Informe del trabajo práctico especial

Publicidad de Juegos

Fuhr, Federico
Genaro, Facundo

28 de Junio de 2020

Índice

Ajuste del esquema

Se ajustó el esquema según lo solicitado en el enunciado, aplicando las pautas de nomenclatura solicitadas modificando el esquema con el prefijo **GR10**.

Elaboración de Restricciones

Todas las restricciones tienen su caso de prueba comentadas dentro del SQL **GR10_Cambios.sql**, por lo que se omiten referencias a las pruebas de las restricciones.

a. *La fecha del primer comentario tiene que ser anterior a la fecha del último comentario si este no es nulo.*

   Para la resolución de este inciso pensamos en la siguiente tabla

Tabla/s	Atributo/s	Tipo de Restricción
Comenta	fecha_primer_com fecha_ultimo_com	Tupla

Esta restricción se aplica sobre la tabla Comenta ya que esta es la que posee los datos necesarios para hacer la comparación correspondiente.
Es una restricción de Tupla ya que afecta solo a una fila y es soportada por PostgreSQL, por lo tanto no fue necesaria la implementación de un trigger.

```
alter table gr10_comenta
   add constraint GR10_CHK_PRIMER_COMENTARIO
       check ( (fecha_primer_com < fecha_ultimo_com) or
(fecha_ultimo_com is null));
```

Esta restricción verifica que no haya comentarios que tengan una **fecha_ultimo_com** menor que **fecha_primer_com** para así preservar la integridad de los datos, ya que un comentario “del pasado” podría afectar dicha integridad.
En el caso de prueba que está comentado en el archivo .sql podemos apreciar que se intenta agregar un comentario que incumple la restricción pero el DBMS muestra el error correspondiente al incumplimiento de la misma.

b. *Cada usuario sólo puede comentar una vez al día cada juego.*

Tabla/s	Atributo/s	Tipo de Restricción
Comentario	fecha_comentario	Tabla

Esta restricción se aplica sobre la tabla **GR10_Comentario** ya que es la que almacena los datos que queremos controlar.
Es una restricción de tabla pero la implementación no es soportada por Postgresql ya que implica el uso de una subquery dentro del Check, por ende debemos implementar un trigger ubicado en **G10_Cambios.sql**.
En esta RI declarativa se busca un usuario que haya comentado más de una vez en la tabla comentario en la fecha actual.

```
alter table gr10_comentario
    add constraint GR10_CHK_COMENTARIO
        check (not exists (select c.id_usuario
                           from gr10_comentario c
                           group by c.fecha_comentario, c.id_usuario, c.id_juego
                           having count(*) > 1
                           where (c.fecha_comentario = CURRENT_DATE)
               );
```

El trigger que implementamos en la tabla Comentario, se aplica antes de insertar en la tabla y verifica si existe dentro de ella un usuario que haya comentado el mismo juego dos veces en un día mediante la comparación entre los campos **id_usuario**, **id_juego** y **fecha_comentario** con los datos que están intentando insertarse (.new) entonces si se produce un triple match entre los datos, la función retorna un mensaje de error.

c. Un usuario no puede recomendar un juego si no ha votado previamente dicho juego.

Tabla/s	Atributo/s	Tipo de Restricción
Voto Recomendación	id_usuario id_juego	Global

Esta restricción se aplica sobre las tablas Voto y Recomendación, por lo tanto al haber más de una tabla involucrada es una RI de global, la cual no es soportada por postgreSQL.
En la siguiente RI declarativa se busca que para que un usuario pueda recomendar tiene que estar su voto en la tabla **GR10_Voto** para el mismo juego haciendo una consulta a la tabla **GR10_Recomendacion** y dentro de está haciendo otra consulta en el **WHERE** en la tabla voto donde tanto el id_usuario como el **id_juego** tienen que estar en ambas tablas.

```
create assertion as G10_CHK_VOTO
    check (not exists (
                select 1
                from gr10_recomendacion r
                where not exists(
                      select 1
                      from gr10_voto v
                      where (v.id_usuario = r.id_usuario) AND (v.id_juego = r.id_juego)
                )
           );
```

El trigger que implementamos se dispara al momento de insertar o actualizar una **id_usuario** e **id_juego** dentro de la tabla Recomendación.
La forma de determinar si el juego fue votado fue mediante el uso de las PK/FK que existen entre Voto y Recomendación para ver si hay un vínculo válido.
La función que dispara devuelve un mensaje de error si no encuentra un match entre el **id_usuario** e **id_juego** que intentan ingresar en la tabla (.new) y los **id_juego** e **id_usuario** almacenados en la tabla Voto.

A la hora de testear este trigger, se intentaron recomendar juegos que no tuvieran votos y también se intentó recomendar juegos que sí los tuvieran, luego de intentar estas dos inserciones, como era de esperarse, el trigger cortó la ejecución del primer insert dando el mensaje de error correspondiente. Mientras que el segundo insert pasó correctamente por el trigger y se añadió a la tabla satisfactoriamente.

d. Un usuario no puede comentar un juego que no ha jugado.

