

Trabajo práctico N° 1

PostgreSQL

UNIVERSIDAD CATÓLICA DE SANTIAGO DEL ESTERO

Departamento Académico Rafaela

Carrera: Ing. en Informática

Materia: Base de Datos II

Profesores: Marcela Vera, Alejandro Aguirre

Fecha: 16/05/2016

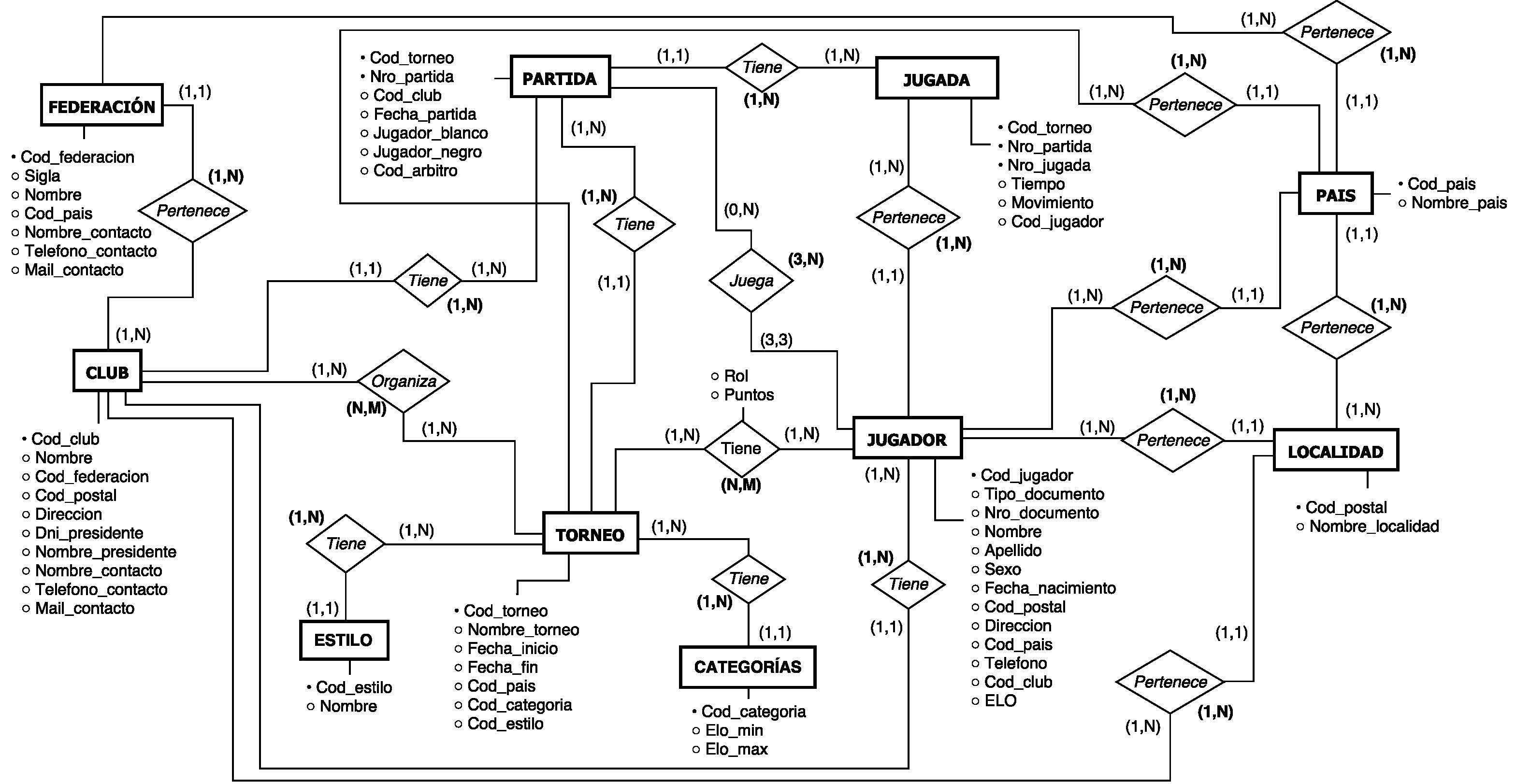
Alumnos: Wendy Sclerandi

Camila Kopech

Miguel Delpuppo

Giorgina Castagno

**a. Diagrama Entidad Relación**

****

**b. Decisiones de implementación**

Tomamos la decisión de particionar la base de datos en cuatro dispositivos físicos de almacenamiento, a los que llamamos D1, D2, D3 y D4. En PostgreSQL estos lugares de almacenamiento son conocidos como tablespaces, en donde se guardan todos los objetos de la base de datos.

