GRUPO MATARRATAS - DDSI A2

Cristian Fernández Jiménez - *Gestión de Ubicaciones*
Francisco Javier Jiménez Legaza - *Gestión de Valoraciones y quejas*
Raul Castro Moreno - *Gestión de premios*
Santiago Muñoz Castro - *Gestión de encuentros*
Ángel Gómez Ferrer - *Gestión de equipos*
Raúl Rodríguez Pérez - *Gestión de usuarios*

APLICACIÓN DE ENCUENTROS DEPORTIVOS
MEET 'N' MATCH

ÍNDICE

DESCRIPCIÓN INICIAL DEL SISTEMA 3

FUNCIONALIDAD DE LOS SUBSISTEMAS 4

- GESTIÓN DE JUGADORES 4
- GESTIÓN DE EQUIPOS 10
- GESTIÓN DE UBICACIONES 17
- GESTIÓN DE ENCUENTROS 22
- GESTIÓN DE VALORACIONES/ALEGACIONES 28
- GESTIÓN DE PREMIOS 32

DISEÑO DE DATOS DE LOS SUBSISTEMAS 37

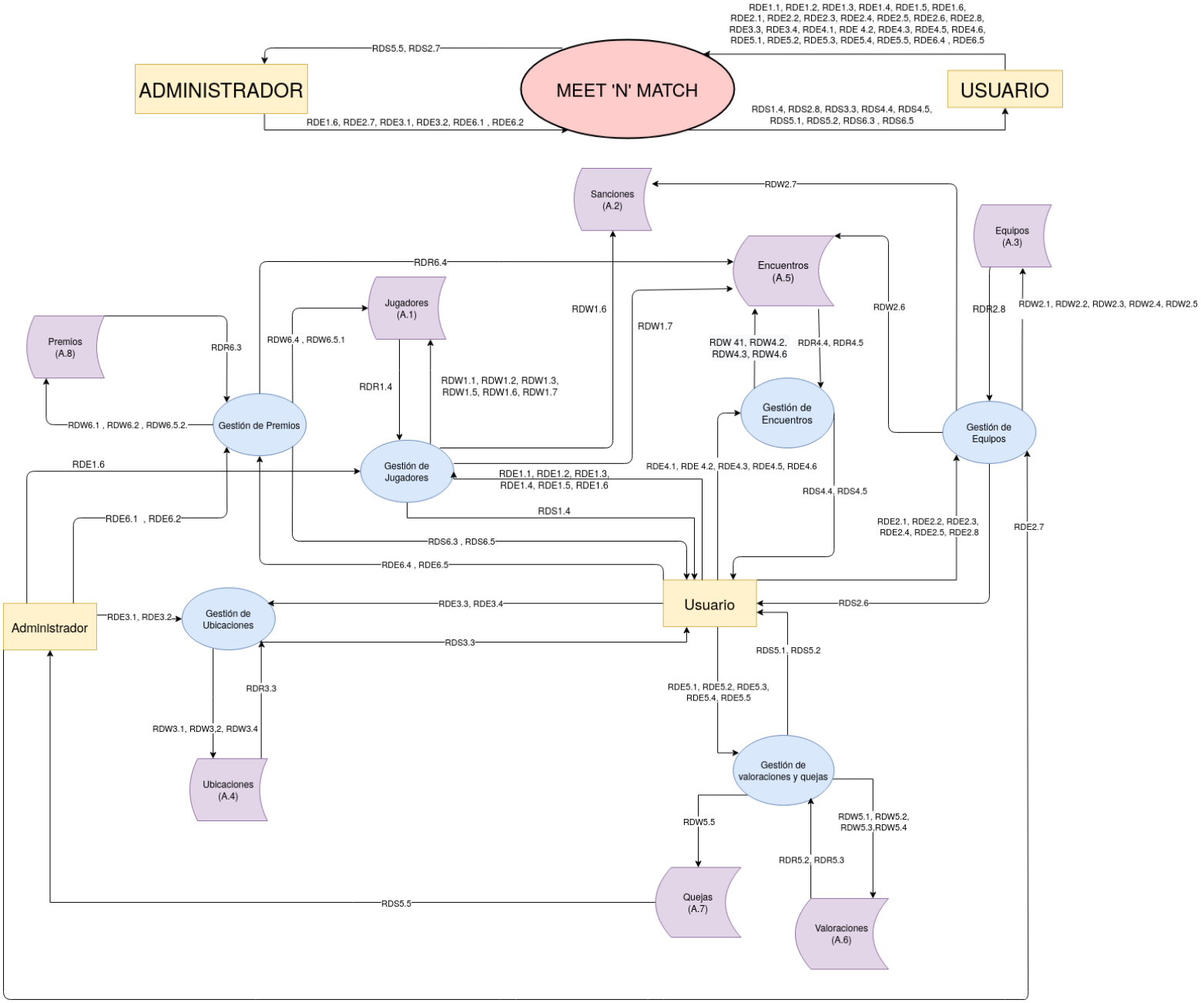
- GESTIÓN DE JUGADORES 37
- GESTIÓN DE EQUIPOS 38
- GESTIÓN DE UBICACIONES 39
- GESTIÓN DE ENCUENTROS 40
- GESTIÓN DE VALORACIONES 41
- GESTIÓN DE PREMIOS 42
- ESQUEMA EXTERNO ARMAZÓN (DFD0) 43
- ESQUEMA ENTIDAD RELACIÓN DEL SISTEMA 44
- NORMALIZACIÓN DE TABLAS 45
- CREACIÓN DE TABLAS SQL 48
- INSERCIÓN DE TUPLAS EJEMPLO 50

DESARROLLO DEL SUBSISTEMA 53

- MOTIVACIÓN DE LA ELECCIÓN DE SOFTWARE 69

DESCRIPCIÓN INICIAL DEL SISTEMA

Deseamos crear un sistema de información para proporcionar las herramientas necesarias a los usuarios del mismo y que estos puedan, de manera rápida y sencilla, tanto organizar cómo participar en actividades deportivas.



FUNCIONALIDAD DE LOS SUBSISTEMAS

1. GESTIÓN DE JUGADORES: *(RAÚL R)*

Para cada usuario, el sistema almacenará diversos campos que serán necesarios para **darse de alta:** su nombre (en una serie de hasta 20 caracteres), su nombre de usuario, el cual deberá ser único y además seguir una serie de restricciones en cuanto a los caracteres que pueden usar para crear el mismo: no pueden incluir el signo et (&), el signo igual (=), guiones bajos (_), apóstrofos ('), guiones (-), corchetes (< >), el signo más (+), comas (,) ni dos o más puntos seguidos. Sus apellidos (en una serie de hasta 40 caracteres), un número de teléfono (en una serie de hasta 20 caracteres que pueden ser todos numéricos menos el primero que puede ser numérico o un signo +), su altura en metros, su edad, su peso en kilogramos, y por último, si tiene/ha tenido una carrera profesional, y en ese caso en que clubes ha estado/está (en una serie de hasta 500 caracteres).

De la misma forma el sistema almacena otros campos que inicialmente el usuario no rellena, pero que después serán muy utilizados por los mismos. Por un lado tenemos el caso de que el jugador ingrese en algún equipo organizado por el sistema, en dicho caso se almacenará el nombre del equipo en los datos del usuario(vacío por defecto). Por otro lado, existe el campo para saber si el jugador tiene alguna sanción o no (por defecto “no sancionado”), y por último, la cantidad de puntos que posee el jugador, los cuales puede canjear para conseguir premios exclusivos que otorga el sistema (inicializado a 0).

Además de estos campos, el usuario puede establecer **los deportes en los que tenga más interés**, tanto para buscar encuentros, como para crearlos ellos mismos. Incluso cada uno, podrá registrar si posee el material necesario para cada deporte, y en caso de que lo posea, establecer exactamente de que dispone. Cabe destacar que el sistema de información da libertad absoluta al usuario para que en cualquier momento pueda tanto **consultar**, como si así lo desea, **modificar** cualquiera de los datos anteriormente mencionados.

Cada jugador tiene la opción de **inscribirse a diversos encuentros** que buscará dependiendo de sus preferencias deportivas. Destacando que el sistema no permitirá inscribirse en dos encuentros si dichos encuentros coinciden en día y hora. Además si los jugadores están bajo sanción, no podrán inscribirse a ningún encuentro hasta que finalice la misma.

RF1.1: Alta jugador

Entrada: Agente externo: usuario. Acción: solicitar inserción.

Requisito de datos de entrada RDE1.1.

BD: Requisito de datos de escritura RDW1.1.

Salida: Agente externo: usuario. Acción: confirmación resultado.

Requisito de datos de salida: ninguno.

RDE1.1: Datos de entrada de alta de jugador

Nombre: Cadena de caracteres (20)

Apellidos: Cadena de caracteres (40)

Número de teléfono: Serie de 20 caracteres que pueden ser todos numéricos menos el primero que puede ser numérico o un signo +

Nombre de usuario (único): Cadena de caracteres (20)

Edad: Número entero

Altura: Número decimal (metros)

Peso: Número decimal (kilogramos)

Carrera profesional o clubes: Cadena de caracteres (500)

Equipo: Cadena de caracteres (20) (por defecto vacío)

Puntos: Número entero (10) (por defecto 0)

Sancionado: 1 -> si el jugador está sancionado o 0 -> si el jugador no lo está (opción por defecto)

RDW1.1: Datos almacenados del jugador

Los mismos datos que RDE1.1

RF1.2: Baja jugador

Entrada: Agente externo: usuario . Acción: solicitar borrado.

