BASES DE DATOS INTELIGENTES *

RAMON BRENA

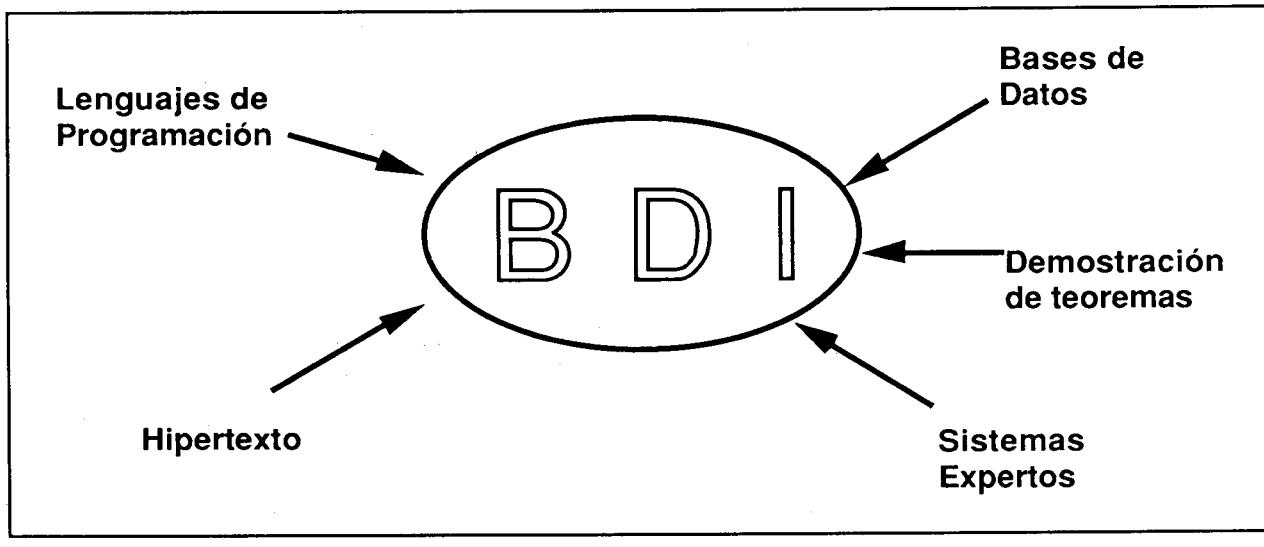
- Ingeniero en Computación en la Facultad de Ingeniería de la Universidad Nacional Autónoma de México.
- Diploma de Estudios Avanzados en Informática, en Grenoble, Francia.

* Documento presentado en el Simposio SICTE'91. Medellín: Universidad EAFIT, 1991.

1. LAS BDI: UNA CONFLUENCIA DE TECNOLOGIAS

Al hablar de Bases de Datos Inteligentes no estamos hablando de una tecnología única o coherente, sino de la conjunción de varias tecnologías, tales como los sistemas deductivos de la Inteligencia Artificial, o los lenguajes de programación basados en objetos, o bien los procesadores avanzados de texto. Bases de datos Inteligentes (BDI) es por lo tanto un término muy amplio que enfatiza el hecho de superar las limitaciones de rigidez de las Bases de Datos tradicionales.

Las principales tecnologías que intervienen en las BDI aparecen en la siguiente figura:



2. FUNCIONALIDADES DE LAS BDI

Al definir las BDI lo que nos interesa en primer lugar es lo que pueden ofrecernos en tanto que usuarios de ellas, es decir, sus funcionalidades. [Parsaye et al. 90].

2.1 Query processing flexible

Una de las más importantes ventajas que se quieren obtener mediante las BDI es poder hacer consultas de información de una manera más flexible que como se hacen con las BD tradicionales.

Palabras parecidas

Si hacemos en una BD tradicional una interrogación:

Nombre = "Rodrígez" **and salario** > 10000

Donde el apellido Rodríguez está mal escrito, la BD no nos regresará la información que nos interesa, sin darse cuenta de lo "cerca" que estuvo de encontrar el resultado. En una BDI con facilidades de comparación de palabras, es posible indicarle al sistema que recupere las palabras "que se parezcan" a una palabra dada.

Esta característica no solamente es importante para la corrección de errores, sino también para hacer al sistema inmune a variaciones en las terminaciones de palabras (por ejemplo, si se busca en un catálogo de biblioteca "almacenamiento", cuando el término conocido por el sistema es "almacenar")

Conceptos relacionados

Las BDI pueden extender las respuestas a una interrogación para considerar los términos que están de alguna manera relacionados con el término propuesto. Por ejemplo, en un sistema de información escolar es necesario que al hacer una consulta sobre **"alumnos"** se obtengan las informaciones sobre **"estudiantes"**.

Relaciones difusas

En una BDI se pueden hacer consultas usando conceptos imprecisos (por ejemplo **"edad alrededor de 30 años"**), que permiten obtener una primera aproximación a una interrogación. Esto es particularmente importante cuando el usuario no sabe exactamente lo que quiere, y prefiere irlo precisando en función de lo que encuentra en la BD (Esta es la situación más común en bancos de datos!).

Acceso asociativo de la información

Los sistemas basados en hipertexto permiten accesar un grupo de informaciones basándose en el contenido. Esta es una forma de acceso asociativo a la información, que puede brindarse en las BDI.

Consecuencias lógicas

Una BDI es capaz de considerar no sólamente las informaciones que están explícitamente almacenadas, sino también otras informaciones que son una consecuencia lógica de lo que está almacenado. Por ejemplo, si tenemos una base con información sobre parentescos (quiénes son padres de quiénes), la BDI debe inferir quién es abuelo de quién.

2.2 Interfaces Inteligentes

Las BDI pueden brindar una mayor facilidad de uso que las BD tradicionales. Al examinar y eventualmente organizar una BDI el administrador cuenta con herramientas de alto nivel para hacerlo.

Navegación flexible por la información

Las BDI ofrecen formas de "browsing" mucho más sofisticadas que aquellas de las BD relacionales. Por ejemplo, es posible manejar el nivel de detalle con que se consulta una información mediante comandos de "zoom-in", para aumentar el detalle, o de "zoom-out", para abstraer detalles innecesarios y dar una vista de conjunto.

Integración con multimedios

La incorporación de imágenes (fijas o en movimiento) y sonido a la interface con que se explota un sistema lo hace más intuitivo (y atrativo) en su uso. Sin embargo, la incorporación de estas tecnologías presenta una serie de problemas nuevos que hay que superar, por ejemplo, ¿cómo distribuir automáticamente una pantalla en la que hay que presentar informaciones gráficas y textuales, sin saber de antemano las dimensiones y cantidades de éstas?

2.3 Integridad y corrección de errores

La integridad de la información almacenada en el sistema es de un gran valor para quien la posee. Mientras las BD relacionales ofrecen herramientas con las que el diseñador puede hacer ciertas verificaciones elementales de los datos introducidos (por ejemplo, que los números se encuentren en rangos preestablecidos), las BDI ofrecen herramientas mucho más poderosas para verificar la calidad de los datos almacenados o capturados.

Detección automática de errores

Las BDI pueden detectar inconsistencias en la información más allá de la simple validación de capturas. Es posible determinar qué hacer si una nueva información contradice informaciones ya existentes o bien informaciones implicadas por las existentes.

Casos no típicos

Existen ciertas inconsistencias en la información que pueden ser detectadas por tratarse de casos no usuales, los cuales pueden ser examinados en detalle por el administrador del sistema. Por ejemplo, si en una base con información sobre personas casadas se detecta que la esposa es 40 años mayor que el esposo, esto puede considerarse motivo para notificar el hecho al administrador.

