

CÁLCULO RELACIONAL

Francisco Ruiz González

Dep. de Informática,

Escuela Superior de Informática

Univ. de Castilla-La Mancha

INTRODUCCIÓN:

A continuación presentamos una introducción al cálculo relacional (CR), herramienta formal utilizada como lenguaje de consulta a bases de datos relacionales (BDR). Frente al álgebra relacional (AR) que provee de una colección de operadores que actúan sobre relaciones para obtener otras relaciones, el CR formula la relación resultante en términos de las relaciones originales. Es decir, mientras el AR es procedural (se indica un procedimiento para resolver el problema), el CR es descriptivo (se indica cuál es el problema y no cómo resolverlo).

El CR está basado en una rama de la lógica matemática, llamada *Lógica de Predicados*, ó *Cálculo de predicados de primer orden*. Es por ello, que al principio haremos una breve exposición de los fundamentos de dicha lógica de predicados.

La primera propuesta de usar la lógica de predicados (LP) en bases de datos relacionales la hizo Codd en [CODD72].

Según el tipo de variables que se manejan, existen dos tipos de CR. El cálculo relacional de tuplas (CRT) emplea variables-tupla, que designan a tuplas de relaciones. En el cálculo relacional de dominios (CRD) se utilizan variables-dominio, que toman valores de los dominios asociados a los atributos de las relaciones. Estudiaremos ambos tipos de lenguajes y los compararemos entre sí, y con el AR.

Por último, presentaremos dos lenguajes comerciales fundamentados en dichos tipos de CR, el lenguaje SQL que utiliza aspectos del CRT (Aunque está basado principalmente en el AR), y el lenguaje QBE basado en el CRD.

1 Lógica de Predicados de Primer Orden

En este primer apartado vamos a repasar los conceptos fundamentales de la lógica de predicados de primer orden. Fundamentalmente nos centraremos en los aspectos sintácticos concernientes con las fórmulas bien formadas (expresiones correctas) del lenguaje. También se comentarán más brevemente los aspectos semánticos, referidos al significado de dichas fórmulas y de los símbolos que las forman. Introduciremos los conceptos de interpretación, evaluación y modelo.

1.1 Símbolos del Lenguaje

Los símbolos básicos del lenguaje son:

- símbolos de puntuación: paréntesis "(", ")" y coma ",", "
- símbolos de variables: se representan mediante letras minúsculas del final del alfabeto: r, s, t, u, v, w, x, y, z.
- símbolos de constantes: se representan mediante letras minúsculas del principio del alfabeto: a, b, c, d, e.
- símbolos de funciones: también se representan con letras minúsculas, pero del centro del alfabeto: f, g, h, i ..
- símbolos de predicados: se representan mediante letras mayúsculas.
- operadores lógicos: \neg (negación), \rightarrow (implicación), \wedge (conjunción) y \vee (disyunción).
- cuantificadores: \forall (cuantificador universal) y \exists (cuantificador existencial).

La jerarquía en la precedencia de los operadores y cuantificadores es (de mayor a menor):

- 1) \neg, \forall, \exists 2) \vee 3) \wedge 4) \rightarrow

Dicha precedencia se puede alterar utilizando los paréntesis.

Los operadores \rightarrow y \wedge y el cuantificador \forall se introducen en el lenguaje por conveniencia, ya que en realidad no son necesarios pues:

$$F \rightarrow G \equiv (\neg F) \vee G$$

$$\forall x(F) \equiv \neg \exists x(\neg F)$$

$$F \wedge G \equiv \neg(\neg F \vee \neg G)$$

1.2 Fórmulas.

Vamos a definir las fórmulas (expresiones) correctas del lenguaje.

1.2.1 Términos.

Empezamos definiendo lo que son los términos:

- i) Los símbolos de constantes y variables son términos.
- ii) Si f es un símbolo de una función n -aria y $t_1, t_2 \dots t_n$ son términos, entonces $f(t_1, t_2 \dots t_n)$ es también un término.

Ejemplos:

c (constante)
 x (variable)
 $f(c_1, x_1, c_2, x_2, x_3)$
 $f(c, g(x, y, z), d, x, z)$

1.2.2 Fórmulas atómicas.

Si P es un símbolo de predicado n -ario y $t_1, t_2 \dots t_n$ son términos, entonces $P(t_1, t_2 \dots t_n)$ es una fórmula atómica, también llamada átomo.

Ejemplos:

Padre(juan, ana) $C(f(x, y), a, z)$

1.2.3 Fórmulas bien formadas.

De todas las expresiones o fórmulas que podemos construir empleando los símbolos de este lenguaje, solamente son correctas las que se construyen siguiendo las siguientes reglas. Estas fórmulas se denominan fórmulas bien formadas (fbf):

- i) Una fórmula atómica es una fbf.
- ii) Si F_1 y F_2 son fbf's, entonces también lo son:

$$F_1 \wedge F_2$$

$$F_1 \vee F_2$$

$$F_1 \rightarrow F_2 \text{ y}$$

$$\neg F_1$$

iii) Si F es una fbf, entonces también lo son:

$$\forall x F \text{ y}$$

$$\exists x F$$

iv) Si F es una fbf, también lo es (F) .

v) Nada más es una fórmula bien formada.

Ejemplos:

$$\forall x (\exists y (P(x,y) \rightarrow Q(x)))$$

$$\neg (\exists x (P(x,a) \wedge Q(f(x))))$$

$$\forall x ((Q(x) \wedge \neg R(x)) \rightarrow P(x,g(x)))$$

La semántica informal de estas fórmulas viene dada directamente, por ejemplo, la última fórmula significa que: *para todo x , si $Q(x)$ es cierto y $R(x)$ es falso, entonces $P(x,g(x))$ es cierto.*

Podemos añadir tres reglas más que nos permiten simplificar la escritura de fbf:

vi) $\neg(F_1 \rightarrow \neg F_2)$ es equivalente a $F_1 \wedge F_2$.

vii) $\neg F_1 \rightarrow F_2$ es equivalente a $F_1 \vee F_2$.

viii) $\neg \forall x \neg F$ es equivalente a $\exists x F$.

Ejemplos:

$$\neg (P(x,y) \rightarrow \neg R(z)) \equiv P(x,y) \wedge R(z)$$

$$\neg P(x,g(x)) \rightarrow Q(x,z) \equiv P(x,g(x)) \vee Q(x,z)$$

$$\neg (\forall x (\neg P(a,f(x,y),z))) \equiv \exists x P(a,f(x,y),z)$$

1.2.4 Forma normal prenexa.

Se dice que una fbf está escrita en forma normal prenexa si tiene el siguiente formato:

$$Q_1 x_1 \dots Q_n x_n M$$

donde los Q_i son cuantificadores sobre las variables x_i y M es una fbf que no contiene cuantificadores.

Ejemplos:

$$\forall y \exists x (P(x,f(a,x),b,y) \rightarrow Q(a,b,z))$$

$$\forall x \forall y \forall z ((\text{Padre}(x,y) \wedge \text{Padre}(y,z)) \rightarrow \text{Abuelo}(x,z))$$

1.3 Variables.

Como se puede observar por su definición, en las fórmulas aparecen variables. A cada una de estas apariciones se les denomina *ocurrencia de una variable*. En una misma fórmula puede haber más de una ocurrencia de una misma variable o de distintas.

1.3.1 Variables libres y ligadas.

El *alcance* de $\forall x$ en la fórmula $\forall x F$ es F . Igualmente, el alcance de $\exists x$ en la fórmula $\exists x F$ también es F .

La regla iii) de la definición de fbf permite distinguir dos clases de ocurrencias de las variables: libres y ligadas.

