Capítulo 9. IAGP 2005/06 Lenguajes Comerciales Relacionales en Bases de Datos

Nota: Para este capítulo se usan como ejemplos las relaciones definidas en el capítulo anterior, esto es: cliente, sucursal, prestamo y deposito.

1.- Lenguajes Comerciales más Usados

QBE (Query By Example)

Basado en el cálculo relacional orientado a dominio. Consiste en dar un ejemplo del tipo de tupla que se quiere, con lo que se obtiene un lenguaje muy amigable para el usuario.

Quel (Query Language).

Se desarrolló para el *SBD INGRES* estando basado en el cálculo relacional orientado a tuplas. Cuando nos queremos referir al atributo A de la tupla t, lo hacemos escribiendo t.A.

SQL (Structured Query Language).

Está basado sobre todo en el álgebra relacional. Tiene también facilidades del cálculo relacional, de manera que ciertas tareas pueden ser hechas de varias formas. Es por tanto un lenguaje procedimental y aprocedimental al mismo tiempo. Se creó para *System R*.

2.- SQL (Structured Query Language)

Los lenguajes comerciales permiten definir las estructuras, los limitantes de integridad y los limitantes de seguridad. Es decir incorporan la parte DDL completa y de la parte DML incluyen la modificación de la base de datos. A continuación vamos a estudiar la parte de consulta(DML)del SQL.

Una consulta se expresa mediante tres cláusulas:

SELECT seguida de una lista de atributos que interese obtener como respuesta (SELECT A_1 ,..., A_n).

FROM seguida de la lista de las relaciones que usamos en la consulta (FROM r_1,...,r_m).

WHERE (cláusula opcional) va seguida del predicado de selección (WHERE p).

Nota: El nombre de la cláusula SELECT es engañoso, ya que lo que hace no es una selección, sino una proyección.

Equivalencia entre una consulta en SQL, y en álgebra relacional:

$$\text{SELECT } A_1,\ldots,A_n \quad \text{FROM } r_1,\ldots,r_m \approx \Pi_{A1,\ldots,An}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m)) \quad \text{WHERE } P$$

A continuación se muestran las operaciones posibles en SQL, por medio de ejemplos:

"Seleccionar todos los atributos de préstamo para los préstamos de más de mil €. (*Selección*)

```
SELECT nombre_sucursal, num_prestamo, nombre_cliente, importe
FROM prestamo
WHERE importe > 1000
```

Una de las posibilidades que ofrece SQL, es que cuando se quieren todos los atributos de una relación, en la cláusula SELECT no hace falta que se todos, basta con poner un asterisco (*). De esta manera, la selección anterior quedaría:

```
SELECT *
FROM prestamo
WHERE importe > 1000
```

Ahora realizaremos una consulta para obtener los nombres y las ciudades en que viven de los clientes que tienen un préstamo en la sucursal principal. (*Proyección sobre una selección realizada sobre un producto cartesiano*).

```
SELECT prestamo.nombre_cliente, ciudad
FROM cliente, prestamo
WHERE nombre_sucursal = "Principal" and
prestamo.nombre_cliente = cliente.nombre_cliente
```

SQL incluye también unión, intersección y diferencia. Veamos como podemos realizar dichas operaciones:

"Queremos obtener los nombres de los clientes que tengan cuenta, prestamo o ambas cosas en la sucursal principal." (*Unión*).

```
(SELECT nombre_cliente
FROM prestamo
WHERE nombre_sucursal = "Principal")
```

UNION

```
(SELECT nombre_cliente
FROM deposito
WHERE nombre_sucursal = "Principal")
```

La cláusula usada para la intersección es INTERSECT y la de la diferencia es MINUS.

Aunque en la unión, intersección y diferencia se eliminan los duplicados, no ocurre así con los productos cartesianos en la mayoría de los lenguajes comerciales, si se desea que en SQL no aparezcan duplicados hemos de especificarlo en la cláusula SELECT mediante la orden *distinc* (en algunas versiones antiguas *unique*). Por ejemplo si queremos saber los nombres de los clientes que o tienen cuenta, o tienen depósito o ambas, sin obtener duplicados en el caso en que un cliente tiene cuenta y depósito, podemos hacerlo mediante la unión o bien como sigue:

```
SELECT distinc nombre_cliente
FROM prestamo, deposito
```

Las vistas hasta ahora son básicamente las operaciones de SQL basadas en el álgebra relacional, a continuación veremos algunas basadas en el cálculo relacional. Por ejemplo las operaciones referentes a pertenencia (Cálculo relacional orientado a dominios).