3. LAS BDI COMO UNA EXTENSION DE LAS BD TRADICIONALES

Los trabajos menos revolucionarios encaminados hacia una nueva generación de sistemas de BD tienden a considerar esta última como un perfeccionamiento de las BD tradicionales. Es decir, a las BD tradicionales se les agregan nuevas funcionalidades. En los párrafos siguientes examinaremos algunas de estas funcionalidades.

3.1 Flexibilidad en la identificación de datos

Los sistemas de BD tradicionales son excesivamente rígidos al hacer identificaciones de datos que aparecen en un query contra datos que se encuentran en la base. Esta rigidez es conveniente cuando se comparan claves (como por ejemplo números de cuenta bancaria), pero no es conveniente en muchos otros tipos de datos. Considérese, por ejemplo, que la Sra. Gumersinda Bravo vive en una dirección almacenada en la BD en la forma siguiente:

"Calle de la Angostura No. 234, edif. 5 interior 2"

Suponga que, sin obtener dicha dirección de la BD, se quiere saber quién vive en ese domicilio, para lo cual se forma un query:

extract nombre
from table Direcciones
where

dirección = "Calle Angostura #234, edificio 5-2"

La dificultad de hacer coincidir dos informaciones especificadas como cadenas de caracteres es evidente. Una solución es separar la dirección en varios campos, por ejemplo:

dirección (Nombre Calle, Número, Interior, Subinterior)

Otra solución es utilizar algoritmos de comparación de cadenas de texto; esto es lo que se discute en seguida.

3.1.1 Expresiones regulares

Diversas BD tradicionales ofrecen la posibilidad de usar caracteres "comodines" ("wild-cards") para facilitar la correspondencia con una cadena de caracteres. Por ejemplo, en "Personal File System" se puede hacer la búsqueda de las direcciones que coincidan con:

dirección = "..Angos..."

En esta notación los dos puntos ".." reemplazan una cadena cualquiera, por lo que la dirección va a coincidir con toda dirección que contenga la subcadena "Angos". (Si hay una calle "Angosta" también aparecerá en la respuesta).

Un grado mucho mayor de generalidad en la especificación de cadenas de caracteres es obtenido con el uso de las Expresiones Regulares (ER) [Hopcroft, Ullman 79]. Estas se pueden definir de la manera siguiente:

- Si A y B son ERs, AB denota la concatenación de una cadena de A y una de B, y A + B denota las cadenas de A y también las de B
- Si a es una Er, a* denota la repetición de a cualquier número de veces (incluyendo cero).

Ejemplos:

(a + b)* representa las cadenas formadas por a y por b, como abbab, ba, aaaaabbbba, etc.

(a + b + c)* ción representa las cadenas que empiezan con cualquier número de a, b y c, y que terminan con "ción", como abacabación, ción, acción,etc.

Las ER pueden ser complementadas con caracteres comodines, para facilitar su uso. Por ejemplo , si ? representa cualquier caracter:

?*ción representa las palabras terminadas en ción.

?*ara(z + s) o representa las palabras terminadas con araso o en arazo.

3.1.2 Parecidos sintáticos

Otro recurso para facilitar la correspondencia de cadenas de caracteres es permitir que exista un grado de parecido, en vez de exigir una correspondencia exacta.

Existen diversos algoritmos para detectar coincidencia parcial entre cadenas de caracteres [Knuth 73], pero en la mayoría de casos es necesario adaptar uno al propósito específico de la aplicación.

En los métodos estadísticos, mediante la obtención de índices estadísticos se mide el parecido entre palabras. Diversos índices pueden ser combinados:

- Correspondencia numérica entre posiciones de letras. Por ejemplo, entre las palabras "caba"y "casa" hay una correspondencia perfecta en el 75% de las letras, por lo que podemos asignarles un parecido global de 0.75.
- Otros criterios pueden dar mayor peso al hecho de encontrar secuencias de letras en común más que letras aisladas en común. Por ejemplo, "ahuecar" tendría mayor parecido con "hueco" que con "abuela", a pesar de que ambas tienen 4 letras en común con aquellas.

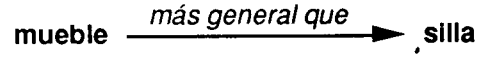
Este tipo de algoritmos es fácil de programar en computadora, pero su implementación debe ser muy eficiente para no deteriorar el tiempo de respuesta del sistema.

3.1.3 Conceptos relacionados (tesauros)

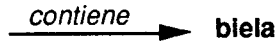
Mientras que la cuestión de parecidos en la forma de las palabras no tiene relación con sus significados, hay sistemas de información avanzados que toman en cuenta la relación entre el significado de las palabras del query con las palabras almacenadas en la BD. Un caso muy importante de facilidad informática para tratar esta cuestión de la relación entre conceptos es el Tesauro.

Un tesauro es una estructura de información que contiene ligas de diversos tipos entre los conceptos que conoce. Una de las ligas más evidentes y más importantes-es la de los sinónimos (casi sinónimos, en la práctica). Así, los términos "estudiante" y " alumno" estarían mutuamente ligados por sinonimía.

Existen otras ligas usuales, tales como la de "más general que", e inversamente "más particular que", como por ejemplo.



Otra liga es "parte de", o inversamente "contiene", como por ejemplo:



Es útil estudiar las propiedades de las diferentes ligas; por ejemplo, la relación de sinonimia es reflexiva, simétrica y transitiva, por lo cual al saber que "puerco" es sinónimo de "cochino", podemos inferir que "cochino" es sinónimo de "puerco" (simetría), o también que si "cochino" es sinónimo de "marrano", entonces "puerço" es sinónimo de "marrano" (transitividad). Similarmente, la propiedad contiene es transitiva y antisimétrica.

Los tesauros son de hecho una implementación informática de las redes semánticas, usadas en Inteligencia Artificial para modelizar relaciones entre conceptos, y así representar el conocimiento.

3.1.4 Estudio de casos

Vamos a presentar ejemplos de sistemas de información, desarrollados en el ITESM, México, que ponen en práctica las ideas arriba expuestas.

Tutor para Macintosh

En el contexto de un contrato del ITESM con Genetec, la empresa que distribuye en México las computadoras Macintosh, se desarrolló un prototipo de ayuda al usuario para la mejor utilización del hardware y software. Dicho sistema, desarrollado en HyperCard, en su etapa inicial se presentaba como una serie de menús descendentes donde el usuario podía navegar hasta localizar los puntos sobre los que requería información.

Sin embargo, el cliente de este proyecto objetó que el recorrido de los menús puede resultar muy engorroso, además de que es necesario pasar por muchos conceptos intermedios que el usuario no necesariamente conoce.

La alternativa a los menús es, evidentemente, la búsqueda por palabras claves que el usuario teclea. Sin embargo, esta posibilidad tiene la desventaja de que el usuario debe teclear literalmente un término que el sistema conozca, y la más mínima diferencia (en terminación, en ortografía, o bien el uso de un sinónimo) hacen que la búsqueda fracase.

Nos propusimos entonces combinar los menús con unos "atajos", bajo la forma de palabras que el usuario teclea, pero de tal forma que el sistema sea inmune a pequeñas diferencias en escritura, así como al uso de sinónimos.

La implementación fue emprendida por dos equipos independientes, que de hecho adoptaron soluciones distintas, tanto para el problema de las diferencias en escritura como para tomar en cuenta los sinónimos., [García, Attias 91], [Marroquín et al. 91]

Tesauro para biblioteca

Otro sistema que ejemplifica la tecnología de los conceptos relacionados es el Tesauro para la biblioteca del ITESM (Monterrey, México) [García det al. 91]. Existe en el campus Monterrey del ITESM una red de servicios computacionales basados en el esquema cliente-servidor, el cual ha sido implementado mediante RPCs. El catálogo de la biblioteca ha sido incorporado a los servicios de la red del campus, y es accesible desde microcomputadoras PS/2. Sin embargo, un problema al que se han enfrentado los usuarios de dicho catálogo es que las consultas son excesivamente rígidas para quien no sabe el título exacto o el autor de determinada obra. La búsqueda por palabras claves ayuda en cierta medida, pero la menor diferencia entre la palabra clave almacenada y la tecleada por el usuario (por ejemplo, "máquinas paralelas" y "computadoras paralelas", respectivamente), da como consecuencia que no se produce la correspondencia deseada.

