcan a ninguna ocurrencia de SET, que no tengan propietarios, habrá hijos sin padre.

Aplicando algunas de las estructuras anteriores también se puede representar en CODASYL un esquema plex descomponiéndolo en SET.

Cuando hay tipos de interrelación del tipo N:M , el problema se resuelve creando un tipo de registro ficticio, que puede o no, contener datos y definiendo dos SET donde este tipo de registro (llamado de enlace) es miembro. Si la interrelación es reflexiva N:M, se puede representar mediante dos SET unidos mediante un tipo de registro ficticio, dichos SET se denomina de explosión y de implosión, y si por último, la interrelación es reflexiva 1:N, es igual que el caso N:M solo que uno de los SET puede tener como propietario el registro de enlace.

Definición de datos a nivel lógico

Estructura general del lenguaje de definición

La definición de un esquema de una base de datos consta de varias partes diferentes, a las que CODASYL se refiere como entradas, cada una de las entradas se refiere a un elemento distinto del esquema. Algunas de las entradas tienen subentradas, y tanto unas como otras contienen una o varias sentencias que se denominan cláusulas, las cuales pueden tener cláusulas subordinadas, y estas, a su vez, están compuestas por opciones.

Existen cuatro entradas:

- *Esquema (schema)*: Describe las características del esquema en general (nombre, contraseña, etc...), por tanto, solo habrá una entrada de este tipo para cada base de datos cuyo esquema se defina.
- *Área (Area)*: Describe las características de las diferentes áreas consideradas. existen tantas entradas de área como áreas tiene el esquema, en cada una de la cuales se definirán los elementos y opciones correspondientes a la misma.
- *Registro (RECORD)*: Describe cada registro especificando su nombre y sus elementos de datos. hay una entrada de registro por cada uno de los existentes en el esquema.
- *Conjunto (SET)*: Define cada SET dentro del esquema, por lo que existirá una entrada de SET por cada uno de los que comprende el esquema.

Las entradas de registro o de SET contienen subentradas. Las subentradas de registro permiten describir los elementos de datos que componen el correspondiente registro; las subentradas de SET describen el (o los) miembro (s) de dicho SET.

En un esquema CODASYL debe de haber al menos una entrada de registro, ya que no puede existir una base de datos que no contenga ni una sola entidad. Sin embargo, es posible que no existan entradas de SET, ya que no es obligatorio que una base de datos contenga interrelaciones entra las entidades que la componen. Las especificaciones de 1973 no decían nada al respecto y el informe de 1981 tampoco se concreta excesivamente, por lo que no se puede considerar como norma para los productos comerciales.

En cuanto a la formación de los nombres que el diseñador puede dar a los elementos de la base de datos, CODASYL especifico las siguientes reglas:

- Pueden utilizarse cualquier combinación de caracteres del alfabeto, ya sean letras o números para construir los nombres de registros, SET, datos, etc...
- La longitud de los nombres no puede exceder de 30 caracteres.
- Los nombres deben ser únicos, con la sola excepción de los nombres de elementos de datos pertenecientes a diferentes registros.
- El lenguaje de definición contiene palabras reservadas que no deben utilizarse en nombres facilitados por el usuario.

Las más importantes convenciones en la descripción de la sintaxis de las distintas entradas y cláusulas son las siguientes.

- Las palabras que aparezcan en mayúsculas y subrayadas son las palabras reservadas del leguaje y no pueden omitirse en la escritura de la cláusula correspondiente.
- Las palabras que aparezcan en mayúsculas sin subrayar son palabras pertenecientes al lenguaje, pero pueden omitirse, con ellas solo se pretende hacer la descripción del esquema más fácilmente legible.
- Las palabras escritas en minúsculas se refieren a nombres que serán proporcionados por el usuario.
- Las cláusulas (o cualquier otra estructura, como opciones o palabras) encerradas entre corchetes ([]) son opcionales, es decir, pueden incluirse en la definición o ser omitidas.
- Las llaves ("{}") implican que una, y solo una, de las opciones encerradas entre ellas debe se elegida.
- Las cláusulas u opciones encerradas entre barras dobles ("|| ||") pueden seleccionarse todas, si así se desea, pero al menos una debe estar presente.
- Los puntos suspensivos dentro de las llaves, corchetes o barras dobles significan que los elementos que encierran pueden repetirse dentro de la cláusula.
- El punto separa entradas y subentradas.
- El punto y coma separa cláusulas dentro de una misma entrada.

Modelo Jerárquico

En el modelo de red, los datos se representan mediante colecciones de *registros* y las relaciones entre los datos por medio de *enlaces*. Esta representación también es valida para el modelo jerárquico. La única diferencia es que el modelo jerárquico los registros se organizan par formar colecciones de árboles en vez de grafos arbitrarios.

Conceptos Básicos

Una base de datos jerárquica consiste en una colección de *registros* que se conectan entre si por medio de *enlaces*. Los registros son similares a los del modelo en red. Cada registro es un conjunto de campos (atributos), cada uno de los cuales contiene un solo valor. Un enlace es una asociación entre dos registros exclusivamente. Por tanto, el concepto de enlace es similar al modelo de red. La relación de los conceptos que trataremos aquí, son dos conceptos básicos en el modelo jerárquico, que son conceptos básicos principales de estructuración de datos: registros y vínculos padre−hijo. Un registro es una colección de valores de campo que proporcionan información sobre una cantidad o un ejemplar de vínculo. Los registros del mismo tipo se agrupan en tipos de registros. Cada tipo de registro recibe un nombre, y su estructura se define en termino de una colección de campos o elementos de información con. Cada campo tiene cierto tipo de datos, como entero, real o cadena.

Un tipo de vínculos padre−hijo (tipo de VPH) es un vinculo 1:n entre dos tipos de registros. El tipo de registros del lado 1 se denomina tipo de registro padre, y el del lado n se llama tipo de registros hijo del lado de VPH. Una ocurrencia (o ejemplar) del tipo VPH consiste en un registro del tipo de registros padre y varios

registros (cero o mas) del tipo de registros hijo.

Considérese una base de datos que representa una relación *cliente−cuenta* en un sistema bancario. Existen dos tipos de registro, *cliente* y *cuenta*. El tipo de registro *cliente* puede definirse de la misma manera que en el modelo de red. Consta de tres campos: *nombre, calle* y *ciudad*. De manera similar, el registro *cuenta* consta de dos campos: *numero* y *saldo*.