De esta manera podemos expresar una consulta de varias formas: "Queremos conocer los nombres de los clientes que tienen cuenta y préstamo en la sucursal principal".

La 1ª forma basada en el álgebra relacional sería:

```
(SELECT nombre_cliente
FROM prestamo
WHERE nombre_sucursal = "Principal")
INTERSECT
(SELECT nombre_cliente
FROM deposito
WHERE nombre_sucursal = "Principal")
```

Una 2ª forma basada también en el álgebra relacional sería esta otra:

```
SELECT prestamo.nombre_cliente
FROM prestamo, deposito
WHERE prestamo.nombre_cliente = deposito.nombre_cliente and
prestamo.nombre_sucursal = "Principal"
```

Podemos usar una 3ª forma basada en el cálculo relacional, que sería la siguiente:

(IN es el operador de pertenencia).

```
SELECT nombre_cliente
FROM prestamo
WHERE nombre_sucursal = "Principal" and
nombre_cliente IN (SELECT nombre_cliente
FROM deposito
WHERE nombre_sucursal = "Principal")
```

Incluso una 4ª forma:

```
SELECT nombre_cliente
FROM prestamo
WHERE nombre_sucursal="Principal" and <nombre_cliente,
nombre_sucursal>
IN (SELECT nombre_cliente, nombre_sucursal
FROM deposito)
```

SQL, también permite realizar algunas operaciones propias del cálculo relacional orientado a tuplas. Las variables de tupla se definen en la cláusula FROM, la mejor forma de ver esto es por medio de algunos ejemplos:

"Queremos saber los nombres de los clientes que tengan un préstamo en la sucursal principal y sus ciudades".

```
SELECT t.nombre_cliente, ciudad_cliente
FROM cliente t, prestamo s
WHERE t.nombre_cliente = s.nombre_cliente and nombre_sucursal
= "Principal"
```

Nótese que los atributos que son comunes a las relaciones cliente y prestamo, cuando son utilizados ha de ser especificado a cual de las dos relaciones nos estamos refiriendo. En el caso anterior definimos una tupla t de la relación cliente, y una tupla s de la relación prestamo, y nos quedamos con todas aquellas tuplas que existen al mismo tiempo en ambas relaciones y cuyo valor para el atributo nombre_sucursal en la relación prestamo es "Principal".

Hay algunos casos en los que las variables de tupla resultan muy interesantes, como aquellos en los que queremos comparar entre sí tuplas de una misma relación. Veamos un ejemplo: "Nombre de los clientes que tengan una cuenta en la misma sucursal que el Sr. López".

```
SELECT t.nombre_cliente
FROM deposito t, deposito s
WHERE s.nombre_cliente ="Lopez" and t.nombre_sucursal =
s.nombre_sucursal
```

Esta misma consulta podría también ser hecha de esta otra forma:

```
SELECT nombre_cliente
```

```
FROM deposito
WHERE nombre_sucursal IN (SELECT nombre_sucursal
FROM deposito
WHERE nombre_cliente = "Lopez")
```

Otro caso en que las variables de tupla pueden ser de interés es cuando lo que nos interesa no es comprobar si un elemento pertenece a un conjunto, sino comparar un elemento con todos los elementos de un conjunto. Por ejemplo: "Queremos conocer las sucursales que tienen un activo mayor que alguna de las sucursales de Murcia"

```
SELECT t.nombre_sucursal
FROM sucursal t, sucursal s
WHERE t.activo > s.activo and s.ciudad_sucursal = "Murcia"
```

SQL incorpora otras cláusulas para poder hacer esto sin tener que usar el cálculo relacional de tuplas. La cláusula *some* situada delante de un conjunto, se refiere a algún elemento de ese conjunto. De esta manera la consulta anterior quedaría:

```
SELECT nombre_sucursal
FROM sucursal
WHERE activo > some (SELECT activo
FROM sucursal
WHERE ciudad_sucursal = "Murcia")
```

En algunas versiones antiguas en lugar de some podemos encontrar *any*.