Como solución a estos problemas se propuso la construcción de un tesauro computarizado. Dicho tesauro se implementó bajo la forma de un conjunto de rutinas compatibles con los formatos de la red de servicios del campus, de modo que pueden ser utilizadas no sólo por el servicio de catálogo de biblioteca, sino por cualquier otro servicio de la red del campus. Se implementaron las relaciones siguientes:

- Sinónimo
- Usado-para (parte-de) /Usa (tiene parte)
- Más-general-que / Más-particular-que

Finalmente existe una relación "conceptos relacionados", que generaliza las otras relaciones. Las pruebas de este sistema se harán en los próximos meses.

3.1.5 Queries inexactos

En muchas ocasiones el usuario de una BD no sabe exactamente qué es lo que quiere. Por ejemplo, es el caso de un cliente tratando de identificar el producto que quiere dentro de un catálogo. En estos casos, se quisiera detectar los registros de la base que mejor corresponden a los criterios proporcionados por el usuario.

Para esto se puede asociar un grado de confianza a la correspondencia entre un query y una respuesta. [Parsaye et al. 90] Es decir, en vez de que la correspondencia se mida con "si" o "no", se asignan números -por ejemplo de 0 a 100. En la medición del grado de correspondencia se hacen intervenir tres factores: (Sea R un registro y C la condición que debe satisfacer) 1) Qué tan bien cada campo de R satisface C; 2) qué tan bien la relación entre los diversos campos es satisfecha en C, y 3) Qué tan importante es cada campo de R para C.

La importancia de cada campo respecto a un querey es especificada por un peso ("weight"), como en el ejemplo siguiente:

select nombre, edad, teléfono
from personal
where

nombre ~ = ~ Juan Pérez, **weight** = 80 and
edad ~ = ~ 18, **weight** = 30;

Los pesos son combinados con el grado de correspondencia de cada campo; en este ejemplo se usa el operador ~ = ~ para indicar la correspondencia aproximada.

Las correspondencias aproximadas a condiciones dadas son estudiadas en los *conjuntos difusos y relaciones difusas*. Estas últimas son caracterizadas por una función que obtiene un grado de correspondencia a partir de los datos. Por ejemplo,para el operador ~=~ Se puede definir una función tabulada como sigue:

Este tipo de tablas constituyen una aproximación polinomial a una función continua; en algunos casos se prefiere trabajar directamente con una fórmula para calcular dicha función continua; una fórmula de este tipo es la siguiente:

$$f(x) = \frac{1}{1 + (x - k)^2}$$

Esta fórmula describe una curva en forma de campana con un máximo alrededor de k.

3.2 Manejo de objetos complejos

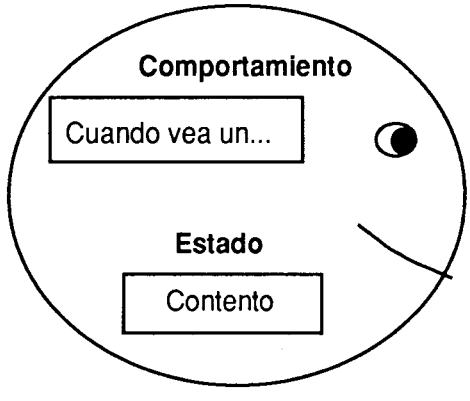
Las BD tradicionales no han podido extender su aplicación al conjunto de usuarios potenciales que requieran almacenar en forma organizada grandes volúmenes de información. Las BD relacionales están bien adaptadas a los sistemas administrativos, pero no lo están tanto a otras áreas. Más aún, el auge de la tecnología relacional ha alejado en vez de acercar la posibilidad de aplicar las BD a ciertas aplicaciones. Un ejemplo típico son los sistemas

n1 - n2	n1 ~ = ~ 2
0	100
5	90
10	60
15	30
20	10
25	5
30	2

CAD (Computer-aided desing). En efecto, en CAD se requiere almacenar entidades altamente estructuradas, con partes que contienen a su vez parte, etc. Aún cuando es posible almacenar la información de partes imbricadas en una relación "parte-de", no es una alternativa atractiva, porque la información se encuentra excesivamente dispersa.

En aplicaciones del tipo CAD se han propuesto BD basadas en el concepto de objeto. Las tecnologías por objetos tuvieron su origen en los lenguajes de programación, donde el ejemplo más típico es Smalltalk. [Goldberg, Robson 83].

La idea esencial de objeto en programación es que en cápsula un área local de memoria de trabajo (también llamada "estado"), una estructura y una serie de mecanismos con los que puede reaccionar a la recepción de mensajes (código de comportamiento):



En lo que se refiere a las BD, lo esencial no es tanto el comportamiento de los objetos, sino la forma en que está encapsulada y estructurada la información. Considérese el siguiente ejemplo:

Togo es fiel;
Togo y Minu se lamen las patas

Una organización relacional agrupa en una misma tabla las entidades que poseen una misma característica; tendríamos las tablas siguientes:

Come ratones
Minú
...

Es fiel
Togo
...

Lame sus patas
Togo
Monou
...

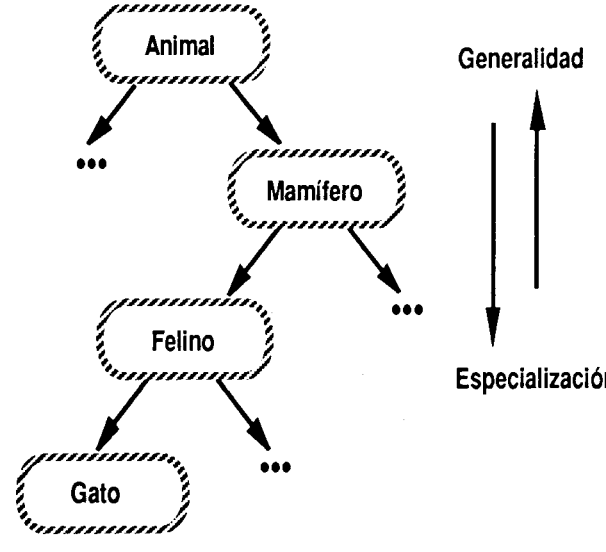
En una organización orientada-objeto, asociamos a los individuos sus características:

Minu:
Come ratones
Lame sus patas

Togo:
Es fiel
Lame sus patas

En los sistemas orientados a objetos, los individuos de características similares son agrupados en clases. Es en las clases donde se definen las características que tienen los individuos que le pertenecen.

Las clases de objetos pueden estructurarse en jerarquías, de acuerdo con su generalidad, donde las clases más generales se encuentran más arriba en la jerarquía, como en el siguiente ejemplo.



Una de las propiedades más interesantes de estas jerarquías es que las subclases (las clases que están abajo de una clase) heredan las propiedades de todas las clases que están arriba en la jerarquía. Por ejemplo, siguiendo la figura de arriba, los felinos heredan las características de los mamíferos (pues la clase de los felinos pertenece a la de los mamíferos). La relación de herencia es transitiva, y así la clase gato hereda las propiedades de mamífero.

Las BD orientadas a objeto no son la única posibilidad para almacenar entidades complejas, y hay otras propuestas igualmente plausibles. Tal es el

caso del modelo de relaciones anidadas, que tiende a ganar aceptación recientemente. El modelo de las relaciones anidadas se base en la idea de que la información almacenada en un campo de un registro puede ser a su vez una relación. En consecuencia, las tablas de relaciones anidadas no cumplen con la primera forma normal de Codd. Como el nivel de anidamiento de las relaciones no está limitada en principio, este modelo se presta a la representación de objetos de una complejidad cualquiera. Por otra parte, el modelo de las relaciones anidadas tiene la ventaja sobre el de objetos de ser más próximo al modelo relacional clásico, tanto conceptualmente como en lo referente a la puesta en práctica del DBMS correspondiente.