Una ocurrencia *ligada* de una variable en una fbf es una ocurrencia sobre la que actúa un cuantificador, o una ocurrencia dentro del alcance de un cuantificador que actúa sobre la misma variable. Cualquier otra ocurrencia es *libre*.

Ejemplos:

En la fórmula $\exists x P(x,y) \wedge Q(x)$, las dos primeras ocurrencias de x son ligadas, mientras que la tercera ocurrencia es libre, ya que el alcance de $\exists x$ es $P(x,y)$.

En $\exists x (P(x,y) \wedge Q(x))$, todas las ocurrencias de x son ligadas, dado que el alcance de $\exists x$ es $P(x,y) \wedge Q(x)$.

1.3.2 Fórmulas abiertas y cerradas.

Una *fórmula abierta* es aquella que tiene ocurrencias libres de variables. Una *fórmula cerrada* es la que no las posee. Habitualmente, si F es una fórmula que tiene una ocurrencia libre de la variable x , se suele expresar como $F(x)$.

Ejemplos:

La fórmula $\forall y \exists x (P(x,y) \wedge Q(x))$ es cerrada. En cambio, $\exists x (P(x,y) \wedge Q(x))$ es abierta, dado que hay una ocurrencia libre de la variable y .

1.4 Interpretación.

Dado un conjunto de fórmulas, una *interpretación* consiste en un conjunto no vacío de elementos de D (dominio), sobre el cual pueden tomar valores las variables; así como un conjunto de reglas de asignación de:

- a) cada símbolo de constante a un elemento de D.
- b) cada símbolo de función n-aria a una función D^n sobre D.
- c) cada símbolo de predicado n-ario a una relación sobre D^n .

Ejemplo:

Consideremos el dominio D definido por:

$$D = \{ \text{juan, pedro, roberto, miguel, jaime, ramón} \}$$

y la fórmula:

$$\forall x \forall y \forall z ((\text{Padre}(x,y) \wedge \text{Padre}(y,z)) \rightarrow \text{Abuelo}(x,z))$$

Dar una interpretación consiste en dar significado a los diferentes símbolos, en este ejemplo únicamente a los símbolos de predicados (ya que no aparecen constantes o funciones), asignándoles los correspondientes objetos. En el ejemplo, a Padre y Abuelo se les pueden asignar las relaciones definidas sobre D^2 , que podrían tener las siguientes extensiones (conjunto de tuplas):

$$\text{Padre} = \{ (\text{juan,pedro}), (\text{pedro,roberto}), (\text{miguel,ramón}), (\text{ramón,jaime}) \}$$

$$\text{Abuelo} = \{ (\text{juan,roberto}), (\text{miguel,jaime}) \}$$

1.4.1 Evaluación.

Una vez se ha determinado la interpretación y, por tanto, se han realizado las correspondientes asignaciones a los diferentes objetos (constantes, funciones y predicados) del conjunto de fórmulas, se pueden evaluar las fórmulas en dicha interpretación y obtener un resultado.

1.4.1.1 Evaluación de fórmulas cerradas.

La evaluación de una fórmula cerrada nos dará un valor cierto o falso, que nos indica respectivamente, si la fórmula es cierta o falsa para la interpretación dada.

Para obtener el resultado de una fórmula cerrada se aplican las siguientes reglas de evaluación:

1) *Evaluación de una fórmula atómica:*

Dado un predicado n-ario P asociado a una relación n-aria R, la evaluación de $P(x_1, x_2, \dots, x_n)$ asignando valores a_i a las variables x_i , es decir, $P(a_1, a_2, \dots, a_n)$, es cierta si la tupla (a_1, a_2, \dots, a_n) es un elemento de la relación R. En otro caso es falsa.

2) Evaluación de fórmulas con operadores lógicos:

Las siguientes tablas ilustran la evaluación de una fórmula en función de los valores de las fórmulas operandos para cada operador lógico:

F_1	$\neg F_1$
f	c
c	f

F_1	F_2	$F_1 \wedge F_2$	$F_1 \vee F_2$	$F_1 \rightarrow F_2$
f	f	f	f	c
f	c	f	c	c
c	f	f	c	f
c	c	c	c	c

3) Evaluación de fórmulas con cuantificadores:

- *Cuantificador universal*: si x es una variable libre en la fórmula F , la fórmula $\forall x F(x)$ tomará el valor cierto si $F(a_i)$ es cierta para todos los elementos a_i del dominio D ; en otro caso será falsa. Es decir, el resultado obtenido al evaluar $\forall x F(x)$ es idéntico al obtenido al evaluar:

$$F(a_1) \wedge F(a_2) \wedge \dots \wedge F(a_n)$$

siendo a_1, a_2, \dots, a_n todos los elementos del dominio D .

- *Cuantificador existencial*: De forma semejante, $\exists x F(x)$ será cierta si existe al menos un elemento a_i del dominio D tal que $F(a_i)$ es cierta; en otro caso será falsa. Es decir, el resultado obtenido al evaluar $\exists x F(x)$ es equivalente al obtenido al evaluar:

$$F(a_1) \vee F(a_2) \vee \dots \vee F(a_n)$$

siendo, como anteriormente, a_1, a_2, \dots, a_n todos los elementos del dominio D .

Ejemplo:

Vamos a evaluar la fórmula del ejemplo anterior:

$$\forall x \forall y \forall z ((\text{Padre}(x,y) \wedge \text{Padre}(y,z)) \rightarrow \text{Abuelo}(x,z)) \quad (a)$$

Para que sea cierta, según la regla 3), se debe evaluar a cierto

$$(\text{Padre}(x,y) \wedge \text{Padre}(y,z)) \rightarrow \text{Abuelo}(x,z) \quad (b)$$

para todos los posibles valores de x,y,z del dominio D . La evaluación de cada predicado Padre/Abuelo para cada par de valores $(a_1,a_2) \in D$, según la regla 1), será cierto si la tupla (a_1,a_2) aparece en la extensión de Padre/Abuelo dada en la interpretación. El resultado será una expresión de la forma:

$$(cf_1 \wedge cf_2) \rightarrow cf_3 \quad (c)$$

siendo cf_i constantes con valor cierto o falso. Esta expresión se puede evaluar aplicando las tablas dadas con la regla 2).

Vamos por tanto, a evaluar (b) para todos los posibles valores $x,y,z \in D$:

- $x=juan, y=pedro, z=roberto$:

$$\begin{aligned} & (\text{Padre}(\text{juan},\text{pedro}) \wedge \text{Padre}(\text{pedro},\text{roberto})) \rightarrow \text{Abuelo}(\text{juan},\text{roberto}) \\ & \quad \Downarrow \\ & (\text{cierto} \wedge \text{cierto}) \rightarrow \text{cierto} \\ & \quad \Downarrow \\ & \text{cierto} \rightarrow \text{cierto} \\ & \quad \Downarrow \\ & \text{cierto} \end{aligned}$$

- $x=juan, y=pedro, z=miguel$:

$$\begin{aligned} & (\text{Padre}(\text{juan},\text{pedro}) \wedge \text{Padre}(\text{pedro},\text{miguel})) \rightarrow \text{Abuelo}(\text{juan},\text{miguel}) \\ & \quad \Downarrow \\ & (\text{cierto} \wedge \text{falso}) \rightarrow \text{falso} \\ & \quad \Downarrow \\ & \text{falso} \rightarrow \text{falso} \\ & \quad \Downarrow \\ & \text{cierto} \end{aligned}$$

Si probamos con todos los casos restantes, veremos que se evalúan a cierto, por lo que la fórmula (a) es cierta para la interpretación dada.