Tabla/s	Atributo/s	Tipo de Restricción
Comentario Juega	id_usuario id_juego	Global

Esta restricción se aplica sobre las tablas Comentario y Juega, por lo tanto al haber más de una tabla involucrada es una RI de Global, la cual no es soportada por PostgreSQL.

La RI declarativa tiene una estructura similar al del inciso anterior en cuanto a la formulación, los cambios que tiene es que se hace una consulta en la tabla **GR10_Comenta** y luego una sub consulta en la tabla **GR10_Juega**.

```
create assertion G10_CHK_VOTO
    check(not exists(
                    select *
                    from gr10_comenta c
                    where (not exists(
                                    select *
                                    from gr10_juega j
                                    where (c.id_usuario = j.id_usuario) and
                                    (c.id_juego = j.id_juego)
                                    )
                    )
    );
```

El trigger que implementamos se dispara al momento de insertar o actualizar una **id_usuario** e **id_juego** dentro de la tabla Comentario.
Al igual que en el inciso anterior usamos los pares de PK/FK para determinar si un usuario puede comentar y también la implementación y funcionamiento del trigger fue muy similar.

Servicios

Se nos solicitó la implementación de los siguientes servicios:

Se debe mantener sincronizadas las tablas COMENTA y COMENTARIO en los siguientes aspectos:

a. *La primera vez que se inserta un comentario de un usuario para un juego se debe hacer el insert conjunto en ambas tablas, colocando la fecha del primer comentario y último comentario en en nulo.*
b. *Los posteriores comentarios para sólo deben modificar la fecha de último comentario e insertar en COMENTARIO*

Para la resolución de estos dos incisos optamos por utilizar un trigger que realice las dos tareas solicitadas.
Nuestro nuevo trigger se implementa sobre la tabla Comentario y se dispara cuando se actualiza o inserta un **id_juego** y/o **id_usuario** almacenados en la tabla.

a. La función que invoca primero verifica si el par de datos **id_usuario**, **id_juego** existen en la tabla Comenta, si no existen los genera e inserta con los valores solicitados en este inciso.
b. En caso de que el par de datos existan, la función actualiza los datos de la tabla Comenta y permite el insert en la tabla Comentario tal y como lo solicita este inciso

Vistas

1. *Listar Todos los comentarios realizados durante el último mes descartando aquellos juegos de la Categoría “Sin Categorías”.*

En la implementación de este inciso usamos un ensamble entre la tabla **GR10_Comentario**, **GR10_Juego** y **GR10_Categoria** por lo que en este caso no es actualizable. A este ensamble elige aquellos que no tienen en la descripción **'Sin categorías'** y se hicieron en el mismo mes y año actual.

2. *Identificar aquellos usuarios que han comentado todos los juegos durante el último año, teniendo en cuenta que sólo pueden comentar aquellos juegos que han jugado.*

La implementación de este inciso es actualizable porque no contiene ningún ensamble. La consulta principal es la proyección del **id_usuario** de la tabla **GR10_Usuario** cuando esté es igual al id_usuario . Id que se obtiene de buscar un usuario en una consulta en la tabla **GR10_Comentario** que sea del último año. Y entonces agrupa y pregunta que tenga la cantidad de comentarios de la tabla **GR10_Comentario** igual a la cantidad de juegos que se registran en **GR10_Juego**. Se pudo haber hecho más simple con ensambles y sería más intuitiva su explicación pero perdería la capacidad de actualizarse automáticamente.

3. *Realizar el ranking de los 20 juegos mejor puntuados por los Usuarios. El ranking debe ser generado considerando el promedio del valor puntuado por los usuarios y que el juego hubiera sido calificado más de 5 veces.*

La implementación de este inciso no es actualizable porque se usa un ensamble entre la tabla **GR10_Juego** y **GR10_Voto** con el **id_juego**, luego de agrupar hace la consulta del having en la que la cantidad de **id_juego** es mayor a 5, es decir, fue votado más de 5 veces ordenado descendentemente con un límite de 20.

Sitio

El sitio fue implementado en tiempo y forma, se encuentra en el siguiente LINK.

Conclusiones

Se pudo realizar de forma exitosa todas las RI / Reglas de negocio, servicios y vistas solicitadas. Donde en los casos de prueba que se han realizados se respalda el correcto funcionamiento de cada implementación, documentado con una explicación sobre el objetivo de la función, un caso de prueba afirmativo y otro negativo.
Durante el desarrollo de este trabajo práctico especial (que se basa en una sistema en menor escala que uno utilizado en la realidad) propuesto por la cátedra aprendimos las nociones básicas de cómo administrar un sistema y así comprendimos la complejidad que presenta mantener y administrar un sistema completo.