Árboles de ocurrencias jerárquicas

En correspondencia con un esquema jerárquico, hay muchas ocurrencias jerárquicas en la base de datos. Cada ocurrencia jerárquica, llamada también árbol de ocurrencias, es una estructura de árbol cuya raíz es un solo tipo de registros raíz. Igualmente, el árbol de ocurrencias contiene todas las ocurrencias de registros hijo del registro raíz, todas las ocurrencias de registros hijo dentro de los VPH de cada uno de los registros hijo del registro raíz, y así sucesivamente, hasta los registros de los tipos de registros hoja

Un ejemplo de esta base de datos sería la siguiente, que muestra que el cliente Lowman tiene la cuenta 305, el cliente Camp las cuentas 226 y 177 y el cliente Kahn la cuenta 155.

Nótese que el conjunto de todos los clientes y cuentas esta organizado en forma de un árbol con raíz donde la raíz del árbol es un nodo ficticio. Como veremos, una base de datos jerárquica es una colección de árboles de este tipo, formando así un bosque. Cada uno de estos árboles con raíz se denominara árbol de base de datos.

El contenido de un registro específico puede tener que repetirse en varios sitios. Por Ejemplo, en el sistema bancario *cliente−cuenta*, una cuenta puede pertenecer a varios clientes. La información correspondiente a esa cuenta, o la correspondiente a los clientes a los que puede pertenecer, tendrán que repetirse. Esta repetición puede ocurrir tanto en el mismo árbol de base de datos como en varios árboles distintos. La repetición de registros tiene dos desventajas principales:

- Puede producirse una inconsistencia de los datos al llevar a cabo la actualización.
- El desperdicio de espacio es inevitable.

Propiedades de los esquemas Jerárquicos

Un esquema jerárquico de tipo de registro y tipos de VPH debe tener las siguientes propiedades:

- Un tipo de registros, la raíz del esquema jerárquico, no participa como tipo de registros hijo en ningún tipo de VPH.
- Todo tipo de registros, con excepción de la raíz, participa como tipo de registro hijo de un y solo un tipo de VPH.
- Un tipo de registros puede participar como tipo de registros padre en cualquier cantidad (cero o mas) de tipos de VPH.
- Un tipo de registros que no participa como tipo de registros padre en ningún tipo de VPH se denomina hoja del esquema jerárquico

- Si un tipo de registro participa como padre en más de un tipo de VPH, entonces sus tipos de registros hijo están ordenados. El orden se visualiza, por convención, de izquierda a derecha en los diagramas jerárquicos.

Vínculos virtuales padre − hijo

El modelo jerárquico tiene problemas cuando se modelan ciertos tipos de vínculos. Entre ellos están los siguientes vínculos y situaciones:

- Vínculos M:N
- El caso en que un tipo de registro participa como hijo en mas de un tipo de VPH
- Vínculos n−arios con mas de dos de registros participantes.

Restricciones de integridad en el modelo Jerárquico

Siempre que especificamos un esquema jerárquico, habrá varias restricciones inherentes al modelo jerárquico. Entre ellas se cuentan;

- Ninguna ocurrencia de registro, con excepción de los registros raíz, puede existir si no esta relacionada con una ocurrencia de registro padre. Esto tiene las siguientes implicaciones
- Ningún registro hijo puede insertarse si no esta enlazado a un registro padre
- Un registro hijo se puede eliminar independientemente de su padre; pero la eliminación de un padre causa automáticamente la eliminación de todos sus registros hijos y descendientes
- Las reglas anteriores no se aplican a los registros hijo y padre virtuales. Las regla en este caso es que un apuntador en un registro hijo virtual debe apuntar a una ocurrencia real de un registro padre virtual. No debe permitirse la eliminación de un registro en tanto existan apuntadores a el en registros hijo virtuales, lo que lo convierte en un registro virtuales, lo que lo convierte en un registro padre virtual.

- Si un registro hijo tiene dos o mas registros padre del mismo tipo de registros, el registro hijo debe duplicarse una vez bajo cada registro padre.
- Un registro hijo que tengo dos o mas registros padre de diferentes tipos de registros solo puede tener un padre real; todos los demás deben representarse como padres virtuales

Diagramas de estructura de árbol

Un diagrama de estructura de árbol en el esquema de una base de datos jerárquica. Este tipo de diagrama esta formado por dos componentes básicos:

- Cajas, que corresponden a tipos de registro.
- Líneas, que corresponden a enlaces.

Un diagrama de estructura de árbol tiene el mismo propósito que un diagrama de entidad−relación; a saber, especificar la estructura lógica global de la base de datos. Un diagrama de estructura de árbol es similar a un diagrama de estructura de datos en el modelo de red. La principal diferencia es que en este último los tipos de registro se organizan en forma de grafo arbitrario, mientras que en el primero se organizan en forma de árbol con raíz.

Tenemos que ser más precisos en lo que quiere decir árbol con raíz. En primer lugar, el grafo no puede contener ciclos. En segundo lugar, Las relaciones formadas en el grafo deben ser de tal forma que solo existan relaciones uno a muchos o uno a uno entre un padre y un hijo. La forma general de una estructura de árbol. Nótese que las flechas están apuntando de padres a hijos. Un padre puede tener una flecha apuntando a un hijo, pero un hijo siempre debe tener una flecha apuntando a su padre.

...

...

El esquema de base de datos se representa como una colección de diagramas de estructura de árbol. Para cada diagrama existe una única instancia del árbol de base de datos. La raíz de este árbol es un nodo ficticio. Los hijos de este nodo son instancias del tipo de registro adecuado. Cada una de esas instancias de hijo puede tener, a su vez, varias instancias de varios tipos de registro, según se especifica en el diagrama de estructura de

árbol correspondiente.

Para comprender como están formados los diagramas de estructura de árbol, veremos como transformar los diagramas de entidad−relación en sus correspondientes diagramas de estructura de árbol. Primero mostraremos como pueden aplicarse estas transformaciones a una sola relación. Después trataremos el tema de cómo asegurar que los diagramas resultantes tengan forma de árboles con raíz.

Relaciones únicas

Las relaciones entre los atributos en la entidad relación, que consta de los dos conjuntos de entidades cliente y cuenta relacionados por medio de la relación binaria uno a muchos CliCta sin atributos descriptores. Este diagrama especifica que un cliente puede tener varias cuentas pero que una cuenta solo puede pertenecer a un cliente. El diagrama de estructura de árbol correspondiente se muestra en la Figura.

El tipo de registro cliente corresponde al conjunto de entidades cliente. Incluye tres campos: nombre, calle y ciudad. De manera similar, cuenta es el tipo de registro que corresponde al conjunto de entidades cuenta. Incluye dos campos: numero y saldo. Finalmente, la relación CliCta se ha sustituido por el enlace CliCta, con una flecha apuntando a tipo de registro cliente.

Así una instancia de una base de datos que corresponda al esquema que se acaba de Describir puede contener varios registros cliente enlazados a varios registros cuenta, como se muestra en la Figura. Puesto que la relación es uno a muchos de cliente a cuenta, un cliente puede tener más de una cuenta.