Requisito de datos de entrada RDE1.2.

BD: Requisito de datos de escritura RDW1.2.

Salida: Agente externo: usuario. Acción: confirmación resultado.

Requisito de datos de salida: ninguno.

RDE1.2: Datos de entrada de baja de jugador

Los mismos que datos de RDE1.1

RDW1.2: Datos almacenados del jugador

Los mismos datos que RDE1.2.

RF1.3: Modificar datos del jugador

Entrada: Agente externo: usuario. Acción: modificar datos del jugador.

Requisito de datos de entrada RDE1.3

BD: Requisito de datos de escritura RDW1.3

Salida: Agente externo: usuario. Acción: confirmación resultado.

Requisito de datos de salida: ninguno.

RDE1.3: Datos de entrada de modificar jugador

Los datos que se quieran modificar de RDE1.1

RDR1.3: Datos de lectura de modificar jugador

Nombre de usuario (único): Cadena de caracteres (20)

RDW1.3: Datos almacenados de jugador

Los mismos datos que RDE1.3

RF1.4: Mostrar datos jugador

Entrada: Agente externo: usuario. Acción: solicitar información.

Requisito de datos de entrada RDE1.4.

BD: Requisito de datos de lectura RDR1.4.

Salida: Agente externo: usuario. Acción: confirmación resultado.

Requisito de datos de salida: RDS1.4.

RDE1.4: Datos del jugador

Nombre de usuario (único): Cadena de caracteres (20)

RDR1.4: Datos del jugador leídos

Los mismos que RDW1.1

RDS1.4: Muestra de datos del jugador

Los mismos que RDR1.4

RF1.5: Preferencias de juego

Entrada: Agente externo: usuario. Acción: añadir información del jugador.

Requisito de datos de entrada RDE1.5.

BD: Requisito de datos de escritura RDW1.5.

Salida: Agente externo: usuario. Acción: confirmación resultado.

Requisito de datos de salida: ninguno.

RDE1.5: Datos en relación a las preferencias del juego del jugador

Nombre de usuario (único): Cadena de caracteres (20)

Elección de deporte/s: Cadena de caracteres (100)

Disponibilidad de material: SI/NO

Descripción del material: Cadena de caracteres (100)

RDW1.5: Datos de jugador almacenado

Los mismos que RDW1.5

RF1.6: Aplicar sanción

Entrada: Agente externo: administrador. Acción: aplicar sanción a jugador.

Requisito de datos de entrada RDE1.6.

BD: Requisito de datos de escritura RDW1.6.

Salida: Agente externo: administrador. Acción: confirmación resultado.

Requisito de datos de salida: ninguno.

RDE1.6: Datos relativos a la sanción

Nombre de usuario (único): Cadena de caracteres (20)

Descripción de la sanción: Cadena de caracteres (500)

RDW1.6: Datos almacenados de sanción

Para el jugador cuyo Nombre de usuario (único) sea el proporcionado por RDE1.6, se modifica el pertinente campo de sanción en RDW1.1

RF1.7: Inscribirse a encuentro

Entrada: Agente externo: usuario. Acción: Solicitar inscripción al encuentro.Requisito de datos de entrada: RDE1.7.

BD: Requisito de datos de escritura RDW 1.7.

Salida: Agente externo: usuario. Acción: Confirmar encuentro.

Requisitos de datos de salida: ninguno.

RDE1.7: Datos del código del encuentro

Código de encuentro: cadena de caracteres alfanuméricos (5)

RDR1.7: Datos del jugador leídos

Sancionado: 1 -> si el jugador está sancionado o 0 -> si el jugador no lo está (opción por defecto)

RDW 1.7: Datos del jugador inscrito al encuentro.

Nombre de usuario (único): Cadena de caracteres (20)

RS1.1: Todos los campos de jugador, excepto los que se inicializan por defecto, no pueden ser vacíos al crearse

RF: RF1.1

RD(s): RDE1.1

Descripción: “Si alguno de los campos necesarios para crear un jugador (menos los que no se completan sino que se inicializan por defecto) es vacío, no se realiza la inserción y se pide que se corrija”

RS1.2: Todos los campos de jugador, excepto los que se inicializan por defecto, no pueden ser vacíos al modificarse

RF: RF1.1

RD(s): RDE1.1

Descripción: “Exactamente igual que el RS1.1 pero en vez de crear los jugadores, a la hora de modificarlos”

RS1.3: Un jugador solo puede estar relacionado con un único nombre de usuario

RF: RF1.1

RD(s): RDE1.1

Descripción: “Cada jugador debe tener un único nombre de usuario, por lo que si se crea un nuevo jugador con un nombre de usuario ya existente en la BD, se notifica, no se realiza la inserción y se pide que se añade uno nuevo”.

RS1.3: El nombre de usuario tiene limitaciones en cuanto a caracteres especiales

RF: RF1.1

RD(s): RDE1.1

Descripción: “Los nombres de usuario pueden incluir letras (a-z), números (0-9) y puntos (.). Además dichos nombres de usuario no pueden incluir el signo et (&), el signo igual (=), guiones bajos (_), apóstrofos ('), guiones (-), corchetes (< >), el signo más (+), comas (,) ni dos o más puntos seguidos”.

RS1.5: Un jugador sancionado no puede inscribirse a encuentros

RF: RF1.6 RF1.7

RD(s): RDE1.1

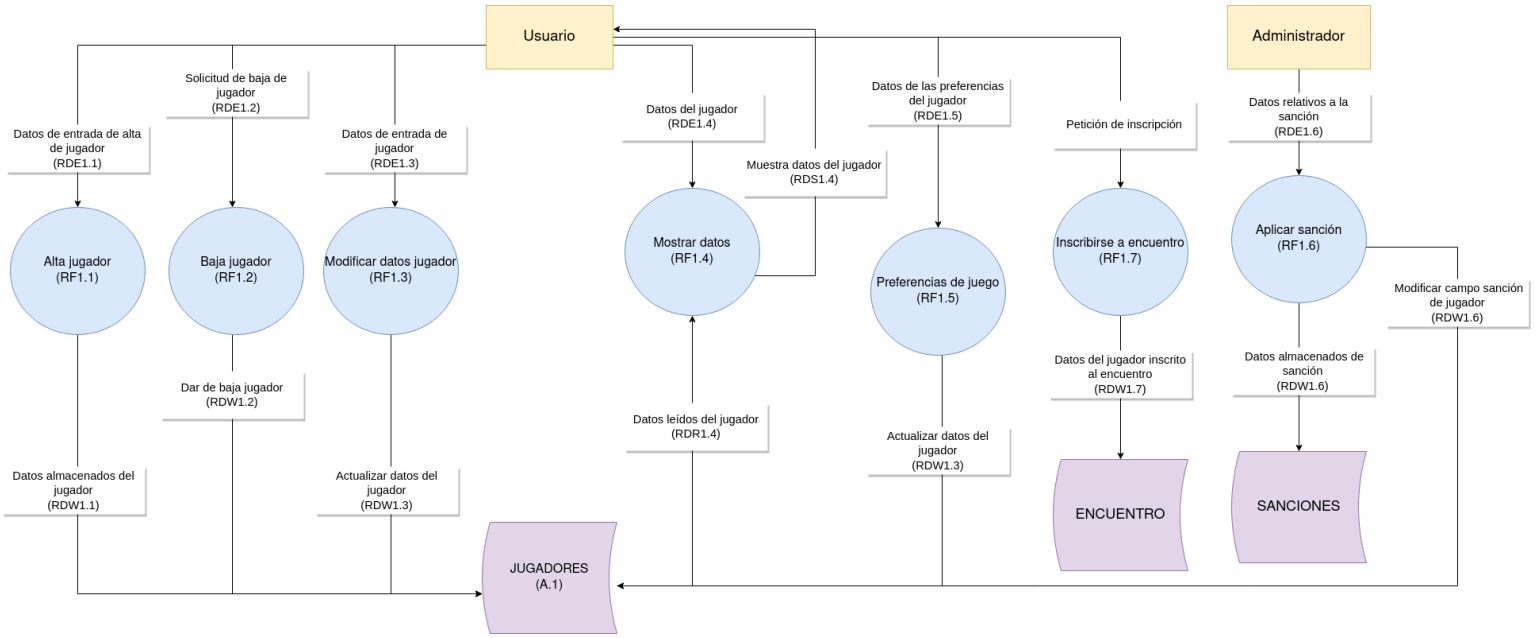
Descripción: “Un jugador que esté sancionado, no podrá buscar encuentros hasta que dicha sanción termine y se modifique el pertinente campo de ‘sanción’ ".