1.4.1.2 Evaluación de fórmulas abiertas.

Al evaluar una fórmula abierta con n variables libres para una interpretación I se obtiene como resultado una relación n -aria R , sobre el dominio D , de forma que al sustituir cada una de las variables libres por los valores de los atributos de una de las tuplas de esta relación, da como resultado una fórmula cerrada que es cierta en la interpretación. Si la relación resultado es vacía, se dice que la fórmula abierta es falsa. si la relación resultado coincide con D^n , entonces se dice que la fórmula abierta es cierta.

Esta noción de evaluación de una fórmula abierta, que define un conjunto de tuplas, es fundamental para la comprensión de los lenguajes de interrogación del cálculo relacional ya que, como veremos más tarde, mediante fórmulas abiertas se definen las preguntas del usuario, y las respuestas a estas preguntas son conjuntos de tuplas que hacen cierta la fórmula.

Ejemplo:

$$\exists x \exists z (\text{Padre}(x,y) \wedge \text{Padre}(y,z))$$

Esta fórmula es abierta, ya que hay una ocurrencia libre de la variable y . El resultado será una relación con los valores de y que la hacen cierta, es decir, aquellas personas que a la vez son hijos de algún x y padres de algún z .

Para la interpretación dada previamente con este ejemplo los valores de y que la hacen cierta son: pedro y ramón.

1.4.2 Modelos.

Una interpretación de un conjunto Γ de fórmulas es un modelo, si y sólo si, toda fórmula $F \in \Gamma$ es cierta en la interpretación.

Ejemplo:

La interpretación del ejemplo anterior de padres y abuelos constituye un modelo para el siguiente conjunto de fórmulas, ya que todas las fórmulas son ciertas para esa interpretación:

- i) $\forall x \forall y \forall z (\text{Padre}(x,y) \wedge \text{Padre}(y,z) \rightarrow \text{Abuelo}(x,z))$
- ii) $\exists x \text{Abuelo}(\text{juan},x)$

La fórmula i) ya se demostró previamente que era cierta. La fórmula ii) es cierta ya que existe un $x=\text{roberto}$ de modo que la tupla $\text{Abuelo}(\text{juan},\text{roberto})$ aparece en la interpretación.

2. Cálculo relacional de tuplas.

Como ya se comentó, este lenguaje está basado en el cálculo de predicados utilizando variables-tupla que representan tuplas, esto obliga a introducir algunos cambios en las definiciones comentadas anteriormente, y que pasamos a exponer.

2.1 Variables-tupla.

Anteriormente habíamos supuesto una lógica homogénea, es decir, el dominio de las variables coincidía con el conjunto de constantes del lenguaje. En el CRT ya no es así. Cada variable-tupla x se define sobre una intensión (esquema) de una relación R :

$x : R$ tal que $R(a_1:\text{dom}_1, a_2:\text{dom}_2, \dots, a_n:\text{dom}_n)$

De esta forma, se especifica de que relaciones tomarán valores las variables. La variable x tomará valores en el producto cartesiano de todos los dominios de R , $\text{dom}_1 \times \text{dom}_2 \times \dots \times \text{dom}_n$.

2.2 Definición del lenguaje.

Los símbolos del lenguaje son los mismos que los vistos en la lógica de predicados de primer orden, con la única diferencia ya comentada de que las variables son variables-tupla.

2.2.1 Términos.

Veamos cuáles son ahora los términos:

- i) Los símbolos de constantes.
- ii) Términos de proyección: $x.A$, donde x es una variable-tupla y A es el nombre de un atributo de la relación sobre la que se declaró x . El término $x.A$ representa el valor del atributo A en la tupla x . También se representa como $x[A]$.

2.2.2 Fórmulas.

La definición de las fórmulas atómicas o átomos viene dada por las siguientes reglas:

- a) Términos de dominio: $R(x)$, donde R es el nombre de una relación y x es el nombre de una variable-tupla que se declaró sobre la relación R o sobre otra relación compatible con R . $R(x)$ limita el rango de variación de la variable-tupla x al conjunto de tuplas que pertenecen a la relación R , es decir, la tupla instanciada por x pertenece a la relación R . También se representa por R_x .
- b) $t_1 \theta t_2$, donde t_1 y t_2 son términos y θ es un operador de comparación ($<$, $>$, $=$, \geq , \leq , \neq). Representa la aserción de que el término t_1 está en relación θ con el término t_2 .

La definición de fbf de este lenguaje coincide totalmente con la ya vista.

2.2.3 Evaluación de fórmulas.

Las definiciones de variable libre y ligada, y de fórmula abierta y cerrada coinciden con las ya vistas.

Ejemplos:

- En una comparación simple todas las ocurrencias son libres:

$x.A = y.B$

$x.A = \text{'Dedalo'}$

Todas las ocurrencias de x e y son libres.

- Las ocurrencias de variables-tupla en (F) y $\neg F$ son libres o ligadas según sean libres o ligadas en F :

$\neg (x.ciudad = \text{'Londres'})$

La ocurrencia de x es libre.

- Las ocurrencias de variables-tupla en $F \wedge G$ y en $F \vee G$ son libres o ligadas según sean libres o ligadas en F o en G , según donde ocurran:

$(x.codigo = y.codigo \wedge y.numero <> z.numero)$

Todas las ocurrencias de x, y, z son libres.

- Las ocurrencias de x que son libres en F son ligadas en $\exists x(F)$ y en $\forall x(F)$. Otras ocurrencias de variables-tupla en F serán libres o ligadas en dichas fbf según lo sean en F .

$\exists x (x.codigo = y.codigo \wedge x.numero = \text{'123'})$

Las ocurrencias de x son ligadas, mientras que la de y es libre.

$\forall x (x.color = \text{'rojo'})$

Las dos ocurrencias de x son ligadas.

El concepto de interpretación es totalmente equivalente al ya visto, y sólo se producen algunos cambios en la forma de realizar la evaluación de los fórmulas. Estos cambios son debidos principalmente al tipo de variable que se maneja. Las reglas para evaluar fórmulas cerradas son:

- 1) La *evaluación de una fórmula atómica* dependerá de la clase de fórmula atómica:
 - a) $R(x)$: cuando x se instancia a una tupla t , tal que $t \in R$, entonces se evalúa a cierto. En caso contrario se evalúa a falso.
 - b) $t_1 \theta t_2$: se evaluará a cierto o falso según el resultado de la comparación. En el caso de que alguno de los términos sea de proyección, es decir, de la forma $x.A$, el valor que se utilizará para realizar la comparación será el del atributo A de la tupla que en ese momento instancie la variable.
- 2) La *evaluación de fórmulas con operadores lógicos* no cambia nada respecto del método comentado anteriormente.
- 3) La *evaluación de fórmulas con cuantificadores* se realiza igual que para la lógica de primer orden. La única diferencia se da en el dominio de las variables, ya que ahora las variables cuantificadas tienen un dominio determinado, tal como se explicó al definir las variables-tupla y su declaración. Este dominio es el que hay que utilizar para realizar la evaluación.

2.3 Consultas en el cálculo relacional de tuplas.

Teniendo en cuenta las definiciones y elementos del CRT explicados previamente, se deduce que podemos escribir consultas (preguntas) a una base de datos relacional utilizando expresiones del tipo

$$\{ x \mid F(x) \}$$

donde F es una fórmula que tiene como única variable-tupla libre a x.

