Departamento de Cs. e Ingeniería de la Computación

Universidad Nacional del Sur

Elementos de Bases de Datos

Segundo Cuatrimestre de 2014

Clase Práctica: **Cálculo Relacional de Tuplas (CRT)**

Notación

y	$\wedge$
o	$\vee$
existe X en R	$\exists x \in R$
para todo X en R	$\forall x \in R$
implica	$\rightarrow$
negación lógica	$\neg$

1. Algunos conceptos básicos

Una expresión CRT tiene la forma $\{t|P(t)\}$, donde t es una variable de tupla y $P(t)$ es un predicado. Por lo general, en una expresión CRT la única variable libre es t (variable de tupla), *las demás deben estár ligadas*, es decir, asociadas a algún cuantificador ($\exists$ o $\forall$).

Dependiendo del formato de las tuplas a generar, las expresiones en CRT pueden tener una de estas formas:

1. $\{t \in R|P(t)\}$: Utilizada cuando se desea que las tuplas t contengan todos los atributos presentes en el esquema de la relación R.

2. $\{t|P(t) \wedge t[a_1] = v_1 \wedge \ldots \wedge t[a_i] = v_i\}$: Utilizada cuando se desea que las tuplas t contengan ciertos atributos específicos, pudiendo estos ser constantes o atributos presentes en diferentes esquemas de relación utilizados en P. Es necesario que al momento de realizar la asignación a la tupla t, los valores aún se encuentren ligados, por ello, deberá realizarse la asignación dentro del ámbito (scope) del cuantificador asociado a dicha variable.

1.1. Cuantificadores

Si existen múltiples cuantificadores en una misma expresión, puede ser que ninguno de ellos se encuentre dentro del ambiente (scope) de otro cuantificador o que los cuantificadores se encuentren anidados (uno ambiente dentro del otro ambiente). Un cuantificador *no* puede extender su alcance por fuera del ambiente que lo contiene.

Si consideramos que los cuantificadores pueden estar anidados, el orden en que se encuentran escritos en la expresión es importante y sólo bajo ciertas condiciones se podrán intercambiar el orden de los cuantificadores en forma segura sin afectar la semántica de la expresión.

1.1.1. Orden de los cuantificadores

Consideremos los siguientes predicados considerando que S y T representan conjuntos.

1. $\forall x \in S(\exists y \in S(y = x))$: Para todo (o para cada) elemento x de S existe un elemento y de S que verifica que x e y son iguales.

2. $\exists y \in S(\forall x \in S(y = x))$: Existe un elemento y en S tal que para todo (o para cualquier) elemento x en S se verifica que x e y son iguales.

3. $\forall x \in S(\exists y \in T(y = x))$: Para todo elemento x de S existe un elemento y en T que verifica que x e y son iguales.

4. $\forall x \in T(\exists y \in S(y = x))$: Para todo elemento x de T existe un elemento y en S que verifica que x e y son iguales.

El valor de verdad de los predicados anteriores puede intentar determinarse cualquiera fuera el valor de S y T, en cuyo caso estaríamos hablando de tautologías o contradicciones; o podría determinarse el valor de verdad para valores de S y t específicos, por lo que estaríamos utilizando una interpretación particular. Es importante recordar que para que un predicado sea una tautología, éste debe ser verdadero para cualquier interpretación.

Consideremos una posible interpretación: Dados los conjuntos $S = \{1, 2, 3, 4\}$ y $T = \{1, 2\}$ el significado de los siguientes predicados es: Si evaluamos el valor de verdad de los predicados:

1. **Verdadero**. Ya que, sea $v \in S$ el valor de x (cualquiera sea) entonces podemos asegurar que $\exists y \in S$ que hace que $x = y$ y ese valor es el mismo v, es decir, tomamos el mismo elemento de S como x e y.

2. **Falso**. Cuando y toma un valor sólo existe un único valor en S que sea igual a y. Por ejemplo, si $y = 1$ no es cierto que si le asigno a x cada valor de S entonces $x = y$, ya que si $x = 2$ entonces $x \neq y$.