RS1.6: Un jugador no puede inscribirse a más de un encuentro para la misma franja horaria en un día en concreto

RF: RF1.7

RD(s): RDE1.1

Descripción: “Si un jugador está inscrito en un encuentro para una franja horaria en (por ejemplo de 17:00 - 19:00) un dia en concreto, no puede inscribirse a otro con las mismas características”



2. **GESTIÓN DE EQUIPOS:** *(ÁNGEL G)*

El sistema contará con un subsistema de gestión de equipos con distintas funcionalidades. De cada equipo **dado de alta** será almacenado un nombre único en el sistema (identificación) de entre 3 y 20 caracteres, también dispondrá de un tag es decir una abreviación de 3 caracteres, el deporte al que pertenece tal equipo (cadena de hasta 20 caracteres), y una lista con el nombre de usuario (único) de los jugadores pertenecientes a el equipo.

Para dar de alta al equipo el usuario introducirá únicamente el nombre identificador del equipo, la abreviatura y el deporte al cual pertenece ese equipo, el usuario creador del equipo tendrá permisos de líder es decir podrá **modificar los datos del equipo,** así como **dar de baja al equipo** o **inscribir al equipo en un encuentro** ya existente. Los administradores del sistema podrán **sancionar a un equipo** dado un motivo (cadena de hasta 100 caracteres) y la sanción será indicada con 0 si no está sancionado y 1 si sanción, en el motivo se concretará si la sanción es temporal o permanente por tanto también se tendrá en cuenta la hora a la que se aplique la sanción (date).

Será necesario también poder **mostrar la información del equipo** así como los datos de entrada definidos con anterioridad sobre el equipo y una lista con el nombre de los miembros que componen al mismo.

RF 2.1: Alta equipo

Entrada: Agente externo: usuario. Acción: solicitar registro de un nuevo equipo

Requisito de datos de entrada RDE 2.1

BD**:** Requisito de datos de lectura RDW 2.1

Salida:Agente externo:usuario. Acción:confirmación resultado.

Requisito de datos de salida: ninguno.

RDE 2.1**:** Datos de entrada del equipo

Nombre de equipo: cadena caracteres (20)

Deporte: cadena de caracteres (20)

Abreviatura del nombre: cadena de caracteres (3)

RDW 2.1**:** Datos almacenados de equipo

lider: cadena de caracteres (20)

Nombre del usuario creador del equipo: cadena de hasta 20 caracteres.

RF 2.2: Baja equipo

Entrada**:** Agente externo: usuario. Acción: solicitar borrado de equipo.

Requisito de datos de entrada RDE 2.2

BD: Requisitos de datos de escritura RDW 2.2.

Salida: Agente externo: usuario. Acción: confirmación resultado.

Requisito de datos de salida: ninguno.

RDE 2.2**:** Datos de entrada del equipo

Nombre de equipo: cadena caracteres (20)

RDW 2.2: Datos almacenados de equipo

Nombre de equipo, abreviatura, deporte, líder, lista de nombres de jugadores asociados.

RF 2.3: **Invitar jugador a equipo**

Entrada: Agente externo: usuario (líder). Acción: solicitar inserción.

Requisito de datos de entrada RDE 2.3.

BD: Requisito de datos de escritura RDW 2.3.

Salida: Agente externo: usuario (líder). Acción: confirmación resultado. Datos de invitación a equipo. Requisitos de datos de salida: ninguno.

RDE 2.3: Datos de entrada de jugador

Nombre de usuario: cadena de 20 caracteres.

RDW 2.3: Nombre del usuario añadido a la lista de jugadores miembros del equipo bajo confirmación.

jugador aceptar o rechazar invitación (mandar invitación)

RF 2.4: **Echar jugador de equipo**

Entrada: Agente externo: usuario (líder de equipo). Acción: solicitar borrado.

Requisito de datos de entrada RDE 2.4.

BD: Requisito de datos de escritura RDW 2.4.

Salida: Agente externo: usuario. Acción: confirmación resultado.

Requisito de datos de salida: ninguno.

RDE 2.4: Datos de entrada de baja de jugador en equipo

Nombre de usuario: cadena de hasta 20 caracteres.

RDW 2.4: Se elimina de la lista de miembros el jugador seleccionado.

Nombre de usuario: cadena de hasta 20 caracteres.

RF 2.5: **Modificar datos del equipo**

Entrada: Agente externo: usuario. Acción: solicitar modificación de datos del equipo.

Requisito de datos de entrada RDE 2.5.

BD:Requisito de datos de escritura RDW 2.5.

Salida: Agente externo: usuario. Acción: confirmación resultado.

Requisito de datos de salida: ninguno.

RDE 2.5: Datos de entrada de modificar datos de equipo

Nombre de equipo: cadena caracteres (20)

Deporte: cadena de caracteres (20)

Abreviatura del nombre: cadena de caracteres (3)

Nombre de usuario de nuevo líder: cadena de hasta 20 caracteres.

RDW 2.5: Datos modificados del equipo

Los mismos datos de RDE 2.5. y

Nombre de usuario de nuevo líder: cadena de hasta 20 caracteres.

RF 2.6: **Inscripción a encuentro**

Entrada: Agente externo: usuario (líder de equipo). Acción: Solicitar inscripción al encuentro.

Requisito de datos de entrada: ninguno.

BD: Requisito de datos de escritura RDW 2.6.

Salida: Agente externo: Usuarios (líderes de equipos). Acción: Confirmar encuentro.

Requisitos de datos de salida: RDS 2.6.

RDE2.6: Datos del código del encuentro

Código de encuentro: cadena de caracteres alfanuméricos (5)

Nombre de equipos: cadena de caracteres (20).

RDW 2.6: Datos del equipo inscrito al encuentro.

Nombre de equipo: cadena caracteres (20).

RDS 2.6: Nombres de los equipos implicados en el encuentro.

Nombre de equipo: cadena caracteres (20).

Nombre de equipo: cadena caracteres (20).

RF 2.7: **Aplicar sanción**

Entrada: Agente externo: administrador. Acción: Aplicar sanción.

Requisitos de datos de entrada: RDE 2.7.

BD: Requisito de datos de escritura: RDW 2.7 (nombre)

Salida: Agente externo: administrador. Acción: confirmación resultado.

Requisito de datos de salida: ninguno.

RDE 2.7: Datos de entrada de aplicar sanción.

Descripción sanción: cadena caracteres (100).

Tipo de sanción: 1 -> si el equipo está sancionado o 0 -> si el equipo no lo está (opción por defecto)

Hora: Date

RDW 2.7: Datos almacenados de la sanción

Los mismos datos que en RDE 2.7.

RF 2.8: **Mostrar equipo**

Entrada: Agente externo: Usuario. Acción: Solicitar información de equipo.

Requisitos de datos de entrada: RDE 2.8.

BD: Requisito de datos de lectura: RDR 2.8.

Salida: Agente externo: Usuario. Acción: confirmación resultado. Requisito de datos de salida RDS 2.9.

RDE 2.8: Datos de entrada de mostrar equipo

Nombre de equipo: cadena de caracteres (20).

RDR 2.8: Datos de equipo.

RDS 2.8: Datos de RDE 2.1 y nombres listados de los miembros que lo componen.

RS 2.1: Nombre de equipo, deporte y abreviatura no pueden ser vacíos.

RF: RF 2.1

RD(S): RDE 2.1

Descripción: “Si los campos de nombre de equipo, abreviatura y deporte no han sido rellenados no se realiza la inserción y se pide que se corrija. ”

RS 2.2: Nombre de equipo debe corresponderse con ese único equipo y longitud de nombre entre 3 y 20 caracteres.

RF: RF 2.1

RD(S): RDE 2.1

Descripción: “Si ya existe un equipo con ese nombre, no se realiza la inserción y se devuelve error”

RS 2.3: Nombre del jugador debe existir en el sistema

RF: RF 2.3

RD(S): RDE 2.3

Descripción: “Si el nombre del jugador no se coincide con ningún jugador (usuario) registrado en el sistema no se realiza la inserción y se devuelve error.”

RS 2.4: El jugador no debe de pertenecer anteriormente al equipo

RF: RF 2.3

RD(S): RDE 2.3, RDW 2.3

Descripción: “Si el jugador ya pertenece al equipo no se inserta en la lista de miembros y se devuelve un error.”

RS 2.5: El jugador debe de pertenecer al equipo

RF: RF 2.4

RD(S): RDE 2.4

Descripción: “El jugador debe pertenecer al equipo de lo contrario no se hace nada y se muestra un error”

RS 2.6**:** Nombre de equipo, deporte, abreviatura y nombre de usuario del lider no pueden ser vacíos.

RF: RF 2.5

RD(S): RDE 2.5

Descripción: “Si los campos de nombre de equipo, abreviatura y deporte no han sido rellenados no se realiza la inserción y se pide que se corrija. ”

RS 2.7: Nombre de equipo debe corresponderse con ese único equipo

RF: RF 2.5

RD(S): RDE 2.5

Descripción: “Si ya existe un equipo con ese nombre, no se realiza la modificación y se devuelve error”

RS 2.8: El equipo no debe estar inscrito al encuentro con anterioridad.