Sin embargo, una cuenta no puede pertenecer a más de un cliente, como puede verse en el ejemplo de base de datos. Si la relación CliCta es uno a uno, entonces el enlace CliCta tiene dos flechas, una apuntando al tipo de registro cuenta y otra apuntando al tipo de registro cliente. En la figura aparece un ejemplo en el que la relaciones uno a uno, una cuenta puede pertenecer a exclusivamente a un cliente, y un cliente puede tener exclusivamente una cuenta, como puede verse en el ejemplo de base de datos.

Si la relación CliCta es muchos a muchos, entonces la transformación del diagrama E−R a un diagrama de estructura de árbol es más complicada. Esto se debe a que en el modelo jerárquico solo pueden representarse directamente las relaciones uno a muchos y uno a uno.

Existen varias formas distintas de transformar este diagrama E−R en un diagrama de Estructura de árbol. Sin embargo, todos estos diagramas comparten la propiedad de que el árbol (o árboles) de base de datos tendrá(n) registros repetidos.

La decisión de que método de transformación debe utilizarse depende de muchos factores, entre los que se incluyen:

- El tipo de consultas esperadas en la base de datos.
- El grado al que el esquema global de base de datos que se esta modelando se ajusta al modelo E−R dado.

Si una relación incluye también un atributo descriptivo, la transformación de un diagrama E−R a un diagrama de estructura de árboles mas complicada. Esto se debe a que un enlace no puede contener un valor de un dato. En este caso se necesita crear un nuevo tipo de registro y establecer los enlaces adecuados. La forma de establecer los enlaces dependerá de cómo se defina la relación CliCta.

Si la relación CliCta fuera uno a uno con el atributo fecha, entonces el algoritmo de transformación seria similar al descrito arriba. La única diferencia es que los dos enlaces CliFecha y FechaCta serian uno a uno.

Si la relación CliCta fuera muchos a muchos con el atributo fecha, entonces habría de nuevo varias transformaciones alternativas. Usaremos la transformación más general, similar a la que se aplico en el caso en que la relación CliCta no tenia atributo descriptivo. Los tipos de registro cliente, cuenta y fecha necesitan repetirse y se deben crear dos diagramas de estructura de árbol.

Cliente

Fecha

Cuenta

Cliente

Cuenta

Sucursal

Considérese el diagrama de entidad−relación de la figura, que consta de tres conjuntos de entidades, cliente, cuenta y sucursal, relacionados mediante el conjunto de relaciones general sin atributo descriptivo. Este diagrama especifica que un cliente puede tener varias cuentas, cada una de las cuales se localiza en una sucursal bancaria especifica, y que una cuenta puede pertenecer a varios clientes distintos.

Existen varias formas de transformar este diagrama E−R en un diagrama de estructura de árbol. De nuevo, todos tienen la propiedad de que el árbol (o árboles) de la base de datos se encuentran registros repetidos. La transformación mas directa es crear dos diagramas de estructura de árbol, como se ilustra en la Figura.

Este algoritmo de transformación puede extenderse de una manera directa para tratar relaciones que abarcan más de tres conjuntos de entidades. Simplemente repetimos los distintos tipos de registros y generamos tantos diagramas de estructura de árbol como sea necesario. Este enfoque, a su vez, puede extenderse para tratar una relación general que tenga algunos atributos descriptivos. Todo lo que hace falta es crear un nuevo tipo de registro con un campo para cada atributo descriptivo, y después insertar ese tipo de registro en el lugar apropiado en el diagrama de estructura de árbol.

Varias relaciones

El esquema que se acaba de describir para transformar un diagrama E−R en un diagrama de estructura de árbol garantiza que para cada relación sola, la transformación resultara en diagramas que tienen la forma de un árbol con raíz. Desafortunadamente, aplicar esta transformación de manera individual a cada una de las relaciones de un diagrama E−R no resulta necesariamente en diagramas que son árboles con raíz.

A continuación tratamos medios para resolver el problema. La técnica es dividir los diagramas en cuestión de varios diagramas cada uno de los cuales es un árbol con raíz. Debido al gran número de posibilidades, en vez de presentar un algoritmo general, presentamos dos ejemplos que ilustran la estrategia global que puede aplicarse para tratar este tipo de transformaciones.

Facilidad de recuperación de datos

En esta sección presentamos un lenguaje de consultas para bases de datos jerárquicas que esta basado en DL/I, el lenguaje de manipulación de datos de IMS (Information Management System, creado por IBM). Par simplificar la presentación, nos desviaremos de la sintaxis de DL/I y utilizaremos una notación simplificada. El lenguaje consta de varias órdenes que están incorporadas en Pascal.

Área de trabajo del programa

Todo programa se aplicación que se ejecuta en el sistema consta de una secuencia de sentencias. Algunas de estas son sentencias de Pascal, mientras que otras son sentencias de órdenes del lenguaje de manipulación de datos. Estas sentencias acceden y manipulan los elementos de la base de datos, así como variables declaradas localmente. El sistema mantiene un área de trabajo del programa para cada programa de aplicación, un área de almacenamiento en registros intermedios (buffer) que contiene las siguientes variables:

- Plantillas de registros: un registro (en el sentido de Pascal) por cada tipo de registro al que acceda el programa de aplicación.
- Punteros de actualidad: un conjunto de punteros, uno para cada árbol de base de datos, el cual contiene la dirección del registro en ese árbol particular (sin importar el tipo) al que accedió el programa de aplicación mas recientemente.
- Indicador de estado: una variable asignada por el sistema para indicar al programa de aplicación el resultado de la última operación en la base de datos. Este indicador se llama DB−status y se utiliza el mismo convenio que en el modelo DBTG para indicar fallo; es decir, si DB−status = 0, la ultima operación se realizo con éxito.

Para el ejemplo de sucursal−cliente−cuenta, un área de trabajo contiene:

- Plantillas: un registro para cada uno de los tres tipos de registro:

- Registro sucursal.
- Registro cliente.
- Registro cuenta.

- Puntero de actualidad: un puntero al último registro accedido de tipo sucursal, cliente o cuenta.
- Estado: una variable de estado.

La orden get

La recuperación de datos se realiza mediante la orden get. Las acciones que se toman en respuesta a un get son las siguientes:

- Localizar un registro en la base de datos y hacer que el puntero de actualidad apunte hacia el.
- Copiar ese registro de la base de datos a la plantilla apropiada en el área de trabajo.

La orden get debe especificar en cual de los árboles de la base de datos se va a buscar. Supóngase que se solicito una orden get para localizar el registro cliente que pertenece a Freeman. Una vez que se haya ejecutado con éxito esta orden, los cambios que tienen lugar en el estado del área de trabajo del programa son:

- El puntero de actualidad apunta ahora al registro Freeman.
- La información que pertenece a Freeman se copia en la plantilla del registro cliente en el área de trabajo.
- DB−status se pone a valor 0.