4. LAS BDI COMO UN COMPROMISO ENTRE BD Y QA

En los 60 y 70 se dieron en el ámbito de la Inteligencia Artificial una serie de progresos relacionados con las técnicas de deducción automática. [Robinson 65], [Darlington 68], [Reiter 71], etc.

Aún cuando dichas investigaciones estaban enfocadas hacia la demostración automática de teoremas, pronto se puso en evidencia el hecho de que al demostrar teoremas es posible obtener resultados de mayor importancia práctica que la satisfacción matemática de una prueba. Dicho de otra manera, hay muchos problemas de búsqueda de información que pueden ser expresados como un problema de demostración de teoremas.

Así, Cordell Green [Green 69] mostró que es posible utilizar el procedimiento de refutación basado en la resolución de cláusulas para obtener respuesta a preguntas. Veamos un ejemplo (simplificado) de este tipo de respuesta deductiva a preguntas.

Ejemplo.- Se sabe que un perro, Fido, está siempre donde está su amo. Se sabe también que su amo se encuentra en la biblioteca. La pregunta es ¿dónde está Fido? La información disponible debe ser expresada en el lenguaje de la lógica (de predicados de primer orden):

estaEn(X,Y) expresa que X está en el lugar Y

Entonces la información disponible queda como:

(1) estaEn(amo,X) $\Rightarrow$ estaEn(fido,X)
(2) estaEn(amo,biblioteca)

donde el símbolo "$\Rightarrow$ " quiere decir que el lado izquierdo implica el lado derecho ("si el amo está en un lugar X, esto implica que fido también está en ese lugar X"). La respuesta también se expresa en lógica, con el predicado respuesta(X), que significa que X es la respuesta buscada.

(3) estaEn(fido,X) $\Rightarrow$ respuesta(X)

Este ejemplo se puede resolver por la regla del "Modus Ponens"[1], que consiste en que si sabemos que A $\Rightarrow$ B y también sabemos que A es cierto, entonces concluimos que B es cierto. Aplicando esta regla, obtenemos:

(4) estaEn(fido,biblioteca), que viene de (1) y (2), haciendo que la variable X tome el valor de biblioteca.

(5) respuesta(fido,biblioteca), que viene de (3) y (4), haciendo que X tome el valor de biblioteca.

Lo cual es la solución: "Fido está en la biblioteca". Por trivial que pueda parecer este ejemplo, la idea brillante de Green fue de darse cuenta de que la respuesta se obtuvo a partir de la demostración del teorema siguiente:

$\exists$X respuesta(fido,X)

que puede leerse como "existe algún lugar X tal que Fido está en ese lugar"

El método deductivo usado por Green, la resolución de cláusulas, recibió continuas mejoras en esos años, y alcanzó una eficiencia computacional que hizo factible desarrollar sistemas computacionales de respuesta deductiva a preguntas en inglés "deductive Question Answering", abreviado "QA". Sin embargo, la eficiencia de la resolución de cláusulas no fue suficiente para resolver el principal problema que la aquejaba: la combinatoria.

En efecto, existen en cada paso de la deducción muchos pares de fórmulas que podemos combinar[2], y la cantidad de combinaciones crece exponencialmente con la cantidad de fórmulas existentes. En consecuencia, un pequeño aumento en el número de

[1] El método del Modus Ponens no es el seguido por Green.

[2] Esto no resulta evidente en el ejemplo de fido, que es demasiado reducido.

fórmulas se traduce en un gran aumento en la cantidad de combinaciones de fórmulas posibles. Esta limitación impidió que se desarrollaran sistemas de QA del orden de magnitud de las BD, y finalmente aquellos nunca pasaron de ser prototipos de laboratorio.

Sin embargo, los sistemas de QA permitieron desarrollar una serie de técnicas y esclarecer unos principios que luego serían de utilidad para las BDI. En primer lugar, el análisis de las BD tradicionales en tanto que sistemas de QA deductivo, revela varios puntos poco conocidos por la comunidad de las BD. Esto es lo que se toca en el párrafo siguiente:

4.1 Un enfoque lógico del modelo relacional

Al analizar los sistemas de BD desde el enfoque de la lógica, se pone de manifiesto que hay dos puntos de vista posibles para interpretar el resultado de un query: puede ser visto como una interpretación o modelo -en el sentido de la lógica- del query, o bien como una fórmula lógica deducida a partir del query. [Reiter 84] Para empezar damos algunas definiciones:

Interpretación de una fórmula lógica: intuitivamente, una interpretación es el significado de la fórmula; así, por ejemplo, en el predicado estaEn(X,Y) entendemos que X es el sujeto Y es el lugar donde está. Formalmente, una interpretación se define como un mapeo de la manera siguiente:

- Primero debe darse un conjunto llamado dominio (D), al cual van a pertenecer los elementos de la interpretación.

- Se mapea a cada constante k de la fórmula un elemento d_k del dominio D;

  A cada predicado P de la fórmula corresponde una relación R_P, que contiene las tuplas para las cuales P se considera cierto. [3] R_P se conoce también como la extensión de P.

Ejemplo: Sea la fórmula a<b, donde a y b son constantes. Definimos una interpretación por D = {0, 1, 2, 3}, a = 2, b = 3, y R < = {0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)}. En esta interpretación, la fórmula a<b es cierta. En este caso, se dice que la interpretación satisface a la fórmula. En cambio, si hacemos a = 2 y b = 0, la misma fórmula en esa interpretación es falsa.

Modelo: Se llada modelo de una fórmula a una interpretación que hace cierta dicha fórmula. Si una fórmula no tiene modelos, se dice que es una contradicción. Si toda interpretación de una fórmula es un modelo, se dice que esa fórmula es válida o idénticamente cierta.

4.1.1 Interpretación de los queries

Regresemos al problema de los queries en las BD. Supongamos, para simplificar, que tenemos un query de la forma $\sigma\varepsilon$ (R), es decir una selección de la tabla R con una condicción c. Vamos a decir que una relación (tabla) R satisface una condición c cuando ésta se evalúa a cierto para todas las tuplas de R. En consecuencia, el resultado del query $\sigma_\varepsilon(R)$ puede ser caracterizado como una interpretación cierta o modelo de la condición c. En esta óptica, se supone que el dominio de interpretación está formado por los elementos de las tuplas de R. De este modo , una tupla t ∈ R pasa a formar parte del resultado cuando t ∈ Rc.

Este punto de vista en que el resultado es una interpretación lógica del query corresponde al enfoque habitual de la comunidad de las BD. Sin embargo, no es el único posible, y de hecho el otro enfoque tiene ventajas muy interesantes.

4.1.2 Respuesta deductiva a los queries

El punto de vista deductivo de la respuesta a un query de la forma σ_c (R) comienza por ver a las tuplas de la relación como proposiciones de la lógica. Así, por ejemplo suponga que en una tabla SALARIOS hay la siguiente información:

Empleado	Salario
Juan Pérez	1050
Ana Torres	1876
Evaristo Pérez	945
Pedro Cordero	979

[3] Por simplicidad no vamos a considerar la interpretación de las funciones.

Entonces se puede expresar la información de la tabla usando un predicado lógico SALARIO:

SALARIO (Juan Pérez, 1050).
SALARIO (Ana Torres, 1876).
SALARIO (Evaristo Pérez, 945).
SALARIO (Pedro Cordero, 979).

