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.

Esta representación no impone en principio ninguna restricción ni al tipo ni al número de los arcos permitiéndote el modelado de estructuras de datos tan complejas como se desee. Podemos definir el modelo en red general con una mayor formalización, como un conjunto finito de tipos de entidades {E1,E2, ... ,En} con sus respectivas propiedades (atributos): {A11,A12,A13...,Anm} y un conjunto finito de interrelaciones (al igual que las entidades y que los atributos tienen un nombre {Ihj,k...n}.Con la notación anterior representamos la interrelación entre los elementos j,k...,n (bien sean entidades y / u otras interrelaciones), donde el superíndice h permite diferenciar dos interrelaciones distintas entre los mismos elementos, ya que se refiere al nombre de la interrelación.

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 practica. 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 mas de dos tipos de entidad y ocurrencias de la misma

Propuesta CODASYL

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.

Historia

El termino CODASYL proviene de COnference on DAta SYstem Languajes, una organización no oficial compuesta por persona privadas (entre las que se encontraban también miembros de varios departamentos del gobierno de estados unidos), apoyadas por instituciones, con el fin de estudiar y normalizar aspectos relativos a la utilización y diseño de bases de datos. a pesar de su carácter no oficial, los informes y normas CODASYL tuvieron una amplia difusión y respaldo, existiendo muchos sistemas comerciales que se adaptaban a dichas

normas. la aceptación las normas CODASYL se debió en gran medida al apoyo brindado por el departamento de defensa de los estados unidos.

Los orígenes del grupo CODASYL se remontan al año 1959. en el mes de abril de dicho año se celebro una reunión de un grupo de constructores y usuarios de ordenadores con el fin de abordar la tarea de definición de un lenguaje de programación común orientado al tratamiento y resolución de problemas de gestión (el que llegaría a ser el lenguaje COBOL). Un poco mas tarde, a finales del mes de Mayo del mismo año, se decidió la continuación de los trabajos y el grupo tomo el nombre de COnference on DAta SYstem Languajes (CODASYL).

En principio las tareas se distribuyeron en tres comités coordinados por un comité ejecutivo. el grupo CODASYL obtuvo un gran éxito con la especificaciones del COBOL, por lo que se decidió a ampliar el espectro de sus estudios y actividades a todos aquellos tema que pudiesen ser de utilidad a COBOL.

En 1964 el grupo sufrió una reestructuración y poco tiempo después en junio de 1965, se creo el List Processing Task Force, dedicado a estudiar las capacidades de proceso de listas para el lenguaje COBOL. en ese mismo año se presento el documento List Processing Extension to Mass Storage, en el que se proponían los cambios necesarios para hacer posible el trabajo con listas en COBOL. Este grupo cambio su nombre dos años más tarde por el de DataBase Task Group (DBTG), con cuyas siglas fue conocido. Este cambio de nombre muestra el creciente interés que despertó hacia mediados y finales de los años setenta el desarrollo de normas y especificaciones en el campo de las bases de datos.

Dentro del grupo destacaron tres personas, a las que se considera sus creadores y máximos impulsores: G.G Dodd, de los laboratorios de la General Motors, con su trabajo sobre un lenguaje de programación asociativo; el presidente del grupo, W.G Simmons, de la United States Steele Corporation, que con gran acierto animo y coordino todos los trabajos y actividades, y C.W Bachman, padre del SGBD de la compañía General Electric, IDS (Luego IDS II de Bull), que además de su labor en el área de almacenamiento integrado de datos propuso la estructura de datos propia del modelo CODASYL y los conocidos diagramas que llevan su nombre.

En el verano de 1968, el comité ejecutivo de CODASYL decidió reorganizar de nuevo el grupo, modificando la estructura de los comités. el resultado fue la formación de tres comités permanentes:

- Comité de sistemas: encargado del desarrollo de lenguajes y técnicas avanzadas para el proceso eficiente de datos.
- Comité de planificación: cuya misión consistía en obtener información de los usuarios e instrumentadores sobre aspectos relacionados con los objetivos perseguidos por CODASYL, de forma que pudieran servir de ayuda en la tarea de planificación de sistemas
- Comité de lenguajes de programación (PLC): que asumió y extendió los objetivos planteados por un subcomité anterior y se responsabilizo del desarrollo y mantenimiento de las especificaciones del COBOL. de este comité encargado también de todas las actividades relativas a las bases de datos, dependía el grupo DBTG.

Después de una primera versión en 1968, en octubre de 1969 e grupo DBTG presentó al PLC el primer informe , que suscito un gran interés, pero que también fue objeto de multitud de criticas. el informe recogía la especificación de la sintaxis y semántica de un lenguaje de definición de datos (DDL) que permitía la descripción de una base de datos, así como de un lenguaje de manipulación de datos (DML); el DML se configuro como una extensión del lenguaje anfitrión, pero en la practica su orientación a COBOL era indiscutible. En estas propuestas de 1969 se definía una arquitectura de base de datos estructurada a dos niveles, esquema y subesquemas, lo cual hacia bastante difícil conseguir la independencia lógico / física deseable en un sistema de base de datos. quizás fue este, junto con su orientación a COBOL, el aspecto mas criticado de las especificaciones contenidas en el informe.

Los trabajos del DBTG continuaron activamente hasta abril de 1971, incluyéndose varios cambios y ampliaciones del informe anterior. se proponían dos lenguajes de descripción de datos, uno para el esquema DDL−Schema, y otro para el subesquema, DDL−SubSchema, con el fin de mejorar la independencia entre los aspectos lógico y físico de la base de datos. En ese mismo año, el comité ejecutivo de CODASYL acepto la recomendación del PLC sobre la creación de un comité independiente encargado del desarrollo y mantenimiento de las especificaciones del lenguaje de definición de datos, al que se llamo comité de lenguaje de definición de datos CODASYL (DDLC). la tarea de añadir a las especificaciones del COBOL el lenguaje de definición de datos del subesquema y las facilidades del lenguaje de manipulación de datos fue encomendada a un subgrupo del DBTG (más tarde convertido en grupo del lenguaje de manipulación DMLTG).

Esta dualidad dio lugar a que se produjesen inconsistencias entre la definición del esquema y la del subesquema y por otro lado con la manipulación de datos. tampoco la terminología empleada por estos dos grupos era uniforme.

En 1973, el DDLC publico en el journal of developement su propuesta para el lenguaje de definición de datos del esquema. En este informe se reconocía que el desarrollo de los lenguajes de bases de datos debía realizarse de forma evolutiva, al igual que lo había sido el COBOL.

Ese mismo año, el DDLC comenzó su colaboración con un grupo de trabajo de la British Computer Society, el Database Administration Working Group (DBAWG), creándose un grupo de trabajo conjunto, denominado BCS/CODASY DDLC. La contribución más importante de este grupo fue el desarrollo y especificación de un lenguaje de almacenamiento de datos (DSDL), que define la representación de la base de datos desde el punto de vista del almacenamiento físico. El informe que contenía las especificaciones del DSDL fue incluido como un anexo en los informes de 1978 y 1981. En estos informes se presentaban los lenguajes de definición y manipulación de datos para una arquitectura a tres niveles (similar a la del grupo ANSI / SPARC de 1975 y 1977). Además de las especificaciones del DSDL, se incluían también las modificaciones necesarias para eliminar ciertas cláusulas del DDL que se consideraba tenían excesivas implicaciones físicas, por lo que se hacia recomendable su desaparición del DDL para introducirlas en el lenguaje de definición del almacenamiento de datos.

Mientras se realizaban estos trabajos e informes, continuaban las actividades de los diferentes comités de CODASYL en otras áreas y se seguían creando nuevos comités encargados de diferentes tareas. en 1974 se formó el comité del lenguaje FORTRAN para bases de datos (FDBLC), que realizo la especificación del lenguaje de datos del subesquema y del lenguaje de manipulación de datos para FORTRAN. Sus trabajos se publicaron en 1977 y 1980. E 1976 se creó un nuevo comité para investigar los requisitos de las bases de datos con vistas a su utilización por usuarios finales, denominado comité de facilidades par usuarios finales (EUFC), cuyas conclusiones fueron publicadas en 1980. Finalmente, cabe destacar la creación en 1978 del comité de lenguajes de mandatos para sistemas operativos comunes (COSCL), el cual publicó sus trabajos en 1980. También se formaron otros grupos específicos dentro de los distintos comités permanentes, con la misión de estudiar aspectos puntuales concernientes al modelo; la mayor parte de ellos fueron disueltos una vez publicados los resultados.

El grupo CODASYL se disolvió en 1983 debido a que la organización oficial ANSI estaba trabajando sobre diferentes propuestas de lenguajes de datos para modelos en red culminando estos con su recomendación NDL/ANS de un lenguaje normalizado de datos en red.

la historia de CODASYL se puede resumir así:

- 1959 aparece CODASYL− COnference on DAta SYstems Languajes
- 1965 Creacion del Lsit Processing Task Group
- 1967 Cambio de nombre por el de DataBase Task Group (DBTG)

- 1968 Informe preliminar sobe LDD y LMD
- 1969 Versión mas completa
- 1971 Versión revisada. creación del comité DDCL
- 1973 Versión aprobada oficialmente
- 1975 Informe Grupo de Trabajo de Administración de Bases de Datos (DBAWG)
- 1976 Informe sobre selección y adicción de SGBD
- 1978 Modificaciones a las propuestas de 1973
- 1980 LDD del subesquema y LMD para FORTRAN
- 1981 Nuevas propuestas del DIC
- 1983 Disolución del grupo

Una de las razones, entre otras, por las que le grupo CODASYL fue elogiado, es la excelente descripción que hizo de los lenguajes que propuso, tanto de los de definición como del de manipulación. Este hecho ha tenido, sin duda, una considerable influencia en la difusión de sistemas de gestión de bases de datos apoyados en estas recomendaciones.

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 la 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, puedan tener interfaces con múltiples lenguajes.

Medios para conseguir los objetivos

Para que un sistema de gestión de base de datos pueda conseguir los objetivos anteriormente citados, CODASYL propone los siguientes lenguajes:

- *Lenguaje de definición del esquema (Schema DDL)*: permite describir la estructura de la totalidad de los elementos que forman parte de la base de datos a nivel lógico (entidades, atributos, interrelaciones, etc...), permitiendo al usuario la elección de su estructura, nombres, etc... su primera especificación fue presentada en los informes de 1968, 1969 y 1971. Contenía muchas cláusulas con claras implicaciones físicas. Las propuestas de 1978 se realizaron con el fin de eliminar dichas cláusulas conservando solamente la parte correspondiente a consideraciones lógicas y pasando los aspectos físicos al nuevo nivel interno que surge con la arquitectura a tres niveles. Si bien, estos cambios supusieron un importante avance en relación con la independencia, le problema solo se resuelve en

parte y aunque en 1981 se introducen nuevos cambios, la independencia datos / aplicaciones en CODASYL es relativa.

