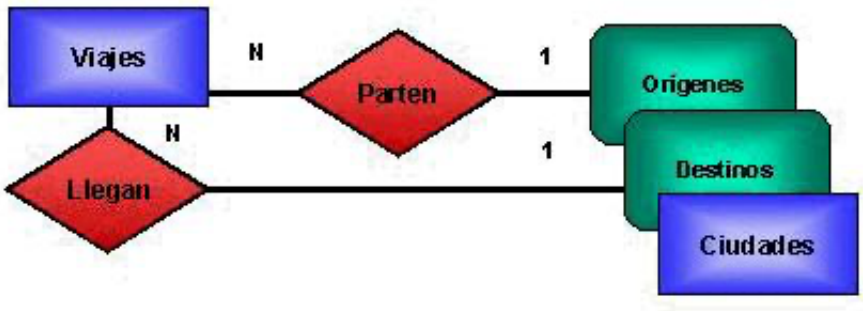
TC2005B Construcción de software y toma de decisiones

Consultas en SQL Usando Roles y Consultas

Consultas con roles.

En ocasiones se presenta la necesidad de hacer consultas que involucran a una misma tabla más de una vez, ya sea para comparar entre sí subconjuntos de tuplas de la tabla o bien para que ésta juegue más de un papel en la consulta.

Tomaremos como primer ejemplo el siguiente modelo Entidad - Relación.



De acuerdo al procedimiento de **traslado** del modelo Entidad-Relación, las tablas derivadas del ejemplo serían

viajes(idviaje, idorigen, iddestino, fecha)
ciudades(idciudad, nombreciudad)

Nota: idorigen e iddestino son valores que pertenecen al dominio de idciudad, heredados para instrumentar las asociaciones Parten y Llegan, jugando los dos roles necesarios en la tabla que representa a la entidad viajes.

Al plantear la consulta "identificador del viaje, nombre de la ciudad de origen, nombre de la ciudad de destino y fecha" utilizando SQL, se presenta la siguiente dificultad: ciudades debe aparecer más de una vez en la cláusula FROM, con la consecuente ambigüedad.

¿Cómo resolver esto?
De manera dinámica, es posible utilizar alias para denotar los roles que la tabla ciudades juega:

```
SELECT idviaje, origen.nombre, destino.nombre, fecha
FROM viajes, ciudades origen, ciudades destino
```

```
WHERE viajes.idorigen = origen.idciudad AND
      viajes.iddestino = destino.idciudad
```

Adicionalmente, es posible hacer definiciones alternativas permanentes de las tablas usando la sentencia:

```
CREATE SYNONYM sinónimo FOR nombretabla
```

Un sinónimo no es una copia de la tabla sino un nombre alternativo para ser utilizado en consultas. En nuestro caso sería de utilidad crear los sinónimos:

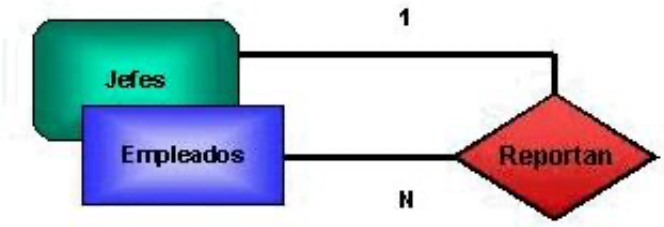
```
CREATE SYNONYM origen FOR ciudades
CREATE SYNONYM destino FOR ciudades
```

y posteriormente podríamos plantear la consulta anterior sencillamente como:

```
SELECT idviaje, origen.nombre, destino.nombre, fecha
FROM viajes, origen, destino
WHERE viajes.idorigen = origen.idciudad AND
      viajes.iddestino = destino.idciudad
```

Nota: en este caso no utilizamos alias sino el sinónimo en sustitución de la tabla ciudades.

Consideremos ahora el ejemplo:



En este caso la tabla que instrumenta el modelo es:

empleados(idempleado, nombre, idjefe)

En la que la asociación **Reportan** se instrumenta a través de la inclusión de idjefe en la tabla, que proviene precisamente del dominio de **idempleado**.

Para plantear la consulta "Nombre del empleado, nombre del jefe" en SQL nos enfrentamos a un problema similar al anterior, sólo que en esta ocasión sólo hay una tabla. La solución a este requerimiento utilizando **alias** es:

```
SELECT e.nombre empleado, j.nombre jefe
FROM empleados e, empleados j
WHERE e.idjefe=j.idempleado
```

En este caso se usan los alias de tabla **e** para **empleados** y **j** para **jefes**. Obsérvese que se incluyen los **alias** de **columna** empleado y jefe para que en los encabezados de las columnas de los resultados aparezcan estas leyendas sobre los nombres de los empleados y los jefes respectivamente.

Si definiésemos el sinónimo:

```
CREATE SYNONYM jefes FOR empleados
```

Podríamos expresar la consulta anterior como:

```
SELECT empleados.nombre empleado,jefes.nombre jefe
FROM empleados,jefes
WHERE empleados.idjefe=jefes.idempleado
```

Supongamos que tenemos ahora las tablas:

ventasdiarias(idproducto, fecha, cantidad)
productos(idproducto, descripción, precio)

y que queremos obtener la diferencia entre las ventas de cada producto entre los días 1/09/2000 y 2/09/2000, necesitamos renombrar en la consulta a la tabla **ventasdiarias** a fin de poder referirnos a ella más de una vez:

```
SELECT p.idproducto,p.descripcion,vprimero.cantidad-vsegundo.cantidad diferencia
FROM productos p,ventasdiarias vprimero,ventasdiarias vsegundo
WHERE p.idproducto=vprimero.idproducto AND
    p.idproducto=vsegundo.idproducto AND
    vprimero.fecha = '1-SEP-00' AND
    vsegundo.fecha = '2-SEP-00'
```

Subconsultas

En algunas ocasiones requerimos utilizar el resultado de una consulta como criterio de selección. Un primer uso es el de descubrir tuplas no asociadas. Considera los esquemas:

ventasdiarias(idproducto, fecha, cantidad)
productos(idproducto, descripción, precio)

¿Cómo encontrar qué productos no se han vendido?