Ahora supongamos un query de la forma: σ salario<1000 (SALARIOS). Desde el punto de vista deductivo, las tuplas elementos de la solución cumpliría un teorema de la forma:

$\exists X, Y$ SALARIO(X,Y) $\wedge$ Y < 1000

Decimos que una prueba de teorema es constructiva cuando no sólo se muestra que existe algún elemento en el universo que satisface la condición requerida, sino que además se muestra de qué elemento se trata. En el caso de los teoremas como el de arriba, nos interesan las pruebas constructivas que nos permitan encontrar los valores de X e Y. Está claro que cada valor de X e Y que satisface el query corresponde a una prueba constructiva. Si queremos todos los valores que satisfacen el query (para reunir todos los elementos de σ salario<1000 (SALARIOS)), entonces es necesario poder efectuar todas las pruebas (exactamente 2) del teorema arriba citado.

La ejecución de muchos queries en las BD relacionales requiere que añadamos hipótesis adicionales cuando trasladamos aquellos a su equivalente lógico. Por ejemplo, supóngase que queremos saber cuántas personas ganan más de 1000. Claramente la respuesta es 2. Sin embargo, estamos suponiendo que Juan Pérez no es la misma persona que Ana Torres. Este supuesto se conoce como la hipótesis del nombre único, y para nuestro ejemplo se puede expresar en lógica como:

Juan Pérez $\neq$ Ana Torres$\wedge$
Juan Pérez $\neq$ Evaristo Pérez $\wedge$
Juan Pérez $\neq$ Pedro Cordero$\wedge$
Ana Torres $\neq$ Evaristo Pérez$\wedge$
Ana Torres $\neq$ Pedro Cordero$\wedge$
Evaristo Pérez $\neq$ Pedro Cordero$\wedge$

El lector puede fácilmente imaginar el tamaño que tendría el axioma del nombre único en una BD de tamaño real.

Otros axiomas que también están implícitos en las BD son la "hipótesis del mundo cerrado" (CWA, por las siglas en inglés) y la "hipótesis de cerradura del dominio". La primera quiere decir que las tuplas que no están en la relación es porque no tienen que estar. Esto se traduce en el ejemplo a decir, por ejemplo,que Evaristo Pérez no gana 3245. Esta afirmación evidente no se encuentra explícitamente declarada. La CWA se puede expresar como:

$(x_1,...,x_n) \notin R \Rightarrow \neg R(x_1,...,x_n)$.

La hipótesis de cerradura del dominio (CD) quiere decir que los elementos de cada campo que hay en la BD son todos los elementos que hay. En el ejemplo, esto querría decir que Juan Pérez, Ana Torres, Evaristo Pérez y Pedro Cordero son los únicos empleados. La CD puede expresarse en nuestro ejemplo mediante el axioma siguiente:

SALARIO(X,Y) $\Rightarrow$ (X=Juan Pérez) v (X = Ana Torres) v (X = Evaristo Pérez) v (X = Pedro Cordero).

La CWA y la CD son indispensables para poder obtener deductivamente la respuesta a queries conteniendo negaciones, como ¿qué empleados no ganan 1000?

El uso de los axiomas de nombre único, CWA y CD, permiten obtener a partir de un proceso deductivo (por demostración de teoremas) la respuesta a cualquier query de la BD. Es más, la demostración de teoremas permite contestar preguntas mucho más complejas que lo que puede hacer en el cuadro del álgebra relacional. Esto se debe a que en lógica se puede representar más información que en una tabla relacional. Por ejemplo, supóngase que se sabe que hay una empleada que gana más de 2000, pero no se sabe si es Adriana Rivas o Martha García (una de las dos). Esto no se puede almacenar en una relación. En cambio, en lógica se puede escribir:

[SALARIO(AdrianaRivas,X)v SALARIO (Martha-García, X)] X >2000.

Dicha información puede ser utilizada para contestar a queries tales como ¿cuántos empleados ganan más de 2000?

4.2 Inferencia limitada: un compromiso

Como conclusión del párrafo anterior podríamos decir que la respuesta deductiva a los queries es más flexible y poderosa que la respuesta usual de las BD, basada esta última en la noción de interpretación. Sin embargo, el problema con los procesos

deductivos es que consumen demasiados recursos computacionales, debido a la combinatoria de que adolecen.

Se han propuesto varias soluciones a esta situación, con la idea de encontrar un compromiso entre la eficiencia de las BD y la flexibilidad de los sistemas deductivos [Reiter 78], [Ceri et al. 89]. La idea básica es siempre la de limitar las deducciones que pueden hacerse. Lo que varía en los diversos métodos es la clase de limitación que se impone al proceso deductivo. Estos sistemas intermedios entre las BD y los sistemas deductivos que QA han sido llamados "Bases de Datos Deductivas" [4]

Una de las limitaciones que se han impuesto a la deducción en las BDD está en el lenguaje de la lógica: se ha tomado una parte solamente de la lógica de predicados a saber, las llamadas "cláusulas de Horn", que no permiten expresar cualquier frase de la lógica, pero que son computacionalmente más simples de tratar. Las cláusulas de Horn (CH) son la base del lenguaje de programación Prolog [Colemrauer et al. 79], y dan lugar a toda la tecnología que relaciona Prolog con las BD. Este enfoque será discutido en la sección 5.

Otra forma de limitar la deducción consiste en distinguir el carácter de cada fórmula lógica, y tratar a cada clase en una forma apropiada. Así, se distinguen tres tipos de informaciones:

i) Informaciones factuales, que declaran un hecho elemental que se considera conocido. Ejemplo: salario (Evaristo Pérez, 945).

ii) Restricciones de integridad, que se usan solamente para garantizar la calidad de los datos contenidos en la base. Ejemplo: SALARIO (X,Y) $\Rightarrow$ Y>500 sirve para invalidar las tuplas en que un empleado gane 500 o menos -se supone que el salario mínimo es de 500.

iii) Leyes generativas, que se usan para producir nuevas informaciones que son consecuencias lógicas de los datos conocidos. Ejemplo: SALARIO(X,Y) $\wedge$ Y > 2000 $\Rightarrow$ rico (X); esta fórmula nos permite obtener los empleados que son ricos.

Las fórmulas de tipo i son las que se almacenan realmente en la BD. A la hora de efectuar procesos deductivos, los hechos nunca se hacen interactuar entre sí, y sólo se ponen en combinación con las fórmulas de tipo ii o iii. Esto reduce enormemente la cantidad de combinaciones de fórmulas que se forman al efectuar deducciones, haciendo el proceso mucho más eficiente.

La distinción entre ii y iii no obedece al tipo de fórmulas,. sino a la forma en que se pretende usarlas. Al diferenciar ii y iii, en el momento en que vamos a verificar integridad haremos caso de ii y no de iii, mientras que cuando vamos a obtener la respuesta a un query, tomaremos en cuenta iii pero no ii. Esto permite reducir aún más la cantidad de fórmulas que intervienen en la deducción, disminuyendo por ende la combinatoria.

La forma de poner en práctica estos principios será discutida en los párrafos siguientes.

5. LA TECNOLOGIA BD-PROLOG

> It is worth observing... that logic programming shows
> how to extend any query language to a programming
> language [Kowalski 78].

Uno de los compromisos más interesantes entre las BD y los sistemas deductivos de QA tiene como base el lenguaje de programación Prolog, desarrollado primero en Marsella por A. Colmerauer [Colmerauer et al. 79] y luego en Edimburgo por D. Warren, a partir de los principios delineados en [Kowalski 74].

5.1 La programación lógica

Prolog (PROgramación en LOGica) es un lenguaje de programación que permite expresar de una manera "declarativa" el conocimiento sobre un problema, en un lenguaje cercano a la lógica, y la parte "acción" es tomada a su cargo por el intérprete Prolog, pero en principio el usuario no se tiene que preocupar por este aspecto.[5]

---

[4] Las BDD son el corazón de este curso; nosotros hemos considerado que BDI es un término más general que BDD en el sentido de que la deducción no es la única característica que hace "Inteligentes" a las BDI... También están la identificación flexible de datos, el uso de multimedios, objetos complejos, etc.