Frente a some tenemos la cláusula *all*, que permite comparar un elemento con todos los elementos de un conjunto; un ejemplo de aplicación de esta cláusula podría ser: "Obtener las sucursales cuyo activo sea mayor que todos los activos de las sucursales de Murcia".

```
SELECT nombre_sucursal
FROM sucursal
WHERE activo > all (SELECT activo
FROM sucursal
WHERE ciudad_sucursal = "Murcia")
```

Las cláusulas some y all, nos permiten comparar un valor con un conjunto de valores, pero si lo que queremos es un conjunto de valores con otro, podemos usar la cláusula *contains*, que indica si un conjunto está contenido en otro. Como ejemplo para el uso de esta cláusula sería válido el siguiente: "Encontrar los clientes que tengan una cuenta en todas las sucursales de Murcia". Lo que buscamos es un conjunto de clientes que tienen cuentas en todas las sucursales de un conjunto de sucursales que contiene a todas las sucursales de Murcia.

```
SELECT t.nombre_cliente
FROM deposito t
WHERE (SELECT s.nombre_sucursal
```

```
FROM deposito s
WHERE t.nombre_cliente = s.nombre_cliente) contains
(SELECT nombre_sucursal
FROM sucursal
WHERE ciudad_sucursal = "Murcia")
```

Otra cláusula que incluye SQL es la cláusula *exists*, que devuelve verdadero cuando la subconsulta que se pone detrás devuelva un valor que no sea vacío, y devuelve falso si la subconsulta que se pone detrás devuelve un conjunto vacío. Ejemplo: "Clientes que tengan un préstamo y cuenta en la sucursal principal".

```
SELECT nombre_cliente
FROM cliente
WHERE exists (SELECT *
FROM prestamo
WHERE prestamo.nombre_cliente = cliente.nombre_cliente and
nombre_sucursal = "Principal")
and exists (SELECT * FROM deposito
WHERE deposito.nombre_cliente = cliente.nombre_cliente and
nombre_sucursal = "Principal")
```

SQL nos permite también, obtener el resultado de todas nuestras operaciones por orden (basándose en uno de los atributos de la relación resultante). Con este fin se introdujo la cláusula ORDER BY, que se pone detrás de las 3 cláusulas SELECT, FROM y WHERE, y que debe ir seguida del nombre del atributo por el cual queremos ordenar. Esta cláusula puede ir acompañada por dos modificadores (*asc* o *dec*), según queramos obtener el resultado en orden ascendente o descendente, respectivamente. Estos modificadores irán emplazados detrás del atributo por el cual queremos ordenar nuestra consulta. Ejemplo: "Queremos obtener los nombres de los clientes de Murcia por orden alfabético".

```
SELECT nombre_cliente
FROM cliente
WHERE ciudad_cliente = "Murcia"
ORDER BY nombre_cliente
```

El modificador por defecto es asc.

Si queremos ordenar por varios atributos (es decir, en caso de que nuestra primera elección de atributo fuese igual para dos tuplas, recurriríamos a la segunda elección para saber cual de las tuplas debe salir antes) lo podemos hacer de la siguiente forma:

```
SELECT nombre_cliente
FROM prestamo
ORDER BY nombre_cliente asc, importe dec
```

Con esa orden conseguiríamos obtener los nombres de los clientes que tienen un préstamo ordenados por orden alfabético, pero en caso de que dos clientes

tuviesen el mismo nombre, saldría primero aquel cuyo préstamo fuese más alto.

Además de todo esto, SQL permite hacer una serie de cálculos que no se encuentran en los lenguajes puros, como calcular determinadas funciones para grupos de tuplas esas funciones son: *avg* (calcula la media), *max* (calcula el máximo), *min* (calcula el mínimo), ***sum*** (calcula la suma), ***count*** (cuenta el número de elementos del grupo).