Para revisar todos los registros de forma consistente, debemos imponer un orden en los registros. El que normalmente se utiliza es el preorden. Una búsqueda de preorden empieza en la raíz y busca los subarboles de la raíz de izquierda a derecha, recursivamente. Así pues, empezamos en la raíz, visitamos el hijo más a la izquierda, y así sucesivamente, hasta que alcancemos un nodo hoja (sin hijos). A continuación movemos hacia atrás al padre de la hoja y visitamos el hijo mas a la izquierda no visitado. Continuamos de esta forma hasta que se visite todo el árbol

Acceso dentro de un árbol de base de datos

Existen dos ordenes get diferentes para localizar registros en un árbol de base de datos. La orden más simple tiene la forma:

La cláusula where es opcional. La < condición > que se adjunta es un predicado que puede implicar a cualquier tipo de registro que sea un antecesor de < tipo de registro > o el < tipo de registro > mismo.

La orden get localiza el primer registro (en preorden) del tipo < tipo de registro > en la base de datos que satisfaga la < condición > de la cláusula where, entonces se localiza el primer registro del tipo < tipo de registro >. Una vez que se encuentra ese registro, se hace que el puntero de actualidad apunte a ese registro y se copia el contenido del registro en la plantilla adecuada dentro del área de trabajo. Si no existe ese registro en el árbol de la base de datos, la búsqueda falla y se pone un mensaje de error apropiado en la variable DB−status.

Para ilustrarlo, construyamos la consulta de base de datos que imprime la dirección del cliente Fleming.

Como un ejemplo mas, considérese la consulta que imprime una cuenta que pertenece a Fleming con saldo mayor de 10.000 dólares (si es que existe).

Pueden existir varios registros similares en la base de datos que deseemos recuperar. La orden get first localiza uno de estos. Para localizar a los demás registros de la base de datos puede emplearse la siguiente orden:

Esta orden localiza al siguiente registro (en preorden) que satisface la < condición >. Si se omite la cláusula where, entonces se localiza el siguiente registro del tipo < tipo de registro >. Obsérvese que el sistema utiliza el puntero de actualidad para determinar desde que punto debe reanudarse la búsqueda. Como antes se verán afectados el puntero de actualidad, la plantilla del tipo < tipo de registro > en el área de trabajo y DB−status.

Para ilustrarlo, construyamos la consulta de base de datos que imprima el número de cuenta de todas las cuentas con un saldo mayor de 500 dólares:

Hemos encerrado parte de la consulta en un bucle while, ya que no sabemos por adelantado cuantas de las cuentas cumplen esa condición. Salimos del bucle cuando DB−status sea distinto de 0. Esto indica que la ultima operación get next fallo, lo que implica que hemos agotado todos los registros de cuentas con cuentas saldo > 500.

Las dos ordenes get anteriores localizan un registro de la base de datos del tipo < tipo de registro >dentro de un árbol de base de datos determinado. Sin embargo existen circunstancias en las que queremos localizar un registro de este tipo dentro de un subarbol en particular. Es decir, queremos limitar la búsqueda a un subarbol específico en vez de buscar en todo el árbol de base de datos. La raíz del subarbolen cuestión es el ultimo registro que se localizo con las ordenes get first o get next. La orden get para localizar un registro dentro de ese subarbol tiene la forma:

Que localiza el siguiente registro (en preorden) que satisface la < condición > en el subarbol cuya raíz es el padre del actual de < tipo de registro >. Si se omite la cláusula where, entonces se localiza el siguiente registro del tipo < tipo de registro >dentro del subarbol indicado. Obsérvese que el sistema utiliza el puntero de actualidad para determinar desde que punto debe reanudarse la búsqueda. Como antes, se verán afectados el puntero de actualidad y la plantilla del tipo < tipo de registro >. Sin embargo, en este caso se pone un valor distinto de 0 en DB−status si no existe el registro dentro del subarbol indicado, no dentro de todo el árbol.

Para ilustrar la ejecucion de la orden get, construyamos la consulta que imprime el saldo total de todas las

cuantas que pertenecen a Boyd:

Nótese que salimos del bucle while e imprimimos el valor de suma solamente cuando DB−status adquiere un valor distinto de 0. Esto ocurre cuando la operación get next within parent falla, lo que indica que hemos agotado todas las cuantas cuyo propietario es el cliente Boyd.

Facilidad de actualización

En la Sección de recuperación de datos describimos las órdenes para consultar la base de datos. En esta sección describiremos los mecanismos de que se dispone para actualizar la información en la base de datos. Estos incluyen la inserción y la eliminación de registros, así como la modificación de registros existentes.

Creación de registros nuevos

Para insertar un registro del tipo < tipo de registro > en la base de datos, primero debemos poner los valores adecuados en la plantilla del <tipo de registro > correspondiente en el área de trabajo.

Una vez hecho esto, añadimos este registro nuevo al árbol de la base de datos ejecutando:

insert < tipo de registro >

where < condición >

Si se incluye la cláusula where, el sistema busca en el árbol de la base de datos (en preorden) un registro que satisfaga la < condición > de la cláusula where. Una vez que encuentre ese registro, llamémosle X, el registro recién creado se inserta en el árbol como hijo más a la izquierda de X. Si se omite la cláusula where, el registro se inserta en la primera posición (en preorden) en el árbol de la base de datos donde se pueda insertar un registro del tipo < tipo de registro > de acuerdo con el esquema especificado en el diagrama de estructura de árbol correspondiente.

Modificación de un registro existente

Para modificar un registro que ya existe del tipo < tipo de registro >, debemos copiar en la plantilla correspondiente del area de trabajo y cambiar los campos deseados en esa plantilla. Una vez que se haya hecho esto, reflejamos los cambios en la base de datos ejecutando:

replace

Nótese que la orden replace no tiene < tipo de registro >como argumento. El registro que se vera afectado es aquel al que apunte el puntero de actualidad, que debe ser el registro deseado.

El lenguaje DL/I requiere que antes de modificar un registro, la orden get incluya la orden adicional hold de forma que el sistema se entere que se va a modificar un registro.

Eliminación de un registro

Para eliminar un registro del tipo < tipo de registro >, debe hacerse que el puntero de actualidad apunte a ese registro. Siguiendo esto, podemos eliminar ese registro ejecutando:

delete

Nótese que como en el caso de la modificación de registros, la orden get debe incluir el atributo hold.