- *Lenguaje de definición de subesquemas (SubSchema DDL)*: permite la descripción de subconjuntos del esquema para diferentes usuarios que no necesitan para su trabajo más que una parte determinada de la base de datos. Este lenguaje hace posible introducir ciertos cambios en el formato y en la estructura de los elementos del esquema que intervienen en el subesquema.
- *Lenguaje de manipulación de datos (DML)*: se ocupa de los aspectos de manipulación de los datos que contiene la base de datos (recuperación, inserción, modificación, borrado, etc...) su especificación estaba presente en los informes de los años 1971 y 1978, aunque ha ido sufriendo cambios a lo largo de su vida a fin de adaptarlo a las modificaciones que ha ido experimentando el esquema, principalmente con la desaparición de las cláusulas de tipo físico del mismo.
- *Lenguaje de control de dispositivos/ soporte (DMCL)*: En las recomendaciones del año 1971, en las que no existía un lenguaje especifico para el almacenamiento interno de los datos, CODASYL se limito a reflejar la necesidad de un lenguaje que controlase los aspectos relativos a los dispositivos y soportes donde estaban almacenados los datos, pero no definió su sintaxis ni su semántica, algunos de los productos de bases de datos incluyeron este lenguaje, pero al haber sido descrito solo en sus generalidades, existen considerables diferencias entre ellos. No se le hizo mucho caso a las especificaciones del DMLC de 1978.
- *Lenguaje de definición del esquema de almacenamiento (DSDL)*: considera todos los aspectos físicos del almacenamiento de los datos recogidos en la base de datos. fue descrito en el informe de 1978, habiéndose desarrollado por el grupo DBAWG.

Principales criticas al modelo CODASYL

Las criticas al modelo se han centrado principalmente en la falta de independencia, confidencialidad, integridad, y ausencia de un lenguaje de interrogación auto contenido.

Una de las más importantes criticas se refiere a la independencia. este objetivo de cualquier SGBD, queda muy lejos de ser alcanzado en las especificaciones de CODASYL de 1971 y 1973, la razón principal de esta dependencia entre los niveles físico y lógico deriva de la arquitectura a dos niveles, la cual se cambio en 1978.

Otro aspecto cuestionado era la necesidad de que el programador de aplicaciones tuviese que abrir las áreas en las que se encontraban las ocurrencias a las que tenia que acceder su programa, así como a las áreas referidas indirectamente; en el modelo de 1978 se introducen modificaciones para mejorar la situación.

También el concepto de modo de emplazamiento o ubicación de los registros (LOCATION MODE) en el lenguaje de definición de datos y los mecanismos de selección de un registro que utilizan la clave de base de datos, guardando la dirección física y recuperando después el registro indicando dicha dirección, han sido muy controvertidos en razón de la pérdida de independencia que produce y de los problemas que causa cuando se reorganiza la base de datos.

Todo el tema de la clave de base de datos ha sido muy discutido, ya que aún cuando es generalmente admitida la necesidad de la existencia de un identificador único para los registros, se cuestiona mucho que este deba ser una dirección física. Los problemas derivados de este hecho son muchos, desde una gran limitación a la independencia físico−lógica hasta un grave peligro para la integridad de los datos de la base, al poder acceder el programador a las direcciones físicas de los datos. En esto también se introdujeron modificaciones en 1978.

Otra de las criticas que se han hecho al modelo es la ausencia de mención expresa a un diccionario de datos, así como la falta de especificaciones concretas para el mismo. Esta deficiencia fue subsanada en mayor o en menor medida por los distintos productos, pero al no existir normas concretas en cuanto a sus características o a su definición, las diferencias entre los diccionarios son notables.

Se han hecho también objeciones en relación con el acceso concurrente ya que se pueden producir errores en la actualización debido a que los mecanismos especificados en los lenguajes de definición y de manipulación, además de solaparse son diferentes.

Por ultimo, citar la fuerte dependencia de COBOL y la ausencia de una especificación para un lenguaje de manipulación de auto contenido

Arquitectura

En primer lugar destacar el cambio sufrido a través de las distintas especificaciones, ya que se empezó por una arquitectura de dos niveles (esquema y subesquema) y que terminó en una de tres niveles al separarse los aspectos físicos en un nuevo esquema (esquema de almacenamiento) de las características de tipo lógico que continúan formando parte del esquema.

El esquema de las propuestas de 1971 se transforma en 1978 en una descripción esencialmente lógica de la base de datos al eliminarse de el los aspectos físicos, como son los relacionados con el emplazamiento y la agrupación de las ocurrencias de registro, los punteros, etc... que pasan al esquema de almacenamiento. A fin de describir estas características de tipo físico aparece el lenguaje de definición del esquema de almacenamiento (DSDL), que permite describir como se organizan y almacenan los datos en los soportes, independientemente de los dispositivos y del sistema operativo.

Modelo de datos CODASYL

Elementos del modelo y definiciones

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

- *Elemento de datos (Data item)*: 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 numero 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 mas 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 (area 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 mas 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 una 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.

Conjunto

En el modelo CODASYL, el conjunto, también llamado SET o COSET, constituye el elemento básico para la representación de las interrelaciones. Por medio de los SET, se establecen asociaciones jerárquicas (1:N) a dos niveles, en las cuales el nodo raíz se llama propietario (owner) y los nodos descendientes de uno o mas tipos se denominan miembros (members).

Una representación de un SET y una ocurrencia del mismo podría ser por ejemplo:

La flecha del esquema parte del propietario y apunta al miembro, lo que indica que el tipo de interrelación es 1:N o 1:1 en dicho sentido. El SET tiene un nombre, lo que permite establecer más de una interrelación entre un par de registros propietario−miembro. por tanto, el SET representa un tipo de interrelación nominada entre tipos de registros. En la ocurrencia de SET es una ocurrencia del registro propietario (P) y m1,m2... son ocurrencias del registro miembro (M) asociadas mediante una interrelación que a nivel físico se puede establecer, en principio, mediante cualquier mecanismo, pero en la práctica se realiza mediante punteros, bien embebidos o empotrados.

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.

Definición formal del modelo CODASYL

Se puede definir el modelo CODASYL como un conjunto finito de registros {R1,R2,..., Rn} compuestos cada uno de ellos por un conjunto finito de elementos de datos. entre los tipos de registros se establecen interrelaciones, llamadas conjuntos: {Ckij..h}, el conjunto {Cki,j..h} representa la interrelación entre los registros Ri,Rj.. Rn, en la que el registro Ri es el propietario y los demás son los miembros, el superíndice k indica que entre la misma colección de tipos de registro Ri puede haber mas de un SET, al ser estos nominados. Cada tipo de registro tiene un conjunto finito de ocurrencias, entre las cuales existen las

vinculaciones definidas por los SET del esquema; Una ocurrencia de registro propietario encadenada con las correspondientes ocurrencias de registros miembro constituye una ocurrencia de SET.

las restricciones inherentes del modelo CODASYL son las siguientes:

A) Solo están admitidos tipos de interrelaciones jerárquicas a dos niveles: nivel propietario y nivel miembro. Combinando varios SET se puede llegar a jerarquías multinivel y a estructuras en red (PLEX).

B) En el nivel propietario solo se permite un tipo de registro.

C) Un mismo tipo de registro no puede ser a la vez propietario y miembro de un mismo tipo SET. no se admite la reflexividad de los SET. Esta restricción ha desaparecido en las especificaciones de 1978, pero por lo general los productos siguen manteniéndola.

D) una misma ocurrencia de miembro no puede pertenecer en un mismo tipo de SET a mas de un propietario, con ello se simplifica la instrumentación física de los SET, al poder organizar sus ocurrencias como una cadena.

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.

Punteros en CODASYL

Las ocurrencias de registros se encadenan, de acuerdo con lo especificado en el esquema, por medio de punteros, pueden existir tres tipos de punteros:

- *Puntero al siguiente*: se establece automáticamente, solo por el hecho de definir un SET, se trata de una cadena cerrada, de forma que el ultimo miembro apunta al propietario.
- *Puntero al anterior*: se puede definir opcionalmente, de forma que la cadena se pueda recorrer en ambos sentidos.
- *Puntero al propietario*: permite acceder directamente al propietario si necesidad de recorrer todas las ocurrencias de registro hasta llegar a el.

Los punteros siguiente son intrínsecos a la definición del SET, mientras que los otros dos son opcionales y solo se definen a efectos de eficiencia, ya que tanto el propietario como el anterior pueden obtenerse aunque no existan estos punteros. La definición de punteros en el esquema atenta contra la independencia físico−lógica, ya que estos elementos solo tiene sentido físico.

Definición de datos a nivel lógico

Estructura general del lenguaje de definición

El lenguaje de definición de datos del esquema (LDDE), es el encargado de realizar la descripción de todos los elementos que forman parte de la base de datos (áreas, registros, SET, etc...) CODASYL desarrolló la definición de un lenguaje de autocontenido, de tal forma que un esquema fuente escrito en dicho lenguaje se convierte en un esquema objeto tras la compilación por parte de un SGBD, y puede entonces almacenarse en la biblioteca del sistema, en la que estará disponible para su uso. El lenguaje fue presentado en 1973, pero algunas de sus deficiencias y problemas fueron corregidas en posteriores especificaciones.

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 compuesta 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.

Entrada de esquema

La entrada del esquema es única para la totalidad del esquema. su misión es la de especificar el nombre con el que se va a conocer al esquema y todas las características relativas a la seguridad y confidencialidad global de la base de datos.

La primera cláusula es la utilizada para especificar el nombre por el que será conocido el esquema

La cláusula PRIVACY LOCK en la entrada del esquema sirve para controlar el acceso a la base de datos a nivel global, permitiendo dar una contraseña a fin de limitar el acceso a la base de datos, de forma que solo se podrá acceder al esquema si se proporciona la contraseña correcta que se ha especificado en la definición. El acceso al esquema puede limitarse según el tipo de acción que se desee realizar, dándose la posibilidad de especificar contraseñas distintas para cada una de la siguientes acciones:

- Alter: modificación del esquema.
- Copy: extracción de información de todo o parte del esquema para copiarla en un subesquema.
- Display: visualización del esquema en pantalla o impresión del mismo.
- LocKs: visualización, creación o modificación de las contraseñas.

Cuando se omite todas las operaciones en la escritura de la clausular, significa que se desea que la contraseña tenga que especificarse para la realización de cualquiera de las acciones anteriormente mencionadas. La contraseña puede ser:

- Literal conjunto de caracteres.
- Una variable: la contraseña, en cada momento, es el valor contenido en dicha variable en ese momento.
- Un procedimiento: la contraseña es el valor obtenido tras la ejecución del procedimiento, lo que puede llevar a que la contraseña varié a intervalos de tiempo de acuerdo con el procedimiento.

La cláusula PRIVACY LOCK puede utilizarse en todo tipo de entrada y subentrada además de la entrada del esquema; en cada una de ellas la contraseña controla ciertas acciones sobre zonas parciales de la base de datos. Las operaciones sobre las que se puede establecer control de acceso son diferentes dependiendo de la entrada o subentrada de que se trate, pero la estructura y sintaxis de la cláusula es siempre la misma. En las recomendaciones de 1978 se substituyo el nombre de la cláusula por el de ACCESS_CONTROL, pero no se modifico la funcionalidad de la cláusula.

La cláusula ON obliga a la ejecución de un procedimiento escrito por el diseñador o el administrador de la base de datos si se desea llevar a cabo alguna de las acciones que se especifican; es decir, se impedirá que el usuario pueda realizar estas acciones por medio de procedimientos propios. Las acciones son las mismas que se han enunciado para la cláusula PRIVACY. También la cláusula ON puede utilizarse en otras entradas o subentradas del lenguaje de definición de datos del esquema, con análoga estructura y sintaxis. La cláusula ON se considera una cláusula de interfaz de administración, ya que su misión es la de proporcionar una interfaz para la ejecución de procedimientos escritos por el usuario (el administrador en este caso) ante determinadas operaciones realizadas por el sistema de gestión de la base de datos.

En la normativa de 1978 se permite que se indique en la cláusula el momento en el que se desea ejecutar el procedimiento, puede indicarse si se quiere que se ejecute antes, después, cuando se produce un error... además se cambio la sintaxis por:

CALL procedimiento ON <operación propia de la entrada o subentrada>.