RF: RF 2.6

RD(S): RDW 2.6

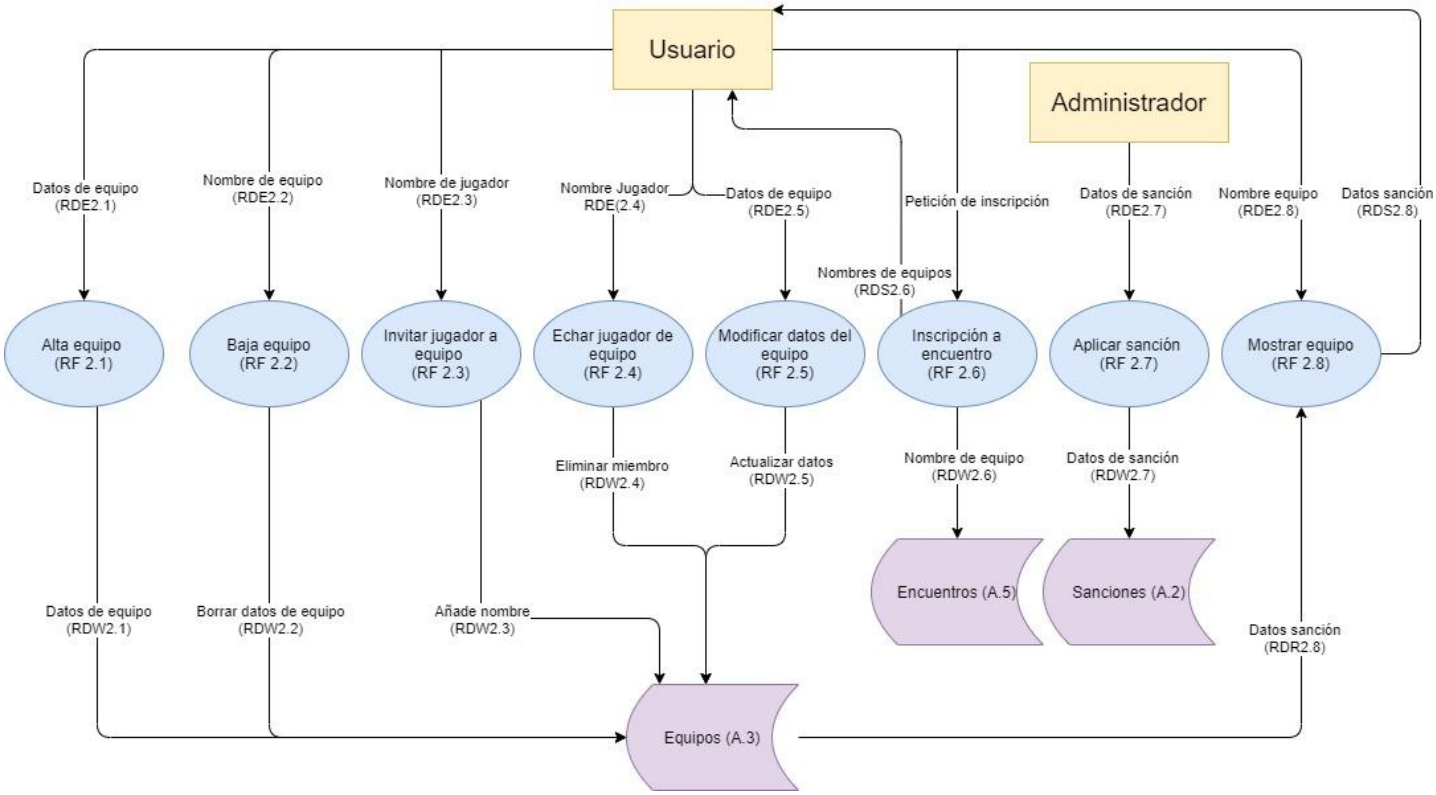
Descripción: “Si el equipo ya está inscrito no se realiza inserción y se devolverá un error (además solo podrá realizar la inscripción el líder del equipo)”

RS 2.9: La descripción, el tipo de sanción no deben ser vacíos.

RF: RF 2.7

RD(S): RDE 2.7

Descripción: “Si la descripción, el tipo o la hora son vacíos no se aplicará la sanción al equipo y se devolverá un error.”



3. **GESTIÓN DE UBICACIONES:** *(CRISTIAN F)*

Para organizar los encuentros, es necesario seleccionar una ubicación registrada con anterioridad. Antes debe de haberse **dado de alta** por un administrador o por un mismo usuario (requerirá verificación por admin), para ello almacenaremos su localización (en una serie de hasta 100 caracteres alfanuméricos). La localización debe ser reconocida por el sistema de mapas de Google. Además los deportes disponibles para practicar en la pista (seleccionar entre todos los registrados en la aplicación). También, como es obvio, se guardará la hora de apertura y cierre (con valores de formato HH:MM). La hora de apertura y cierre no pueden coincidir, y la primera debe ser anterior a la segunda. Es necesario registrar también las dimensiones del campo (en un real para largo, y en otro para ancho, ambos en metros). No debemos olvidar añadir una breve descripción del estado actual de la pista (en una serie de hasta 500 caracteres alfanuméricos). Por último ha de determinarse si la pista ha de ser alquilada mediante un pago (selección entre SÍ y NO, en el caso positivo habrá que añadir un valor real). Para **dar de baja** una ubicación, normalmente por mal estado, el administrador deberá proporcionar justificación e imágenes, que habrá recibido de la retroalimentación producida por las quejas de los usuarios. El sistema también permitirá **mostrar los datos de una ubicación** deportiva con todos sus datos a petición del usuario. Para su utilización por parte de los miembros, se debe **alquilar el uso de la pista**, almacenando fecha y duración, además de los usuarios de las personas responsables (capitanes de equipos o persona involucrada en el deporte individual), si el alquiler conlleva un pago se informará y redirigirá a una plataforma de pago. A la hora de alquilar, la fecha debe de comprobarse en día de apertura y disponibilidad, y las horas deben de seleccionarse una anterior a la otra. Cuando un usuario esté buscando una ubicación concreta para un encuentro, se **mostrará la disponibilidad actual**, mediante un calendario semanal.

RF3.1: Dar de alta una ubicación

Entrada: Agente externo: usuario, administrador. Acción: solicitar inserción.

Requisito de datos de entrada: RDE3.1.

BD: Requisito de datos de escritura RDW3.1

Salida: Agente externo: administrador. Acción: confirmación resultado.

Requisito de datos de salida: RDS3.1

RDE3.1: Datos de entrada de alta de ubicación.

Localización: Cadena de caracteres alfanuméricos (100)

Deportes: Cadena de caracteres (20)

Descripción estado actual: Cadena de caracteres alfanuméricos (500)

Alquiler mediante pago: 0 ó 1 (No ó Sí)

Cantidad de pago: Número real

Hora apertura: DATE con formato HH:MM

Hora cierre: DATE con formato HH:MM

RDW3.1: Datos almacenados de ubicación.

Los mismos datos que RDE3.1

Código ubicación (generado): Cadena de caracteres alfanuméricos (5)

RDS3.1: Código ubicación: Cadena de caracteres alfanuméricos (5)

RF3.2: Dar de baja una ubicación

Entrada: Agente externo: administrador. Acción: solicitar borrado.

Requisito de datos de entrada RDE3.2.

BD: Requisito de datos de escritura RDW3.2

Salida: Agente externo: administrador. Acción: confirmación resultado.

Requisito de datos de salida: ninguno.

RDE3.2: Datos de entrada de baja de ubicación.

Los mismos datos que RDS3.1

Justificación: Cadena de caracteres alfanuméricos (500)

RDW3.2: Datos almacenados de ubicación.

Los mismos datos que RDE3.2

fueraServicio

RF3.3: Mostrar datos de ubicación

Entrada: Agente externo: usuario. Acción: solicitar información.

Requisito de datos de entrada: RDE3.3

BD: Requisito de datos de lectura RDR3.3

Salida: Agente externo: usuario. Acción: confirmación resultado.

Requisito de datos de salida: RDS3.3

RDE3.3:

Localización aproximada: Cadena de caracteres alfanuméricos (100)

RDR3.3: Datos de ubicación almacenada

Los mismos datos que RDW3.1

RDS3.3: Datos de ubicación, los mismos datos de RDR3.3

RF3.4: **Alquilar el uso de la ubicación**

Entrada: Agente externo: usuario. Acción: solicitar modificación.

Requisito de datos de entrada: RDE3.4

BD: Requisito de datos de lectura RDR3.4. Requisito de datos de escritura RDW3.4.

Salida: Agente externo: Usuario. Acción: confirmación alquiler (y pago si necesario)

Requisito de datos de salida: RDS3.4

RDE3.4:

Los mismos datos de RDE3.3

Deporte: Cadena de caracteres (20)

Fecha: Tipo de dato DATE

Hora entrada: DATE con formato HH:MM

Hora de salida: DATE con formato HH:MM

RDR3.4:

Los mismos datos de RDE3.4

Disponibilidad : 0 ó 1 (Libre o Ocupado)

RDW3.4:

Fecha: Tipo de dato DATE

Hora entrada: DATE con formato HH:MM

Hora de salida: DATE con formato HH:MM

Disponibilidad: 1 (Ocupado)

Usuarios responsables: Lista de cadenas de caracteres (20)

RDS3.4:

Precio alquiler: Numero real (por defecto 0.00)

RF3.5: **Mostrar disponibilidad**

Entrada: Agente externo: usuario. Acción: solicitar información.