Una operación de eliminación borra no solo el registro en cuestión, sino todo el subarbol del cual es raíz

Registros virtuales

Hemos visto que en el caso de las relaciones muchos a muchos es necesario repetir registros para conservar la organización de estructura de árbol de la base de datos. La repetición de registros tiene dos inconvenientes importantes:

- La actualización puede producir inconsistencia de los datos.
- El desperdicio de espacio es inevitable.

A continuación tratamos formas de eliminar estos problemas.

Para eliminar la repetición de registros necesitamos relajar el requisito de que la organización lógica de los datos debe tener forzosamente estructura de árbol. Sin embargo, hay que tener cuidado al hacerlo, pues, de lo contrario, iríamos a parar al modelo de red.

La solución es introducir el concepto de registro virtual. Un registro virtual no contiene valores, sino un puntero lógico a un registro físico determinado. Cuando se va a repetir un registro en varios árboles de la base de datos, mantenemos una sola copia de ese registro en uno de los árboles y sustituimos cada uno de los otros registros por un registro virtual que contiene un puntero a ese registro físico.

Para eliminar la repetición de datos, creamos dos tipos de registros virtuales: cliente−virtual y cuenta−virtual. A continuación sustituimos el tipo de registros cuenta por el tipo de registro cuenta−virtual en el primer árbol y el tipo de registro cliente por el tipo de registro cliente−virtual en el segundo árbol. También añadimos una línea discontinua desde el registro cliente−virtual al registro cliente y otra línea discontinua desde el registro cuenta−virtual al registro cuenta, para especificar la asociación entre un registro virtual y su correspondiente registro físico.

El lenguaje de manipulación de datos para esta nueva configuración sigue siendo el mismo que en el caso en que se permite la repetición. Así, el usuario no necesita enterarse de estos cambios. Solo se afecta a la implementación interna

IMS

El sistema de gestión de información (IMS, Information Management System) de IBM es uno de los sistemas de base de datos más antiguos y más ampliamente utilizados. Trabaja con el sistema operativo MVS en maquinas grandes basadas en la arquitectura IBM 370.

Las consultas en base de datos IMS se hacen por medio de llamadas incorporadas en un lenguaje principal. Las llamadas incorporadas son parte del lenguaje de base de datos DL/I del IMS.

Puesto que el rendimiento es de importancia critica en las bases de datos grandes, IMS permite al diseñador de base de datos un gran número de opciones en el lenguaje de definición de datos. El diseñador de base de datos define una jerarquía física como el esquema de la base de datos. Pueden definirse varios subesquemas (o vistas) construyendo una jerarquía lógica a partir de los tipos de registro que constituyen el esquema. Las diversas opciones de que dispone el lenguaje de definición de datos (tamaños de bloques, campos de punteros especiales, etc...) permiten al administrador de la base de datos afinar el sistema con el fin de mejorar su rendimiento.

IMS cuenta con varios esquemas de acceso a registros:

- HSAM (método jerárquico de acceso secuencial, hierachical sequential−access method) se utiliza para archivos secuénciales físicamente (como los archivos en cinta). Los registros se almacenan físicamente en preorden.
- HISAM (método jerárquico de acceso secuencial indexado, hierachical indexed−sequential−access method) es una organización indexada en el nivel de raíz de la jerarquía.
- HIDAM (método jerárquico indexado de acceso directo, hierachical indexed−direct−access method) es una organización indexada en el nivel de la raíz con punteros a los registros hijos.
- HDAM (método jerárquico de acceso directo, hierachical direct−access method) es similar al HIDAM, pero con acceso por asociación (hash) al nivel de la raíz.

La versión original de IMS antecede al desarrollo de la teoría de control de concurrencia. Las primeras versiones de IMS tenían una forma sencilla de control de concurrencia. Solo podía ejecutarse un programa que incluyera actualizaciones a la vez. Sin embargo, podía ejecutarse de manera concurrente cualquier número de aplicaciones de lectura con una aplicación de actualización. Esta característica permitía a las aplicaciones leer actualizaciones no cometidas y también permitía ejecuciones no serializables. La única opción disponible para las aplicaciones que requerían un grado mayor de aislamiento contra las anomalías del procesamiento concurrente era el acceso exclusivo a la base de datos.

Versiones posteriores de IMS incluyeron una característica de aislamiento de programas más sofisticada que permitía un mejor control de la concurrencia y técnicas de recuperación de transacciones en línea en vez de las transformaciones por lote que eran la norma en un principio.

La necesidad de un procesamiento de transacciones de alto rendimiento condujo a la introducción de IMS Fast Path (Camino rápido). Fast Path utiliza una organización física de datos alternativa diseñada para permitir que las partes mas activas de la base de datos residan en la memoria principal. En vez de forzar la salida de actualizaciones al disco al final de una transacción (como lo hace el IMS estándar), la actualización se posterga hasta un punto de verificación o de sincronización. En el caso de una caída, el subsistema de recuperación debe volver a hacer todas las transacciones cometidas cuyas actualizaciones no se grabaron en disco. Estos y otros trucos permiten una velocidad de procesamiento de transacciones muy alta. Como la memoria principal ha llegado a ser más grande y menos cara, ha habido un interés creciente en desarrollar sistemas de base de datos de memoria principal. IMS Fast Path es un precursor de gran parte de este trabajo.

Sistema 2000

El Sistema 2000 es un sistema de base de datos jerárquico desarrollado en u principio por MRI Corporation y mas tarde adquirido por Intel. El Sistema 2000 se ejecuta en computadores tipo IBM 370, Univac 1100, CDC 6000 y Cyber. Aunque utiliza el mismo modelo de datos que IMS, las características del lenguaje que ofrece el Sistema 2000 presentan varias diferencias interesantes con respecto al lenguaje DL/I de IMS.

1* nombre−cliente (name);

2* calle (name);

3* ciudad−cliente (name);

4* cuenta (repeating group);

5* numero−cuenta (integer);

6* saldo (money);

Name, integer y money son tres de los tipos incorporados en el Sistema 2000. Cada campo puede

especificarse como key (clave) o nonkey (no clave). El Sistema 2000 construye un índice por cada uno de los campos clave.

Considérese la consulta Encontrar el nombre y la ciudad de todos los clientes que tienen una cuenta cuyo saldo es mayor de 10000 dólares. Si se utiliza el lenguaje DL/I, debemos revisar todos los registros de cuentas de cada cliente escribiendo un bucle while en el lenguaje principal. EL lenguaje de consultas del Sistema 2000, QUEST, nos permite escribir esta consulta utilizando una sola sentencia:

list nombre−cliente, ciudad−cliente

while cliente has saldo > 10000

También es posible incorporar las consultas de Sistema 2000 en u lenguaje principal. Un precompilador procesa el programa. El análisis sintáctico de las consultas se hace durante la fase de Pre−compilación. Al igual que la mayor parte de los sistemas comerciales, el Sistema 2000 incluye facilidades que ayudan en la generación de informes.