Área de entrada

El concepto de área no debería situarse en el LDDE ya que tiene un significado puramente físico. En los informes CODASYL de 1971 y 1973, se define el área como subdivisión nominada del espacio de almacenamiento direccionable en la base de datos que puede contener ocurrencias de registros SET, o partes de SET de varios tipos. Posteriormente se modifico considerablemente esta definición, a fin de eliminar en lo posible todo aquello que tuviese alusiones demasiado físicas, eludiendo, por ejemplo, la expresión espacio de almacenamiento direccionable.

Las áreas pueden se páginas de disco, cilindros, etc... mientras que sean parte del soporte físico que posee el sistema. Por lo tanto, la especificación de los elementos de la base de datos, que deben asignarse a cada una de las áreas se realiza en base a criterios de rendimiento tales como que los registros que se utilizan con frecuencia juntos deben colocarse en la misma área para no retardar al sistema en las búsquedas sobre el soporte; que deben situarse solos en un área los tipos de registros que previsiblemente van a tener un número elevado de ocurrencias, incluso si este número de ocurrencias es muy elevado, podrían dividirse las

ocurrencias de un mismo tipo de registro en dos o más áreas; podría hacerse lo mismo con aquellas ocurrencias que vayan a ser accedidas conjuntamente por determinados procedimientos, etc...

La asignación de los elementos a las áreas se rige por las siguientes reglas:

- Una ocurrencia de registro puede ser asignada a cualquier área.
- Una determinada ocurrencia de registro solo puede ser asignada a un área.
- Una vez que una ocurrencia ha sido asignada a un área no puede transferirse a otra, a no ser que se reorganiza la base de datos.
- Las ocurrencias de un mismo tipo de registro pueden ser asignadas a diferentes áreas
- Las ocurrencias de distintos tipos de registros pueden ser asignadas a una misma área.

Dada la clara implicación física del concepto de área, es necesario controlar de alguna forma el acceso a cada zona de almacenamiento por parte del conjunto de usuarios, para que un usuario pueda acceder a los datos contenidos en un área, debe abrir esta previamente y ocuparse de cerradla cuando no necesite su uso, lo que, como ya hemos señalado, ha sido muy criticado. De estas operaciones se ocupa el lenguaje de manipulación de datos. la sintaxis de una entrada de área es como sigue:

Como puede observarse, existe una cláusula opcional para indicar si este área es temporal. Si no se escribe esta cláusula, el área se considera permanente, las áreas temporales solo tienen existencia como almacenamiento real durante el tiempo que dura la ejecución del programa que las utiliza y desaparecen cuando termina este, por lo que siempre han de estar asociadas a una unidad de ejecución. CODASYL define una unidad de ejecución como la realización del conjunto de funciones necesarias para la ejecución de un programa concreto. Al igual que las áreas permanentes, también las áreas temporales han de abrirse antes de ser utilizadas. existe, sin embargo, una diferencia entre la apertura de un área permanente y una temporal , ya que para la primera, la apertura consiste en el permiso de acceso a los datos que contiene; mientras que para la segunda la apertura crea una copia de los elementos definidos en el área temporal en la zona del disco asignada al usuario que la abre.

También existen para esta entrada las cláusulas opcionales ON y PRIVACY LOCK. En este caso las operaciones sobre las que se pueden utilizar son las dos permitidas para un área: RETRIEVAL (recuperación) para la lectura de los datos que contiene y UPDATE (actualización) para la escritura de nuevos datos o modificación de datos ya existentes. para cada una de estas operaciones se puede especificar el modo de acceso al área:

- PROTECTED: cualquier otro usuario puede acceder a la misma área, pero no puede realizar modificaciones o borrados de registros, solo puede recuperar datos que se encuentren en ella.
- EXCLUSIVE: con esta opción ningún otro usuario puede acceder al área; no se permite que realice ninguna operación con los datos que contiene.

Entrada de registro

La entrada de registro se utiliza para dar nombre a cada uno de los tipos de registros existentes en la base de datos y especificar sus elementos de datos así como algunas otras características importantes del modelo.

Hay una entrada de registro para cada uno de los tipos de registro que componen la véase de datos (uno como mínimo) y cada entrada tiene tantas subentradas de datos como elementos contenga el registro, además de las correspondientes cláusulas asociadada a la entrada.

La primera cláusula de la entrada tiene como misión dar nombre al registro. A continuación hay dos clausular obligatorias y otras dos opcionales. Las cláusulas opcionales son las ya mencionadas en las entradas de esquema y arrea. ON y PRIVACY LOCK. Las operaciones que pueden controlar a nivel de registro son

CONNECT (conectar), DISCONNECT (desconectar), RECONNECT (reconectar), STORE (almacenar), ERASE (borrar); FIND (encontrar, localizar) y GET (recuperar).

De las cláusulas obligatorias, la primera se utiliza para especificar el o las áreas en las que debe almacenarse el registro. Cuando un tipo de registro puede estar en varias áreas es necesario indicar cuáles son las condiciones que deciden en que área se almacena cada una de las ocurrencias. Esta cláusula desapareció definitivamente del LDDE en 1981, quedando exclusivamente en el lenguaje de almacenamiento de datos. es posible dejar al sistema la responsabilidad de decidir que área se va a asignar el registro mediante la opción WITHIN ANY AREA.

La segunda de las cláusulas obligatorias se refiere a la forma en la que se almacenaran las ocurrencias de los registros; dicho de otro modo, proporciona un mecanismo para controlar la asignación de las claves de base de datos. Esta cláusula tiene unas innegables connotaciones físicas, ya que permite al usuario elegir la forma en la que se hace la correspondencia (mapping) entre las ocurrencias de registro y su situación física sobre el soporte. Debido a ello y a su relación con la clave de base de datos, en las especificaciones de 1978 se eliminó del LDDE, pasando con el nombre de PLACEMENT al lenguaje de definición del almacenamiento de datos, para conseguir un modo de acceso directo a un registro aparece una nueva cláusula KEY, cuyo papel es parecido al modo de ubicación DIRECT o CALC, pero sin sus implicaciones físicas, ya que trata únicamente de una clave lógica que no determina la ubicación del registro. su sintaxis es:

KEY nombre [ASCENDING || DESCENDING] elemento de datos

DUPLICATES ARE [NOT || FIRST || LAST || SYSTEM] ALLOWED

El modo de ubicación se examina cada vez que se almacena una ocurrencia de un registro cualquiera en la base de datos. Dependiendo de los modos de ubicación la recuperación de los registros contenidos en la base de datos se podrá hacer de una forma u otra. los modos de ubicación disponibles en las especificaciones de 1973 son:

A) Modo "DIRECT"

Este modo de ubicación permite al usuario intervenir en la asignación del lugar de almacenamiento de las ocurrencias por medio de la clave de base de datos. la sintaxis de la cláusula es LOCATION MODE IS DIRECT elemento de datos, el valor de elemento de datos determina la ubicación del registro.

Para hacer uso de este modo de ubicación es necesario que el elemento de datos que aparece en la cláusula LOCATION MODE se defina en la subentrada de datos del registro y su tipo ha de ser DATABASE_KEY.

El elemento de datos que se utiliza como clave de base de datos debe ser numérico, no puede tomar valores duplicados, y además, debe mantenerse dentro del espacio físico direccionable, es decir, debe ser denso en el sentido de que ha de cubrir dicho espacio sin dejar demasiados huecos. Hemos de observar que cuando un registro se define como DIRECT es el usuario el responsable de asignar el emplazamiento físico a las ocurrencias, mientras que con los modos CALC y VIA el usuario solo indica el procedimiento y con SYSTEM es el sistema el único responsable. En cualquier caso, después de asignadas las claves, CODASYL propone un mecanismo para que el programador pueda leerlas y utilizarlas en los programas.

En la especificación de 1978 no se llegó a eliminar el concepto de clave de base de datos, pero se prohibió su almacenamiento en variables o ficheros normales propios de los usuarios. El mecanismo que se propuso para el manejo de las clave de base de datos es una keep−list o lista de almacenamiento, este tipo de datos se utiliza para guardar los valores de las claves de la base de datos que son necesarias para le ejecución de u programa, pero su valor solo puede ser leído , no puede ser guardado en un fichero, ni asignarse a una variable, ni modificarse, además solo tiene validez durante la ejecución del programa, mientras exista la correspondiente

unidad de ejecución y cuando esta termina los valores almacenados se borran automáticamente.

B)Modo "CALC"

El modo de ubicación CALC (abreviatura de calculated) determina la clave de base de datos, es decir, el lugar en el que debe almacenarse la ocurrencia de registro que se inserta en la BD, mediante la aplicación de un procedimiento de calculo sobre uno o varios elementos de datos del registro que devuelve el valor de la clave (es el modo de almacenamiento que se conoce como Hash).

El procedimiento que se utiliza para calcular la ubicación puede ser definido por el programador o puede ser proporcionado por el SGBD, en caso de ser definido por el programador, el procedimiento debe ser escrito previamente, compilado e introducido en la librería del sistema.

Los SGBD proporcionan procedimientos propios para el modo de ubicación CALC, que en general son algoritmos comunes de hashing. Las especificaciones de CODASYL no son excesivamente claras al respecto, pero en la mayor parte de los SGBD comerciales se han instrumentado algoritmos que utilizan el valor del elemento de datos indicado en la sentencia del CALC en forma aritmética, e ya que si no lo es el algoritmo lo convierte previamente. La función mas común es la que divide el calor del elemento de datos entre el numero primo mas grande que, a su vez, sea menor que el número de direcciones físicas disponibles. La sintaxis de la cláusula es la siguiente:

LOCATION MODES IS CALC procedimiento USING elemento de datos

Cuando en una base de datos alguno de los registros se define con modo de ubicación CALC hay que tener en cuenta la posible aparición de claves duplicadas. es decir, debe decidirse si se va a permitir la existencia de valores iguales de la clave que se utiliza en el calculo o si tal circunstancia no va a considerarse valida. La decisión respecto a los duplicados corresponde al administrador de la base de datos, debe guiarse por las características especificas de la base de datos y el elemento de datos elegido como clave. si este corresponde a un valor que se supone único para cada ocurrencia del registro no será necesario permitir los duplicados, sin embargo, si es fácil que se presenten duplicados, por lo que será conveniente prever su existencia y permitirlos, con objeto de evitar problemas en el momento de introducir los datos en la base.

La existencia o inexistencia de duplicados puede controlarse mediante la cláusula DUPLICATES cuya sintaxis es:

DUPLICATES ARE [NOT] ALLOWED

Es necesario distinguir entre duplicado y sinónimo, un sinónimo produce una ubicación igual a otra ya obtenida para un valor igual o diferente de la clave. el procedimiento de calculo de la clave se ha de encargar de la gestión de los sinónimos, decidiendo el lugar en el que se debe almacenar una ocurrencia de registro cuando el algoritmo proporciona una dirección que ya había sido previamente asignada a esta ocurrencia.

C)Modo "VIA"

Este modo de ubicación tiene mucha relación con el concepto de SET y solo puede ser utilizado con registros que sean miembros de algún SET.