[5] Todo programador en Prolog se dará cuenta de cuán lejana está la realidad de esto ideales!

Aquí no estamos interesados en Prolog en tanto que lenguaje de programación sino en tanto que lenguaje para cumplir las funciones de un sistema de BD, a saber:

(1) Representar los datos y estructurarlos
(2) Expresar queries y resolverlos.

En cuanto a 1, Prolog permite expresar información organizada en forma relacional (conjuntos de tuplas) exactamente como en la lógica: [6] Un programa Prolog con la información de la relación de salarios sería como sigue:

```
salario(JuanPérez, 1050).
salario(AnaTorres, 1876).
salario(EvaristoPérez, 945).
salario(PedroCordero, 979).
```

Como el nombre de la relación acompaña al dato para cada tupla, no es necesario reservar áreas especializadas para cada relación. Así, si otra relación POSICION(EMPLEADO, PUESTO) va a definirse, sus datos pueden situarse en el mismo programa que la información anterior:

```
salario(JuanPérez, 1050).
salario(AnaTorres, 1876).
posicion(AnaTorres,secre).
posicion(EvaristoPérez,sindico)
salario(EvaristoPérez, 945).
salario(PedroCordero, 979).
posicion(PedroCordero, labo).
```

Como se puede observar, las relaciones pueden estar "mezcladas" en el programa; el sistema Prolog se encarga de reservar internamente almacenamiento separado a cada relación.

Los queries en Prolog se hacen utilizando el nombre de la relación, y dejando variables en el lugar del que queremos información. Por ejemplo, para saber el sueldo de Ana Torres se haría la pregunta:

```
salario(AnaTorres, x)?
```

Y el sistema respondería con: X = 1876.

Esta pregunta correspondería a un query relacional σ (1=AnaTorres)(SALARIOS) [7]

Para preguntas simples como ésta, el sistema Prolog se limita a buscar una tupla que se unifique con el query, donde unificar quiere decir reemplazar las variables por valores, de modo que las dos partes consideradas queden idénticas. La respuesta del sistema es precisamente la substitución o reemplazamiento que resulta de dicha unificación.

Se pueden hacer queries más complejos, que involucren una combinación de varias condiciones, como por ejemplo: ¿qué empleados ganan más de 1000?

```
salario(X,Y), Y>1000?
```

Se pueden igualmente hacer preguntas que involucren varias relaciones: ¿cuáles son los sueldos de las secretarias?

```
salario(X,Y),posicion(X,secre)?
```

El equivalente en álgebra relacional necesita de operaciones de "joint":

$$SALARIO[X](1=1)\sigma(2=secre)POSICION$$

Negación como fallo

La implementación del operador relacional de diferencia en Prolog necesita de la negación de algunas de las condiciones que aparecen en el query. Por ejemplo, si se quiere obtener los nombres y sueldos de los empleados con salario pero sin posición, tendríamos:

```
salario(X,Y), not posicion(X,Z)?
```

En el álgebra relacional esto se puede expresar como:

$$\Pi_1 SALARIO - \Pi_1 POSICION$$

El operador de negación "not" de Prolog debe tratarse con precaución, pues no corresponde al conectivo lógico $\neg$, que se define por $\neg$cierto = falso, $\neg$falso = cierto. En efecto , el operador not se evalúa a cierto cuando el sistema trata de hacer cierto el predicado al que se aplica (posicion(X,Z) en el ejemplo), y al haber examinado todas las maneras posibles concluye que no se puede. Por esto se dice que en Prolog la negación se define como fallo del sistema para obtener el valor cierto: "Si no pude mostrar que es cierto, entonces es falso", tal es la filosofía de Prolog.

---

[6] Aquí no definiremos la sintaxis exacta de Prolog, no siendo la intención del curso.

[7] No estamos tomando en cuenta el paso de proyección del resultado.

En la práctica hay muchas maneras de expresar queries equivalentes a una diferencia de relaciones, sin para ello utilizar explícitamente la negación. Por ejemplo, para obtener los empleados que no ganen más de 1500, podemos cambiar la negación de Y>1500 por Y≤1500, suponiendo que el operador ≤ está disponible en el Prolog en que se trabaja:

salario(X,Y), Y≤1500?

Definición de reglas en Prolog

Además de definir relaciones explícitamente, enumerando sus tuplas, en Prolog es posible definir nuevas relaciones en forma implícita, a partir de otras relaciones. Esto se hace por medio de las reglas. Una regla es de la forma:

cabeza(Argumento):-cuerpo$_1$ (Arg$_1$),...,cuerpo$_n$ (Arg$_n$).

donde la parte a la izquierda de :- es la cabeza de la cláusula, y la parte a la derecha es su cuerpo. El predicado que se define es el que aparece en la cabeza. En el cuerpo puede aparecer cualquier relación, incluyendo a la que se define en la cabeza, caso en el cual se trata de una definición recursiva.

Desde un punto de vista lógico, una regla Prolog expresa el hecho de que juntas las proposiciones contenidas en el cuerpo de la regla implican la cabeza de la regla.

Las definiciones no-recursivas de relaciones son de hecho abreviaturas de queries complejos, además de que implementan el equivalente a la proyección del álgebra relacional en el sentido de que permiten restringir la respuesta del sistema a las variables que aparecen en la cabeza. Por ejemplo, si hacemos el query:

salario(X,Y),Y≤1500?

la respuesta del sistema será (siguiendo con nuestro ejemplo):

X = JuanPérez, Y= 1050;
X = EvaristoPérez, Y= 945;
X = PedroCordero, Y = 979;
no more.

En cambio, si definimos una relación hasta 1500, como:

hasta 1500(X) :-salario(X,Y), Y≤ 1500.

entonces al hacer el query hasta 1500 (X)? obtendremos únicamente el nombre del empleado:

X =JuanPérez, etc.

Las reglas recursivas permiten expresar muchos queries que no se pueden expresar directamente en el álgebra relacional, como por ejemplo la cerradura transitiva de una relación. Suponga que tenemos almacenadas informaciones indicando quién es jefe de quién en la empresa:

jefe(JuanPérez, EvaristoPérez).
jefe(AnaTorres, EvaristoPérez).
jefe(EvaristoPérez, PedroCordero).

Si queremos saber quiénes son todos los subordinados de una persona, habría que ver quiénes son sus subordinados inmediatos, luego quiénes son los subordinados de sus subordinados, etc. Esto se puede expresar en forma simple mediante una regla recursiva:

subordinado(X,Y) :- jefe(Y,X).
subordinado(X,Y) :- jefe(Y,Z), subordinado(X,Z).

Para saber quiénes son los subordinados de juanPérez se haría el query siguiente:

subordinado(X,=JuanPérez)?

(El sistema respondería con X = EvaristoPérez; X = PedroCordero;).

Al hacer un query recursivo en Prolog hay que tener cuidado con ciertas sutilezas resultantes del mecanismo de ejecución de Prolog; de otro modo es posible caer en ciclos infinitos o bien en ejecuciones muy ineficientes. Esto lo discutiremos más adelante.

5.2 Tratamiento uniforme datos-programa

Como se puede observar de lo arriba expuesto, en Prolog no hay distinción formal entre datos y programa: [Kowalski 78] los hechos y las reglas son ambos simplemente cláusulas de Horn, que se pueden entremezclar en cualquier orden. Las relaciones se pueden definir en extensión (dando explícitamente sus tuplas) o en intención, mediante el uso de reglas, o mediante una combinación de ambas. Esta última posibilidad facilita la definición de relaciones que contienen una regla general con excepciones. Suponga, por ejemplo que todos los empleados que ganan menos de 1500 cobran su

sueldo en caja, mientras que para los que ganan 1500 o más hay que especificar el lugar. La relación lugar Cobro se definiría de la forma siguiente:

```
lugarCobro(X,caja) :-salario(X,Z), Z<1500.
lugarCobro(AnaTorres,tesorería).
```

5.3 Acoplamiento BD-Prolog

El lenguaje Prolog es muy eficaz para implementar en poco tiempo un prototipo de un sistema relacional; en unos cuantos minutos es posible definir un sistema simple y hasta ejecutar queries. Sin embargo, al aumentar la cantidad de datos, la velocidad de ejecución de los queries puede degradarse, o los datos pueden ocupar más memoria de la disponible en la computadora. En estos casos, es necesario separar los hechos de las reglas, guardando los primeros, que son quienes constituyen la casi totalidad del volumen de información, en una BD externa al programa Prolog.