Una posibilidad es utilizar el operador diferencia:

```
(SELECT idproducto FROM productos)
minus
(SELECT idproducto FROM ventasdiarias)
```

Una segunda forma de plantear este problema es mediante una relación de pertenencia, esto es podemos replantear el problema como:

"Encontrar todos los productos que no pertenecen al conjunto de los productos vendidos"

Esto puede plantearse utilizando una subconsulta como criterio de búsqueda:

```
SELECT idproducto FROM productos
WHERE idproducto NOT IN (SELECT idproducto FROM ventasdiarias)
```

El operador IN establece una relación de pertenencia de conjuntos y desde luego NOT IN puede entenderse como "no pertenencia".
Otra forma de plantear esta consulta es enfocándola como
"Encontrar los productos para los que no existen ventas"
que se expresa utilizando una subconsulta basada en el operador EXISTS.

```
SELECT idproducto FROM productos p
WHERE NOT EXISTS
  (SELECT * FROM ventasdiarias v WHERE v.idproducto = p.idproducto)
```

Observa que en este caso es necesario utilizar alias para evitar la ambigüedad en la columna idproducto de la subconsulta.

Las subconsultas también pueden ser utilizadas, con ciertas reglas en lugar de una expresión.

Por ejemplo para obtener el identificador y la descripción de los productos de los que se han vendido más de 1,000,000 pesos, puede plantearse la siguiente consulta que incluye una subconsulta para calcular el importe vendido de cada producto:

```
SELECT idproducto,descripcion FROM productos p WHERE 100000 < (SELECT sum(v.cantidad*p.precio) FROM ventasdiarias v WHERE v.idproducto=p.idproducto)
```

Observa que en este caso también utilizamos un alias y el precio utilizado en la subconsulta proviene de la tupla analizada de la consulta principal.

Por otra parte, el resultado de la subconsulta se usa para comparar con una constante, como cualquier otra expresión, pero en este caso la constante debe ponerse primero, por lo que para restringir los productos con ventas mayores a 100000, escribimos "que 100000 sea menor al importe total".

La sintaxis general para utilizar subconsultas en una consulta principal es:

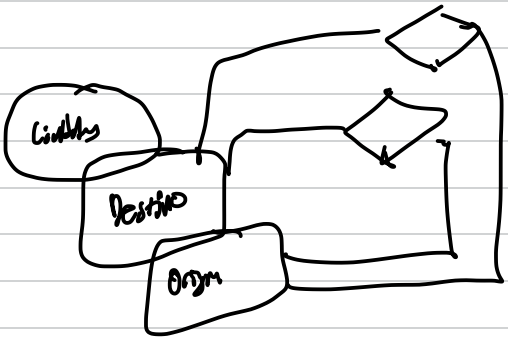
```
SELECT columnas FROM tablas WHERE condicionsubconsulta
```

en donde las alternativas para la condición de subconsulta son:
condicionsubconsulta :
expresión [NOT] IN (SELECT columnas FROM tablas WHERE condiciones)
|
expresión {> | < | = | != | <> | >= | <= } [{ANY | SOME | ALL }]
(SELECT columnas FROM tablas WHERE condiciones)
|
[NOT] EXISTS
(SELECT * FROM tablas WHERE condiciones)

Donde el texto en azul indica los elementos que realmente aparecen en las sentencias y los símbolos tienen el siguiente significado:

[] : elemento opcional {} : lista de alternativas

Powered by Materialize.



@NOT { IN / EXIST

Subconsulta => SELECT

(SELECT ...)

1.

```
SELECT nombre
FROM Actor
WHERE sexo = 'F'
AND nombre IN (SELECT nombre
               FROM Pelicula
               WHERE titulo = 'Los brujos de Salem')
```

2.

```
SELECT nombre
FROM Pelicula
WHERE año = 1995
AND titulo IN ( SELECT titulo
                FROM Peliculas
                WHERE nombre = 'Hugh')
```

3.

```
SELECT titulo, año
FROM Peliculas
WHERE duracion > (SELECT duracion
                  FROM Peliculas
                  WHERE titulo = 'Lo que el viento se llevó')
                  AND año = 1939)
```

4.

```
SELECT nombre
FROM Productor PD, Relación P
WHERE PD.idproductor = P.idproductor
HAVING COUNT(*) > (SELECT COUNT(*)
                   FROM Productor PD, Relación P.
                   WHERE PD.idproductor = P.idproductor.
                   AND nombre = 'Gregg Lucas')
```

5.

```
SELECT nombre
FROM Productor
WHERE idproductor IN (SELECT idproductor
                      FROM Relación
                      WHERE idfilm IN (SELECT idfilm
                                       FROM Film
                                       WHERE nombre = 'Star
                                       Wars'))
```

6.

```
WHERE 1 < (SELECT COUNT (*)
           FROM Reviews
           GROUP BY movie_id)
```