Si se elige para un registro el modo de ubicación VIA respecto a uno de los SET del cual dicho registro es miembro, se asigna a cada ocurrencia un emplazamiento físico lo más próximo posible al lugar en el que se encuentra una ocurrencia del correspondiente registro propietarios. la sintaxis de la cláusula es la siguiente:

LOCATION MODE IS VIA conjunto SET

El registro que tenga este modo de ubicación es forzosamente miembro del SET cuyo nombre se indica en la sentencia. Dicho registro puede ser también miembro o propietario de otros SET, pero la ubicación vendrá determinada solo en relación de las ocurrencias del registro propietario del SET que figure en la cláusula. En general, el registro propietario y el registro miembro que se haya declarado VIA se encuentran en la misma área.

No existe en principio restricción para la utilización del modo de ubicación VIA, lo cual puede conducir a estructuras realmente complejas cuando se aplica a registros que son a su vez propietarios de SET en los que los miembros también se declaran VIA; únicamente existe la necesidad de conocer en el momento de la inserción de una ocurrencia cual es la ocurrencia del registro que es le propietario en el correspondiente SET, lo que obliga a que el modo de inserción del miembro en el SET sea automático. Otra restricción aparece en el caso de ciclos o cuando los registros se reverencian mutuamente.

El usuario no tiene ningún control sobre la ubicación de los registros que declara VIA, solo sabe que el SGBD tratará de colocar las ocurrencias tan cerca como sea posible de las del registro propietario. Las recomendaciones CODASYL solo indican esto, lo que hace que cada implementador haya optado por la forma que ha creído más conveniente.

D)Modo "SYSTEM"

Este modo de ubicación deja la responsabilidad de la asignación de la clave de base de daos totalmente a manso del SGBD. Puede utilizarse, por ejemplo, en los registros que son miembros de SET singulares, de forma que su gestión sea realizada por completo por el sistema, en tanto que no sea necesario acceder a ellos directamente por el valor de un campo (para eso tenemos DIRECT o CALC).

Subentrada de datos

La subentrada de datos se utiliza para describir las características de los elementos de datos que han de contener el registro. existe una subentrada de diferente para cada elemento de datos, distinguiéndose por el numero que las precede. estos números, denominados números de nivel, pueden omitirse, en el caso de estructuras planas, y suelen comenzar, a partir del 2 ó del 3 e incrementarse de 2 en 2 para facilitar inclusiones posteriores. tras cada número de nivel debe indicarse el nombre del elemento de datos.

Es necesario especificar el formato que va a tener el elemento de datos. En los primeros informes CODASYL, la cláusula que realizaba estas tareas se denominaba PICTURE, pero en las recomendaciones posteriores a 1978 se cambio por TYPE.

A lo largo del esquema presentado pueden encontrase los diferentes tipos de especificación de elementos de datos.

El tipo DATABASE_KEY es muy dependiente de cada instrumentación en particular, y en los informes de 1978 este tipo desapareció, pues en el momento que se elimina el modo de ubicación DIRECT, dejaba de tener sentido.

El LDD CODASYL permite la definición de grupos repetitivos mediante la cláusula OCCURS, de sintaxis:

OCCURS {entero, elemento} TIMES.

Esta cláusula hace posible la definición de un elemento de datos como un grupo de un numero indicado de elementos. El número de elementos que componen el grupo repetitivo puede especificarse con un numero entero directamente o por medio de otro elemento de datos.

Para realizar validaciones directas sobre los valores de los elementos de datos se utiliza la cláusula CHECK. Esta cláusula permite la especificación de un procedimiento que se encargue de comprobar el valor del elemento de datos y realizar las operaciones pertinentes. Este procedimiento ha de ser escrito por el administrador y debe estar compilado y almacenado en la librería del sistema cuando se defina el esquema. La síntesis es:

CHECK IS procedimiento

En las especificaciones de 1978 se permite especificar en la cláusula las condiciones que ha de cumplir el elemento de datos, en lugar de escribir un procedimiento. Las condiciones se puede expresar mediante operadores como AND, OR, >, <, =, etc... de esta forma se simplificaba la validación para casos sencillos en los que la escritura de un procedimiento resultaba excesiva.

CHECK IS ((CONT<540) AND (CONT >200))

La cláusula ENCODING / DECODING se utiliza para realizar la codificación o descodificación de un elemento de datos mediante un procedimiento previamente definido. Su sintaxis es:

FOR {ENCODING, DECODING} [ALWAYS] CALL procedimiento

Su papel consiste en llevar a cabo la conversión entre la imagen de un elemento de datos en el esquema y la que tiene en el subesquema, que pueden ser diferentes. es decir, se utiliza para realizar conversiones internas. El procedimiento de codificación (ENCODING) se utiliza cuando se almacena o modifica el registro, y el de decodificación (DECONDING) cuando este se recupera. Puede aplicarse, por ejemplo, para compactar un elemento de datos a la hora de almacenarlo, realizando la operación inversa a la hora de recuperarlo, cuando el usuario necesite que se inteligible; pero la principal finalidad que persigue con esta cláusula es criptografiar los datos para proteger la confidenciabilidad de los mismo, de modo que si son accedidos "puenteando" el SGBD no pueda entenderse su contenido. En la práctica muy pocos SGBD comerciales implantaron esta cláusula debido a su complejidad.

En el conjunto de la base de datos es posible que existan elementos de datos que tengan valores idénticos a otros, o cuyo valor deba obtenerse a partir de otros, esto se hace mediante las cláusulas SOURCE y RESULT. Además, obliga a indicar si se desea que el elemento de datos tenga existencia real en la base de datos o que se obtenga solo cuando se vaya a recuperar. Lo que se consigue median te las opciones ACTUAL o VIRTUAL. La sintaxis es:

La cláusula SOURCE hace que el elemento de datos del registro miembro del SET tome el valor del elemento de datos especificado en la cláusula, elemento que pertenece al registro propietario del SET. Se utiliza la cláusula SOURCE en el elemento de datos del registro miembro. Si se especifica ACTUAL en la cláusula, el valor del elemento de datos estará almacenado en la base de datos, mientras que si se especifica VIRTUAL, el valor no estará realmente almacenado, sino que se copiará del registro propietario cuando se acceda al registro miembro. Es decir, cuando se solicite una recuperación, la opción ACTUAL / VIRTUAL tiene claramente un objetivo de eficiencia, ya que si predominan las actualizaciones sobre las recuperaciones será más conveniente utilizar VIRTUAL, siendo preferible REAL en caso contrario.

La función de la cláusula RESULT es parecida a la de la cláusula SOURCE, pero si se define en el elemento de datos de un registro propietario de un SET, su valor se calcula a partir de los valores de uno o varios elementos de datos del registro miembro. El cálculo se realiza mediante un procedimiento que ha de definirse previamente incluyéndolo en la librería del sistema. Los cálculos puede realizarse también sobre elementos de datos del mismo registro.

Las opciones ACTUAL y VIRTUAL tienen el mismo significado que en el caso anterior, determinando el

momento en el que se calcula el valor y si este se almacena realmente en el registro o no.

Estas clausular sirven para controlar la redundancia, con los elementos de datos especificados como ACTUAL no existe redundancia lógica, aunque si física, ya que los valores iguales están almacenados realmente en la base de datos; se trata, por tanto, de una redundancia controlada por el propio sistema que no puede dar lugar a incoherencias. En el caso de los datos VIRTUAL no existe redundancia lógica ni tampoco física, aunque el usuario percibe la base de datos como si existiese la redundancia.

Otro aspecto de las opciones ACTUAL y VIRTUAL es su incidencia en el rendimiento de la base de datos, para decidir cual de las dos opciones es más conveniente hay que considerar la relación que existe entre el número de modificaciones y el número de lecturas sobre ese elemento de datos. si se van a realizar muchas más modificaciones que lecturas, es conveniente declarar el elemento de datos VIRTUAL, ya que como el elemento de datos no está realmente en la base de datos, solo hay que calcular su valor en el momento que vaya a ser accedido, por el contrario, si las recuperaciones superan a las modificaciones, conviene declararlo como ACTUAL, para no tener que calcularlo o copiarlo en cada recuperación.

El objetivo de las clausular SOURCE y RESULT es principalmente el mantenimiento de la consistencia de la información de la base de datos, El programador no ha de preocuparse de que los elementos de datos tengan los valores que deben, el SGBD lo hace por él, si se han utilizado las correspondientes cláusulas SOURCE o RESULT en la definición del esquema, de todas formas, la dificultad de estas clausular hizo que pocos SGBD las incluyesen. En la especificación de 1978 las cláusulas SOURCE y RESULT sufrieron modificaciones, y ACTUAL y VIRTUAL desaparecieron del LDE por sus características de modificación del rendimiento de la base de datos, pasando al lenguaje de definición del esquema de almacenamiento, para realizar una labor similar, se introduce la cláusula EVALUATION, cuya sintaxis es:

EVALUATION IS ON {ACCESS STORAGE, UPDATE} IS [NOT] REQUIRED

Con esta cláusula se indica al sistema que el calculo del elemento de datos se haga en el momento de la actualización o bien, en el momento de la lectura.

Entrada de SET

La entrada de SET especifica los registros que forman parte del SET, el nombre que recibe este y otras características. Existe una entrada de SET para cada uno de los SET que contenga el esquema, en cada entrada se puede distinguir unas clausular en las que se definen características generales del SET, y tras ellas tantas entradas de miembro como sean necesarias.

La primera cláusula se utiliza para dar nombre al SET, es decir, para fijar el identificador por el que se va a conocer al SET dentro de la base de datos.

La segunda cláusula, OWNER, especifica el propietarios del SET, este puede ser un registro cualquiera de la base de datos o la palabra SYSTEM, con la que se indica que el propietarios del SET es el sistema, es decir, permite definir los SET singulares. Esta cláusula no ha sufrido variación alguna en las diferentes especificaciones.

La tercera cláusula es la denominada SET MODE, su misión es definir el tipo de enlace que se establecerá entre las ocurrencias del propietarios y el miembro del SET. En el primer informe de 1971 se permitía declarar de forma directa la estructura de los punteros, en 1973 esta cláusula fue modificada, permitiéndose elegir entre dos opciones:

- DYNAMIC: Cualquier tipo de registro de la base de datos puede formar parte del SET, declarándose su estructura en el lenguaje de manipulación.

- PRIOR PROCESSIBLE: En la cadena de ocurrencias, además del puntero al siguiente (obligatorio), existe otro al anterior. esta opción permite mejorar el rendimiento de la base de datos, acelerando las recuperaciones dentro del SET. En las especificaciones de 1978 esta cláusula fue eliminada del LDE debido a su clara relación con la estructura física de la base de datos.