Ejemplos:

La unión de dos relaciones $R \cup S$ puede expresarse en CRT así:

$$\{ t \mid R(t) \vee S(t) \}$$

La diferencia de dos relaciones $R - S$ se expresa de la siguiente forma:

$$\{ t \mid R(t) \wedge \neg S(t) \}$$

Dado el esquema de base de datos siguiente:

sucursal(nombre, activo, ciudad)

cliente(nombre, calle, ciudad)

depósito(sucursal, n-cuenta, cliente, saldo)

préstamo(sucursal, n-prestamo, cliente, importe)

a) Encontrar la sucursal, n-prestamo, cliente e importe para los préstamos mayores de 300.000 pts:

$$\{ t \mid \text{préstamo}(t) \wedge t.\text{importe} > 300000 \}$$

b) Obtener todos los clientes que tienen préstamos de más de 300.000 pts:

$$\{ t \mid \text{cliente}(t) \wedge \exists s(\text{préstamo}(s) \wedge t.\text{nombre} = s.\text{cliente} \wedge s.\text{importe} > 300000) \}$$

c) Encontrar a todos los clientes que tienen una cuenta en la sucursal 'Paz', pero que no han sacado un préstamo en esa sucursal:

$$\{ t \mid \text{cliente}(t) \wedge \exists s(\text{depósito}(s) \wedge t.\text{nombre} = s.\text{cliente} \wedge s.\text{sucursal} = \text{'Paz'}) \wedge \neg \exists u(\text{préstamo}(u) \wedge t.\text{nombre} = u.\text{cliente} \wedge u.\text{sucursal} = \text{'Paz'}) \}$$

2.4 Cálculo relacional de tuplas restringido.

El CRT, tal y como lo hemos definido, permite expresiones como

$$\{ t \mid \neg R(t) \}$$

que denota todas las posibles tuplas tales que no estén en R. Hay que tener en cuenta que mientras que el conjunto de tuplas que pertenecen a una relación es perfectamente conocido, el conjunto de tuplas que no pertenecen a ella dependerá de los dominios de la definición de dicha relación. Puesto que en las bases de datos suele ser habitual que aparezcan dominios de cardinalidad infinita o prácticamente

infinita, esto nos podría llevar a intentar manipular relaciones de cardinalidad también infinita. Por otra parte, el sentido de tales preguntas difícilmente se puede aceptar en las bases de datos relacionales.

Por todo ello es necesario evitar tales expresiones, restringiendo el CRT a fórmulas que no tengan esta característica, y que llamaremos fórmulas seguras.

2.4.1 Dominio de una fórmula.

El dominio de una fórmula F , que se representa por $\text{dom}(F)$, corresponde al rango de valores permitidos para una fórmula. Su definición es la siguiente:

Dada una fórmula F que contiene los símbolos de predicados P_1, P_2, \dots, P_k (siendo R_1, R_2, \dots, R_k las relaciones que tienen asignadas respectivamente) y los símbolos de constantes a_1, a_2, \dots, a_p , el dominio de F se define como:

$$\text{dom}(F) = \{a_1, a_2, \dots, a_p\} \cup \{\text{valores de los atributos de las tuplas de } R_1, R_2, \dots, R_k\}$$

Puesto que las relaciones R_1, R_2, \dots, R_k son conjuntos finitos, también lo será $\text{dom}(F)$.

2.4.2 Fórmulas seguras.

Una fórmula F es segura si satisface las tres condiciones siguientes:

- Si t es una tupla tal que $F(t)$ es cierta, entonces cada valor de cada atributo de t debe pertenecer al dominio de F , $\text{dom}(F)$.
- Para cada subfórmula F' de F tal que $F = \exists x F'(x)$, si t es una tupla tal que $F'(t)$ es cierta, entonces cada valor de cada atributo de t debe pertenecer a $\text{dom}(F')$.
- Para cada subfórmula F' de F tal que $F = \forall x F'(x)$, si t es una tupla tal que $F'(t)$ es cierta, entonces algún valor de alguno de los atributos de t no debe pertenecer a $\text{dom}(F')$.

Las reglas b) y c) permiten asegurar que se puede determinar el valor de una subfórmula F' cuantificada de la forma $\exists x F'(x)$ o $\forall x F'(x)$ considerando únicamente aquellas instancias de x compuestas por elementos de $\text{dom}(F')$.

Para entender el sentido de la tercera condición escribimos $\forall x F'(x)$ en su forma equivalente: $\neg \exists x \neg F'(x)$. La segunda condición obliga, para que esta última expresión represente información contenida en la base de datos, a que:

$$\forall x (\neg F'(x) = \text{'cierto'} \Rightarrow \text{todos los valores de atributos de } x \text{ pertenecen a } \text{dom}(\neg F'))$$

Invirtiendo la implicación obtenemos:

$$\forall x (\text{un valor de un atributo de } x \text{ no perteneciente a } \text{dom}(\neg F') \Rightarrow F'(x) = \text{'cierto'})$$

y puesto que $\text{dom}(F') = \text{dom}(\neg F')$, podemos escribir:

$$\forall x (\text{un valor de un atributo de } x \text{ no perteneciente a } \text{dom}(F') \Rightarrow F'(x) = \text{'cierto'})$$

expresión equivalente a la tercera condición.

Ejemplos:

Veamos todos los conceptos anteriores con el siguiente ejemplo:

Sean R y S dos relaciones con las extensiones siguientes:

R	
A	B
1	2
3	4
2	5

S	
A	B
3	4
2	2

y dos preguntas E1 y E2 representadas por las dos expresiones siguientes:

$$\begin{array}{ll} E1 : \{ x \mid R(x) \vee \neg S(x) \} & \text{dom}(E1) = \{1,2,3,4,5\} \\ E2 : \{ x \mid R(x) \wedge \neg S(x) \} & \text{dom}(E2) = \text{dom}(E1) \end{array}$$

La primera expresión E1 contiene una fórmula abierta que no es segura ya que, por ejemplo, la tupla (3,7) hace cierta la fórmula y, sin embargo, el valor 7 no pertenece a $\text{dom}(E1)$.

La segunda expresión E2 contiene una fórmula abierta que sí es segura. Si t es una tupla que hace cierta dicha fórmula, también debe ser cierta $R(t)$ y, por tanto, cada valor de cada atributo de t pertenece a $\text{dom}(E2)$.

En los ejemplos anteriores se ha presentado algún caso que se puede generalizar. A continuación presentamos tres reglas generales que nos pueden simplificar el averiguar si una fórmula es segura:

- i) $\{ x \mid F(x) \wedge G(x) \}$, donde F es una fórmula segura y G es una fórmula cualquiera, es una fórmula segura. Para comprobarlo, basta con tener en cuenta que cualquier tupla t que verifique la fórmula $F(x) \wedge G(x)$, también verifica F(t), por lo que los valores de todos los atributos de t pertenecen a $\text{dom}(F(x))$, y por consiguiente también pertenecerán a $\text{dom}(F(x) \wedge G(x))$.
- ii) $\exists x (F(x) \wedge G(x))$, donde F es una fórmula segura y G es una fórmula cualquiera (segura o no segura), es una fórmula segura.
- iii) $\forall x (\neg F(x) \wedge G(x))$, donde F es una fórmula segura y G es una fórmula cualquiera, es una fórmula segura.

