Universidad Nacional del Centro de la
Provincia de Buenos Aires

**FACULTAD DE CIENCIAS EXACTAS**

TUDAI


# Trabajo Práctico

## Base de datos

*Profesores/ayudantes a cargo: Ferraggine Viviana, Villar Sebastian*

**GRUPO GR02**

Casado Paula: pcasado@alumnos.exa.unicen.edu.ar

Caballeri Eliana: ecaballeri@alumnos.exa.unicen.edu.ar

30/11/2

El siguiente informe se realizó en base a las pautas del proyecto entregado por la cátedra Base de datos. Dicho proyecto consiste en la resolución de un conjunto de controles y servicios sobre una base de datos que mantiene un Sistema de Publicidad de Juegos en Línea de la empresa "PJL Sistemas Tandil".

En él se explican y justifican las decisiones asociadas a la implementación de cada una de las restricciones de integridad, la definición de las vistas y de los servicios requeridos.

**A)** Para comenzar con el trabajo se pidió ajustar los script de creación SQL de acuerdo a nuestro grupo, y luego ejecutarlos junto con las inserciones también ajustadas.

**B)** En el primer inciso se pide las siguiente restricciones/reglas del negocio:

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

Este tipo de restricción consideramos que es de tupla, la cual es soportada por PostgresSQL. Para demostrar que funcionan los controles se realizaron las siguientes inserciones que proceden como: ingresando fecha_ultimo_com en NULL(permitido por el atributo en la tabla), que la fecha_primer_com sea menor que fecha_ultimo_com, así mismo en los UPDATE se cambió la fecha_primer_com por una menor a la fecha_ultimo_com. Con el fin de garantizar la funcionalidad de la restricción, luego se decidió agregar sentencias que fallan para comprobarlo al insertar o actualizar un registro donde la fecha_primer_com sea mayor que la de fecha_ultimo_com.

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

Según el enunciado de esta restricción consideramos que es de tipo tabla, y como no es soportado por PostgresSql, se decidió hacer un triggers para que dicha restricción se complete. Se hicieron inserciones y actualizaciones que proceden como un usuario que comente el mismo juego pero con fechas distintas. También se probó registros que fallen para comprobar su funcionalidad como: que un usuario trate de insertar un segundo comentario en el mismo día.

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

La restricción que se implementó para este enunciado es de tipo general, y como no es soportado por PostgresSql, se decidió hacer un triggers para que dicha restricción proceda. Se hicieron inserciones y actualizaciones que proceden como que un jugador recomiende cualquiera de los juegos que efectivamente juega siempre y cuando los haya votados. También se probó registros que fallen para comprobar su funcionalidad como que un usuario trate de recomendar un juego que no ha votado.

**IV. Un usuario no puede comentar un juego que no ha jugado.**

Dicha restricción consideramos que es general, y al igual que las anteriores no es soportada por PostgresSql, por lo que se decidió hacer un triggers para que proceda. Se hicieron inserciones y actualizaciones que proceden como que un jugador haga un comentario sobre cualquiera de los juegos que efectivamente juega. También se probó registros que fallen para comprobar su funcionalidad como que un usuario trate de hacer un comentario respecto a un juego que no juega.

**C)** En el segundo inciso se pide los siguientes servicios:

**I. Se debe mantener sincronizadas las tablas COMENTA y COMENTARIO.**

Para llevar a cabo dicho servicio se decidió implementar un trigger en la tabla comentario que se despierte bajo las acciones de inserciones y actualizaciones de sus registros. Una vez que este es activado, llama a la función que es la responsable de la sincronización, y lo hace mediante sentencias que se ocupan de que cada vez que un registro quiera ser insertado o actualizado en la tabla comentario al mismo tiempo lo realice sobre los registros de la tabla comenta.

Cabe destacar que además de la sincronización se pidió la organización de las fechas ya que la tabla comenta posee dos campos que requieren de ellas, por lo que en las sentencias también se regula el llenado de dichos campos de acuerdo al orden en que son insertados los registros de la tabla comentario. Si es el primer registro se llena el campo de primer comentario de comenta con la fecha que trae el comentario, en cambio si ese usuario ya comentó la fecha pasa a ocupar el campo de fecha_ultimo_com de la tabla comenta.

**II. Realizar un patrón de búsqueda.**

Ante el pedido de este servicio se creó una función que retorne una tabla en la que se proyecte los campos del usuario requerido a través de su id de usuario. Cabe mencionar que se podía buscar por distintos campos de la tabla usuario, pero a nuestro criterio decidimos hacerlo por ese solo campo.

Además de los campos se pedía contar la cantidad de juegos que jugaba ese usuario pedido y la cantidad de votos que había realizado, por lo que a las filas que ya retornaba la tabla usuarios, se le agrego dos filas que proyectan dichas cantidades.

**D)** En el tercer inciso se pide la definición de las siguientes vistas:

I. **COMENTARIOS_MES: Listar todos los comentarios realizados durante el último mes descartando aquellos juegos de la Categoría "Sin Categorías".**

Se realizó una vista actualizable y soportada por postgresSQL ya que contiene el id de la tabla que muestra, no tiene sentencias de agrupación en la primer consulta y utiliza en el primer from una sola tabla. Luego a través de subconsultas se obtienen los comentarios hechos por usuarios, siempre y cuando hayan sido un mes atrás de la fecha actual, y que los juegos que dicho usuario juega tengan una categoría.

II. **USUARIOS_COMENTADORES: Listar aquellos usuarios que han comentado TODOS los juegos durante el último año, teniendo en cuenta que sólo pueden comentar aquellos juegos que han jugado.**

Se realizó una vista actualizable y soportada por postgresSQL ya que contiene el id de la tabla que muestra, no tiene sentencias de agrupación en la primer consulta y utiliza en el primer from una sola tabla. Posteriormente a través de subconsultas se obtienen los usuarios que si o si comentaron cada uno de los juegos a los que juegan, la vista no puede proyectar usuarios que no hayan comentado algún juego de los que juega. Además se debe tener en cuenta solo los comentarios realizados durante el último año corriente.

**III. LOS_20_JUEGOS_MAS_PUNTUADOS: 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 haya sido calificado más de 5 veces.**

Se realizó una vista actualizable y soportada por postgresSQL ya que contiene el id de la tabla que muestra, no tiene sentencias de agrupación en la primer consulta y utiliza en el primer from una sola tabla. Por consiguiente a través de subconsultas se obtienen los juegos más puntuados por los usuarios. Cabe aclarar que para que sea un ranking se debería mostrar en una fila junto al juego su puntaje, pero decidimos que para que sea actualizable se mantenga oculta por las subconsultas. Para ello se debió hacer un barrido de los juegos que habían sido calificados más de 5 veces, para luego obtener el promedio de cada uno, y ordenarlos de manera descendente mostrando en la parte superior el id del juego que más puntaje obtuvo.

**IV. LOS_10_JUEGOS_MAS_JUGADOS: Generar una vista con los 10 juegos más jugados.**

Para esta vista no se pudo realizar una vista actualizable ya que posee en la consulta un group by. A través de esta se proyecta aquellos juegos que son los más jugados por los usuarios, para ello se agrupó por juego y se contó la cantidad de usuarios que lo jugaban ordenándolos de manera descendente quedando por encima los 10 más jugados.