La siguiente cláusula, ORDER, se refiere a la colocación que tendrán las ocurrencias de los registros miembros en la cadena; es decir, cuando una ocurrencia de miembro deba ser insertada en la base de datos, no solo es necesario determinar la ocurrencia de propietario a la que debe conectarse, sino también fijar el lugar que debe ocupar en la cadena de miembro. Esta cláusula permite especificar si el orden de los miembros dentro de la ocurrencia de SET será permanente (PERMANET), con lo que el usuario o el programador no podrá cambiar el orden en el que se haya elegido; o si se permite que el orden pueda ser modificado (TEMPORARY) con objeto de adaptarse a aplicaciones particulares.

una vez decidido esto, es necesario indicar la posición en la que se ha de colocar cada nueva ocurrencia de un miembro, para ello se ofrecen varias opciones:

- *FIRST*: Cada nueva ocurrencia de miembro se insertará inmediatamente después de la ocurrencia del propietarios, quedando como primera ocurrencia de mimbro de la cadena (LIFO).
- *LAST*: Cada nueva ocurrencia de miembro se colocará inmediatamente después de la última cadena, pasando a ser la ultima del SET (FIFO).
- *PRIOR*: Cada nueva ocurrencia de miembro se colocara inmediatamente antes de la ocurrencia que esté activa en ese momento.
- *NEXT*: Cada nueva ocurrencia de miembro se colocará inmediatamente después de la ocurrencia activa en ese momento
- *INMATERIAL*: Cada nueva ocurrencia de miembro se colocará en el lugar que el sistema internamente elija, es decir, no se especificará una ubicación determinada; cada SGBD coloca la nueva ocurrencia en la forma que considera más eficiente.
- *SORTED*: Esta opción sólo puede utilizarse con la opción PERMANET. Permite especificar una ordenación diferente de las anteriores mediante el uso de diferentes elementos de clasificación entre los que se encuentran los siguientes:
    - *BY DEFINED KEYS*: El elemento por el que se realiza la clasificación es uno o varios de los elementos de datos del registro miembro. se tomará el valor de dicho elemento o elementos y se situará la ocurrencia en el lugar correspondiente según una secuencia ascendente o descendente. Si el SET es multimiembro no se tiene en cuenta el tipo de registro miembro al que pertenece la ocurrencia. El elemento de datos que va a servir para la clasificación debe ser especificado como tal cuando se defina cada registro miembro del SET.
    - *WITHIN RECORD−NAME*: se utiliza para SET multimiembros y permite especificar el orden que han de tener las ocurrencias. Para cada tipo de registro miembro, dentro de cada tipo de registro miembro, las ocurrencias se ordenarán de acuerdo con la clave definida. Los elementos que se utilizan como clave se identifican de la misma forma que en el caso anterior, mediante la cláusula KEY. Aparecerán primero las ocurrencias del primer tipo de registro miembro que se ha definido en la subentrada del correspondiente conjunto, a continuación las del segundo y así sucesivamente.
    - *BY DATA−BASE−KEY*: el elemento que se utiliza para realizar la calcificación es la base de datos.

En las diferentes especificaciones, la opción SORTED ha sufrido bastantes variaciones, eliminándose de ella las opciones que tenían más implicaciones físicas como la ordenación por la clave de base de datos. Algunos ejemplos de ORDER serian:

ORDER IS PERMANTE SORTED BY DEFINED KEYS

ORDER IS TEMPORARY LAST

ORDER IS PERMANENT INMATERIAL

A continuación de las cláusulas que acaban de definirse se sitúan las subentradas de miembro, existiendo tantos como tipos de registro miembro contenga el SET. Un ejemplo de sintaxis sería :

La primera cláusula de la subentrada especifica el nombre del registro miembro. a continuación es obligatorio definir como se va a comportar este miembro respecto a la conexión con el propietario mediante la especificación del modo de inserción y el modo de retención.

El modo de inserción indica si la conexión de las ocurrencias de los registros miembros de un SET se lleva a cabo en el momento de su almacenamiento o posteriormente cuando lo decida el usuario. Si se elige la opción AUTOMATIC, cada vez que se almacena una ocurrencia en la base de datos, el programador no debe ocuparse de conectarla a la correspondiente ocurrencia SET, ya que es el sistema el que se ocupa de hacerlo, aunque el programador ha de proporcionar la información necesaria para ello, por el contrario, si la opción es MANUAL, será el programador el que, en el momento que lo considere oportuno, conectará las ocurrencias mediante la adecuada sentencia del lenguaje de manipulación. En ambos casos el programador deberá ocuparse de seleccionar la ocurrencia de SET a la que debe conectarse el registro miembro.

El modo de retención especifica el comportamiento que tendrán las ocurrencias de miembro respecto a la modificación de su conexión en un SET. La opción MANDATORY permite que una ocurrencia de miembro conectada a una ocurrencia de SET pueda cambiarse, a otra ocurrencia; sin embargo, no permite que la ocurrencia de miembro quede suelta, una vez que ha estado conectada a una ocurrencia de SET; es decir, podrá cambiar de propietario pero no podrá quedarse sin ninguno. Si se define el miembro como OPTIONAL, puede desconectarse de su propietario y quedar suelta. En las normas de 1978 se ha incluido una tercera opción FIXED que no permite que el registro quede suelto ni que tampoco cambie de ocurrencia de SET, la restricción es más fuerte que MANDATORY, y estaba contemplada en la mayoría de los SGBD comerciales.

La combinación de los diferentes modos de inserción y de retención permite al esquema CODASYL reflejar ciertas restricciones semánticas que se presentan en el mundo real que se pretende modelar.

Existen determinadas reglas que es preciso seguir en la elección de estas características. Por ejemplo, cuando un registro se ha definido con modo de ubicación VIA, su modo de inserción en ese SET ha de ser AUTOMATIC. Cuando se define un SET en modo DYNAMIC, los miembros han de ser OPTIONAL y MANUAL, aunque no se especifiquen estos miembros, ya que en este modo de SET cualquier registro de la base de datos puede ser miembro del SET. Cuando se define un SET singular, los miembros han de ser FIXED y AUTOMATIC.

Los modos de inserción y de retención tiene mucha relación con las operaciones de almacenamiento, conexión y desconexión de registros que proporcionan todos los SGBD. Las diferencias de instrumentación de estas operaciones en cada uno de los productos comerciales producen también diferencias en la utilización de cada una de las operaciones del modo de inserción y retención, estas diferencias no son realmente sintácticas, sino semánticas. Algunos de los sistemas han instrumentado modos de inserción y retención muy interdependientes, mientras que en otros su relación es bastante escasa.

La cláusula LINKED TO OWNER, que aparece a continuación de los modos de inserción y retención es opcional, y con ella obliga al sistema a mantener, desde cada ocurrencia del miembro, un puntero al propietario, lo cual aumenta el rendimiento de ciertas recuperaciones en la base de datos. Por sus implicaciones físicas, esta cláusula fue eliminada en 1978.

Otra de las cláusulas opcionales es KEY, de la que ya hemos hablado, con ella se identifican los elementos de

datos del registro miembro que se utilizarán como claves de clasificación si se ha elegido la opción SORTED en el orden del SET. La sintaxis de esta cláusula es:

KEY IS {ASCENDING, DESCENDING} elemento {ASCENDING, DESCENDING} elemento]

En donde puede observarse que es necesario indicar si la ordenación es ascendente o descendente para cada uno de los elementos de datos. CODASYL proporciona en el LDDE un medio para definir estructuras indexadas permitiendo indicar los elementos de datos del registro miembro que se utilizará a modo de claves para este tipo de acceso. El sistema crea con los elementos de datos especificados en la cláusula SEARCH−KEY, una tabla de índices o lista invertida por cada ocurrencia del correspondiente SET. La sintaxis es:

SEARCH−KEY IS elemento1, elemento2,...

Se puede especificar el permiso o prohibición de claves duplicadas, aunque es el sistema el que se encarga de su gestión en el caso de que se permitan. Es conveniente definir este tipo de listas invertidas con la cláusula SEARCH−KEY, cuando cumplen las siguientes condiciones:

- se tienen pocas ocurrencias de SET
- el número de miembros de cada ocurrencia es elevado
- es habitual buscar por el correspondiente elemento de datos.

La utilidad de esta cláusula es exclusivamente acelerar las búsquedas por determinados elementos de datos, aumentando la eficiencia, por este motivo desapareció del LDDE en 1978.

En determinadas operaciones, el SGDB se ve obligado a seleccionar una de las ocurrencias de un SET entre todas las existentes en la base de datos, para indicarle al sistema de que forma ha de elegir la ocurrencia de SET se ha de especificar el modo de selección de SET mediante la cláusula SET SELECTON, cuya sintaxis es:

SET SELECTION THRU set OWNER IDENTIFIED BY

{DATABASE_KEY

SYSTEM

CURRENT OF SET }

donde:

- *DATABASE_KEY*: Se selecciona la ocurrencia cuyo propietario tiene la clave de base de datos que indicará el usuario.
- *SYSTEM*: Es la que se aplica para los SET singulares, El SGBD es el que se ocupa de la selección de la ocurrencia.
- CURRENT OF SET: Se selecciona la ocurrencia activa en el momento.

Las especificaciones de 978 introdujeron otras modificaciones, la más importante fue STRUCTURAL CONSTRAINT (restricción estructural), se incluye con la idea de dotar al modelo CODASYL de cierto aspecto relacional. Su sintaxis es:

STRUCTURAL CONSTRAINT IS elemento1 EQUAL elemento2

El elemento de datos indicado en primer lugar debe corresponder al registro miembro, mientras que el segundo ha de pertenecer al registro propietario. Como muestra su sintaxis, se utiliza para asegurar que cierto elemento de datos del registro miembro de un SET tenga el mismo valor que otro elemento de datos del propietario, de esta forma es posible imponer restricciones a los miembros del SET, siguiendo las necesidades de determinados casos presentes en la realidad.

Lenguaje de definición de datos del subesquema

El lenguaje de definición de datos del subesquema (LDDS) CODASYL permite realizar modificaciones respecto al esquema del que se obtiene. las modificaciones pueden ser:

- *En las contraseñas*: Es posible asignar contraseñas al subesquema que sean diferentes de las del esquema, con objeto de aumentar la protección de la base de datos.
- *En el SET*: Se puede elegir los SET que sean necesarios, ignorando el resto, además de poder modificar el modo de selección de los mismos.
- *En el registro*: Se puede construir un registro en el subesquema utilizando solo parte de los elementos de datos que contenga en el esquema, e incluso modificar su orden dentro del registo.
- *En el elemento de datos*: Se puede modificar el formato de un elemento de datos en el subesquema para adaptarlo a las necesidades de la aplicación particular qeu lo vaya a utilizar. también pueden modificarse los grupos repetitivos cambiando el orden de los elementos de datos que lo forman e incluso dándole un nuevo nombre a todo el grupo.

Una vez definido el subesquema, debe ser compilado y almacenado de forma independiente de los demás para poder ser llamado posteriormente por los programas de aplicación que lo necesiten.

Ejemplo de definición de esquema

SCHEMA NAME IS GESTION_VIDEOCLUB

ON DISPLAY CALL VISUAL

PRIVACY LOCK FOR ALTER IS CAMBIO

PRIVACY LOCK FOR COPY IS CLP

AREA NAME IS VID_AREA

ON RETRIEVAL CALL RECUP

AREA NAME IS CLIENTE_AREA

PRIVACY LOCK FOR UPDATE EXCLUSIVE IS ORDEN

AREA NAME IS PELIC_AREA

RECORD NAME IS VIDEOCLUB

WITHIN VID_AREA

LOCATION MODE IS CALC USING ID_VIDCL

03 ID_VIDCL TYPE CHAR(2)

03 DIRECCION TYPE CHAR(20)

RECORD NAME IS CLIENTE

WITHIN CLIENTE_AREA

LOCATION MODE IS VIA ES_SOCIO

03 TIPO TYPE CHAR(1)

03 COD_CARNET TYPE CHAR(10)

03 FECHA_INGRESO TYPE FIXED(6)

03 ANTIGUEDAD TYPE 99 IS VIRTUAL RESULT OF PROC_CALC ON THIS PERIOD USING FECHA_INGRESO

CHECK IS COMPRUEBA_ANTIGUEDAD

SET NAME IS ES_SOCIO

OWNER IS VIDEOCLUB

ORDER IS PERMANENT INMATERIAL

MEMBER IS CARNET

AUTOMATIC FIXED

LINKED TO OWNER

Manipulación de datos

Las operaciones que constituyen la parte de manipulación de datos en un modelo de datos son de dos tipos, selección y acción, en selección lo que se hace es localizar un registro de la base de datos, para realizar distintas acciones sobre el; en acción las acciones que podemos realizar sobre el registro seleccionado son, recuperar (GET), insertar (STORE), borrar (ERASE), Modificar un registro (MODIFY), conectar (CONNECT), desconectar (DISCONNECT), reconectar (RECONNECT), y un largo etc.