Requisito de datos de entrada: RDE3.5

BD: Requisito de datos de lectura: RDR3.5

Salida: Agente externo: usuario. Acción: ninguna

Requisito de datos de salida: RDS3.5

RDE3.5:

Los mismos datos de RDE3.4

RDR3.5: Datos de disponibilidad de ubicación

Los mismos datos de RDR3.4

nomUsuario

Disponibilidad : 0 ó 1 (Libre o Ocupado si la fecha y el deporte no estén ocupados)

RDS3.5:

Si Disponibilidad 1 → NO DISPONIBLE.

Si Disponibilidad 0 → DISPONIBLE.

RS3.1: Localización, deportes, largo, dimensión ancho y alquiler mediante pago no pueden ser vacíos.

RF: RF3.1

RD(s): RDE3.1

Descripción: “Si Localizacion, Deportes, Dimensión largo, Dimensión ancho o Alquiler mediante pago son vacíos, no se realiza la inserción y se pide que se corrija”

RS3.2: La localización debe existir.

RF: RF3.1

RD(s): RDE3.1

Descripción: “La dirección de una pista debe de ser reconocida por Google Maps, si no existe, no se realiza la inserción y se pide que se corrija”

RS3.3: La hora de apertura y cierre no se pueden pisar

RF: RF3.1

RD(s): RDE3.1

Descripción: “Si hora de cierre es igual o anterior a hora de apertura, no se realiza la inserción y se pide que se corrija. Si no se rellenan estos campos se supone pista abierta 24h”

RS3.4: La hora de alquiler y de finalización no se pueden pisar

RF: RF3.4

RD(s): RDE3.4

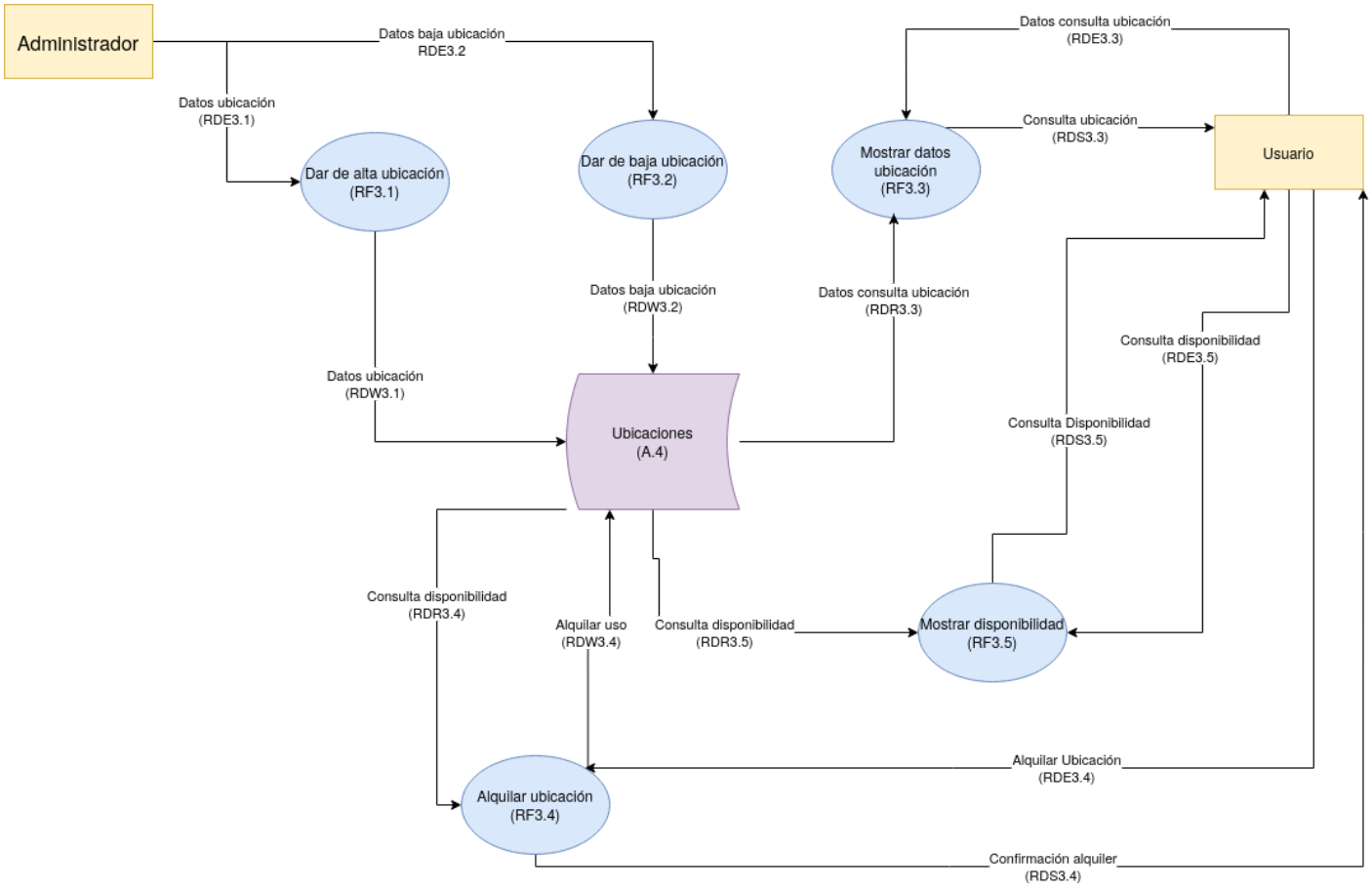
Descripción: “Si hora de finalización es igual o anterior a hora de alquiler, no se realiza la reserva y se pide que se corrija.”

RS3.5: La fecha introducida debe de estar disponible para el alquiler de la pista

RF: RF3.4

RD(s): RDE3.4, RDR3.4

Descripción: “Si para la fecha introducida no está disponible la pista, no se realiza la reserva y se pide que se añada otra fecha”



4. **GESTIÓN DE ENCUENTROS:** *(SANTIAGO M)*

Por otro lado este sistema de información deberá contener un subsistema que se encargue de la gestión de **encuentros deportivos**. Para cada encuentro, almacenaremos el deporte que se vaya a jugar(en una serie de hasta 20 caracteres), la hora de comienzo(en un valor de tipo date), su ubicación(en una serie de hasta 100 caracteres alfanumericos), el material y personal requerido(en una serie de hasta 100 caracteres) y la apuesta de puntos(en una valor entero de hasta 6 dígitos). Para **crear un nuevo encuentro**, un usuario deberá especificar la hora y el lugar , el material y personal necesario y la apuesta en puntos del sistema o en dinero de dicho encuentro, si dicho usuario tiene ya un equipo deberá indicarlo, el usuario que cree el encuentro será el líder del encuentro, dicho encuentro tendrá asignado un código de encuentro(cadena de 5 caracteres alfanuméricos), si algunos de estos campos menos apuesta son vacíos,no se realiza la inserción y se

pide que se corrija, si la ubicación no existe o no consta como lugar para realizar el deporte,no se realiza la inserción y se pide que se corrija. Tanto en la creación de un encuentro **o su modificación** debe realizarse 24 horas antes de su diputación, si no se cumple no se realiza la inserción y se pide que se corrija. Un **encuentro se puede cancelar** hasta 12 horas antes del partido y para ello el usuario deberá explicar los motivos de dicha decisión, si estos no son suficientes para el sistema, esté cancelará la solicitud, tanto la modificación como la creación de un encuentro solo puede ser realizada por el usuario líder del encuentro. El subsistema también permitirá **mostrar la información del encuentro** con todos sus datos a petición del usuario. Un usuario o equipo pueden **buscar un encuentro que se ajuste a unos criterios de búsqueda** tales como distancia, hora, edad media, puntos en juego , media de nivel y deporte.La hora debe estar dentro del rango de 8 horas de la mañana hasta las 20 horas de la tarde, si no se cumple no se realiza la búsqueda. Al final del encuentro el usuario líder del encuentro **debe establecer el resultado del equipo local y visitante**, si el resultado del local o el del visitante es negativo, el sistema no realiza la inserción y se pide que se corrija, el sistema solo dejará poner el resultado una hora después de su comienzo.

RF 4.1: **Crear encuentro**

Entrada: Agente externo:usuario. Acción:solicitar creación de encuentro.

Requisito de datos de entrada RDE 4.1

BD: Requisito de datos de lectura RDW 4.1

Salida: Agente externo:usuario. Acción:confirmación resultado.

Requisito de datos de salida:ninguno.

RDE 4.1: Datos de entrada de creación de encuentro

- Deporte: cadena de caracteres(20).
- Hora: valor DATE.
- Ubicación: cadena de caracteres alfanumericos (100).
- Material y personal: cadena de caracteres (100).
- Apuesta: valor entero (6), por defecto 0.

RDW 4.1: Datos almacenados del encuentro

- Los mismos datos que RDE 4.1.
- Código de encuentro: cadena de caracteres alfanuméricos (5)
- Líder del encuentro(Nombre de usuario):cadena de caracteres(20).
- Puntuación local:número entero(3 dígitos), por defecto 0.
- Puntuación visitante:número entero(3 dígitos), por defecto 0.

RF4.2: **Cancelar encuentro**

Entrada: Agente externo: usuario líder del encuentro. Acción: solicitar cancelación.