Ejemplos:

Para el siguiente esquema de base de datos vamos a escribir varias expresiones que representan restricciones o requerimientos (consultas) a la base de datos:

Alumno(n_mat, nombre, acceso)
 Asignatura(codigo, nombre, curso)
 Notas(n_mat, codigo, nota)

- Todos los alumnos han de estar matriculados al menos de una asignatura (Restricción):

x : Alumno
 y : Notas
 $\forall x (\text{Alumno}(x) \rightarrow \exists y (\text{Notas}(y) \wedge x.\text{num_mat}=y.\text{num_mat}))$

- Todo alumno que se matricule de la asignatura 'Bases de Datos' (BDa) ha de tener aprobada la asignatura 'Estructuras de Datos' (EDa) (restricción):

x : Alumno
 y ,z : Notas
 $\forall x ($
 $(\text{Alumno}(x) \wedge \exists y (\text{Notas}(y) \wedge x.\text{n_mat}=y.\text{n_mat} \wedge y.\text{codigo}='BDa')) \rightarrow$
 $\exists z (\text{Notas}(z) \wedge x.\text{n_mat}=z.\text{n_mat} \wedge z.\text{codigo}='EDa' \wedge z.\text{nota} \geq 5))$

- Obtener las asignaturas de las que se ha matriculado algún alumno que tenga modalidad de acceso de 'Formación Profesional' (FP) (consulta):

x : Asignatura
 y : Alumno
 z : Notas
 $\{ x \mid \text{Asignatura}(x) \wedge \exists y (\text{Alumno}(y) \wedge y.\text{acceso}='FP' \wedge$
 $\exists z (\text{Notas}(z) \wedge z.\text{codigo}=x.\text{codigo} \wedge z.\text{n_mat}=y.\text{n_mat})) \}$

- Obtener los alumnos matriculados en todas las asignaturas de primero (consulta):

x : Alumno
 y : Asignatura
 z : Notas
 $\{ x \mid \text{Alumno}(x) \wedge \forall y ((\text{Asignatura}(y) \wedge y.\text{curso}=1) \rightarrow$
 $\exists z (\text{Notas}(z) \wedge z.\text{codigo}=y.\text{codigo} \wedge z.\text{n_mat}=x.\text{n_mat})) \}$

3. Cálculo relacional de dominios.

Esta segunda modalidad de cálculo relacional se diferencia de la ya vista en que se emplean variables-dominio. Las expresiones en el cálculo relacional de dominios (CRD) se construyen siguiendo unos principios análogos. Los cambios respecto de lo ya visto son bastante pequeños, por lo que la exposición será más reducida y se limitará a señalar dichas diferencias.

3.1 Variables-dominio.

Las variables en este lenguaje se denominan variables-dominio ya que toman valores en el dominio asociado a alguno de los atributos de una relación en vez de representar una tupla entera, como en el CRT. La definición de una variable-dominio será del tipo:

$$x : \text{dom}_k$$

de esta forma la variable x tomará valores en el dominio dom_k .

3.2 Definición del lenguaje.

Aclarado el nuevo tipo de variable empleado, pasamos a ver cuáles son los diferentes objetos de este lenguaje. Los símbolos son los mismos que para el CRT salvo la diferencia ya comentada de las variables.

3.2.1 Fórmulas.

Los *términos* del lenguaje son los símbolos de constantes o de variables.

Ejemplo:

a
x
'hola'

Los átomos o *fórmulas atómicas* toman una de los dos formas siguientes:

- a) $R(t_1, t_2, \dots, t_n)$, donde R es una relación de grado igual a n o superior, y t_1, t_2, \dots, t_n son términos. Los términos han de pertenecer a los dominios asociados a los atributos de la relación.
- b) $t_1 \theta t_2$, donde t_1 y t_2 son términos y θ es un operador de comparación.

La definición de *fórmula bien formada* (fbf) sigue siendo la misma que para el CRT.

3.2.2 Evaluación de fórmulas.

También aquí se mantienen las definiciones de ocurrencia de una variable libre y ligada, y de fórmula abierta y cerrada.

La forma de construir la *interpretación* es la misma que la comentada anteriormente, y sólo hay que introducir algunas modificaciones, causadas por el tipo de variable que se usa, al evaluar las fórmulas. A continuación se indican las reglas para evaluar una fórmula cerrada en este lenguaje:

- 1) *Evaluación de fórmulas atómicas:* Depende de la clase de átomo que sea:
 - a) $R(t_1, t_2, \dots, t_n)$: Si se puede encontrar una tupla de R, tal que los valores de los atributos a_1, a_2, \dots, a_n de la tupla coinciden con los de los respectivos términos t_1, t_2, \dots, t_n , entonces la fórmula se evalúa a cierto. En otro caso se evalúa a falso.
 - b) $t_1 \theta t_2$: La evaluación es equivalente a la vista en el CRT, dependiendo del resultado de la comparación.

- 2) *Evaluación de fórmulas con operadores lógicos:* Es idéntica a la ya estudiada.

- 3) *Evaluación de fórmulas con cuantificadores:* La única diferencia es que ahora el dominio de las variables es el dominio asociado a uno de los atributos de una relación. Este es el dominio que se debe usar para realizar su asignación.

3.3 Consultas en el cálculo relacional de dominios.

La forma de expresar en este lenguaje una pregunta a la base de datos es:

$$\{ x_1, x_2, \dots, x_k \mid F(x_1, x_2, \dots, x_k) \}$$

tal que x_1, x_2, \dots, x_k son las únicas variables libres en F. La evaluación de esta expresión da como resultado una relación k-aria, de forma que al sustituir cada una de las variables libres por los valores de los atributos de una de las tuplas de dicha relación, da como resultado una fórmula cerrada que se evalúa a cierto.

Ejemplos:

Reproducimos aquí los mismos ejemplos utilizados en el apartado 2.3 con el fin de que sirvan para poder comparar cómo son las mismas expresiones en los dos lenguajes:

La unión de dos relaciones $R \cup S$ se expresa en CRD así:

$$\{ t_1, t_2, \dots, t_k \mid R(t_1, t_2, \dots, t_k) \vee S(t_1, t_2, \dots, t_k) \}$$

La diferencia de dos relaciones $R - S$ se expresa de la siguiente forma:

$$\{ t_1, t_2, \dots, t_k \mid R(t_1, t_2, \dots, t_k) \wedge \neg S(t_1, t_2, \dots, t_k) \}$$

a) Encontrar la sucursal, n-prestamo, cliente e importe para los préstamos mayores de 300.000 pts:

$$\{ s, t, u, v \mid \text{préstamo}(s, t, u, v) \wedge v > 300000 \}$$

b) Obtener todos los clientes que tienen préstamos de más de 300.000 pts:

$$\{ t_1, t_2, t_3 \mid \text{cliente}(t_1, t_2, t_3) \wedge \exists s_3 s_4 (\text{préstamo}(s_1, s_2, s_3, s_4) \wedge t_1 = s_3 \wedge s_4 > 300000) \}$$

c) Encontrar a todos los clientes que tienen una cuenta en la sucursal 'Paz', pero que no han sacado un préstamo en esa sucursal:

$$\{ t_1, t_2, t_3 \mid \text{cliente}(t_1, t_2, t_3) \wedge \exists s_1 s_3 (\text{depósito}(s_1, s_2, s_3, s_4) \wedge t_1 = s_3 \wedge s_1 = \text{'Paz'}) \wedge \neg \exists u_1 u_3 (\text{préstamo}(u_1, u_2, u_3, u_4) \wedge t_1 = u_3 \wedge u_1 = \text{'Paz'}) \}$$

3.4 Cálculo relacional de dominios restringido.

Análogamente al CRT, es necesario restringir el CRD para no permitir expresiones como la siguiente:

$$\{ x \mid \neg R(x) \}$$

que podría dar como resultado una relación de cardinalidad infinita, con los consiguientes problemas que esta situación plantea. Se hace necesario, por tanto, restringir el CRD para que únicamente sean válidas las fórmulas llamadas *seguras*.

Para definir estas fórmulas también nos hace falta el concepto de dominio de una fórmula, que coincide totalmente con el indicado anteriormente para el CRT.

3.4.1 Fórmulas seguras.

Una fórmula F es segura, si cumple las siguientes tres reglas:

- Sea F una fórmula abierta con x_1, x_2, \dots, x_k variables libres, si (a_1, a_2, \dots, a_k) es una tupla de valores tales que $F(a_1, a_2, \dots, a_k)$ es cierta, entonces se cumple que $a_i \in \text{dom}(F)$, $i=1..k$.
- Para cada subfórmula F' de F de la forma $F = \exists x F'(x)$, si x tiene un valor tal que hace cierta $F'(x)$, entonces se cumple que $x \in \text{dom}(F')$.
- Para cada subfórmula F' de F de la forma $F = \forall x F'(x)$, si x tiene un valor tal que $x \notin \text{dom}(F')$, entonces $F'(x)$ es cierta.