Modelo de Datos Orientado A Objetos

La importancia de la orientación a objeto recorta el tiempo que toma a obtener resultados temporales.

La orientación a objetos permite la forma más simple en el manejo de los datos tanto para el diseño relacional como la elaboración de interfaz entre los diversos lenguajes orientados a objetos de base de datos.

Bases de datos orientadas por objetos.

Las aplicaciones de las bases de datos en áreas como el diseño asistido por computadora, la ingeniería de software y el procesamiento de documentos no se ajustan al conjunto de suposiciones que se hacen para aplicaciones del estilo de procesamiento de datos. El modelo de datos orientado por objetos se ha propuesto para tratar algunos de estos nuevos tipos de aplicaciones.

El modelo de bases de datos orientado por objetos es una adaptación a los sistemas de bases de datos. Se basa en el concepto de encapsulamiento de datos y código que opera sobre estos en un objeto. Los objetos estructurados se agrupan en clases. El conjunto de clases esta estructurado en sub. y superclases basado en una extensión del concepto ISA del modelo Entidad − Relación. Puesto que el valor de un dato en un objeto también es un objeto, es posible representar el contenido del objeto dando como resultado un objeto compuesto.

El propósito de los sistemas de bases de datos es la gestión de grandes cantidades de información. Las primeras bases de datos surgieron del desarrollo de los sistemas de gestión de archivos. Estos sistemas primero evolucionaron en bases de datos de red o en bases de datos jerárquicas y, más tarde, en bases de datos relaciónales.

Estructura de objetos.

El modelo orientado por objetos se basa en encapsular código y datos en una única unidad, llamada objeto. El interfaz entre un objeto y el resto del sistema se define mediante un conjunto de mensajes.

Un objeto tiene asociado:

- Un conjunto de variables que contienen los datos del objeto. El valor de cada variable es un objeto.
- Un conjunto de mensajes a los que el objeto responde.
- Un método, que es un trozo de código para implementar cada mensaje.
- Un método devuelve un valor como respuesta al mensaje.

El término mensaje en un contexto orientado por objetos, no implica el uso de un mensaje físico en una red de computadoras, si no que se refiere al paso de solicitudes entre objetos sin tener en cuenta detalles específicos de implementación.

La capacidad de modificar la definición de un objeto sin afectar al resto del sistema está considerada como una de las mayores ventajas del modelo de programación orientado a objetos.

Jerarquía de clases.

En una base de datos existen objetos que responden a los mismos mensajes, utilizan los mismos métodos y tienen variables del mismo nombre y tipo. Sería inútil definir cada uno de estos objetos por separado por lo tanto se agrupan los objetos similares para que formen una clase, a cada uno de estos objetos se le llama instancia de su clase. Todos los objetos de su clase comparten una definición común, aunque difieran en los valores asignados a las variables.

Así que básicamente las bases de datos orientadas por objetos tienen la finalidad de agrupar aquellos elementos que sean semejantes en las entidades para formar un clase, dejando por separado aquellas que no lo son en otra clase.

Por ejemplo: Retomemos la relación alumno−cursa−materia agregándole la entidad maestro; donde los atributos considerados para cada uno son alumno: Nombre, Dirección, Teléfono, Especialidad, Semestre, Grupo; Maestro: Nombre, Dirección, Teléfono, Número económico, Plaza, RFC; Materia: Nombre, Créditos, Clave.

Los atributos de nombre, dirección y teléfono se repiten en la entidad alumno y maestro, así que podemos agrupar estos elementos para formar la clase Persona con dichos campos. Quedando por separado en alumno: Especialidad, semestre, Grupo. Y en maestro: Número económico, Plaza y RFC; la materia no entra en la agrupación (Clase persona) ya que la clase específica los datos de solo personas, así que queda como clase materia.

Herencia.

Las clases en un sistema orientado por objetos se representan en forma jerárquica como en el diagrama anterior, así que las propiedades o características del elemento persona las contendrán (heredaran) los elementos alumno y maestro. Decimos que tanto la entidad Alumno y maestro son subclases de la clase persona este concepto es similar al utilizado en la de especialización (la relación ISA) del modelo E−R.

Se pueden crear muchas agrupaciones (clases) para simplificar un modelo así que una jerarquía (en forma gráfica) puede quedar muy extensa, en estos casos tenemos que tener bien delimitados los elementos que intervienen en una clase y aquellos objetos que las heredan.

Consultas orientadas por objetos:

Los lenguajes de programación orientados por objetos requieren que toda la interacción con objetos se realiza mediante el envío de mensajes.

Consideremos el ejemplo de alumno−cursa−materia deseamos realizar la consulta de los alumnos que

cursan la materia de Base de Datos 1, para realizar esta consulta se tendría que enviar un mensaje a cada instancia alumno

Así un lenguaje de consultas para un sistema de bases de datos orientado por objetos debe incluir tanto el modelo de pasar el mensaje de objeto a objeto como el modelo de pasar el mensaje de conjunto en conjunto.

Complejidad de Modificación.

En base de datos orientados por objetos pueden existir los siguientes cambios:

Adición de una nueva clase: Para realizar este proceso, la nueva clase debe colocarse en la jerarquía de clase o subclase cuidando las variables o métodos de herencia correspondientes.

Eliminación de una clase: Se requiere la realización de varias operaciones, se debe de cuidar los elementos que se han heredado de esa clase a otras y reestructurar la jerarquía.

En sí la estructuración de modelos orientados por objetos simplifica una estructura evitando elementos o variables repetidas en diversas entidades, sin embargo el precio de esto es dedicarle un minucioso cuidado a las relaciones entre las clases cuando en modelo es complejo, la dificultad del manejo de objetos radica en la complejidad de las modificaciones y eliminaciones de clases, ya que de tener variables que heredan otros objetos se tiene que realizar una reestructuración que involucra una serie de pasos complejos.

CLASIFICACION DE LENGUAJES ORIENTADOS A OBJETOS

Los principales lenguajes de programación utilizados actualmente para sistema de tiempo real son C y ADA.

ADA: Fue diseñado específicamente para la implementación de sistemas en tiempo real, especialmente empotrados.

Los lenguajes de programación orientado a objetos son: Eiffel, Prolog, Simula, Pascal, Smalltalk, C++ y Object.

Ventajas del Modelo orientado a objetos con respecto al modelo estructurado

- Un modelo de objetos es más cercano a la realidad que un modelo funcional.
- Un desarrollo realizado con el modelo orientado a objetos es más fácil de mantener y de reutilizar.
- El modelo orientado a objetos evita la redundancia en los procesos luego los códigos son más entendibles y resumidos
- La integridad que dan los objetos a los datos evita ambigüedades en su uso, dando mayor seguridad en los resultados
- El modelo orientado a objetos facilita la integridad de módulos que hallan sido realizados por separado sin correr riesgos en el manejo de los datos.