El LMD CODASYL es el que realiza la interfaz entre el programa escrito en lenguaje anfitrión y la base de datos, debiendo suministrar los medios necesarios para poder trabajar con la base. La interacción entre el usuario y la base de datos se realiza a través de un área de trabajo del usuario (ATU), en cada programa de aplicación que se esta ejecutando se encuentran definidas un área de trabajo de usuario, un área de comunicación con el SGBD destinada a recibir los mensajes y la información de control procedente de éste, y una llamada que hace referencia al subesquema o vista externa permitida a tal programa.

El área de trabajo de usuario contiene las áreas de entrada / salida, especie de plantillas con las estructuras de los datos que se van a consultar o actualizar, y una zona de indicadores que el sistema modifica automáticamente. Existen varios tipos de indicadores:

- Indicadores de condición de error: contienen la causa del error ocurrido.
- Indicadores de pertenencia: proporcionan información sobre si un registro pertenece a una ocurrencia

de SET.

- Indicadores de SET vacío: informan sobre si una ocurrencia de SET tiene o no miembros.
- Indicadores de registro activo: son punteros que señalan el último registro seleccionado en la unidad de ejecución, existen cuatro tipos de registros activo:
    - ◊ Indicador de registro activo de la correspondiente unidad de ejecución, que señalará al último registro que haya sido accedido en dicha unidad.
    - ◊ Indicadores de registro activo de área, que contendrán la dirección del último registro de cada una de las áreas.
    - ◊ Indicadores de registro activo de tipo de registro, que apuntarán al último registro de cada tipo que haya sido accedido.
    - ◊ Indicadores de registro activo de SET, cada uno de los cuales señala a la última ocurrencia de propietario o de miembro que haya sido accedida en el correspondiente SET, determina la ocurrencia de SET activa en cada momento.

Sentencias

FIND

Es la sentencia fundamental del LMD, por medio de esta sentencia se localiza un registro que cumpla unas determinadas condiciones. La sentencia tiene múltiples formatos y puede tener varias variables. Una vez localizado el registro , la sentencia adecuada se ocupará de mostrarlos, borrarlo, recuperarlo...

GET

Sirve para recuperar un registro seleccionado, los datos los obtiene de la ocurrencia

de registro activo en la unidad de ejecución y los lleva al área de trabajo de usuario.

STORE

STORE inserta una nueva ocurrencia de registro en la base de datos, conectándola a las ocurrencias de donde el registro es miembro, y crea ocurrencias de SET en aquellos tipos de SET donde el registro estas definido como propietarios.

ERASE

Borra la ocurrencia de registro activo de la unidad de ejecución, pone a nulo el valor de el indicador correspondiente.

MODIFY

Modifica uno o más elementos de datos de la ocurrencia de registro activa de la unidad de ejecución o cambia la condición de pertenencia del registro asignándola a otra ocurrencia de SET.

CONNECT

Conecta una ocurrencia de registro existente en la base de datos a uno o más SET.

DISCONNECT

Desconecta una ocurrencia como miembro de un tipo de SET.

RECONNECT

Cambia la ocurrencia de registro activo de la unidad de ejecución de la ocurrencia de SET a la que pertenecía a la ocurrencia activa del SET indicado, seleccionada mediante el SET SELECTION.

READY

Abre una o más áreas para recuperación o actualización.

FINISH

Libera una ó más áreas de la unidad de ejecución cuando ya no se necesitan.

ORDER

Ordena las posiciones lógicas de los miembros de un SET de acuerdo con la clave que se especifique.

RETAINING CURRENCY

Inhibe la actualización de los indicadores de registro activo que se especifiquen.

ACCEPT

Se utiliza para extraer la clave de base de datos de los indicadores de registro activo.

Esquema de almacenamiento

La modificación en 1978 de la arquitectura CODASYL supuso la aparición de un nuevo nivel de descripción de los datos donde se definen aspectos físicos relativos a su almacenamiento. Este nuevo nivel, denominado *esquema de almacenamiento*, se usa para proporcionar:

- Independencia del esquema respecto a los aspectos puramente físicos.
- Independencia del sistema operativo, haciendo posible la adaptación a sistemas .operativos que realizan la gestión del almacenamiento físico de diferentes formas. También proporciona independencia respecto a los dispositivos físicos.
- Eficiencia en los accesos y en el almacenamiento de datos al proporcionar un medio para definir la ubicación de los datos en el soporte físico.
- Posibilidad de reestructuraciones de la base de datos a fin de mejorar su eficiencia sin necesidad de modificar su definición lógica.

Lenguaje de definición del esquema de almacenamiento

En el lenguaje de definición de almacenamiento (LDEA), se engloba el control de soporte/dispositivo y los caminos de acceso y emplazamiento de los registros. −del control de soporte/dispositivo solo se llegó a definir cual era su función, pero nunca se definió su sintaxis y su implementación comercial fue muy irregular; Los modos de acceso y emplazamiento de los registros se encargan de determinar la estrategia que se utilizará en el acceso y almacenamiento de los datos, con independencia de los dispositivos físicos de la instalación.

Esquema de almacenamiento

La base de datos puede considerarse físicamente dividida en áreas de almacenamiento, que ocupan partes concretas del soporte físico que se utilice. En un área de almacenamiento se encuentran los registros

almacenados, que son los que verdaderamente contienen la información de la base de datos.

El esquema de almacenamiento permite que la asignación de registros lógicos a registros físicos se pueda hacer utilizando tanto fragmentación vertical como horizontal, para aumentar la eficiencia de los accesos. La fragmentación vertical, que solo puede realizarse en la arquitectura de tres niveles, consiste en la aparición de un mismo registro lógico repartido en varios dispositivos de almacenamiento. La fragmentación horizontal consiste en situar parte de los registros lógicos en una zona de almacenamiento y parte en otra, por ejemplo, según cumplan, o no, una condición. Este tipo de fragmentación ya era posible en las bases de datos con arquitectura a dos niveles, que permitían asignar las ocurrencias de un mismo tipo de registro a dos áreas distintas.

Cada área de almacenamiento se divide, a su vez, en páginas, cada una de las cuales puede contener uno o varios registros almacenados. Normalmente, las páginas suelen hacerse coincidir con las unidades de las operaciones de entrada / salida, pero no siempre ha de ser así.

El esquema de almacenamiento se divide a su vez en las siguientes entradas:

- Entrada de esquema de almacenamiento

Es la que da nombre al esquema de almacenamiento.

- Entrada de descripción de correspondencia

Indica las correspondencias entre los tipos de registro del esquema y los tipos de registro de almacenamiento.

- Entrada de área de almacenamiento

Identifica y describe las áreas de almacenamiento

- Entrada de registro de almacenamiento

Esta formada por tres subentradas

- Descripción de los elementos de datos que forman parte de los registros, especificándose características físicas.
- STORAGE RECORD tipo−registro−almacenamiento

LINK TO otro−tipo−registro−almacenamiento−del−mismo reg−del−esquema

Permite especificar la conexión entre varios registros de almacenamiento

que se corresponden con el mismo registro lógico del esquema.

- Entrada de conjunto

Su misión es la especificación de punteros que puedan mejorar los tiempos de recuperación de los registros en los conjuntos.

- Entrada de índice

Se Utiliza para definir índices para la clave de registro para los registros del esquema y los índices de miembros o propietarios de conjuntos.

Gestores

IDMS

IDMS es un sistema de base de datos de red desarrollado por Cullinane database systems, Inc. Corre sobre mainframes IBM sobre sistemas operativos como VSE, MVS, etc... Es probablemente el mejor ejemplo en cuanto a implementación del modelo CODASYL, y frecuentemente es citado como sistema CODASYL.

Incluye un lenguaje de manipulación de datos detallado que permite al diseñador de base de datos un alto grado de control sobre la organización física de la base de datos. El lenguaje de manipulación de datos incluye características idénticas al DML. Se incluyen características adicionales para facilitar la labor del programador y para que los programadores experimentados sean capaces de escribir consultas más eficientes. También incluye un editor para DC (Data Communications), el IDMS−DC, integra un diccionario, el IDD, y un interfaz de Queries, el OLQ, un interfaz para el uso de lenguaje natrual (solo en ingles), generador de aplicaciones (application Developement System / OnLine) y muchas cosas más.

Un ejemplo de una característica de IDMS, es la orden *obtain*, que combina las ordenes *find* y *get* en una solicitud. Puede añadírsele una cláusula *where* opcional a la orden *obtain* para encontrar el siguiente registro que satisface el predicado *where*. Esto le ahorra al programador la tarea de escribir una prueba explícita para un registro que se haya localizado por medio de la orden *find*.

En 1983 Cullinet anunció una versión avanzada de IDMS llamada IDMS/R (R= relacional). IDMS/R incluía todas lacs características del modelo original CODASYL con algunos toques relacionales.

Una base de datos IDMS esta definida por un esquema, este esquema está escrito en el DDL de IDMS, y una vez creado y compilado, es guardado en el diccionario de IDMS, al igual que el DDL del subesquema.

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 del modelo de red.

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*.

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. En este capitulo, cada uno de estos árboles con raíz se denominara árbol de

base de datos.

El contenido de un registro especifico 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, tendrá 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.

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 ultimo 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 se ilustra en la figura 2. 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 arboles con raíz.

Relaciones únicas

Considérese el diagrama entidad relación de la Figura 3 a, 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 3 b.

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.

(a)

Cliente

Cuenta

(b)

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 1. Puesto que la relación es uno a muchos de cliente a cuenta, un cliente puede tener más de una cuenta , como en el caso de Camp, que tiene las cuentas 226 y 177. Sin embargo, una cuenta no puede pertenecer a mas 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 (Figura 4.). En la figura 5 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 (vease la Figura 6 a.), entonces la transformación del diagrama E−R a un diagrama de estructura de árbol es mas 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 tendra(n) registros repetidos.

(a)

Cliente Cuenta

Cuenta Cliente

(b)

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.

Presentaremos una transformación lo mas general posible. Es decir, las demás transformaciones posibles seran casos especiales de esta.

Para transformar el diagrama E−R de la figura 6 a. a uno de estructura de árbol, necesitaremos hacer lo siguiente:

- Crear dos diagramas de estructura de árbol distintos, T1 y T2, cada uno de los cuales incluye los tipos de registro cliente y cuenta, En el árbol T1, la raíz es cliente, mientras que en el árbol T2 la raíz es cuenta.
- Crear los dos enlaces siguientes:

    - CtaCli, un enlace muchos a uno del tipo de registro cuenta al tipo de registro cliente, en T1.
    - CliCta, un enlace muchos a uno del tipo de registro cliente al tipo de registro cuenta, en T2.

El diagrama de estructura de árbol resultante se muestra en la Figura 6 b. En la figura 7.se muestra un ejemplo de base de datos correspondiente al diagrama de estructura de árbol de la Figura 6 b. Exsiten dos arboles de base de datos. El primer árbol (Figura 7 a.) corresponde a al diagrama de estructura de árbol T1, mientras que el segundo árbol corresponde al diagrama de estructura de árbol T2. Como puede verse, todos los registros cliente y cuenta están repetidos en ambos arboles de base de datos. Ademas, el registro cuenta 347 aparece dos veces en el primer árbol, mientras que los registros cliente Katz y Doner aparecen dos veces en el segundo árbol.