La división la realizamos de la siguiente manera:

D1 🡪 Entidad JUGADA. Esta tabla crecerá de manera abrupta en el tiempo, ya que habrá muchas jugadas para una sola partida. Es por esto que decidimos aislarla.

D2 🡪 Entidades PARTIDA y JUGADOR. Estas dos tablas están guardadas en el mismo tablespace porque poseen una relación muy fuerte ya que cada partida debe almacenar tres jugadores.

D3 🡪 Entidades TORNEO, TORNEO/CLUB, TORNEO/JUGADOR, ESTILO y CATEGORÍA. Todas estas tablas están relacionadas por torneo y debido a que su crecimiento no es abrupto pueden coexistir en un mismo dispositivo físico.

D4 🡪 Entidades FEDERACIÓN, CLUB, LOCALIDAD y PAÍS. Las tablas federación y club tienen una relación fuerte por eso hemos decidido almacenarlas en el mismo tablespace. A su vez, las cuatro tablas son las menos actualizadas.

Los índices se encuentran en el mismo tablespace que su tabla correspondiente.

Además de los índices asociados a las claves primarias creados por el DBMS, agregamos las siguientes estructuras de acceso, pretendiendo facilitar las consultas sobre la base de datos:

* Club.Cod\_federacion
* Torneo.Fecha\_inicio
* Partida.Cod\_club
* Jugada.Cod\_jugador
* Jugador.Nro\_documento (Clustered)

**c.Esquemas externos**

* Directorio de FIDE: La vista “vw\_datos\_torneos” le facilita a la FIDE los datos de todos los torneos, y para consultar todas las federaciones juntos con sus correspondientes jugadores, se muestra la vista “vw\_jugadores\_federaciones”.
* Directivos de cada Federación:Para que cada federación pueda visualizar los detalles de los torneos desarrollados en su federación, junto con los participantes y su ubicación, se mostrará la vista “vw\_directivos\_federacion”, la cual es llamada desde la función “fn\_federacion” para filtrar por código de federación.
* Participante/jugador: Por un lado, los jugadores tendrán acceso a la función “fn\_inscripcion\_jugadores” para poder inscribirse. Para ver su ubicación final en los torneos podrán visualizar la vista “vw\_jugadores\_ubicacion”, la cual es llamada desde la función “fn\_jugador\_ubicacion” para filtrar por un jugador determinado.Para ver las respectivas jugadas de un jugador, se mostrará la vista “vw\_jugadores\_jugadas” que es llamada desde la función “fn\_jugador\_jugadas” para el filtrado por jugador.

**d. Esquemas Discrecionales**

La base de datos utiliza un acceso discrecional en el cual se especifica a qué objetos pueden acceder los diferentes usuarios. Para ello creamos grupos de usuarios y asignamos los usuarios a cada uno de estos. Todos los usuarios pueden leer todos los datos de la base de datos.

Los grupos de roles o usuarios son:

* “Clubes”: Tienen permitido ingresar los datos de su propio club y de sus jugadores.
* “Directivos\_Federaciones”: Pueden cargar sus propios datos y los datos de sus clubes, además de todas las jugadas correspondientes a cada federación.
* “FIDE”: Tiene permiso para manipular los datos de todas las relaciones.
* “Participantes”: No tienen permiso de escritura o modificación sobre las tablas de la base de datos, pero pueden acceder a la función “fn\_inscripcion\_jugadores” para inscribirse en el torneo.

El usuario “Postgres” es el administrador de la base de datos, por lo que se encargará de crearla y mantenerla.

**e. Consultas**

* Para consultar el detalle de todas las partidas de un determinado jugador, se deberá acceder a la función “fn\_partida\_jugador”.
* Para ver los datos de los jugadores de un determinado torneo, junto con su posición final y sus respectivas partidas, se deberá llamar a la función “fn\_datos\_jugadores\_torneo”.
* Con la función “fn\_datos\_jugador\_max\_torneo” se mostrará el nombre y el ELO de todos los jugadores que participaron en más torneos durante el último año.

**f. Política de respaldo**

PostgreSQL ofrece Ficheros WAL (Write Ahead Log), en los cuales guarda toda la información sobre las transacciones y cambios realizados en la base de datos. Estos ficheros se utilizan para garantizar la integridad de los datos grabados en la base de datos y para reparar automáticamente posibles inconsistencias después de una caída del servidor. Además, ofrece un tipo de backup para estos ficheros WAL, llamado Point in Time Recovery (PITR).