CARACTERISTICAS DE LOS LENGUAJES ORIENTADOS A OBJETOS

La orientación a objetos se basa en tres pilares básicos: − encapsulación y ocultación de la información − abstracción y clasificación:

Herencia y Polimorfismo

TIPIFICACIÓN ESTRICTA (FUERTE):

Es el proceso de declarar el tipo de información que puede contener una variable.

ENCAPSULAMIENTO:

La encapsulación significa agrupar en una sola entidad ('caja') datos y operaciones sobre esos datos. Evidentemente todo programa es una encapsulación, pero por encapsulación queremos decir algo mas preciso: en concreto, cada componente ha de tener nada más que los datos relacionados con un tema y su manipulación. Por ejemplo cualquier programa de facturación puede leer directamente un registro de artículo y modificar el 'stock' al mismo tiempo que imprime la factura y modifica el saldo del cliente. Tenemos tres actividades sobre tres entidades diferentes en un único componente ('el programa'). Este programa no tiene una encapsulación adecuada.

Es deseable que el lenguaje soporte ocultación de la información, mediante partes independientes.

COMPILACIÓN INCREMENTAL:

Características en el desarrollo de sistemas grandes, en donde el sistema se crea e implementa en un modo sistemático.

GENERICIDAD:

Las clases parametrizadas sirven para soportar un alto grado de reusabilidad (reutilización).

La herencia permite entonces describir entidades (clases) por diferencia entre ellas; por eso se dice que programar con orientación a objetos es programar para diferenciar.

La utilidad de la herencia es múltiple. En primer lugar hay que analizar, diseñar, codificar y poner a punto menos, ya que en cuanto se tiene un concepto definido, jamás lo repetimos. Como mucho, si el concepto nuevo es muy parecido a uno anterior, se define el nuevo como la diferencia respecto al anterior. Si surge un error en la utilización de la nueva entidad, seguro que el error está en lo que se ha añadido porque lo que se ha heredado ya se había probado.

Hay diferentes tipos de herencia: los más importantes son simple y múltiple. Con esto se quiere decir que, al contrario de la herencia biológica donde siempre se hereda de dos padres, en la orientación a objetos se puede heredar de uno, dos o más padres. La definición de cliente minorista a partir de cliente es un ejemplo de herencia simple. Y se podría definir un vehículo anfibio como una herencia (una subclase) de otras dos clases: coche y barco.

POLIMORFISMO:

Los lenguajes deben permitir que existan operaciones con igual nombre, que se utilicen para manejar objetos diferentes en tiempo de ejecución.

CONCURRENCIA:

Es conveniente que el lenguaje soporte la creación de procesos paralelos independientes del sistema operativo.

PERSISTENCIA:

Los objetos deben ser persistentes: es decir que los objetos han de poder permanecer después de la creación del programa.

DATOS COMPARATIVOS:

Los módulos se deben poder comunicar mediante memoria compartida además del paso de mensaje.

LENGUAJES ORIENTADOS A OBJETOS

SIMULA:

Sus propósitos fueron describir sistemas y programar simulaciones. Su desarrollo fue motivado por el deseo de:

- Expresar procesos que son permanentes y activos.
- Crear y distribuir los procesos.
- Extender un lenguaje para incluir procesos.
- Proporcionar a lo procesos un mecanismo de ejecución.

TAXONOMIA DE LENGUAJES ORIENTADOS A OBJETOS

Una taxonomía de lenguajes de programación orientación a objetos fue creada por Wegner.

Proporciona una definición estricta de los lenguajes de programación orientado a objetos. Que han prevalecido en la época actual según esta taxonomía, no es suficiente que un lenguaje soporte la creación de objetos para ser considerado Orientado a Objetos.

La clasificación incluye los siguientes grupos:

BASADOS EN OBJETOS:

Un lenguaje de programación es basado en objetos si su sintaxis y semántica soportan la creación de clases se considera basado en clases.

BASADOS EN CLASES:

Si un lenguaje de programación es basado en objetos y soporta además la creación de clases se considera basada en clases.

ORIENTADO A OBJETOS.

Un lenguaje de programación orientado a objeto es un lenguaje basado en clase que soporta también herencia.

TAXONOMÍA DE LENGUAJES ORIENTADOS A OBJETOS DE WAGNER

OBJETOS BASADO EN OBJETOS + CLASE

Ada−83

Actor BASADO EN CLASES

Clipper 5.X ORIENTADO A OBJETOS

Turbo Borland Pascal

Delphi Visual

Object Cobol

Ada−95

LISP:

Ofrece mecanismos que facilitan directamente la implementación de los conceptos de la orientación a objetos.

Estos mecanismos eran:

Un enfoque altamente dinámico para la creación de objetos.

Gestión automática de memoria.

Disponibilidad de la implementación de estructuras de datos.

Selección de operaciones en tiempo de ejecución.

Modelo Formal Deductivo Orientado a Objetos

Los Diagramas Entidad Relación, como su nombre lo indica están formados, en esencia, de entidades y relaciones entre ellos. Formalmente, ERD = composición (Entidad, Relación)

Entidad = composición(Nombre, Atributo)	Relación = composición(Nombre, Cardinalidad)

Una entidad tiene un nombre representativo y un conjunto de atributos propios de él. Formalmente,

Entidad = atribución (Nombre, Atributo)

El nombre de la entidad está conformado por número y letras, con la restricción que éste no puede empezar con número, sino con letra, además del símbolo ``_". También está conformado por un tamaño máximo de 32 caracteres. Formalmente,

Nombre = composición ([a − z, A − Z, _]+ [a − z, A − Z, _, 0 − 9]* long)

Long = evaluación (maxlength (32))

Los atributos que conforman a una entidad están formados por un nombre de este atributo, de qué tipo es, su tamaño, una breve descripción y el rango de valores de este atributo. Formalmente,

Atributo = composición (Nombre, Tipo, Longitud, Descripción, Rango)

El nombre de un atributo esta formado de la combinación de letras y números y el carácter especial ``_". Sin embargo, un nombre válido no puede empezar con número, pero sí con los demás caracteres. Formalmente,

Nombre = [a − z, A − Z, _]+ [a − z, A − Z, _, 0 − 9]*

Los tipos que se aceptan para los atributos son numéricos y carácter. El tipo numérico como su nombre lo indica está formado de puros números, mientras que el carácter está formado por números y letras, haciendo destacar que un tipo carácter no puede empezar con un número como primer elemento.