Ejemplos:

Vamos a ver como se escriben en el CRD las mismas expresiones que se utilizaron de ejemplos para el CRT:

- Todos los alumnos han de estar matriculados al menos de una asignatura (restricción):

$$x : \text{dom}(n_mat) \equiv 1..10000;$$

$$y : \text{dom}(\text{codigo}) \equiv \text{char}(3);$$

$$\forall x (\text{Alumno}(x, v_1, v_2) \rightarrow \exists y (\text{Notas}(x, y, v_3))$$

- Todo alumno que se matricule de la asignatura 'Bases de Datos' (BDa) ha de tener aprobada la asignatura 'Estructuras de Datos' (EDa) (restricción):

$x : \text{dom}(n_mat) \equiv 1..10000;$

$y : \text{dom}(nota) \equiv \text{real};$

$\forall x ((\text{Alumno}(x,v_1,v_2) \wedge \text{Notas}(x,\text{'BDa'},u_1)) \rightarrow$
 $\exists y(\text{Notas}(x,\text{'EDa'},y) \wedge y \geq 5))$

- Obtener las asignaturas de las que se ha matriculado algún alumno que tenga modalidad de acceso de 'Formación Profesional' (FP) (consulta):

$x : \text{dom}(\text{codigo}) \equiv \text{char}(3);$

$y : \text{dom}(n_mat) \equiv 1..10000;$

$\{ x \mid \text{Asignatura}(x,v_1,v_2) \wedge \exists y (\text{Alumno}(y,u_1,\text{'FP'}) \wedge \text{Notas}(y,x,w_1)) \}$

- Obtener los alumnos (número de matrícula y nombre) matriculados en todas las asignaturas de primero (consulta):

$x : \text{dom}(n_mat) \equiv 1..10000;$

$y : \text{dom}(\text{nombre}) \equiv \text{char}(30);$

$z : \text{dom}(\text{codigo}) \equiv \text{char}(3);$

$\{ x,y \mid \text{Alumno}(x,y,v_1) \wedge \forall z (\text{Asignatura}(z,u_1,1) \rightarrow \text{Notas}(x,z,w_1)) \}$

4 Comparación entre los diversos lenguajes.

Ya se ha comentado que el AR es un lenguaje procedimental, ya que se detalla qué operaciones se deben realizar y en qué orden deben hacerse. Esto plantea la necesidad de elaborar estrategias de optimización, tema que por su complejidad e importancia no se ha estudiado. La mayor parte de los lenguajes de SGBDR son parecidos al AR, pero de más alto nivel, ya que no son procedimentales, pues el propio sistema incorpora un optimizador que elige en cada momento la secuencia de operaciones más conveniente en términos de tiempo de ejecución.

Por contra, el CR (orientado a tuplas o a dominios) es no procedimental, es decir, de más alto nivel de abstracción que AR, pues se indica qué resultado quiere obtenerse, pero no cómo debe llegarse a él.

Ejemplos:

A continuación se presentan varios ejemplos de equivalencia entre expresiones en AR, CRT y CRD. Para ello se emplea el siguiente esquema de base de datos:

proveedor(p#, nombre, ciudad, calle, numero)

articulo(a#, nombre, precio, descripcion)

pedido(p#, a#, cantidad)

1) Nombre de los proveedores que viven en 'Madrid'.

- Algebra:

$$\Pi_{\text{nombre}}(\sigma_{\text{ciudad}='Madrid'}(\text{proveedor}))$$

- Cálculo de tuplas:

$$\{ r.\text{nombre} \mid \text{proveedor}(r) \wedge r.\text{ciudad}='Madrid' \}$$

- Cálculo de dominios:

$$\{ x \mid \exists y \exists z \exists w (\text{proveedor}(y,x,'Madrid',z,w)) \}$$

2) Nombre de los proveedores que suministran el artículo 20:

- Algebra:

$$\Pi_{\text{nombre}}(\sigma_{a\#=20}(\text{pedido}) _ \text{proveedor})$$

- Cálculo de tuplas:

$$\{ r.\text{nombre} \mid \exists p (\text{proveedor}(r) \wedge \text{pedido}(p) \wedge r.p\#=p.p\# \wedge p.a\#=20) \}$$

- Cálculo de dominios:

$$\{ x \mid \exists y \exists z \exists w \exists p \exists u (\text{proveedor}(p,x,y,z,w) \wedge \text{pedido}(p,20,u)) \}$$

2) Nombre de los proveedores que suministran artículos de precio mayor que 10:

- Algebra:

$$\Pi_{\text{nombre}}(\sigma_{\text{precio} \geq 10}(\text{articulo}) _ \text{pedido} _ \text{proveedor})$$

- Cálculo de tuplas:

$$\{ r.\text{nombre} \mid \exists s \exists t (\text{proveedor}(r) \wedge \text{pedido}(s) \wedge \text{articulo}(t) \wedge r.p\#=s.p\# \wedge s.a\#=t.a\# \wedge t.\text{precio} \geq 10) \}$$

- Cálculo de dominios:

$$\{ x \mid \exists y \exists z \exists u \exists v \exists w \exists k \exists l \exists m \exists n (\text{proveedor}(y,x,z,u,v) \wedge \text{articulo}(w,k,l,m) \wedge \text{pedido}(y,w,n) \wedge l > 10) \}$$

4.1 Completitud relacional.

Un lenguaje se llama '*relacionalmente completo*' si es al menos tan potente como el cálculo relacional de tuplas, es decir, si sus expresiones permiten la definición de cualquier relación definible por las expresiones del CRT. Codd demostró que el CRT es equivalente al AR, en el sentido de que cualquier expresión del CRT tiene su equivalente en el AR y viceversa. Por tanto, el AR es relacionalmente completo.

Lo anterior nos facilita un método para comprobar si un lenguaje cualquiera L es relacionalmente completo: bastará con averiguar si incluye análogos a los cinco operadores fundamentales del AR.

4.2 Algebra versus Cálculo de tuplas.

Tal como se ha comentado, ambos lenguajes son equivalentes según indica el siguiente teorema:

Para cada expresión E del algebra relacional, existe una expresión segura en el

cálculo relacional de tuplas que es equivalente a E.

Dicho teorema es demostrado por inducción en [ULLM82], donde se establecen las siguientes equivalencias, suponiendo que las expresiones del CRT $\{ t \mid F_1(t) \}$ y $\{ t \mid F_2(t) \}$ son equivalentes respectivamente a las expresiones E_1 y E_2 del AR:

- i) $E_1 \cup E_2 \equiv \{ t \mid F_1(t) \vee F_2(t) \}$
- ii) $E_1 - E_2 \equiv \{ t \mid F_1(t) \wedge \neg F_2(t) \}$
- iii) $E_1 \times E_2 \equiv \{ t^{(k+m)} \mid \exists u \exists v (F_1(u) \wedge F_2(v) \wedge t[1]=u[1] \dots \wedge t[k]=u[k] \wedge t[k+1]=v[1] \dots \wedge t[k+m]=v[m]) \}$

denotando E_1 y E_2 relaciones de aridad k y m respectivamente, y significando $t^{(k+m)}$ una tupla de aridad $k+m$. Además $t[i]$ representa el atributo i -ésimo de la tupla t .

- iv) $\Pi_{i_1, i_2, \dots, i_k}(E_1) \equiv \{ t^{(k)} \mid \exists u (F_1(u) \wedge t[1]=u[i_1] \dots \wedge t[k]=u[i_k]) \}$
- v) $\sigma_p(E_1) \equiv \{ t \mid F'_1(u) \}$, donde F'_1 es F_1 sustituyendo cada operando que denota el componente i por $t[i]$.