Existe una cláusula para agrupar tuplas según un determinado atributo que es **GROUP BY**. Como ejemplo de esta cláusula podría servir el siguiente: "Queremos el saldo medio de las cuentas en cada una de las sucursales". Lo que tenemos que hacer es agrupar los depósitos por nombre de sucursal, y calcular el saldo medio de esos grupos.

```
SELECT nombre_sucursal, avg (saldo)
FROM deposito
GROUP BY nombre_sucursal
```

Veamos ahora como se podría contar el número de tuplas de cliente:

```
SELECT count (*)
FROM cliente
```

Hay casos en los que las condiciones que imponemos, nos interesa que las cumpla un grupo de tuplas, y no una sola, en ese caso usamos la cláusula **HAVING**.

Ej: "Queremos obtener los préstamos en la sucursal principal agrupados por nombre de cliente cuya media (la de los préstamos de un solo cliente) supera las 10000 euros".

```
SELECT avg (importe)
FROM prestamo
WHERE nombre_sucursal = "Principal"
GROUP BY nombre_cliente
HAVING avg (importe) > 10000
```

El orden de las cláusulas sería: SELECT, FROM, WHERE, ORDER BY, GROUP BY y HAVING.

La cláusula BETWEEN sirve para hacer comparaciones entre un rango de valores, es decir:

```
SELECT nombre_cliente
FROM deposito
WHERE saldo ≥ 100.000 and saldo ≤ 1.000.000
```

es equivalente a:

```
SELECT nombre_cliente
FROM deposito
WHERE saldo BETWEEN 100.000 and 1.000.000
```

SQL ofrece también una cláusula para tratar cadenas de caracteres, se trata de la cláusula ***like***. Cuando usamos esta cláusula, el símbolo % en una cadena es un comodín de 0, 1, o más caracteres, y el carácter _ es un comodín de un carácter (equivalentes al * y la ? de MS-DOS respectivamente). Ejemplo en el que se usa esta cláusula:

```
SELECT nombre_cliente
FROM cliente
WHERE ciudad_cliente like "Mur%"
```

De esta manera obtenemos los nombres de todos los clientes cuyo atributo ciudad_cliente empiece por Mur. Si queremos que en una cadena aparezca uno de los dos caracteres comodines, tendremos que ponerlos con el carácter \ delante, que servirá de carácter de escape. Si por ejemplo ponemos like "ANT\%2" estaremos comparando con la cadena "ANT%2".

Hasta ahora hemos estado viendo la parte de consulta del DML de SQL, a continuación estudiaremos la parte de actualización.

Las operaciones de actualización son: inserción, modificación y actualización. Las iremos viendo una por una:

A) Eliminación. La sintaxis a seguir para la eliminación es la siguiente:

```
DELETE r
WHERE p
```

donde r es la relación de la que queremos borrar tuplas, y p es el predicado que deben de cumplir las tuplas que queremos borrar. Por ejemplo, para borrar los préstamos de Pepito deberíamos hacer:

```
DELETE prestamo
WHERE nombre_cliente = "Pepito"
```

SQL adolece de que no cumple la integridad de referencia, es decir, que si dos relaciones tienen en común un atributo que en una de las relaciones es clave primaria, y en la otra es clave ajena, este atributo debería ser igual en ambas relaciones en todo momento, sin embargo en SQL esto no es así, ya que cuando borras una tupla en una relación, si tiene un atributo (clave primaria) en común en otra relación, éste no se ve modificado.

B) Modificación. Lo veremos mediante un ejemplo: "Queremos meter a cada cliente de la sucursal 10, 3 €más en su cuenta".

```
UPDATE deposito
SET saldo = saldo + 3
```

```
WHERE nombre_sucursal = "10"
```

C)Inserción. Tiene dos variantes: En primer lugar insertar en una relación tuplas una a una, o la segunda, insertar conjuntos de tuplas enteras que son resultado de una operación SELECT.

- Si queremos insertar una sola tupla en una relación lo haremos siguiendo el siguiente ejemplo:

```
INSERT INTO prestamo
VALUES (102, "Principal", "Manolo", 2.000.000)
```

Los valores se deben introducir en el orden correcto en que están los atributos en la relación.