Requisito de datos de entrada RDE 4.2

BD: Requisitos de datos de escritura RDW 4.2

Salida: Agente externo:usuario. Acción: confirmación resultado.

Requisitos de datos de salida: ninguno.

RDE 4.2: Datos de entrada de cancelar encuentro

Código de encuentro: cadena de caracteres alfanuméricos (5)

Motivos:cadena de caracteres(50).

RDW 4.2: Datos almacenados de encuentro.

Los mismos datos que RDW 4.1.

RF 4.3: **Modificar encuentro**

Entrada: Agente externo:usuario líder del encuentro. Acción:Modificar encuentro.

Requisito de datos de entrada RDE 4.3

BD: Requisitos de datos de escritura RDW 4.3

Salida: Agente externo:usuario. Acción:confirmación resultado.

Requisitos de datos de salida:ninguno.

RDE 4.3: Datos de entrada de modificar encuentro

Código de encuentro: cadena de caracteres alfanuméricos (5)

Los mismos datos que se quieran cambiar de RDE 4.1

/*NUEVO P2

RDR 4.3: Los mismos datos que RDE 4.3

*/

RDW 4.3: Datos modificados de encuentro

Los mismos datos que RDE 4.3

RF 4.4: **Mostrar información del encuentro**

Entrada:Agente externo:usuario. Acción:solicitar información encuentro.

Requisitos de datos de entrada:ninguno

BD:Requisitos de datos de lectura RDR 1.4

Salida:Agente externo:usuario. Acción:confirmación resultado

Requisitos de datos de salida:RDS 1.4

RDE 4.4:Datos de entrada para buscar encuentro

Código de encuentro: cadena de caracteres alfanuméricos (5).

/*NUEVO P2

RDR 4.4: Datos de encuentro leídos

Los mismos datos que RDW 4.1

*/RDS 4.4: Información del encuentro.

Los mismos datos que RDR 4.4

RF 4.5: **Búsqueda de encuentros**

Entrada:Agente externo:usuario. Acción:solicitar búsqueda de encuentros.

Requisitos de datos de entrada: RDE 4.5

BD: Requisitos de datos de lectura: RDR 4.5

Salida: Agente externo:usuario. Acción:confirmación resultado.

Requisitos de datos de salida: RDS 4.5

RDE 4.5: Datos de entrada de búsqueda de encuentros, busca según un filtro.

Ubicacion:cadena de caracteres(100 caracteres),int apuesta=0;

Hora: valor tipo DATE.

Apuesta: valor entero(6 dígitos)

Deporte: cadena de caracteres(20)

RDR 4.5:Datos de encuentros almacenados.

Los mismos datos que RDW 4.1

RDS 4.5:Listado de encuentros con su respectiva información, los mismo datos que RDR 4.5

RF 4.6:**Establecer resultado del encuentro**

Entrada:Agente externo:usuario líder del encuentro. Acción: establecer resultado.

Requisitos de datos de entrada: RDE 4.6

BD: Requisitos de datos de escritura: RDW 4.6

Salida: Agente externo: usuario líder del encuentro. Acción:confirmación resultado.

Requisitos de datos de salida: ninguno

RDE 4.6: Datos de entrada de establecer resultado

Código de encuentro: cadena de caracteres alfanuméricos (5)

Puntuación local:número entero(3 dígitos).

Puntuación visitante:número entero(3 dígitos).

RDW 4.6:Datos de encuentros almacenados.

Los mismos datos que RDE 4.6

RS 4.1:Deporte, hora, ubicación y material no pueden ser vacíos.
RF:RF 4.1
RD:RDE 4.1
Descripción:"Si deporte, hora, edad media, ubicación y material son vacíos,no se realiza la inserción y se pide que se corrija."

RS 4.2:La ubicación debe existir.
RF:RF 4.1
RD:RDE 4.1
Descripción:"Si la ubicación no existe o no consta como lugar para realizar el deporte,no se realiza la inserción y se pide que se corrija."

RS 4.3:La hora del partido debe ser válida para el sistema.
RF:RF 4.1
RD:RDE 4.1
Descripción:"La hora tiene que corresponder al formato DD/MM/YY HH:mm y se pide que se corrija."

/*NO LA TENGO EN CUENTA EN LA P3

RS 4.4:La cancelación del encuentro debe ser 12 horas antes de la realización del mismo.
RF:RF 4.2
RD:RDE 4.2
Descripción:"Si el encuentro se quiere cancelar 12 horas antes del encuentro, el sistema cancelará la solicitud."
/*NO LA TENGO EN CUENTA EN LA P3

RS 4.5: Los motivos para cancelar el encuentro deben ser suficientes para el sistema .
RF:RF 4.2
RD:RDE 4.2
Descripción:"El usuario debe dar los motivos necesarios para cancelar el encuentro, si no son suficientes para el sistema, este cancelará la solicitud."

/*NO LA TENGO EN CUENTA EN LA P3
RS 4.6: La modificación del encuentro debe ser 24 horas antes de la realización del mismo.

RF: RF 4.3

RD: RDE 4.3

Descripción: ”Si el encuentro se quiere modificar 24 horas antes del encuentro, el sistema cancelará la solicitud.”

*/NO LA TENGO EN CUENTA EN LA P3

RS 4.7:La hora del partido debe ser válida para el sistema.

RF:RF 4.5

RD:RDE 4.5

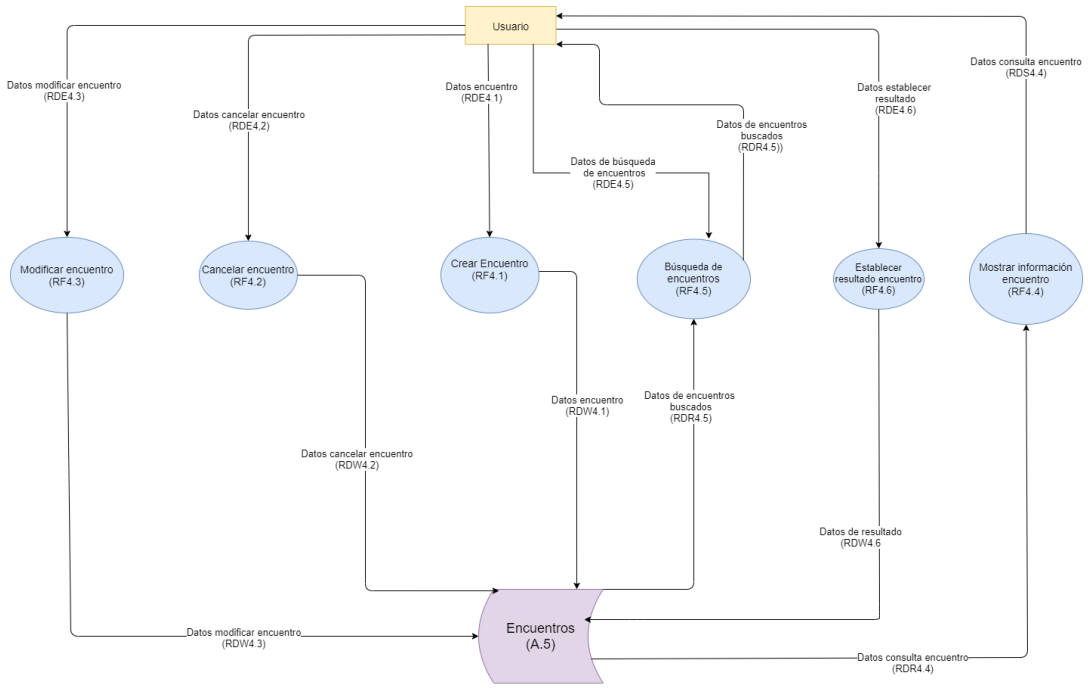
Descripción:"La hora tiene que corresponder al formato DD/MM/YY HH:mm y se pide que se corrija"

RS 4.8:Tanto el resultado local como el visitante no pueden ser negativos.

RF:RF 4.6

RD:RDE 4.6

Descripción:” Si el resultado del local o el del visitante es negativo, el sistema no realiza la



inserción y se pide que se corrija.”

5. GESTIÓN DE VALORACIONES/ALEGACIONES: *(FRAN)*

Aparte el usuario puede **crear valoraciones** de los jugadores que se almacenarán en la base de datos, para ello deberá introducir un título para la valoración (cadena hasta 50 caracteres), el nombre del jugador (cadena hasta 50 caracteres) y un comentario (cadena hasta 500 caracteres), ninguno de estos campos puede estar vacíos al crear la valoración, por cada valoración el sistema creará un código único (cadena de caracteres alfanuméricos de hasta 5 caracteres) para poder identificarlas. Además el usuario podrá **consultar valoraciones** de los jugadores de todos los usuarios incluyendo las suyas, y se le mostrará una lista con las valoraciones. También el usuario puede **modificar sus valoraciones**, el sistema le muestra una lista con sus valoraciones y el usuario modifica el comentario de la que quiera introduciendo el código de la valoración que quiera modificar, el comentario no puede estar vacío al modificar la valoración a partir del código de valoración sino el sistema mandará un mensaje para que se corrija y la valoración debe ser encontrada para modificarla en el sistema de lo contrario se informará con un error.. Además, el usuario puede **borrar sus valoraciones**, el sistema le muestra una lista con sus valoraciones y el usuario borrará la valoración de la que quiera introduciendo el código de la que quiera borrar, la valoración debe ser encontrada para borrarla en el sistema a partir del código de la valoración de lo contrario se informará con un error.. Aparte el usuario podrá **crear quejas** sobre jugadores, en las que deberá introducir título de la queja (cadena hasta 50 caracteres), tipo de accidente (cadena hasta 100 caracteres), nombre Usuario (cadena de caracteres hasta 20) y el comentario (cadena hasta 500 caracteres), el sistema crea un código único para cada queja poder identificarlas (cadena de caracteres alfanuméricos de hasta 5 caracteres), estas quejas se envían al administrador, que tendrá consecuencias en posibles baneos o avisos sobre los jugadores, equipos o eliminación de ubicaciones, si título, tipo de incidente, categoría o comentario son vacíos, no se realiza la inserción y se pide que se corrija

RF5.1: **Crear valoración**

Entrada: Agente externo: **Usuario**. Acción: **Solicitar inserción**. Requisito de datos de entrada: RDE5.1.

