**Aclaraciones DER – TP OPF5**

* El rechazo, se juntó a la tabla Propuesta, ya que por el momento, no nos parecía de demasiada utilidad modelar el rechazo en la base de datos ya que no cuenta con demasiada funcionalidad y puede ser un detalle de la Propuesta.
* Decidimos que en la tabla de Amigos, solo se guarde como un jugador, al jugador que recomienda al amigo. El amigo recomendado no lo modelamos como otro jugador ya que no nos parecía demasiado útil por el momento.
* La tabla de Jugadores, cuenta con la misma estructura que el objeto Jugador en el diagrama de objetos. La tabla no necesita otros campos para reflejar algún tipo de diferencia con el diagrama de objetos.
* De igual forma ocurre con la tabla de Calificaciones y con la tabla de Inscripciones, Equipos, Partido y Administrador.
* Para reflejar la relación entre jugadores y equipos, implementamos la tabla intermedia Equipo\_Jugador que tiene clave foránea a ambas tablas, ya que un jugador puede formar parte de varios equipos, y un equipo tiene muchos jugadores. Es una relación de muchos a muchos.
* Se implementa la tabla tipo\_inscripcion porque no nos parecía correcto poner un string en el tipo de inscripción de la Inscripción, por lo tanto la tabla tipo\_inscripcion cuenta con un id que es el referenciado por la inscripción y además la descripción que dice que tipo de inscripción es.
* Los métodos que correspondían a confirmar una inscripción y un partido, actuarían sobre los campos confirmado/a, poniendo una S en caso de estar confirmado y una N en caso de no estarlo.
* Para resolver el requerimiento del punto d, como elegimos un trigger, decidimos reflejar en el modelo los reemplazos que proponía un jugador para que quede constancia y asi poder utilizarlo en el mismo trigger.

**Justificación sobre elección de componentes**

* Se eligieron **Vistas** para resolver los puntos a) y b) ya que nos permite reificar un conceptos (jugadores malos o jugadores traicioneros) y luego trabajar con la vista como si fuera una tabla.  
   Una ventaja de usar la vista es que no ocupa espacio de almacenamiento en la base de datos, lo cual provoca que no hallan redundancias ya que los datos siguen residiendo en un solo lugar.  
   Otra ventaja es que se puede restringir acesso a un conjunto de filas o columnas en el caso de que el usuario que realice la consulta no cometa errores.
* Se eligió una **función** para el c) ya que devuelve un solo valor, y como el valor es único y se calcula, la función podría llegar a ganar performance.
* Se eligió un **stored procedure** para el d) ya que se necesitaba ingresar varios parametros(id\_partido, id\_jugador\_viejo, id\_jugador\_nuevo) y iba a haber efecto colateral (INSERT en tabla de reemplazo, y DELETE en tabla de inscripcion).
* Se eligió un **trigger** para el e) ya que se necesitaba ejecutar una acción: Agregar una infracción si el jugador no ofrece reemplazante cuando ocurre el evento de que se de de baja un jugador (DELETE)