- Cuando queremos insertar un conjunto de tuplas tendremos que seguir el siguiente ejemplo en el que abrimos a todo aquel que tenga un préstamo en la sucursal principal una cuenta con 1 € en esta misma sucursal, y a dicha cuenta le damos como número el mismo que tenía el préstamo.

```
INSERT INTO deposito
SELECT num_prestamo, nombre_cliente, nombre_sucursal, 1
FROM prestamo
```

En este caso también hay que insertar los atributos en el orden correcto, a no ser que detrás de deposito, pusiésemos entre paréntesis el orden en que vamos a introducir los atributos, es decir que la anterior orden sería equivalente a:

```
INSERT INTO deposito (saldo, num_cuenta, nombre_sucursal,
nombre_cliente)
SELECT 100, num_prestamo, nombre_sucursal, nombre_cliente
FROM prestamo
```

3.- QUEL

Es un lenguaje comercial que se desarrolló para INGRES, está basado en el cálculo relacional de tuplas.

A) Consultas.

La estructura general de una consulta es la siguiente:

```
RANGE OF t1 IS r1
RANGE OF t2 IS r2
.....................................
.....................................
RANGE OF tm IS rm
```

```
RETRIEVE (t_i1.A_j1, t_i2.A_j2, ........, t_im.A_jm)
WHERE P
```

donde t_1,.....,t_m son las tuplas que usamos para la consulta, r_1,...,r_m son las relaciones correspondientes a t_1,...,t_m. La cláusula RETRIEVE es equivalente a la cláusula SELECT de SQL, y P es el predicado de selección.

Ejemplo: Obtener todos los clientes que tienen cuenta en la sucursal principal.

```
RANGE OF t IS deposito RETRIEVE (t.nombre_cliente) WHERE
t.nombre_sucursal = "Principal".
```

Como ejemplo de una consulta en la que aparezcan dos tuplas de dos relaciones distintas podríamos usar el siguiente: " Obtener todos los nombres de los clientes y sus ciudades de residencia que tienen préstamo en la sucursal principal."

```
RANGE OF t IS cliente RANGE OF s IS prestamo RETRIEVE
(s.nombre_cliente, t.ciudad_cliente) WHERE s.nombre_sucursal =
"Principal" AND t.nombre_cliente = s.nombre_cliente
```

QUEL no incluye eliminación de duplicados por defecto, si se quiere conseguir ésta, es necesario indicarlo mediante la cláusula UNIQUE.

```
RANGE ............. RETRIEVE UNIQUE (.............) WHERE
.......
```

Así mismo QUEL tampoco incluye las operaciones de unión, intersección y diferencia , como tampoco permite subconsultas anidadas, por lo que es un poco menos amigable que SQL.

Sin embargo, QUEL sí incluye **operaciones de grupo**. éstas pueden aparecer en la cláusula RETRIEVE o bien en la cláusula WHERE. Las posibles operaciones son las siguientes: *count, sum, max, min, avg, any*.

```
Sintaxis: Operación (t.A WHERE P)
```

Ej: "Obtener la media de los saldos de las cuentas de la sucursal principal."

```
RANGE OF t IS deposito
RETRIEVE avg (t.saldo WHERE t.nombre_sucursal = "Principal")
```

Existe una variante: Operación ($t.A_i$ by $t.A_j$) ,donde $t.A_j$ es la condición de agrupamiento.

Ej: "Media de los saldos agrupadas por nombre de sucursal."

```
RANGE OF t IS deposito
RETRIEVE avg (t.saldo by t.nombre_sucursal)
```

Ej: "Números de cuenta con saldo mayor que el saldo medio de la sucursal a la que pertenecen."

```
RANGE OF t IS deposito
RETRIEVE (t.num_cuenta)
WHERE t.saldo > avg (t.saldo by t.nombre_sucursal).
```

B) Inserción.
- Para la inserción de tuplas individuales:

```
APPEND TO deposito (lista de atributos con valores nuevos).
```

Ej: APPEND TO deposito (num_cuenta = 287, nombre_sucursal = "Murcia"....)