BD: Requisito de datos de escritura: RDW5.1.

Salida: Agente externo: **Usuario**. Acción: **Confirmación de resultado**. Requisito de datos de salida: RDS5.1.

RDE5.1: Datos de entrada de crear valoración

Título: Cadena de caracteres (50)

Nombre Usuario: cadena de caracteres(20)

Comentario: cadena de caracteres (500)

RDW5.1: Datos almacenados en valoraciones.

Los mismos datos que RDE5.1

Añadimos también Código de valoración.

RDS5.1: Código valoración: Cadena de caracteres alfanuméricos (5)

RF5.2: **Consultar valoraciones.**

Entrada: Agente externo: **Usuario** . Acción: **Solicitar información**. Requisito de datos de entrada: RDE5.2.

BD: Requisito de datos de lectura: RDR5.2.

Salida: Agente externo: **Usuario**. Acción:**Confirmación de resultado**. Requisito de datos de salida: RDS5.2.

RDE5.2: Datos de entrada para filtrar las valoraciones.

nombreUsuario : cadena de caracteres (20)

RDR5.2: Datos de valoraciones almacenados, los mismos que RDW5.1

RDS5.2: Listado de valoraciones según la consulta

Mismos datos que RDR5.2

RF5.3: **Modificar valoración**

Entrada: Agente externo: **Usuario**. Acción: **Solicitar modificación** . Requisito de datos de entrada: RDE5.3.

BD: Requisito de datos de lectura: RDR5.3. Requisito de datos de escritura: RDW5.3.

Salida: Agente externo: **Usuario**. Acción: **confirmación resultado** . Requisito de datos de salida: ninguno.

RDE5.3: Comentario modificado de la valoración indicada a partir del código de la valoración

- Código valoración: Cadena de caracteres alfanuméricos (5)
- Comentario: cadena de caracteres (500)

RDR5.3: Datos de las valoraciones actuales.

Los mismos datos que RDR5.2

RDW5.3: Datos almacenamos de las nuevas valoraciones

RF5.4: **Borrar valoración/es**

Entrada: Agente externo: **Usuario**. Acción: **solicitar borrado** . Requisito de datos de entrada: RDE5.4.

BD: Requisito de datos de escritura: RDW5.4.

Salida: Agente externo: **Usuario**. Acción: **confirmación resultado** . Requisito de datos de salida: **ninguno** .

RDE5.4: Listado de valoraciones del usuario.

Lista de códigos de valoraciones a borrar: Lista de cadena de caracteres alfanuméricos (5)

RDW5.4: Datos suprimidos de valoraciones.

Los mismos que RDW5.1

RF5.5: **Crear queja.**

Entrada: Agente externo: **Usuario**. Acción: **Solicitud inserción**. Requisito de datos de entrada: RDE5.5.

BD: Requisito de datos de escritura: RDW5.5.

Salida: Agente externo:**Administrador**. Acción: **confirmación resultado** . Requisito de datos de salida: RFS5.5.

RDE5.5: Datos de entrada de la queja.

Título: Cadena de caracteres (50)

Tipo de incidente: cadena de caracteres (100)

Nombre Usuario: cadena de caracteres(50)

Comentario: cadena de caracteres (500)

RDW5.5: Datos almacenados de quejas. Los mismos que RDE5.5.

Añadimos también Código queja: Cadena de caracteres alfanuméricos (5)

RDS5.5: Código queja: Cadena de caracteres alfanuméricos (5)

RS5.1: Título , nombre Jugador y comentario no pueden estar vacíos

RF: RF5.1

RD(s): RDE5.1

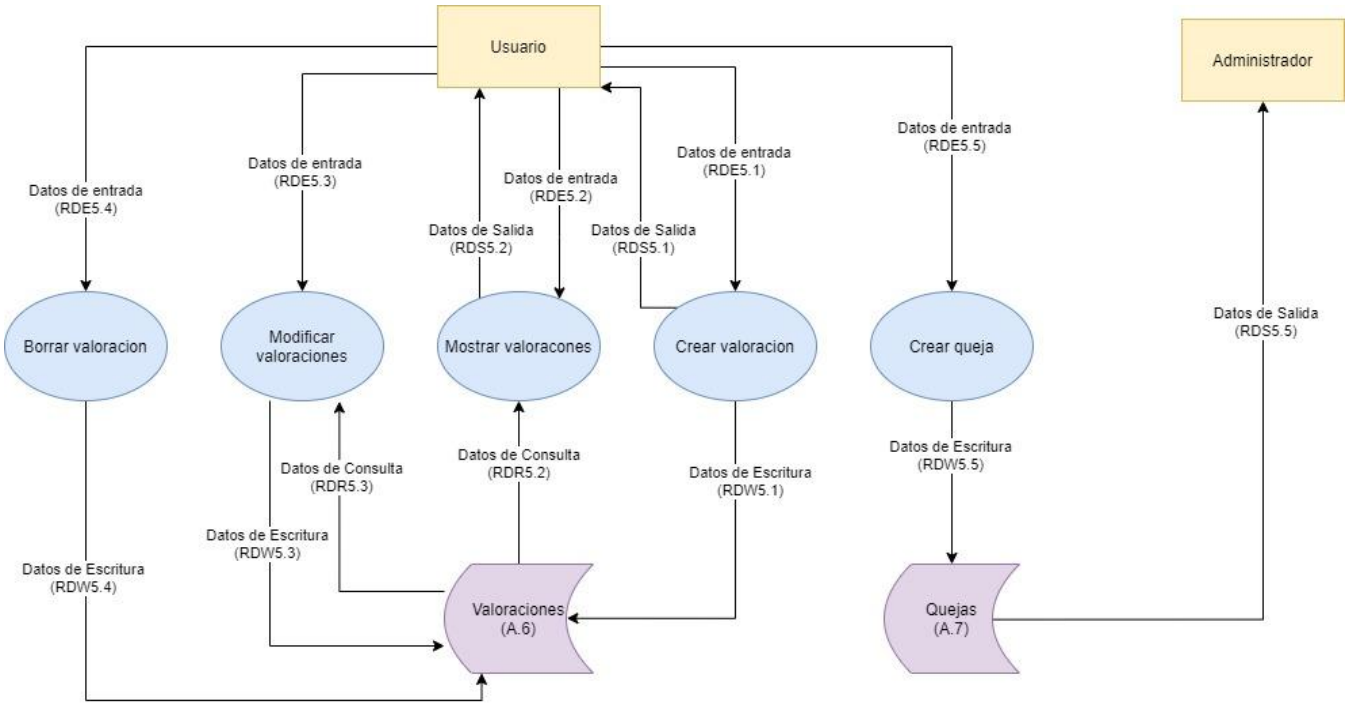
Descripción: si título, nombre Jugador o comentario son vacíos, no se realiza la inserción y se pide que se corrija

RS5.2: Título , tipo de incidente, nombre Jugador y comentario no pueden estar vacíos
RF: RF5.5
RD(s): RDE5.5
Descripción: si título, tipo de incidente, nombre Jugador o comentario son vacíos, no se realiza la inserción y se pide que se corrija

RS5.3: Comentario no puede estar vacío
RF: RF5.3
RD(s): RDE5.3
Descripción: si comentario son vacíos al modificar la valoración, no se realiza la inserción y se pide que se corrija

RS5.4: El código de valoración debe existir
RF: RF5.3
RD(s): RDE5.3
Descripción: La valoración debe ser encontrada para modificarla en el sistema de lo contrario se informará con un error.

RS5.5: El código de valoración debe existir
RF: RF5.4
RD(s): RDE5.4
Descripción: La valoración debe ser encontrada para borrarla en el sistema de lo contrario se informará con un error.



6. **GESTIÓN DE PREMIOS:** *(RAÚL C)*

Finalmente nos encontramos con un subsistema de gestión de premios, los cuales sirven para dar un incentivo a usar la aplicación. Para cada premio,a la hora de **añadirlo**, almacenamos un nombre (cadena de hasta 50 caracteres), una descripción del premio (cadena de hasta 300 caracteres) , la cantidad de premios disponibles para canjear (número natural de hasta 5 dígitos) y el precio en puntos del premio (número natural de hasta 20 dígitos), y el sistema le genera un código único (cadena alfanumérica de 8 caracteres). Ninguno de los campos mencionados puede estar vacío a la hora de crear el premio y además ,tanto el nombre como la descripción no pueden contener caracteres especiales, ya que en caso de alguna de estas circunstancias, no se realizará la inserción.