Tipo = composición (numérico, carácter)

Numérico = [0 −9]*

Carácter = [a − z, A − Z, _]+ [a − z, A − Z, _, 0 − 9]

Dependiendo del tipo seleccionado, se define la longitud que éste va a tener. Formalmente,

Longitud = evaluación (maxlength (Tipo))

La descripción de las entidades es una secuencia lógica de caracteres y números. Formalmente,

Descripción = [a −z, A − Z , _, 0 − 9]*

Por su parte, la relación tiene un nombre y una cardinalidad. Formalmente,

Relación = atribución (Nombre, Cardinalidad)

El nombre está formado de número y caracteres y una longitud. Se cuenta con la restricción en el nombre que no puede empezar con un número. Formalmente,

Nombre = composición ([a − z, A − Z,_]+[a −z, A − Z, _, 0 − 9]*, long)

Long = evaluación (maxlength (32))

Cardinalidad

Está formada por la dirección que tiene, y el tipo de relación de la que se trata. Con respecto a la dirección es la manera en que se presenta desde donde es la relación y hasta donde llega; por otra parte el tipo muestra, como su nombre lo indica, que tipo de relación es, a saber. Formalmente,

Cardinalidad = composición (dirección tipo)

Dirección = interfase (desde, hasta)

Tipo = interfase (0:1, 1:1, 0:N, 1:N)

Glosario Técnico

Algoritmo : Conjunto de pasos a seguir para la solución de un problema

Árboles : Nombre que se le da a una representación de datos de esquema sobre una base de datos

Atributos : Nombre que se diferencian las entidades

Base de datos : Aplicación informática para manejar información en forma de "fichas": clientes, artículos, películas, etc.

Booleano : Lógica que maneja operadores algebraicamente

Bloques : Grupo de instrucciones tratadas conjuntamente

Buffer : Zona de almacenamiento temporal

Código : Lenguaje utilizado por el computador, utilizado para programar un conjunto de instrucciones

Cadena : Conjunto cuyos elementos están ordenados literalmente según un cierto criterio

Caracteres : instrucciones contenidas en un programa, y entendibles por el ordenador

Codasyl : Modelo de datos de tipo red que introduce restricciones inherentes

DL/I : Data Lenguaje 1. Lenguaje de programación para bases de datos jerárquicas

DB−status : Categoría de enchufes y zócalos

DBTG : (Data Base Task Group) estándares para COBOL

DML : Lenguaje de manipulación de datos

Eiffel, Prolog,

Simula, Pascal,

Smalltalk,

C++ y Object : Lenguajes de programación orientados para objetos

Encapsulamiento: Proceso por el cual los datos que se deben enviar a través de una red

E−R : Entidad−Relación

Entidad : objetos concretos o abstractos que presentan interés para el sistema

Enlaces : Unión entre una entidad y atributos

Estructura : Cuerpo de esquema de una base de datos

Grafo : Estructura de datos utilizada en algunos lenguajes de programación

HSAM : Método jerárquico de acceso secuencial

HISAM : Método jerárquico de acceso secuencial indexado

HIDAM : Método jerárquico indexado de acceso directo

HDAM : Método jerárquico de acceso directo

Interrelaciones : Un Atributo se relaciona con muchos atributos a la vez

IMS : Sistema de administración de información

IBM : Internacional Business Machines

ISA : Tipo de arquitectura estándar de placas base

Intel : Fabricante de los procesadores que llevan su nombre

IBM 370 : Modelo de computador de la IBM

Interfaz : Conexión e interacción entre hardware, software y el usuario

LISP : Procesamiento de listas

Long : Longitud palabra en idioma ingles

Maxlength : Máximo de longitud (ingles)

MVS : Sistema operativo de IBM para sus grandes ordenadores o mainframes

Nodo : Punto en donde se producen dos o más conexiones en una red de comunicaciones

Ocurrencia : Datos que se agregan a un registro

Plex : Sinónimo de Estructura de red

Polimorfismo : Operaciones con igual nombre

Pre−compilación : Para crear programas en un cierto lenguaje de programación

Pascal : Lenguaje de programación

SGBD : Sistemas de Gestión de Bases de Datos

Sistemático :

Software : La parte "que no se puede tocar" de un ordenador

Taxonomia : Clasificación jerárquica y ordenada

Univac 1100 : Modelo de Computador

Vector : Línea designada por sus puntos extremos (coordenadas x−y o x−y−z)

VPH : Vinculo Padre−Hijo

Bibliografía

- Estructura de datos y diseño de programas

Robert L. Kruse

- Monografías punto com.
- Diseño de Base de Datos
-

2

Ei

Ej

Iij

ei1

ei2

ei3

ej1

ej2

ej3

ej4

Ei

Ej

Iij

ei1

ei2

ei3

ej1

ej2

ej3

ej4

ej5

ej6

Ei

ei1

ei2

ei3

ej1

ej2

ej3

ej4

ej5

ej6

Iij

Ei

ei1

ei2

ej1

ej2

ej3

Iij

Ei

Ej

Iijk

ei1

ei2

ej1

ej2

ej3

ek2

Ek

ej4

ek1

Editorial

Libro

Editorial_1

Libro_3

Libro_4

Libro_1

Libro_3

Pública

A

B

C

D

E

F

G

A

B

C

D

E

F

G

Set 1

Set2

Set3

Set4

Set5

Cuentas

Lowman

Square

Dallas

Camp

Downridge

Garland

Kahn

Bayside

Philadelphia

305

500

675

155

226

336

A

Bn

B1

B2

Ck

Cm

C1

Nombre

Calle

Ciudad

Saldo

Número

Cliente

Cuenta

CliCta

Ciudad

Calle

Nombre

Número

Saldo

Número

Saldo

Nombre

Calle

Ciudad

Fecha

Número

Saldo

Nombre

Calle

Ciudad

Fecha

Número

Saldo

Nombre

Calle

Ciudad

Nombre

Activo

Ciudad

Número

Saldo

Nombre

Activo

Ciudad

Nombre

Calle

Ciudad

get first < tipo de registro >

where < condicion >

get first cliente

```
where cliente.nombre = Fleming;
print (cliente.direccion);
get first cuenta
where cliente.nombre = Fleming and cuenta.saldo > 10000;
if DB−status = 0 then print (cuenta.numero);
get next < tipo de registro >
where < condicion >
get first cuenta
where cuenta.saldo > 500;
while DB−status = 0 do
begin
print (cuenta.numero);
get next cuenta
where cuenta.saldo > 500;
end;
get next within parent < tipo de registro >
where < condicion >
suma := 0;
get first cliente
where cliente.nombre = Boyd;
get next within parent cuenta;
while DB−status = 0 do
begin
suma := suma + cuenta.saldo;
get next within parent cuenta;
end;
```

print (suma);

-