Desde el punto de vista teórico y como Administradores de Base de Datos,recomendamos emplear un backup de tipoFULL cada 5 días para que el tamaño de los backup diferenciales no crezca demasiado y consuma menos tiempo su restauración. El horario de implementación podría ser a las 00.00hs debido a que nadie estará utilizando el sistema. Vale la pena invertir tiempo y espacio en este proceso porque otorga la ventaja de poder recuperar toda la base de datos en caso de fallas.

Este respaldo se complementaría con un DIFERENCIAL realizado diariamente a las 23.00hs.Si bien en época de torneos existirán muchos registros, los mismos serán pequeños y ocuparán poco espacio. Cabe destacar que es una política de respaldo fácil y rápida de realizar, y es un buen complemento para restaurar la base de datos ante cualquier falla.

Por último, se debería realizar un backup INCREMENTAL o de LOG cada 5 minutos. De esta manera estamos dispuestos a perderlos datos de aproximadamente una partida. Este tipo de backup ayudará a restaurar las últimas transacciones que quedaron inconsistentes debido a alguna falla.

Finalizados los torneos, puede que el sistema se utilice solo para nuevas inscripciones, suponiendo que entre torneos esperan un lapso de un mes. Llegado este caso, las políticas de respaldo estarán guardando pocos datos, por lo que los backups se realizarán con menos frecuencia. Por ejemplo, efectuando un full cada diez días, diferenciales cada 5 días y logs diariamente.

Con respecto al eliminado de backups, recomendamos hacerlo cada 6 meses. Esto asegura que por lo menos los últimos 6 torneos tengan una copia de respaldo.

El trabajo de aumentar o disminuir la intensidad de políticas los backup es responsabilidad del administrador de la base de datos.

Para incrementar el respaldo y disponibilidad de los datos de este sistema podemos recomendar RAID de tipo 1, utilizando un disco de gran capacidad para tener redundancia de la base de datos entera y agilizar la lectura de la misma.

**g. Arquitectura**

La arquitectura de base de datos más adecuada para este sistema sería una distribuida, ya que se trata de un sistema internacional y se podrían distribuir los datos de cada torneo en los diferentes sitios donde se llevaron a cabo. Además, se trata de datos que pueden cambiar constantemente y que serán modificados por cada club, como los jugadores, sus puntajes, los participantes de determinados torneos, etcétera.

Nuestra recomendación para implementar este sistema distribuido es aplicar una fragmentación horizontal por las distintas federaciones, teniendo así localmente la información de su federación y los datos de todos los torneos realizados en sus respectivas localidades. Tener los datos en los sitios donde más se utilizan permite minimizar la transferencia de datos.

Además, es necesario conservar una réplica total de la base de datos en la sede de la FIDE para que la misma pueda gestionar la información de todos los torneos a nivel internacional.

Cabe aclarar que toda información que tienda a ser constantemente accedida debe ser replicada en los sitios donde sea necesario para así evitar la pérdida de datos y garantizar que los mismos estén disponibles en todo momento.

**h. Cifrado de Datos**

Decidimos cifrar los datos más sensibles de esta base de datos que son el número de documento y la dirección de cada jugador, debido a que cualquier persona puede consultar su número ELO y sus jugadas.

Esto se realizaría para que ninguna persona no autorizada pueda tener acceso a dichos valores.

**i. Control de concurrencia**

Dado que en este motor de base de datos el nivel de aislamiento por defecto es Read committed, opinamos que para este caso lo más conveniente sería mantenerlo. Esto se debe, en primer lugar, a que Read committed proporciona mejor rendimiento y proporciona una forma de bloqueo especial que ayuda a evitar los fenómenos fantasmas. Este bloqueo es llamado bloqueo compartido y permite a otras transacciones leer los registros examinados, pero no actualizar o borrar mientras que otra transacción esté interviniendo.

Además, PostgreSQL no tiene un nivel real serializable puesto que solo ve los datos que han sido commited antes de la primera consulta o modificación de datos.

Gracias a la utilización del modelo MVCC (explicado en el punto k),una lectura nunca bloquea una escritura y una escritura nunca bloquea una lectura.PostgreSQL aplica bloqueos sobre tablas y filas automáticamente, aunque también brinda la posibilidad de hacerlo de forma manual en caso de necesitarlo.

Pensamos que los datos de esta base de datos no son muy susceptibles a modificaciones simultáneas, porque a pesar de que puede haber muchas partidas al mismo tiempo, las claves primarias serán distintas. Es por esto que los controles que vienen dados por defecto por este motor de base de datos nos parecen suficientes para mantener una correcta consistencia en la base de datos del caso que estamos trabajando.