Ejemplo:

Dadas dos relaciones binarias R y S , transformar la expresión siguiente del AR al CRT:

$$\Pi_{1,4}(\sigma_{2=3}(R \times S))$$

Aplicando la equivalencia iii) tendremos la siguiente expresión para $R \times S$:

$$\{ t \mid \exists u \exists v (R(u) \wedge S(v) \wedge t[1]=u[1] \wedge t[2]=u[2] \wedge t[3]=v[1] \wedge t[4]=v[2]) \}$$

Para $\sigma_{2=3}(R \times S)$, según la equivalencia v) añadimos a la fórmula anterior el término $\wedge t[2]=t[3]$:

$$\{ t \mid \exists u \exists v (R(u) \wedge S(v) \wedge t[1]=u[1] \wedge t[2]=u[2] \wedge t[3]=v[1] \wedge t[4]=v[2] \wedge t[2]=t[3]) \}$$

La equivalencia iv) nos permite representar $\Pi_{1,4}(\sigma_{2=3}(R \times S))$ como:

$$\{ w \mid \exists t \exists u \exists v (R(u) \wedge S(v) \wedge t[1]=u[1] \wedge t[2]=u[2] \wedge t[3]=v[1] \wedge t[4]=v[2] \wedge t[2]=t[3] \wedge w[1]=t[1] \wedge w[2]=t[4]) \}$$

Si se sustituye cada uno de los componentes de t por los componentes apropiados de u o v , podemos eliminar t :

$$\{ w \mid \exists u \exists v (R(u) \wedge S(v) \wedge u[2]=v[1] \wedge w[1]=u[1] \wedge w[2]=v[2]) \}$$

4.3 Cálculo de tuplas versus Cálculo de dominios.

La construcción de una expresión en el CRD equivalente a una expresión $\{ t \mid F(t) \}$ del CRT es directa. Si t tiene aridad k , basta con introducir nuevas variables-dominio t_1, t_2, \dots, t_k y sustituir dicha expresión por

$$\{ t_1, t_2, \dots, t_k \mid F'(t_1, t_2, \dots, t_k) \}$$

donde F' es F con los átomos $R(t)$ sustituidos por $R(t_1, t_2, \dots, t_k)$, y cada ocurrencia libre de $t[i]$ sustituida por t_i . Podemos por tanto plantear un teorema similar al visto para el AR versus CRT:

Para cada expresión segura del cálculo relacional de tuplas existe otra expresión segura equivalente en el cálculo relacional de dominios.

Ejemplo:

Siguiendo el ejemplo utilizado en el apartado anterior, la expresión del CRT:

$$\{ w \mid \exists u \exists v (R(u) \wedge S(v) \wedge u[2]=v[1] \wedge w[1]=u[1] \wedge w[2]=v[2]) \}$$

se transforma en su equivalente del CRD sustituyendo w por w_1w_2 , u por u_1u_2 y v por v_1v_2 :

$$\{ w_1, w_2 \mid \exists u_1 \exists u_2 \exists v_1 \exists v_2 (R(u_1, u_2) \wedge S(v_1, v_2) \wedge u_2=v_1 \wedge w_1=u_1 \wedge w_2=v_2) \}$$

4.4 Cálculo de dominios versus Álgebra.

Al igual que en los casos anteriores, podemos establecer un teorema de equivalencia:

Para cada expresión segura del cálculo relacional de dominios, existe una expresión equivalente en el álgebra relacional.

La demostración se puede encontrar en [ULLM82], resultando demasiado compleja para nuestros objetivos, por lo que no la analizaremos en detalle. Esta basada en los dos lemas siguientes:

- i) Si F es una fórmula en el CRD, entonces existe una expresión en AR denotando la relación unaria $\text{dom}(F)$.
- ii) Si F es una fórmula en CRD, entonces existe una fórmula equivalente F' del CRD sin ocurrencias de \wedge o de \forall . Si F es segura, también lo será F' .

El segundo lema está basado en las dos siguientes equivalencias, ya estudiadas:

$$F_1 \wedge F_2 \equiv \neg(\neg F_1 \vee \neg F_2)$$

$$\forall x(F(x)) \equiv \neg \exists x(\neg F(x))$$

Ejemplo:

Dadas las relaciones binarias R y S , la expresión del CRD:

$$\{ w, x \mid R(w, x) \wedge \forall y(\neg S(w, y) \wedge \neg S(x, y)) \}$$

denota el conjunto de tuplas en R para las cuales ninguno de sus dos componentes aparece como primer componente de alguna tupla de S .

Utilizando el segundo lema, eliminamos el operador \wedge :

$$\{ w, x \mid \neg(\neg R(w, x) \vee \neg \forall y(\neg(\neg \neg S(w, y) \vee \neg \neg S(x, y)))) \}$$

y el cuantificador \forall :

$$\{ w, x \mid \neg(\neg R(w, x) \vee \neg \exists y(\neg(\neg \neg S(w, y) \vee \neg \neg S(x, y)))) \}$$

y cancelando los pares de negaciones queda:

$$\{ w, x \mid \neg(\neg R(w, x) \vee \exists y(S(w, y) \vee S(x, y))) \}$$

En [ULLM82] se demuestra que dicha expresión se convierte en la siguiente del AR:

$$R - (\Pi_{1,3}(S \times E) \cup \Pi_{3,1}(S \times E))$$

5 Lenguajes comerciales.

Los lenguajes formales que se han presentado permiten representar consultas a bases de datos relacionales, pero el tipo de lenguaje utilizado no es adecuado para su empleo por usuarios inexpertos. Es por esto que se han desarrollado diversos lenguajes más o menos similares al inglés basados en los lenguajes formales comentados. En esta sección presentaremos dos de estos lenguajes: SQL y QBE. SQL utiliza una combinación del AR y del CRT, mientras que QBE está basado en el CRD. Coincide además, que ambos son con mucho, los dos lenguajes más extendidos para acceso a bases de datos relacionales, hasta el punto de que constituyen un estándar de facto.

Aunque se les suele llamar lenguajes de consulta, hay que recordar que no se limitan solamente a esta función sino que también permiten realizar operaciones de definición de datos y de control del SGBD.

5.1 Structured Query Language (SQL).

Este lenguaje, que ha llegado a ser un auténtico estándar internacional para consultas a SGBDR, fue desarrollado en 1974 para el System R de IBM. También es conocido por su nombre más antiguo SEQUEL.

Una consulta en SQL tiene la forma:

```
select A1, A2, ... , An
from R1, R2, ... , Rm
where P
```

Los A_i son atributos, las R_i representan relaciones y P es un predicado construido de forma similar a los explicados en el AR para el operador de selección (σ), pero sustituyendo los operadores lógicos \wedge , \vee y \neg por *and*, *or* y *not* respectivamente. Esta consulta es equivalente a la expresión siguiente del AR:

$$\Pi_{A_1, A_2, \dots, A_n}(\sigma_P(R_1 \times R_2 \times \dots \times R_m))$$

Es decir, SQL forma el producto cartesiano de las relaciones que se indican en la cláusula **from**, a continuación realiza una selección utilizando el predicado de la cláusula **where**, y proyecta el resultado a los atributos de la cláusula **select**. En realidad, el SQL incorpora un optimizador de consultas, de forma que puede ocurrir que realmente la expresión se convierta en otra equivalente más rápida de procesar.

Como es lógico, el resultado de una consulta SQL es una relación. Aunque en los lenguajes formales una relación es un conjunto de tuplas y por tanto, no hay tuplas duplicadas, en la práctica, la eliminación de duplicados tiene unos costes considerables y no se realiza a no ser que se indique expresamente.