- A continuación podemos ver un ejemplo de inserción de tuplas de una relación en otra relación.

```
RANGE OF t IS prestamo
APPEND TO temp (t.nombre_cliente)
WHERE t.nombre_sucursal = "Principal"
```

Donde temp quedará formada como una relación cuyo único atributo es nombre_cliente. Si temp ya existiera cuando se realizó la inserción, entonces los atributos que tenga temp deben coincidir con los atributos que insertamos en la cláusula APPEND.

C) Modificación.

Por su sencillez, veremos la modificación mediante un ejemplo por el cual aumentaremos en un 5% los saldos de la relación deposito.

```
RANGE OF t IS deposito
REPLACE t (saldo = t.saldo x 1.05)
```

D) Eliminación.

Para la eliminación usamos el comando DELETE que borra tuplas completas.

Ej: Eliminar todas las tuplas de prestamo para todos los clientes cuyo nombre sea López.

```
RANGE OF t IS prestamo
DELETE (t)
WHERE nombre_cliente = "López".
```

4.- QBE (Query By Example)

Es un lenguaje comercial desarrollado por IBM y basado en el cálculo relacional de dominios. En él las consultas se hacen por medio de ejemplos, para ello se usan unas tablas que son "*esqueletos*" de relaciones. El sistema generaliza los ejemplos.

A) Consultas.

Para una consulta, el usuario solicita un esqueleto de la relación sobre la que quiere realizar la consulta, y lo rellena con *columnas muestra*, que pueden incluir constantes o variables de dominio. Los nombres de variable van precedidos del carácter de subrayado '_'.

Ejemplo: Obtener todos los nombres de los clientes que tienen prestamo en la sucursal principal

préstamo	num_prestamo	nombre_sucursal	nombre_cliente	importe
		Principal	P.ALL. _X	

El comando P bajo la columna nombre_cliente indica que se muestren los valores, y com podemos ver, delante de la variable X, hemos introducido el carácter de subrayado, como sólo queremos ver los clientes que tienen préstamo en la sucursal principal, ponemos la constante Principal en la columna nombre_sucursal. Por último veamos el significado del comando ALL: QBE sí elimina los duplicados por defecto, por tanto si no queremos que los elimine tendremos que usar el comando ALL.

Si queremos realizar una consulta en la que empleemos dos relaciones,tes/ _la que es

sula que préstamo prestamo nombre_clie4.603430546 0cay lclie7.897530546 0ciudadimporte

B)**Operaciones de Grupo.**

Las operaciones posibles son: CNT, AVG, MIN, MAX, SUM.

Veamos un ejemplo: Obtener la media de los saldos por nombre de sucursal.

deposito	num_cuenta	nombre_sucursal	nombre_cliente	saldo
		P.G._X		P.ALL.AVG._Y

Mediante el comando G, conseguimos agrupar por nombre de sucursal, mientras que el comando AVG, nos da la media aritmética del salso de cada uno de esos grupos.

C) Inserción.

Para la inserción usamos el comando I, que pondremos en la tabla bajo el nombre de la relación.

- Si queremos hacer una inserción individual:

deposito	num_cuenta	nombre_sucursal	nombre_cliente	saldo
I.	207	Principal	García	2.000.000

Para una inserción de conjunto de tuplas, necesitamos dos esqueletos. Si por ejemplo, queremos insertar en una relación ***temp*** los nombres de todos aquellos clientes que tienen un préstamo en la sucursal principal, haríamos lo siguiente:

prestamo	num_prestamo	nombre_sucursal	nombre_cliente	importe
		Principal	_X	

temp	nombre_cliente
I.	_X

D)Modificación.

Para las modificaciones se usa el comando U.

Ejemplo: Multiplicar por 2 los importes de los préstamos.

prestamo	num_prestamo	nombre_sucursal	nombre_cliente	importe
	_X			_Y

	_X			U._Y * 2

E) Eliminación.

Para eliminar tuplas usamos el comando D, colocándolo en la columna del nombre de la relación.

Ejemplo: Eliminar todas las tuplas de la relación cliente, para las que el nombre del cliente sea "López"

cliente	nombre_cliente	calle	ciudad
D.	Lopez		