**j. Vistas materializadas**

En este sistema no utilizaríamos vistas materializadas ya que las mismas ocupan espacio de almacenamiento y requieren de un mantenimiento que, comparado al costo de buscar los datos a través de una vista normal, resulta inconveniente. Además, la mayoría de las tablas son afectadas constantemente por escrituras y modificaciones, lo que implicaría una actualización constante de las vistas y, por lo tanto,un mayor costo de procesamiento.

Por ejemplo,se podría materializar una vista de los datos de todos los torneos, pero,suponiendo que se realicen nuevos torneos cada mes, implicaría actualizar la vista todos los meses.Un caso extremo de vista materializada seríade laspartidas con sus respectivas jugadas ya que esta relación contiene muchos registros y supondría un desperdicio del almacenamiento.

**k.Capacidades de PostgreSQL en materia de control de concurrencia**

La mayoría de los DBMS ofrecen los siguientes niveles de aislamiento: Read uncommitted, Read commited, Repeatable read, Serializable. Internamente, PostgreSQL, a diferencia de la mayoría de los DBMS, ofrece sólo 3 niveles: Read Commited, Repeatable Read y Serializable.

El nivel de aislamiento por defecto de PostgreSQL es **read committed**, donde las modificaciones de otras transacciones se ven sólo si se terminaron correctamente antes de comenzar la consulta,por lo tanto,los cambios no confirmados no son visibles para cualquier otra transacciónhasta que se confirme dicha modificación. En caso de intentar cambiar un dato que otra transacción está cambiando, la actual queda bloqueada hasta saber si proceder con el cambio (en caso de rollback) o si volver a ejecutar la condición de consulta del cambio para comprobar que las filas a cambiar aún la cumplen (en caso de commit).

El nivel **serializable** de PostgreSQL funciona igual que el readCommitted, excepto que monitorea las condiciones que podrían hacer que la ejecución de un conjunto concurrente de transacciones serializables se comporten de una manera incompatible con todas las posibles series de ejecuciones de esas transacciones.Para garantizar este tipo de serialización, PostgreSQL utiliza “bloqueo de predicados”, el cual mantiene bloqueos que le permiten determinar si una escritura pudiera tener un impacto en el resultado de alguna lectura de otra transacción concurrente.Estos bloqueos no causan bloqueos reales ya que se utilizan para identificar dependencias entre transacciones serializables y, por lo tanto, no pueden causar deadlocks.

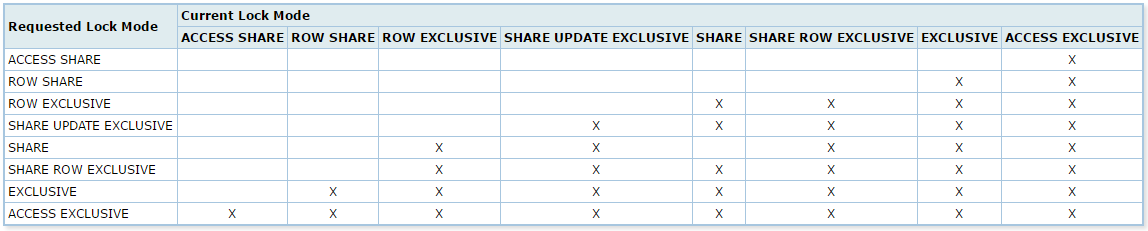
El nivel de aislamiento **repeatableread** sólo procesa los datos confirmadosantes del comienzo de la transacción. Sin embargo, la consulta puede ver los efectos de las actualizaciones ejecutadas anteriormente dentro de su propia transacción, incluso cuando aún no se han comprometido. Este nivel es diferente al de lectura confirmada ya que en que una consulta de una determinada transacción con lectura repetible, esta transacción trabaja sobre una misma instantánea de los datos. Por lo tanto, las sucesivas sentencias select dentro de una sola transacción ven los mismos datos, es decir, que no ven los cambios realizados por otras transacciones sobre esos datos después de que su propia transacción comenzó.