5.1.1 Completitud relacional.

Bastará con comprobar que se pueden realizar las cinco operaciones fundamentales del AR:

i) La unión aparece implementada directamente en SQL con la cláusula **union**, de forma que la operación $R \cup S$ se escribe:

```
select *  
  from R  
union  
select *  
  from S
```

donde * representa a todos los atributos de la relación o relaciones correspondientes.

ii) La diferencia $R - S$ se realiza utilizando el predicado **exists** que devuelve un valor 'cierto' si existe alguna relación que cumple una determinada subconsulta:

```
select *  
  from R  
 where not exists  
   (select *  
     from S  
    where R.A1=S.A1 ... R.An=S.An )
```

siendo $A_1 \dots A_n$ los atributos de R o S (deben ser unión compatibles).

iii) El producto cartesiano $R \times S$ se realiza con la propia sentencia **select** como ya se indicó:

```
select *  
  from R, S
```

iv) La restricción $\sigma_p(R)$ se realiza con la cláusula **where**:

```
select *  
  from R  
 where P
```

donde P es un predicado indicando la condición que deben cumplir las tuplas.

v) Por último, la proyección $\Pi_L(R)$ se representa por:

```
select L  
  from R
```

siendo L la lista de atributos de R sobre los que se hará la proyección.

Con lo anterior se ha demostrado que el SQL es al menos tan potente relacionalmente hablando como el AR. En realidad lo es más, ya que incorpora algunos conceptos que no aparecen en el AR, tales como las funciones, las cláusulas *having*, *group by* y *order by*; y los operadores *in*, *any* y *all*.

Los demás operadores del AR también se pueden representar, a título de ejemplo mostramos

a continuación dos de ellos:

La intersección $R \cap S$ se implementa de forma muy similar a la diferencia, eliminando el operador **not** de la cláusula exists:

```
select *
  from R
  where not exists
    (select *
     from S
     where R.A1=S.A1 ... R.An=S.An)
```

La combinación natural se realiza de la forma siguiente:

```
select A1, A2, ..., An, Bk+1, ..., Bm
  from R,S
  where R.A1=S.A1 and ... R.Ak=S.Ak
```

siendo $A_1 \dots A_k$ los atributos comunes de R y S, $A_{k+1} \dots A_n$ los que solo tiene R y $B_{k+1} \dots B_m$ los que solo tiene S.

Como ya se ha dicho, el SQL no solo se basa en el AR, también toma ideas del CRT; en particular, la sintaxis utilizada y el hecho de ser un lenguaje no procedimental, es decir, no se le dice al SGBD cómo realizar la consulta, indicando el orden adecuado de las operaciones y donde buscar cada dato, esto se deja al albedrío del propio intérprete del lenguaje, que además incorpora un optimizar de consultas.

Existen otros lenguajes comerciales que están basados totalmente en el CRT, siendo el más divulgado el QUEL (Query Language) diseñado para el SGBDR INGRES elaborado por la Universidad de Berkeley en California. Hoy en está en desuso.

5.2 Query by Example (QBE).

Este lenguaje fue desarrollado por IBM. Aunque está basado en el CRD, a diferencia de la mayor parte de los lenguajes de consulta de bases de datos o de programación, el QBE tiene una sintaxis bidimensional y además las consultas se realizan indicando un ejemplo de lo que se quiere, en vez de especificar un procedimiento para obtener la respuesta.

Para hacer consultas en QBE se emplean esqueletos de tablas, que son representaciones gráficas del esquema de las relaciones tal como se ve en la figura siguiente:

relación	atributo ₁	...	atributo _n

El usuario elige los esqueletos que necesita para una determinada consulta y los llena con variables y constantes de dominio de forma similar al CRD. Para distinguir las primeras de los

segundos, las variables se preceden de un símbolo de subrayado ('_'). Todos los comandos en QBE acaban en un punto ('.'), por ejemplo, P_x indica que los valores de la variable x se muestran (print). A diferencia del SQL, en QBE se eliminan automáticamente las filas duplicadas.

Anteriormente se indicó que el QBE es un lenguaje bidimensional, esto significa que cuando las condiciones impuestas a las tuplas están conectadas por un *and* se indican en la misma línea del esqueleto de las tablas, mientras que si están unidas por un *or* se colocan en líneas separadas. Algunas veces es conveniente indicar las condiciones fuera de los esqueletos de las tablas, para ello se dispone de un cuadro de condiciones.

Ejemplos:

A continuación mostramos los mismos ejemplos vistos para el CRD en el apartado 3.3:

a) Encontrar la sucursal, n-prestamo, cliente e importe para los préstamos mayores de 300.000 pts:

$$\{ s,t,u,v \mid \text{préstamo}(s,t,u,v) \wedge v > 300000 \}$$

préstamo	sucursal	n-préstamo	cliente	importe
	P._s	P._t	P._u	p._v

condiciones
v > 300000

b) Obtener todos los clientes que tienen préstamos de más de 300.000 pts:

$$\{ t_1,t_2,t_3 \mid \text{cliente}(t_1,t_2,t_3) \wedge \exists s_3s_4(\text{préstamo}(s_1,s_2,s_3,s_4) \wedge t_1=s_3 \wedge s_4 > 300000) \}$$

cliente	nombre	calle	ciudad
	P._t1	P._t2	p._t3

préstamo	sucursal	n-préstamo	cliente	importe
			_t1	> 300000

c) Encontrar a todos los clientes que tienen una cuenta en la sucursal 'Paz', pero que no han sacado un préstamo en esa sucursal:

$$\{ t_1,t_2,t_3 \mid \text{cliente}(t_1,t_2,t_3) \wedge \exists s_1s_3(\text{depósito}(s_1,s_2,s_3,s_4) \wedge t_1=s_3 \wedge s_1='Paz') \wedge \neg \exists u_1u_3(\text{préstamo}(u_1,u_2,u_3,u_4) \wedge t_1=u_3 \wedge u_1='Paz') \}$$

cliente	nombre	calle	ciudad
	P._t1	P._t2	p._t3

depósito	sucursal	n-cuenta	cliente	saldo
	'Paz'		_t ₁	

préstamo	sucursal	n-préstamo	cliente	importe
¬	'Paz'		_t ₁	

donde la negación (¬) debajo de préstamo indica que se buscan aquellas tuplas que no cumplen la condición.

BIBLIOGRAFÍA:

- [CODD72] E. F. Codd
Relational Completeness of Data Base Sublanguages
in 'Data Base Systems'
Prentice-Hall, New Jersey 1972.
- [DATE90] C. J. Date (cap . 14)
Introducción a los Sistemas de Bases de Datos. Vol I (5ª edic.)
Addison-Wesley Iberoamericana, 1993.
- [FERN87] Covadonga Fernández Baizán (cap. 2)
El modelo relacional de datos
Díaz de Santos, Madrid 1987.
- [HURS89] C.J. Hursch & J.L. Hursch (caps. 10 y 11)
SQL. El lenguaje de consulta estructurado.
RAMA, Madrid 1989.
- [KORT87] Henry F. Korth y Abraham Silberschatz (cap. 3)
Fundamentos de Bases de Datos (2ª edición)
McGraw-Hill, 1993.
- [LLOY87] John Wylie Lloyd
Foundations of Logic Programming
Springer-Verlag, Germany 1987
- [ULLM82] Jeffrey D. Ullman (caps. 5 y 6)
Principles of Database Systems (2nd. ed.)
Computer Science Press, USA 1982.
- [VOSS90] Gottfried Vossen (caps. 8, y 16)
Data Models, Database languages and DBMS.
Addison-Wesley, Great Britain 1990.