(a)

(b)

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 casose necesita crear un nuevo tipo de registro y establecer los enlaces adecuados. La forma de establecer los enlaces dependera de cómo se defina la relación CliCta.

Considerese en diagrama E−R de la Figura 3 a. Supongase que añadimos el atributo fecha a la relación CliCta, para indicar la ultima fecha en la que el cliente tuvo acceso a la cuenta. El diagrama E−R correspondiente se muestra en la Figura 8 a. Para transformar este diagrama a uno de estructura de árbol necesitamos:

- Crear un nuevo tipo de registro, fecha, con un solo campo.
- Crear los dos enlaces siguientes:

    - CliFecha, un enlace muchos a uno del tipo de registro fecha al de registro cliente.
    - FechaCta, un enlace muchos a uno del tipo de registro cuenta al de registro fecha.

(a)

(b)

El diagrama de estructura de árbol resultante se ilustra en la Figura B.8 b. En la Figura B.9. aparece una instancia que corresponde al esquema que se acaba de describir. Muestra lo siguiente:

- Lowman tiene la cuenta 305, que fue accedida por ultima vez el 15 de Septiembre de 1980.
- Camp tiene dos cuentas: la 226, que fue accedida por ultima vez el 1 de Octubre de 1983, y la 177, que fue accedida por ultima vez el 23 de Noviembre de 1984.
- Kahn tiene la cuenta 155, que fue accedida por ultima vez el 15 de septiembre de 1980.

Nótese que es posible abrir dos cuentas el mismo dia, como en el caso de las cuentas 155 y 305. Puesto que estas cuentas pertenecen a dos clientes distintos, el registro fecha debe repetirse para conservar la jerarquia.

Si la relación CliCta fuera uno a uno con el atributo fecha, entonces el algoritmo de transformacion 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 mas 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, como se muestra en la figura B.10.

Cliente Cuenta

Fecha Fecha

Cuenta Cliente

En la Figura 11 se presenta un ejemplo de base de datos correspondiente a este esquema.

(a)

(b)

Hasta ahora hemos considerado únicamente relaciones binarias. Ahora prestaremos atención a relaciones generales. La transformación de diagramas E−R que corresponden a relaciones generales a diagramas de estructura de árbol es bastante complicada. En vez de presentar un algoritmo de transformación general, presentaremos un solo ejemplo para ilustrar la estrategia global que puede aplicarse cuando se necesita hacer una de estas trasformaciones.

(a)

Sucursal Sucursal

Cliente Cuenta

Cuenta Cliente

(b)

Considérese el diagrama de entidad−relación de la figura 12 a, que consta de tres conjuntos de entidades, cliente, cuenta y sucursal, relacionados mediante el conjunto de relaciones general CCS 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 Figura12 b.

En la Figura 13 se ilustra una instancia de la base de datos que corresponde al esquema descrito arriba. Muestra que Beck tiene la cuenta 200 en la sucursal Northcross, Katz las cuentas 256 y 347 en las sucursales Northcross y Highland, respectivamente, y que Doner tiene las 347 y 301 en la sucursal Highland.

Este algoritmo de transformación puede extenderse de una manera directa para tratar relaciones que abarcan mas 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.

(a)

(b)

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 arboles 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 numero 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.

Considérese el diagrama E−R de la Figura 14 a. Aplicando el algoritmo de transformación descrito en la seccion 2.1. por separado a las relaciones SucCta y CliCta, obtenemos un diagrama de base de datos. Este diagrama no es de árbol con raíz, ya que la única raíz posible seria el tipo de registro cuenta, pero este tipo de registro tiene relaciones muchos a uno con sus dos hijos, lo que viola la definición de árbol con raíz (vease la seccion 2.). Para transformar este diagrama en uno que tenga la forma de árbol con raíz, repetimos el tipo de registro cuenta, creamos dos arboles separados. Obsérvese que ambos son arboles con raíz. así, en general, podemos dividir un diagrama de este tipo en varios diagramas, cada uno de los cuales es un árbol con raíz.

Considerese el diagrama E− de la Figura 16 a. Aplicando el algoritmo de transformación descrito en la Seccion 2.1, obtenemos otro diagrama de base de datos. Este diagrama no tiene los forma de árbol con raíz, puesto que contiene un ciclo. Para transformar el diagrama en uno de estructura de árbol, repetimos los tres tipos de registros y creamos dos diagramas separados. Observese que ambos diagramas son arboles con raíz. así, en general, podemos dividir un diagrama de este tipo en varios diagramas, cada uno de los cuales es un árbol con raíz.

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 ordenes que están incorporadas en Pascal. Usaremos un ejemplo sencillo de un esquema cliente−cuenta−sucursal.

El diagrama de estructura de árbol que corresponde a este esquema aparece en la Figura 18. Especifica que una sucursal puede tener varios clientes, cada uno de los cuales puede tener varias cuentas. Sin embargo, una cuenta puede pertenecer únicamente a un cliente, y un cliente puede pertenecer solamente a una sucursal.

Area 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 ordenes 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 ultima 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.

De nuevo hacemos hincapié en que un área de trabajo de programa particular esta asociados a un solo programa de aplicación.

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 suscursal.
- Registro cliente.
- Registro cuenta.

- Puntero de actualidad: un puntero al ultimo 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 area de trabajo.

La orden get debe especificar en cual de los árboles de la base de datos se va a buscar. Para este ejemplo suponemos que el único árbol en el que se va a buscar es el ejemplo de la base de datos de la Figura B.19, y por tanto omitimos esta especificación en las consultas.

Para ilustrar el efecto general que tiene la orden get sobre el área de trabajo de programa, considérese el ejemplo de base de datos de la Figura B.19. 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 area 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 busqueda 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 mas a la

izquierda, y así sucesivamente, hasta que alcancemos un nodo hoja (sin hijos). A continuacion 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 mas simple tiene la forma:

La clausula where es opcional. La < condicion > 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 < condicion > de la clausula 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 regsitro en la plantilla adecuada dentro del area de trabajo. Si no existe ese registro en el árbol de la base de datos, la busqueda 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 direccion del cliente Fleming.

Como un ejemplo mas, considerese 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 < condicio >. Si se omite la clausula where, entonces se localiza el siguiente registro del tipo < tipo de registro >. Observese que el sistema utiliza el puntero de actualidad para determinar desde que punto debe reanudarse la busqueda. Como antes se veran afectados el puntero de actualidad, la plantilla del tipo < tipo de registro > en el area de trabajo y DB−status.

Para ilustrarlo, construyamos la consulta de base de datos que imprima el numero de cuenta de todas las cuetnas con un saldo mayor de 500 dolares:

Hemos encerrado parte de la consulta en un bucle while, ya que no sabemos por adelantado cuantas de las cuentas cumplen esa condicion. 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 cuenta.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 busqueda a un subarbol especifico en vez de buscar en todo el árbol de base de datos. La raíz del subarbolen cuestion es el ultimo registro que se localizo con las ordenes get first o get next. La orden get para localizar un regsitro dentro de ese subarbol tiene la forma:

que localiza el siguiente regsitro (en preorden) que satisface la < condicion > en el subarbol cuya raíz es el padre del actual de < tipo de registro >. Si se omite la clausula where, entonces se localiza el siguiente registro del tipo < tipo de registro >dentro del subarbol indicado. Observese que el sistema utiliza el puntero de actualidad para determinar desde que punto debe reanudarse la busqueda. Como antes, se veran 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 regsitro 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 valos 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 ordenes 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 area de trabajo. Una vez hecho esto, añadimos este registro nuevo al árbol de la base de datos ejecutando:

insert < tipo de registro >

where < condicion >

Si se incluye la clausula where, el sistema busca en el árbol de la base de datos (en preorden) un registro que satisfaga la < condicion > de la clausula where. Una vez que encuentre ese registro, llamemosle X, el registro recien creado se inserta en el árbol como hijo mas a la izquierda de X. Si se omite la clausula where, el registro se inserta en la primera posicion (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.

Considérese el programa para añadir un nuevo cliente, Jackson, a la sucursal Seashore:

cliente.nombre := Jackson;

cliente.calle := Old Road;

cliente.ciudad :=Queens;

insert cliente

where sucursal.nombre = Seashore;

Como un ejemplo mas, considere el programa para crear una cuenta con numero 655 que pertenezca al cliente Jackson:

cuenta.nombre := 655;

cuenta.saldo := 100;

insert cuenta

where cliente.nombre = Jackson;

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 regsitro 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.

Como ejemplo, considerese e programa que cambia la calle de Boyd a Northview:

get hold first cliente

where cliente.nombre = Boyd

cliente.calle :=Northview;

replace;

Nótese que en el lenguaje solamente tenemos un registro que contiene la dirección de Boyd. Si no fuera así, el programa habría incluido un bucle para buscar todos los registros de Boyd.

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.

Para ilustrarlo, considerese el programa que elimina la cuenta 561:

get hold first cuenta;

where cuenta.numero = 561;

delete;

Una operacion de eliminacion borra no solo el registro en cuestion, sino todo el subarbol del cual es raíz. así, para eliminar el cliente Boyd y todas sus cuentas, escribimos:

get hold first cliente

where cliente.nombre = Boyd;

delete;

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 repeticion de registros tiene dos inconvenientes importantes:

- La actualizacion 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.

Como ejemplo, considérese el diagrama E−R de la Figura 6 a y a su correspondiente diagrama de estructura de árbol, que incluye los tipos de registro cliente y cuenta (Figura 6 b).

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 linea discontinua desde el registro cuenta−virtual al registro cuenta, para especificar la asociacion entre unregsitro virtual y su correspondiente registro fisico.

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

Gestores

El modelo jerárquico es importante principalmente debido a la importancia del sistema de base de datos IMS de IBM. EN esta seccion trataremos el IMS y otro sistema jerárquico ampliamente utilizado, el Sistema 2000.

IMS

El sistema de gestion de informacion (IMS, Information Management System) de IBM es uno de los sistemas de base de datos mas antiguos y mas ampliamente utilizados. Trabaja con el sistema operativo MVS en maquinas grandes basadas en la arquitectura IBM 370. Puesto que históricamente las bases de datos IMS han sido de las mas grandes, las personas encargadas del desarrollo de IMS fueron de las primeras en tener que enfrentarse a los problemas de concurrencia, recuperación, integridad y procesamiento eficiente de las consultas. A traves de varias version, IMS adquirio un gran numero de características y opciones. Como resultado, IMS es un sistema de gran complejidad. Aquí consideraremos solo algunas características de IMS.

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 numero 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 (metodo jerárquico de acceso secuencial, hierachical sequential−access method) se utiliza para archivos secuénciales físicamente (como los archivos en cinta). Los registros se almacenan fisicamente en preorden.
- HISAM (metodo jerárquico de acceso secuencial indexado, hierachical indexed−sequential−access method) es una organización indexada en el nivel de raíz de la jerarquia.
- 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 teoria de control de concurrencia. Las primeras versiones de IMS tenian una forma sencilla de control de concurrencia. Solo podía ejecutarse un programa que incluyera actualizaciones a la vez. Sin embargo, podia ejecutarse de manera concurrente cualquier numero 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 caracteristica de aislamiento de programas mas sofisticada que permitía un mejor control de la concurrencia y tecnicas de recuperacion 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 introduccion de IMS Fast Path (Camino rapido). Fast Path utiliza una organización fisica 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 transaccion (como lo hace el IMS estandar), la actualizacion se posterga hasta un punto de verificacion o de sincronizacion. En el caso de una caida, el subsistema de recuperacion 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 mas grande y menos cara, ha habido un interes 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 princiapio 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 caracteristicas del lenguaje que ofrece el Sistema 2000 presentan varias diferencias interesnates con respecto al lenguaje DL/I de IMS.