A diferencia de la mayoría de otros motores de bases de datos que usan bloqueos para el control de concurrencia, PostgreSQL mantiene la consistencia de los datos utilizando el **MVCC** (Control de Concurrencia Multi-Versión). La misma es una técnica de concurrencia optimista en donde ninguna tarea o hilo es bloqueada mientras se realiza una operación en la tabla, debido a que el otro hilo usa su propia copia (versión) del objeto dentro de una transacción. Esto significa que, mientras se consulta una base de datos, cada transacción ve una imagen de los datos como si fuera tiempo atrás (para ello el dato se versiona como un timestamp), sin tener en cuenta el estado actual de los datos que hay por debajo. Esto evita que la transacción vea datos inconsistentes que pueden ser causados por la actualización de otra transacción concurrente en la misma fila de datos, proporcionando aislamiento transaccional.

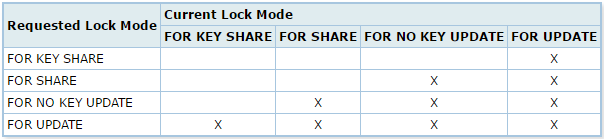
El MVCC nunca modifica o elimina los datos, las nuevas filas de información se van añadiendo según se crea o actualiza la misma y se marca la anterior como “no visible”.Cuando se desea eliminar un dato igualmente se añade una fila de datos y se marca como “no visible” al mismo tiempo. Internamente se identifica a cada transacción con un numero univoco y a cada registro de la tabla, con un numero de versión. Entonces, cada transacción trabaja con su copia y si se modifica un registro de una tabla, el contador de versión del registro se incrementa. Cuando se realiza el commit, la copia del objeto reemplaza a la que existía en la base de datos. Si dos transacciones modificaron la misma tabla, se unen ambas tablas combinando las últimas versiones de cada registro. Si las dos transacciones modificaron exactamente el mismo registro, cuando se hace el commit, el registro que finalmente queda corresponde a la última transacción en realizar una modificación sobre ese registro.

PostgreSQL ofrece varios modos de bloqueo para controlar el acceso concurrente a los datos en tablas. Algunos de estos modos de bloqueo los adquiere automáticamente antes de la ejecución de una transacción, mientras que otros son proporcionados para ser usados por las aplicaciones.

* Bloqueo a nivel de tablas: son adquiridos de manera automática por PostgreSQL o manual mediante el comando *lock*. Hay de diferentes modos de bloqueo: access share, row share, row exclusive, share update exclusive, share, share row exclusive, exclusive, access exclusive. La única diferencia real entre cada modo de bloqueo es el conjunto de modos de bloqueo con el que cada uno trata los conflictos.



* Bloqueo a nivel de filas: este tipo de bloqueos se produce cuando campos internos de una fila son actualizados (o borrados o marcados para ser actualizados). PostgreSQL no retiene en memoria ninguna información sobre las filas modificadas y de este modo no tiene límites para el número de filas bloqueadas sin incremento de bloqueo.Estos bloqueos no afectan a los datos consultado y son usados para bloquear escrituras a la misma fila únicamente. Los diferentes modos de bloqueo que ofrece PostgreSQL son: for update, for no key update, for share, for key share.



* Bloqueo a nivel de páginas: Son bloqueos compartidos/exclusivos usados para controlar los accesos de escritura/lectura a las páginas de la tabla en el buffer pool compartido. Estos bloqueos se liberan inmediatamente después de que una fila haya sido modificada.
* Deadlocks o bloqueos mortales
* Advisory Locks o bloqueos de asesoramiento: son bloqueos para fines específicos de las aplicaciones, pero no son usados normalmente por el sistema. Pueden ser útiles para las estrategias de bloqueo que son un ajuste incómodo para el modelo MVCC.

**l. Mejoras de optimización en la base de datos**

**i.**Una opción para agilizar esta consulta es aplicar vistas materializadas, ya que la información de un torneo no cambiará una vez que éste finalice y, por lo tanto, el mantenimiento sería sencillo. Pero aplicar una vista materializada trae como desventaja un aumento en el almacenamiento de los datos.

Otra opción para obtener de manera rápida los datos de esta consulta sería ordenar físicamente los registros de las jugadas por torneo, lo cual en nuestra base de datos ya está aplicado debido a que la clave primaria es el código del torneo en las 3 entidades que participarían en esta consulta: Torneo, Partida y Jugada.

Los datos que mostraríamos en la vista materializada para facilitar la comprensión serían el nombre del torneo, el número de partida y los nombres de cada jugador junto a su movimiento y tiempo demorado en cada jugada.

**ii.** Si la idea de los usuarios es poder encontrar los torneos sabiendo la fecha en que se realizaron los mismos, se recomienda agregar un índice clustered a la fecha de inicio de la entidad Torneo. De esta manera los datos se encontrarán ordenados físicamente por fecha y la lectura o filtrado de los mismos será más rápida.