Acceso a BD externas desde Prolog

La mayoría de los sistemas avanzados de programación en Prolog (LPA, Quintus, etc.) ofrecen la posibilidad de accesar BD externas, en particular del tipo DBase u Oracle. En estos casos el sistema Prolog es quien mantiene el control, y hace llamados en el momento necesario a la BD. El manejo de estas llamadas puede hacerse completamente transparente al usuario, quien ve estos llamados como referencias a una relación que no está definida en el programa. Supóngase el ejemplo siguiente:

```
external salario(-,-):"HD:FoxBasef:Salarios".
rico(X) :- salario(X,Y), Y>1500.
salario(PepeHerrera,769).
salario(AnaTorres,1789).
```

Primero se declara la relación salario como externa (en BD). El programa en sí contiene la regla para definir la relación rico, y contiene además dos hechos relativos a la misma relación que está en la BD. Según el sistema Prolog del que se trate, esto puede ser válido o no. En este ejemplo supondremos que la política del intérprete es de buscar al final en la BD, como si estuviera escrita hasta abajo del programa Prolog. Entonces, si preguntamos un query rico (X)?, la primera respuesta será X=AnaTorres, que está en el programa, y después iniciará el recorrido de la BD para encontrar las demás respuestas.

Desde el punto de vista del mantenimiento del sistema, puede ser preferible mantener la masa de datos en una BD externa, para que las modificaciones en los datos (por ejemplo, aumentos de sueldos) no afecten al texto del programa Prolog, y se reduzcan a actualizaciones de la BD.

Sin embargo, una desventaja de los sistemas Prolog con acceso a BD externa es que, por la forma en que se ejecutan los queries en Prolog, la información de la BD es extraída registro por registro (en vez de hacer accesos a conjuntos de datos), y esto puede hacer el sistema excesivamente lento para muchas aplicaciones [Ceri et al. 89].

Prolog desde BD: Inference Manager

Aún cuando lo más usual es accesar desde Prolog una BD externa, también es posible hacer lo contrario: desde la BD hacer llamados a Prolog. El Inference Manager (IM), desarrollado por Apple Computer, ofrece una máquina virtual Prolog pequeña (como 50KB) pero muy rápida (30000 inferencias lógicas por segundo en una Mac II) bajo la forma de un comando externo de tipo HyperCara® o bien como una biblioteca de rutinas MPW, que también se puede llamar desde una BD tal que 4th Dimension®. Estamos suponiendo que la BD tiene un lenguaje procedural que le permite mantener el control del sistema en su conjunto.

Las llamadas al IM siguiendo el formato de comandos externos de HyperCard es la siguiente:

infMgr <num> [<parametros>]

donde <num> es un dígito que expresa qué operación efectuar. Por ejemplo, infMgr 0 inicializa el IM. Para cargar una cláusula en IM, se usa el parámetro 4:

infMgr 4, "salario(socratesRizzo,8650)"

Las reglas se introducen en forma similar. Para los queries se tiene el código 6:

infMgr 6, "salario(socratesRizzo,?X)"

(Obsérvese que existen pequeñas diferencias sintácticas entre el Prolog estándar y el de IM).

5.4 Datalog

Como se mencionó antes, el acceso a BD externas desde Prolog tiende a ser ineficiente

porque la base se accesa registro por registro, desaprovechando así las operaciones con conjuntos de datos que ya han sido implementadas en los sistemas relacionales. Varias soluciones han sido propuestas para integrar en forma más efectiva las potencialidades deductivas de Prolog con las BD. Uno de los enfoques más exitosos está constituido por el lenguaje Datalog, [Ceri et al. 89] que ha sido objeto de varios prototipos experimentales. Dados los niveles de eficiencia alcanzados en dichos prototipos, podría esperarse que en muy poco tiempo haya sistemas Datalog disponibles comercialmente.

Sintaxis de Datalog

Las definiciones de relaciones en Datalog son enteramente similares a las de Prolog, pero se añaden ciertas restricciones para garantizar que el conjunto de datos que pueden deducirse de un programa Datalog es finito.

El conjunto de cláusulas en Datalog se divide en dos grupos disjuntos: la BD en extensión (EDB), que está físicamente almacenada en una BD externa, y la BD en intención (IDB), que es propiamente el programa Datalog.

Los queries se escriben en la misma forma que en Prolog, pero su ejecución es muy distinta.

Traducción al Algebra Relacional

Con objeto de aprovechar la eficiencia de los sistemas relacionales para efectuar operaciones con conjuntos de datos, en Datalog se decidió hacer una traducción de un query de Datalog a una expresión del álgebra relacional. La traducción se hace en varias etapas, que serán explicadas con ayuda de un ejemplo.

Ejemplo.- Las personas importantes en la empresa son jefes de departamentos y los gerentes, siempre y cuando ganen más de 2000. Se desea saber quiénes son las personas importantes de la empresa. Se propone el siguiente programa Datalog:

```
importante(X):-salario(X,Y), Y>2000, posicion(X,
jefeDepto).
importante(X):-salario(X,Y),  Y>2000,posicion
(X,gerente).
```

La traducción del query importante(X)? al álgebra relacional se hace en los siguientes pasos:

1) Las cláusulas en que aparecen varias relaciones binarias, ternarias, etc. con variables comunes se identifican con operaciones de "join", mientras que los predicados con una sola variable se identifican con operaciones de selección:

Expr. Datalog	Expr. Alg. Rel.
Salario(X,Y), Y>2000	$\sigma 2>2000$ (SALARIO)
Salario(X,Y), Y>2000, posición(X,jefeDepto)	$\sigma 2>2000$ (SALARIO) $\|X\|(1=1)$ $\sigma 2=$jefe Depto(POSICION)

2) Se forman relaciones de inclusión, en donde las expresiones del paso 1 están a la izquierda del símbolo "$\subseteq$" y la relación de la cabeza de las cláusulas está a la derecha del " $\subseteq$ ", como sigue:

$\sigma 2>2000$(SALARIO) $|X|_{(1=1)}$ σ_2 = jefe Depto (POSICION) $\subseteq$ IMPORTANTE

$\sigma 2>2000$(SALARIO) $|X|_{(1=1)}$ σ_2 = gerente (POSICION) $\subseteq$ IMPORTANTE

3) A partir de estas relaciones de inclusión se forma una ecuación que define el predicado del query como la unión de varias expresiones:

IMPORTANTE = σ_2 >2000(SALARIO) $|\times|_{(1=1)}$ σ_2 = jefeDepto(POSICION) $\cup$

σ_2 >2000(SALARIO) $|\times|_{(1=1)}$ σ_2 = gerente (POSICION)

En este ejemplo, la ecuación es suficiente para producir un query relacional directamente ejecutable. En el caso de los queries recursivos, sin embargo, es necesario resolver de alguna manera laecuación resultante.

Comparación Prolog / Datalog

A pesar de que el aspecto de un programa Prolog se parece mucho al de un programa Datalog, existen muchas diferencias, que discutiremos aspecto por aspecto.

a) Adecuación a BD externas

El sistema Prolog soluciona el problema del orden de ejecución de las cláusulas siguiendo un recorrido de cláusulas "en profundidad" (depth-first), con regreso hacia atrás (backtracking) y negación como fallo.[8] En el contexto de las BD, esto se traduce en un acceso a los registros individuales, lo cual puede ser muy ineficiente, en comparación con los accesos a conjuntos de tuplas que pueden efectuarse en Datalog, dada la traducción que hace el algebra relacional.

b) Orden de cláusulas y predicados.