3. **Falso**. Contraejemplo, si $x = 3$ entonces no existe y en T que sea igual a x.

4. **Verdadero**. $T \subset S$.

1.1.2. Intercambio de Cuantificadores

Se pueden intercambiar en forma segura cuantificadores sucesivos si se cumplen ambas condiciones siguientes:

- Ambos cuantificadores son del mismo tipo.
- Ninguno de los conjuntos sobre los que se cuantifican dependen de una variable ligada por el otro conjunto.

1.2. Equivalencias

$$(P(x) \to Q(x)) \iff (\neg P(x) \vee Q(x))$$

$$\neg(P(x) \wedge Q(x)) \iff (\neg P(x) \vee \neg Q(x))$$
$$\neg(P(x) \vee Q(x)) \iff (\neg P(x) \wedge \neg Q(x))$$

$$\neg\exists x \in S(P(x)) \iff \forall x \in S(\neg P(x))$$
$$\neg\forall x \in S(P(x)) \iff \exists x \in S(\neg P(x))$$
$$\exists x \in S(P(x)) \iff \neg\forall x \in S(\neg P(x))$$
$$\forall x \in S(P(x)) \iff \neg\exists x \in S(\neg P(x))$$

2. Ejemplos

Consideremos el siguiente modelo relacional:

- **Proyectos** (cod_proy, cod_cliente, fecha_inicio, tipo)
- **Personal** (cod_func, nombre, fecha_ingreso)
- **Tareas** (cod_tareas, descrip, tipo)
- **Asignacion** (cod_func, cod_proy, cod_tarea)
- **Registro_Horas** (cod_func, cod_proy, fecha, cant_horas)

1. Tareas de tipo '*Desarrollo*'. (**Selección** $\sigma_{tipo='Desarrollo'}(Tareas)$)

$$\{t \in Tareas \mid t[tipo] = 'Desarrollo'\}$$

2. Nombre de funcionarios que ingresaron a partir del 1/1/2009. (**Proyección** $\pi_{nombre}(\ldots)$)

$$\{t | \exists f \in Personal(f[fecha\_ingreso] \geq '1/1/2009' \wedge t[nombre] = f[nombre])\}$$

Importante: Observe que la expresión $\{t|\exists f \in Personal(f[fecha\_ingreso] \geq '1/1/2009') \wedge t[nombre] = f[nombre]\}$ no es equivalente porque el ambito del $\exists f$ tiene alcance hasta el cierre del paréntesis y por lo tanto $f[nombre]$ no tiene valor por ser una variable libre (está fuera del ambito).

3. Personal asignado en algún proyecto. (**Join**)

$$\{t \in Personal | \exists a \in Asignacion(t[cod\_func] = a[cod\_func])\}$$

4. Personal que no esté asignado en algún proyecto.

$$\{t \in Personal | \neg\exists a \in Asignacion(t[cod\_func] = a[cod\_func])\}$$

La lectura de la expresión anterior sería: las tuplas t de $Personal$ tal que no existe en $Asignacion$ una tupla a que tenga el mismo cod_func que t.

$$\{t \in Personal | \forall a \in Asignacion(t[cod\_func] \neq a[cod\_func])\}$$

La lectura de la expresión anterior sería: las tuplas t de $Personal$ tal que para toda tupla de $Asignacion$, ninguna de ellas tiene un cod_func igual al de t.

5. Personal que *sólo* registren horas a partir del 1/1/2000. (**Implica**)

$$\{p \in Personal | \forall r \in Registro\_Horas(r[cod\_func] = p[cod\_func] \rightarrow r[fecha] \geq '1/1/2000'))\}$$

Observe que la siguiente consulta no es equivalente:

$$\{p \in Personal | \forall r \in Registro\_Horas(r[cod\_func] = p[cod\_func] \wedge r[fecha] \geq '1/1/2000'))\}$$

ya que suponga que existen registros en $Registro\_Horas$ para al menos dos funcionarios, f_1 y f_2, entonces la primer parte de la expresión ($\forall r \in Registro\_Horas(r[cod\_func] = p[cod\_func])$ nunca sería verdadera porque siempre existe al menos una tupla en la cual el código del funcionario es diferente y por lo tanto la expresión booleana se hace falsa, ya que es un $\wedge$.