El concepto fundamental del lenguaje de definicion de datos DL/I es la relación padre−hijo uno a muchos. En el ejemplo bancario, supongase que existe uno a muchos entre cliente y cuenta. Diseñariamos la base de datos jerárquica del banco definiendop cuenta como hijo de cliente. Una manera alternativa de ver esta relación uno a muchos eb considerar un registro que pueda tener campos repetidos. Dentro del registro cliente de un cliente determinado, podemos almacenar el conjunto de cuentas pertenecientes a ese cliente creando un campo cuenta repetido. Este enfoque de campos repetidos es el fundamento del lenguaje de definicion de datos del Sistema 2000.

En el ejemplo bancario, el registro cliente tiene los mismos campos nombre−cliente, calle y ciudad−cliente, y el registro cuenta tiene los campos numero−cuenta y saldo. A continuacion mostramos las sentencias para esta parte de base de datso del banco utilizando el lenguaje de definicion de datos Sistema 2000.

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 indice por cada uno de los campos clave.

Considerese la consulta Encontrar el nombre y la ciudad de todos los clientes que tienen una cuenta cuyo saldo es mayor de 10000 dolares. 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 solasentencia:

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 analisis sintactico de las consultas se hace durante la fase de precompilacion. Al igual que la mayor parte de los sistemas comerciales, el Sistema 2000 incluye facilidades que ayudan en la generacion de informes.

Biografía

− Técnicas de bases de datos

Benet Campderrich

Editores técnicos asociados S.A

1983 ISBN: 84−7146−246−X

Deposito legal B. 7.063−1984

− Concepción y diseño de bases de datos

Adoración de miguel castaño

Editorial Ra−Ma

1993 ISBN: 84−7897−083−−5

Deposito legal M−2975−1993

− Organización de las bases de datos

James Martin

Prentice Hall Internacional

Editorial Dossat S.A

1977 ISBN: 84−237−0493−9

Deposito legal M−537−1982

− Fundamentos de bases de datos (segunda edición)

Henry F.Korth

McGraw−Hill

1993 ISBN: 84−481−0079−4

Deposito legal: M−13.750/1993

− Principles of database systems

Jeffery D.Ullman

Computer Science press (stanford university)

1982 UK ISBN: 0−273−085948

− An Introduction to Database Systems (volume 1 3ºedicion)

C.J Date

Addison−Wesley publising company & IBM Editorial board

1977

− An introducition to Database Systems (volume 1 5ºedicion)

C.J Date

Addison−Wesley publishing company

1990 ISBN: 0−201−52878−9

Bases de datos en red y jerárquicas

66

A

B

C

D

n **I1** m

1

I3N

n

I2

1

n

I7

m

m

n

I5

n

I4

m

n

I6

1

B

D

Ei

Ej

Iij

ei1

ei2

ei3

ej1

ej2

ej3

ej4

ej6

ej5

ej4

ej3

ej2

ej1

ei3

ei2

ei1

Iij

Ej

Ei

ej6

ej5

ej4

ej3

ej2

ej1

ei3

ei2

ei1

Iij

Iij

Ei

ek2

ej3

ej3

ej2

ej2

ej1

ei2

ej1

ei1

Ei

ei2

ej4

ei1

Iijk

Ej

Ei

Ek

ek1

propietario

miembro

nombre

P1

M1

M2

Mn

Editorial

Libro

Editorial_1

Libro_3

Libro_4

Libro_1

Libro_3

Pública

A

B

C

D

E

F

G

G

F

E

D

C

B

A

Set 1

Set2

Set3

Set4

Set5

TYPE IS DECIMAL

BINARY

FIXED

FLOAT

REAL entero

COMPLEX

CHARACTER

BIT

DATABASE_KEY

RECORD NAME IS nombre

WITHIN nombre(s) de area(s)

LOCATION MODE IS DIRECT

CALC

VIA

SYSTEM

[ON . CALL]

[PRIVACY LOCK .]

[PICTURE IS formato]

TYPE IS formato

[OCCURS entero TIMES]

elemento

[CHECKIS procedimiento]

AREA NAME IS nombre

[AREA IS TEMPORARY]

PROTECTED

RETRIEVAL EXCLUSIVE

[ON PROTECTED CALL procedimiento]

UPDATE EXCLUSIVE

PROTECTED

RETRIEVAL EXCLUSIVE literal

[PRIVACY LOCK [FOR PROTECTED] IS variable

UPDATE EXCLUSIVE porced

SCHEMA NAME IS nombre del esquema

[On procedimiento]

[PRIVACY LOCK contraseña]

AREA NAME IS nombre del área

[AREA IS TEMPORARY]

[ON procedimiento]

[PRIVACY LOCK contraseña]

RECORD NAME IS nombre del registro

WITHIN nombre(s) del area(s)

MODE IS DIRECT

CALC

VIA

SYSTEM

[ON procedimiento]

[PRIVACY LOCK contraseña]

Nº de nivel y nombre del elemento de datos

[PICTURE IS formato]

[TYPE IS formato]

[OCCURS IS TIMES]

[ENCODING procedimiento]

DECODING

[ACTUAL SOURCE IS campo]

VIRTUAL RESULT OF procedimiento

SET NAME IS nombre del conjunto

OWNER IS nombre del propietario

SYSTEM

[SET MODE IS DYNAMIC]

PRIOR PROCESSIBLE

ORDER IS PERMANET FIRST

TEMPORARY LAST

PRIOR

NEXT

SORTED

INMATERIAL

[ON procedimiento]

[PRIVACY LOCK contraseña]

MEMBER IS nombre del miembro

MANDATORY AUTOMATIC

OPTIONAL MANUAL

[LINKED TO OWNER]

[KEY IS ASCENDING elemento de datos]

DESCENDING

[DUPLICATE ARE FIRST ALLOWED]

LAST

NOT

[SEARCH KEY IS elemento(s) de datos]

[SET SELECTION THRU nombre del set]

[OWNER IDENTIFIED BY]

SCHEMA NAME IS nombre

[PRIVACY LOCK [FOR ALTER] IS literal]

COPY variable

DISPLAY proced

LOCKS

[ON ALTER CALL procedimiento]

COPY

DISPLAY

LOCKS

IS ACTUAL SOURCE OF elem. OF OWNER set

VIRTUAL

IS ACTUAL RESULT procedimiento [elem(s)] USING [ON MEMBERS reg(s) OF set]

VIRTUAL

SET NAME IS nombre

OWNER IS registro

SYSTEM

[SET MODE IS DYNAMIC]

PRIOR PROCESSIBLE

ORDER IS PERMANET INSERTION IS

TEMPORARY

FIRST

LAST

PRIOR

NEXT

INMATERIAL

SORTED

[NAMES nombre]

BY DATABASE_KEY

WITHIN RECORD_NAME

BY DEFINED KEYS

[DUPLICATES ARE FIRST ALLOWED

LAST

NOT

[ON .]

[PRIVACY LOCK]

MEMBER IS nombre INSERTION IS AUTOMATIC

MANUAL

RETENTION IS MANDATORY

OPTIONAL

FIXED

[LINKED TO OWNER]

[KEY IS ASCENDING elementos de datos]

DESCENDING

[SEARCH KEY IS elem, [elem.]−−−]

[DUPLICATES ARE [NOT] ALLOWED]

[SET SELECTION IS THRU set]

OWNER IDENTIFIED BY DATABASE_KEY

SYSTEM

CURRENT OF SET

[ON]

[PRIVACY LOCK]

[IF condición]

PLACEMENT IS

CALC [procedimiento] USING clave

CLUSTERED VIA SET nombre [NEAR OWNER]

SEQUENTIAL ASCENDING identificador

DESCENDING

WITHIN area−de−almacenamiento

Se utiliza para asignar la ubicación de los registros de almacenamiento dentro de las áreas que se hayan definido. La subintrada se compone de una única cláusula, PLACEMENT, que permite especifica una condición a partir de los elementos de datos del registro; si se cumple la condición el registro es almacenado en el lugar indicado. Si elegimos la opción CALC, los registros se colocan dentro del área según el resultado de aplicar una clave. CLUSTERED, es similar al modo VIA de la cláusula LOCATION MODE, y en SEQUENTIAL, los datos se colocan en orden secuencial.

lo mejor que podéis hacer es buscar en libros viejos

Ignacio Charfole

Mil y un gracias la biblioteca de la UCM es genial.

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

Nombre

Calle

Número

Saldo

Ciudad

Ciudad

Ciudad

Calle

Nombre

Número

Saldo

Ford

Post

Georgetown

Moody

Oxford

Pittsburgh

Nabors

Willow

Philadelphia

710

10.577

675

544

860

400

Nombre

Calle

Ciudad

Saldo

Número

Cliente

Cuenta

CliCta

Nombre

Calle

Número

Saldo

Ciudad

Nombre

Calle

Número

Saldo

Ciudad

Beck

Maple

San Francisco

Katz

North

San José

Doner

Sidehill

Palo Alto

200

55

256

100.0000

347

667

347

667

301

10.533

200

55

256

100.0000

347

667

301

10.533

Beck

Maple

San Francisco

Katz

North

San José

Katz

North

San José

Doner

Sidehill

Palo Alto

Doner

Sidehill

Palo Alto

Nombre

Calle

Ciudad

Saldo

Número

Cliente

Cuenta

CliCta

Fecha

Nombre

Calle

Ciudad

Fecha

Número

Saldo

Lowman

Square

Dallas

Camp

Downridge

Garland

Kahn

Bayside

Plano

200

55

256

100.0000

347

667

301

10.533

1 Oct. 1983

23 Nov. 1984

15 Sept. 1980

15 Sept. 1980

Número

Saldo

Nombre

Calle

Ciudad

Fecha

Número

Saldo

Nombre

Calle

Ciudad

Fecha

Beck

Maple

San Francisco

Katz

North

San José

Doner

Sidehill

Palo Alto

200

55

256

100.000

347

667

347

667

301

10.533

23 Nov. 1980

23 Nov. 1980

23 Nov. 1980

23 Nov. 1980

23 Nov. 1980

200

55

347

667

301

10.533

Beck

Maple

San Francisco

Katz

North

San José

Katz

North

San José

Doner

Sidehill

Palo Alto

Doner

Sidehill

Palo Alto

256

100.000

23 Nov. 1980

23 Nov. 1980

23 Nov. 1980

23 Nov. 1980

23 Nov. 1980

Nombre

Calle

Ciudad

Saldo

Número

Cliente

Cuenta

CliCta

Nombre

Sucursal

Ciudad

Activo

Número

Saldo

Nombre

Calle

Ciudad

Nombre

Activo

Ciudad

Número

Saldo

Nombre

Activo

Ciudad

Nombre

Calle

Ciudad

Beck

Maple

San Francisco

Katz

North

San José

Doner

Sidehill

Palo Alto

200

55

256

100.000

347

667

347

667

301

10.533

Katz

North

San José

Northcross

Highland

Highland

Northcross

200

55

256

100.000

347

667

301

10.533

Beck

Maple

San Francisco

Katz

North

San José

Katz

North

San José

Doner

Sidehill

Palo Alto

Doner

Sidehill

Palo Alto

SucCtaa

Nombre

Activo

Ciudad

Calle

Nombre

Sucursal

Cliente

Numero

Cuenta

Saldo

Ciudad

CliCta

C

B

A

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);
```