Como consecuencia del orden de ejecución de Prolog, la eficiencia o incluso la terminación de los programas puede ser comprometida por un cambio en el orden de las cláusulas o de los predicados dentro de una cláusula. Por ejemplo, haciendo una pequeña modificación al programa para obtener los subalternos de un jefe, tenemos:

```
subordinado(X,Y):-subordinado(X,Z),jefe
(Y,Z).

subordinado(X,Y):- jefe(Y,X).
```

Ahora bien, este programa cae en un ciclo infinito debido a que la meta subordinado(X,Y) genera inmediatamente una submeta subordinado(X,Z), y ésta otra similar, etc.

c) Semántica

Mientras que el significado de un programa Prolog se refiere al conjunto de metas elementales que es posible satisfacer, en Datalog la semántica de un programa se define como un mapeo entre el contenido de la BD y los queries que son respondidos positivamente. Es decir, en Datalog el significado del programa se hace dependiente del contenido de la BD. De esta manera se toma en cuenta el hecho de que la BD puede cambiar frecuentemente por actualización de los datos.

[8] Aquí no pretendemos una introducción al lenguaje Prolog; remitimos al lector a las referencias.

6. BDI COMO CONFLUENCIA DE BD Y SISTEMAS EXPERTOS

A pesar de que las comunidades de BD y de Inteligencia Artificial se han mantenido bastante distantes[9], las necesidades prácticas han forzado una confluencia. Por ejemplo, casi todos los "shells" comerciales de Sistemas Expertos ofrecen acceso a una BD externa. Sin embargo, las facilidades de BD ofrecidas a ese nivel son muy precarias, y aún queda mucho por avanzar en este camino.

En los párrafos siguientes daremos una visión lo más imparcial posible[10] de esta confluencia BD-IA.

6.1 Datos y conocimiento

Las BD manejan datos, mientras que los sistemas expertos (ES) de la IA manejan conocimiento. Planteada de esta manera tan simple, esta frase -muchas veces citada- no tiene mucho sentido. No existe ninguna diferencia formal entre datos y conocimiento -ambos son información; más bien se trata de una cuestión de punto de vista. Mientras que los datos están destinados a ser copiados, transferidos, sumados, etc., los conocimientos tienen por objeto servir de representación del mundo a un agente inteligente con capacidad de inferencia. Este es el punto de vista de la IA, el cual revisamos brevemente a continuación.

IA: conocimiento y deducción

En IA se considera que el conocimiento es la materia prima de la "inteligencia" .[11] El conocimiento trata de representar aspectos de la realidad, cosa que hacen también los datos, pero a diferencia de estos, los conocimientos están generalmente estructurados en forma compleja y poco repetitiva, y pueden incluir leyes generales además de las informaciones particulares.

Mientras que para la definición de datos cada paradigma de BD incluye su Lenguaje de Definición de Datos (DDL), en IA el equivalente es un formalismo para representación del conocimiento.

[9] La gente de BD cree que los de IA son charlatantes, mientras que los de IA creen que los de BD no pasan de sus archivos y tablas.

[10] Aún aceptando que el autor de estas notas proviene del área de IA...

[11] Más formalmente, de la deducción.

6.2 Las KB

El conocimiento de los sistemas de IA -usualmente SE- es almacenado en las llamadas Bases de Conocimiento (KB en inglés), siguiendo un formato que corresponde a alguna representación del conocimiento. Una de las representaciones más usuales es la de las reglas de producción, que son pares de la forma:

si<CONDICION>entonces<ACCION>

Esto significa que si se cumple la condición lógica <CONDICION>, entonces será ejecutada la <ACCION>[12]

Las KB de los SE tienen usualmente varios cientos de reglas, por lo que se hace necesario un sistema que administre su acceso y mantenimiento; esto es, un KBMS -el equivalente al DBMS en BD. Los problemas que debe resolver una KBMS son esencialmente los mismos que los de una DBMS, y esto es un factor de confluencias entre BD y KB.

6.3 Ligas entre KB y BD

Usualmente los SE hacen acceso a datos dispersos. Sin embargo, cuando se presenta el problema de manejar volúmenes masivos de datos, usualmente se emplean BD externas al SE, o inclusive hojas de cálculo del tipo Multiplan.

Por ejemplo, en el shell de SE NExpert-Object, un query tiene la forma:

retrieve <DBfile> <parámetros>

y tiene el efecto de leer un registro de la BD y asignar los valores a los atributos de un objeto de NExpert.

Como anteriormente habíamos mencionado, este tipo de liga BD-SE es relativamente ineficiente, pues accesa los registros uno por uno, además de que el shell no dispone de las funcionalidades completas de un DBMS. Este tipo de ligas BD-SE son ofrecidas bajo el supuesto de que el acceso a los datos es más bien excepcional (por ejemplo, cargar parámetros de operación de un SE al iniciar). Sin embargo, para aplicaciones que combinan un manejo intensivo tanto de datos como de conocimientos, es necesario encontrar una forma de integrar el KBMS y el DBMS en un solo DKBMS.

[12] Por razones de espacio, no podemos hacer una exposición introductoria a los SE; remitimos al lector a las referencias.

REFERENCIAS

S. CERI, G. GOTTLOB, L. TANCA. What you always wanted to know about Datalog (and never dared to ask). Knowledge and Data Engineering, vol. 1 no. 1 IEEE, 1989.

A. COLMERAUER, H. KANOUI, M. VAN CANEGHEM. Etude et réalisation d'un systeme Prolog. Reporte Univ. de Aix-Marseille II. 1979.

J. DARLINGTON. Automatic theorem proving with equality substitutions and mathematical induction. Machive Intelligence Vol. 3. 1968.

M. GARCIA, A, RIVAS, A. GONZALEZ, G. OLAVARRIETA. Thesaurus para catálogo de biblioteca. Monterrey. ITESM, 1991.

T. GARCIA, D. ATTIAS. Reporte del Tutor con sinónimos y parecidos, Monterrey. ITESM,1991.

A. GOLDBERG, D. ROBSON. Smalltalk-80: the language and its implementation. Reading, MS; Addison Wesley 1983.

C. GREEN. Theorem proving by resolution as a basis for question-answering systems. Machive Intelligence, Vol. 4, 1969.

J. HOPCROFT, J. ULLMAN. Introduction to Automata theory, Languages and Computation. Reading, MS: Addison Wesley, 1979.

D. KNUTH. The art of computer programming. Vol. 3. Reading, MS: Reading Addison Wesley 1973.

R. KOWALSKI. Predicate Logic as a Programming Language. IFIP world Congress, 1974.

R. KOWALSKI. Logic for Data Description. en Gallaire, Minker (eds). Logic and Databases. New York: Plenum Press, 1978.

M. MARROQUIN, H. MENDEZ, M. HERRERA, H. BRITO. Genesaurus.Monterrey. ITESM, México, 1991.

K. PARSAYE, M. CHIGNELL, S. KHOSHAFIAN, H. WONG. Intelligent Databases. AI Expert. March 1990.

R. REITER. Two results on ordering for resolution with merging and linear format. JACM 18 No. 4, 1971.

R. REITER. Deductive Question-Answering on Relational Databases. en Gallaire, Minker (eds). Logic and Databases. New York: Plenum Press, 1978.

R. REITER 84. Towards a Logical reconstruction of Relational Database Theory. En Brodie, Mylopulos, Schnmidt (eds.). On conceptual modelling: perspectives from Artificial Intelligence, Databases and Programming Languages. New York: Springer Verlag,1984.

J. ROBINSON. A machine-oriented logic based on the resolution principle. JACM 12 No. 1, 1965.