SQL

SQL Intermedio

L. en C.C. Odín M. Escorza Soria

UNAM

Motivación

¿Cómo calculamos una lista del nombre, apellidos, sexo y el estado (donde viven) de todos los estudiantes nacidos en enero de 1984? Versión con producto cartesiano (`implicit join`)

```
select p.nombre,p.ap1,p.ap2,p.sexo,edo.nombre as edo
  from persona as p,domicilio as d,asentamiento as a,municipio m,estado as edo,ocupacion as
       o
 where p.idDomicilio = d.idDomicilio
       and d.idAsentamiento = a.idAsentamiento
       and a.idMunicipio = m.idMunicipio
       AND m.idEstado = edo.idEstado
       AND o.nombre = 'Estudiante'
       AND fnac between '1984-01-01' and '1984-01-31'
```

y la versión con `explicit join` es:

```
SELECT p.nombre,p.ap1,p.ap2,p.sexo,edo.nombre as edo
  FROM persona as p
       join ocupacion o using (idOcupacion)
       join domicilio as d using (idDomicilio)
       join asentamiento as a using (idAsentamiento)
       join municipio as m using (idMunicipio)
       join estado as edo using (idEstado)
 WHERE o.nombre = 'Estudiante'
       AND fnac between '1984-01-01' and '1984-01-31'
```

Ventajas

- Facilita distinguir entre condiciones “de unión” y condiciones de filtrado.
- Previene inclusión accidental de tablas sin condición de unión.
- Si no hay optimización de consultas, el SMBD podría hacer el producto cartesiano completo.

Tipos


```
SELECT ... FROM <t1> JOIN <t2> ON (<cond>)

SELECT ... FROM <t1> LEFT JOIN <t2> ON (<cond>)

SELECT ... FROM <t1> RIGHT JOIN <t2> On (<cond>)

SELECT ... FROM <t1> FULL JOIN <t2> On (<cond>)
```

Características

- Son como "tablas de mentira" para el SMBD.
- Salvo cuando son materializadas, se ejecutan cada vez que se hace referencia a ellas.
- No permiten actualización de datos (salvo en casos muy particulares).
- Permiten hacer más granular el control de acceso a los datos.
- Simplifican consultas grandes.
- Facilitan darle formato a los resultados de las consultas.
- Promueven el rehuso de código SQL

Gestión

Para crear o eliminar vistas en un SMBD...

```
CREATE VIEW <nombre> AS <expr>

DROP VIEW <nombre>
```

Motivación

El jefe desea saber el número de personas que nacieron entre el 22 de enero del 94 y el 23 de enero del 2010... ¡fácil!

```
select count(*)
from persona
where fnac between '1994-01-22' and '2010-01-23'
```

...10min después... ¡Ups!, era apartir del 2 de septiembre del 93...

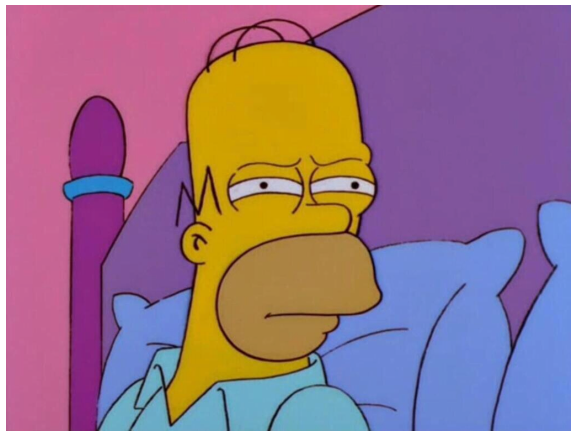
```
select count(*)
from persona
where fnac between '1993-09-02' and '2010-01-23'
```

...5min después... jejeje, perdón, que mejor entre el 2 de marzo de 1976 y hoy...

```
select count(*)
from persona
where fnac between '1975-03-02' and now()
```

Motivación

... un momento...



¿qué tal si hubiera algo como:
"`select cuentacuantos('1975-03-02',now());`"?

Implementación

PostgreSQL

```
CREATE FUNCTION cuentacuantos(f1 DATE, f2 DATE) RETURNS BIGINT AS
  select count(*)
  from persona
  where fnac between f1 and f2;
 LANGUAGE SQL;
```

y para usarla, sólo hay que afinar algunos “detalles”:

```
select cuentacuantos('1976-03-02'::date,now()::date);
```

MySQL

```
DELIMITER
CREATE PROCEDURE cuentacuantos(f1 DATE,f2 DATE)
 BEGIN
  select count(*)
    from persona
   where fnac between f1 and f2;
 END
DELIMITER ;
```

y para usarla:

```
call cuentacuantos('1976-03-02',now());
```

Motivación

Si quisiéramos restringir los valores del atributo sexo en la tabla persona, en PostgreSQL podemos hacer:

```
alter table persona add check(sexo in ('m','f'));
```

... pero en MySQL esto no tiene ningún efecto...
¿podemos lograr algo similar?

Componentes

- Momento
  - ⋆ `BEFORE`: El cuerpo se ejecuta **antes** que el evento.
  - ⋆ `AFTER`: El cuerpo se ejecuta después de completar **exitosamente** el evento.
  - ⋆ `INSTEAD OF`: El cuerpo se ejecuta **en lugar** del evento.
- Evento
  - ⋆ `INSERT`
  - ⋆ `DELETE`
  - ⋆ `UPDATE`
- Cuerpo
  - ⋆ Variable `NEW`: Se habilita para `INSERT` y `UPDATE` en todos los momentos.
  - ⋆ Variable `OLD`: Se habilita para `DELETE` y `UPDATE` en todos los momentos.
- Alcance:
  - ⋆ `ROW`: El cuerpo se ejecuta por cada registro modificado.
  - ⋆ `STATEMENT`: El cuerpo se ejecuta solamente una vez.

Observaciones:

- Un disparador siembre debe estar asociado a una tabla o vista.
- Para el alcance `STATEMENT` no se habilita ninguna variable en el cuerpo.
- El cuerpo debe ser una expresión atómica.

Comando CREATE TRIGGER de PostgreSQL

Estructura

```
CREATE TRIGGER <nombre>
<momento> <evento>[ OR ... ]
ON <tabla>
FOR EACH <alcance>
EXECUTE PROCEDURE <proc> ()
```

Observaciones:

- Se pueden especificar varios eventos para el mismo momento.
- El cuerpo del disparador debe estar encapsulado en un procedimiento almacenado.
- La función "cuerpo" no debe recibir argumentos y regresar TRIGGER

Comando `CREATE TRIGGER` de MySQL

Estructura

```
CREATE TRIGGER <nombre> <momento> <evento>
    ON <tabla> FOR EACH ROW
    [<st>|<bloque>]
```

Observaciones:

- Sólo hay soporte para el alcance `ROW`
- El cuerpo puede ser una expresión en SQL o un bloque de expresiones encapsuladas entre `BEGIN` y `END`