En todo caso, una consulta equivalente (por reescritura) sería:

$$\{p \in Personal | \neg\exists r \in Registro\_Horas(r[cod\_func] = p[cod\_func] \wedge r[fecha] < '1/1/2000'))\}$$

P	Q	$P \wedge Q$	$P \rightarrow Q$	$\neg P \vee Q$
V	V	V	V	V
V	F	F	F	F
F	V	F	V	V
F	F	F	V	V

6. (**Ejercicio**) Proyectos en los que sus funcionarios sólo hayan registrado más de 8 horas de trabajo.

7. Código de los funcionarios que sólo registren horas a partir del 1/1/2000.

$$\{t | \exists p \in Personal(p[cod\_func] = t[cod\_func] \wedge \forall r \in Registro\_Horas(r[cod\_func] = p[cod\_func] \rightarrow r[fecha] \geq '1/1/2000'))\}$$

$$\{t | \exists p \in Personal(p[cod\_func] = t[cod\_func] \wedge \neg\exists r \in Registro\_Horas(r[cod\_func] = p[cod\_func] \wedge r[fecha] < '1/1/2000'))\}$$

8. Código de los funcionarios que no están asignados a proyectos del tipo '*Desarrollo*'.

$$\{t | \exists f \in Personal(\forall a \in Asignacion(f[cod\_func] = a[cod\_func] \rightarrow \neg\exists p \in Proyectos(a[cod\_proy] = p[cod\_proy] \wedge p[tipo] = 'Desarrollo')) \wedge t[cod\_func] = f[cod\_func])\}$$

$$\{t | \exists f \in Personal(\forall a \in Asignacion(f[cod\_func] = a[cod\_func] \rightarrow \forall p \in Proyectos(a[cod\_proy] = p[cod\_proy] \rightarrow p[tipo] \neq 'Desarrollo')) \wedge t[cod\_func] = f[cod\_func])\}$$

$$\{t | \exists f \in Personal(\neg\exists a \in Asignacion(f[cod\_func] = a[cod\_func] \wedge \exists p \in Proyectos(a[cod\_proy] = p[cod\_proy] \wedge p[tipo] = 'Desarrollo')) \wedge t[cod\_func] = f[cod\_func])\}$$

Observe que en lugar de utilizar $\exists f \in Personal$ podría haberse utilizado la expresión $\exists f \in Asignacion$. Sin embargo, la diferencia en el resultado estaría dada si hubiese algún funcionario que esté en $Personal$ pero que no estuviese presente en $Asignacion$.

9. Código de los proyectos en los que todas las personas asignadas al mismo ingresaron el año 2010, verificando que tengan personas asignadas.

$$\begin{aligned}\{t \mid \exists p \in Proyectos(& \\ & \forall a \in Asignacion(\\ & \quad p[cod\_proy] = a[cod\_proy] \rightarrow \\ & \quad \exists f \in Personal(\\ & \quad\quad a[cod\_func] = f[cod\_func] \\ & \quad\quad \wedge f[fecha\_ingreso] = '2010')) \\ & \wedge \exists b \in Asignacion(p[cod\_proy] = b[cod\_proy] \\ & \wedge t[cod\_proy] = p[cod\_proy])\}\end{aligned}$$

10. (**Ejercicio**) Obtener el código de los funcionarios que no están asignados a proyectos del tipo '*Desarrollo*', pero sí a tareas de tipo '*Desarrollo*'.

11. (**Ejercicio**) Obtener el nombre de los funcionarios tales que en todos los proyectos en los que están asignados, tienen tareas de tipo '*Gestión*'.