Se puede **borrar** los premios de la lista de premios indicandolo mediante el código que tenga asignado, pero este código debe de existir en la base de datos, sino, no se realiza nada.

También se tendrá disponible la opción de **mostrar la lista de todos los premios** en la cual el usuario elegirá de entre todos cual quiere canjear con sus puntos.

Para **obtener puntos** se necesitará que el usuario de el código del encuentro (cadena alfanumérica de 5 caracteres) con el cual se obtendrá la información del encuentro

para saber el resultado y más características para dar mayor número de puntos o menos. En este caso, el código de encuentro debe de existir, y en caso de que exista pero no se encuentre finalizado, lo cual significa que no hayan pasado 2 horas desde la hora de comienzo del encuentro, se informará del error y no se realizará nada.

Por último tenemos la **obtención de premios** en el cúal de la lista de premios el usuario selecciona el que desea pasando así el código único del premio. En caso de que el premio seleccionado valga más puntos de los que el usuario tiene, no se realizará nada.

RF6.1: Añadir premio

Entrada: Agente externo: **Administrador**. Acción: **Solicitar inserción**. Requisito de datos de entrada: RDE6.1.

BD: Requisito de datos de lectura:RDW6.1

Salida: Agente externo: **Administrador**. Acción: **Confirmación de resultado**. Requisito de datos de salida: **ninguno**.

RDE6.1: Datos del premio a añadir

Nombre: Cadena de caracteres(50)

Descripción: Cadena de caracteres (300)

Cantidad: Número natural (5)

Precio(en puntos): Número natural (20)

RDW6.1: Datos almacenados del premio

Los mismos datos de RDE6.1

Código único: Cadena alfanumérica (8)

RF6.2: Quitar premio.

Entrada: Agente externo: **Administrador** . Acción: **Solicitar borrado**. Requisito de datos de entrada: RDE6.2.

BD: Requisito de datos de escritura: RDW6.2.

Salida: Agente externo: **Administrador**. Acción: **Confirmación resultado**. Requisito de datos de salida: **ninguno**.

RDE6.2: Datos de entrada de eliminación de premio

Código: Cadena alfanumerica (8)

RDW6.2: Datos almacenados de premio

Los mismos datos que RDW6.1

RF6.3: Mostrar lista de premios

Entrada: Agente externo: **Usuario**. Acción: **Solicitar lista** . Requisito de datos de entrada: **ninguno**.

BD: Requisito de datos de lectura: RDR6.3.

Salida: Agente externo: **Usuario**. Acción: **confirmación resultado** . Requisito de datos de salida: RDS6.3 .

RDR6.3: Datos sobre premios almacenados

Los mismos datos que RDW6.1

RDS6.3: Listado de premios, cada uno de ellos con los mismos datos que RDR6.3

RF6.4: Obtener de puntos.

Entrada: Agente externo: **Usuario**. Acción: **solicitar puntos** . Requisito de datos de entrada: RDE6.4.

BD: Requisito de datos de escritura: RDW6.4. Requisito de datos de lectura: RDR6.4

Salida: Agente externo: **Usuario**. Acción: **confirmación resultado** . Requisito de datos de salida: **ninguno**.

RDE6.4: Datos sobre encuentro almacenado

Código de encuentro: Cadena alfanumérica (5)

RDR6.4: Datos sobre el encuentro identificado con el código de RDE6.4

RDW6.4: Datos del jugador almacenados

Puntos: Número entero (10)

RF6.5: Obtener premio.

Entrada: Agente externo: **Usuario**. Acción: **canjear puntos** . Requisito de datos de entrada: RDE6.5.

BD: Requisito de datos de escritura: RDW6.5.1 , RDW6.5.2.

Salida: Agente externo: **Usuario**. Acción: **confirmación resultado** . Requisito de datos de salida: RDS6.5.

RDE6.5: Datos sobre el premio seleccionado

Código: Cadena alfanumérica (8)

RDW6.5.1: Datos del premio almacenado cambiando la cantidad disponible

Los mismos datos que RDW6.1

RDW6.5.2: Datos del jugador almacenado con nuevos puntos

Los mismos datos que RDW1.1

RDS6.5 Datos sobre el premio obtenido

Código premio: Cadena alfanumérica (16) (se genera por la aplicación automáticamente)(se debe canjear en la tienda respectiva del premio)

RS6.1: Todos los campos del premio no pueden ser vacíos al crearse.
RF: RF6.1
RD(s): RDE6.1
Descripción: “Si alguno de los campos necesarios para crear un premio es vacío, no se realiza la inserción y se pide que se corrija”

RS6.2: El código único de premio tiene que existir
RF: RF6.2
RD(s): RDE6.2
Descripción: “En caso de que el código de premio introducido a la hora de dar de baja un premio no coincida con los códigos almacenados en la BD, se informa del error y no se realiza nada.

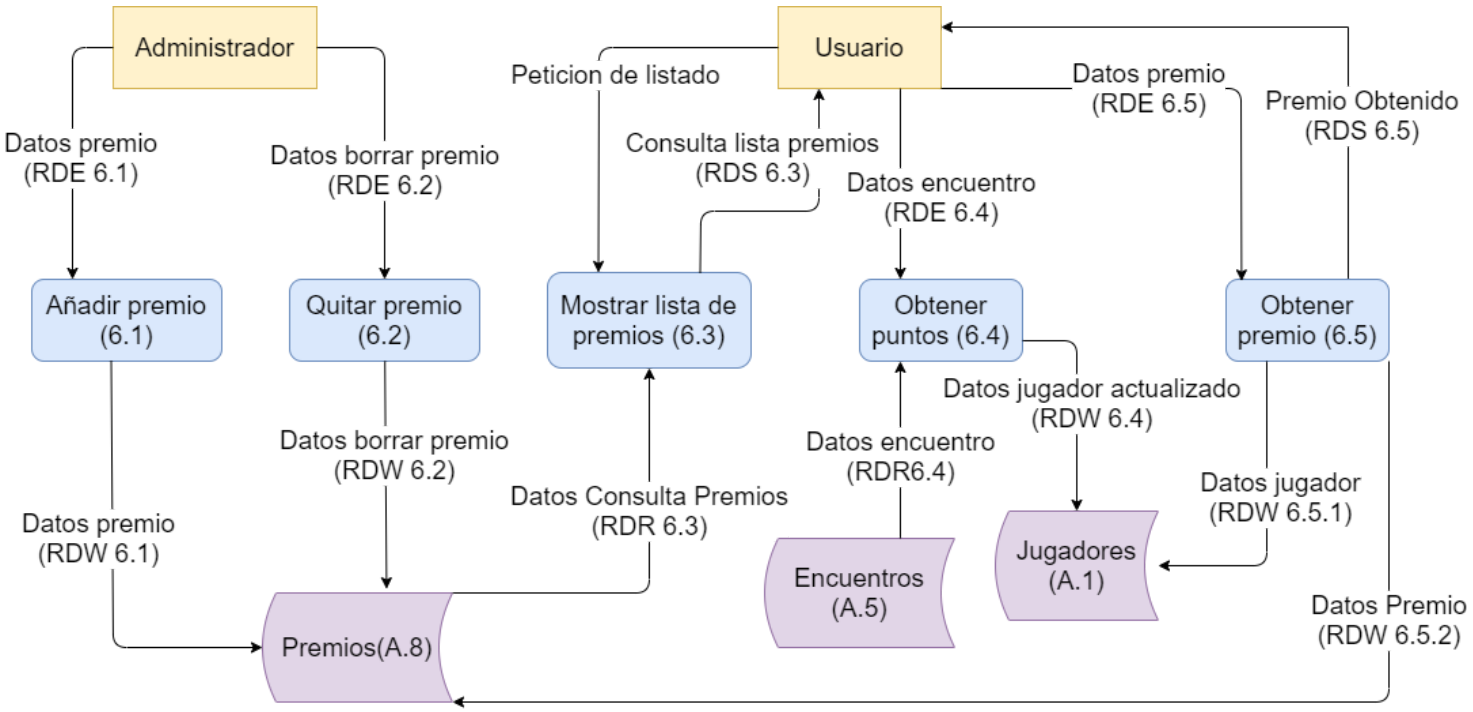
RS6.3: El código de encuentro debe existir
RF: RF6.4
RD(s): RDE6.4
Descripción: “En caso de que el código de encuentro introducido a la hora de obtener puntos, no coincida con los códigos almacenados en la BD, se informa del error y no se realiza nada.

RS6.4: El encuentro debe estar finalizado para obtener los puntos
RF: RF6.4
RD(s): RDE6.4
Descripción: “En caso de que el encuentro el cual se sabe su información por el código de encuentro introducido, no se encuentra finalizado, es decir, que no han pasado 2 horas desde la hora de comienzo que tiene guardado el encuentro, se informa y no se realiza nada.

RS6.5: El premio no puede valer más puntos de los que tiene el jugador
RF: RF6.5

RD(s): RDE6.5

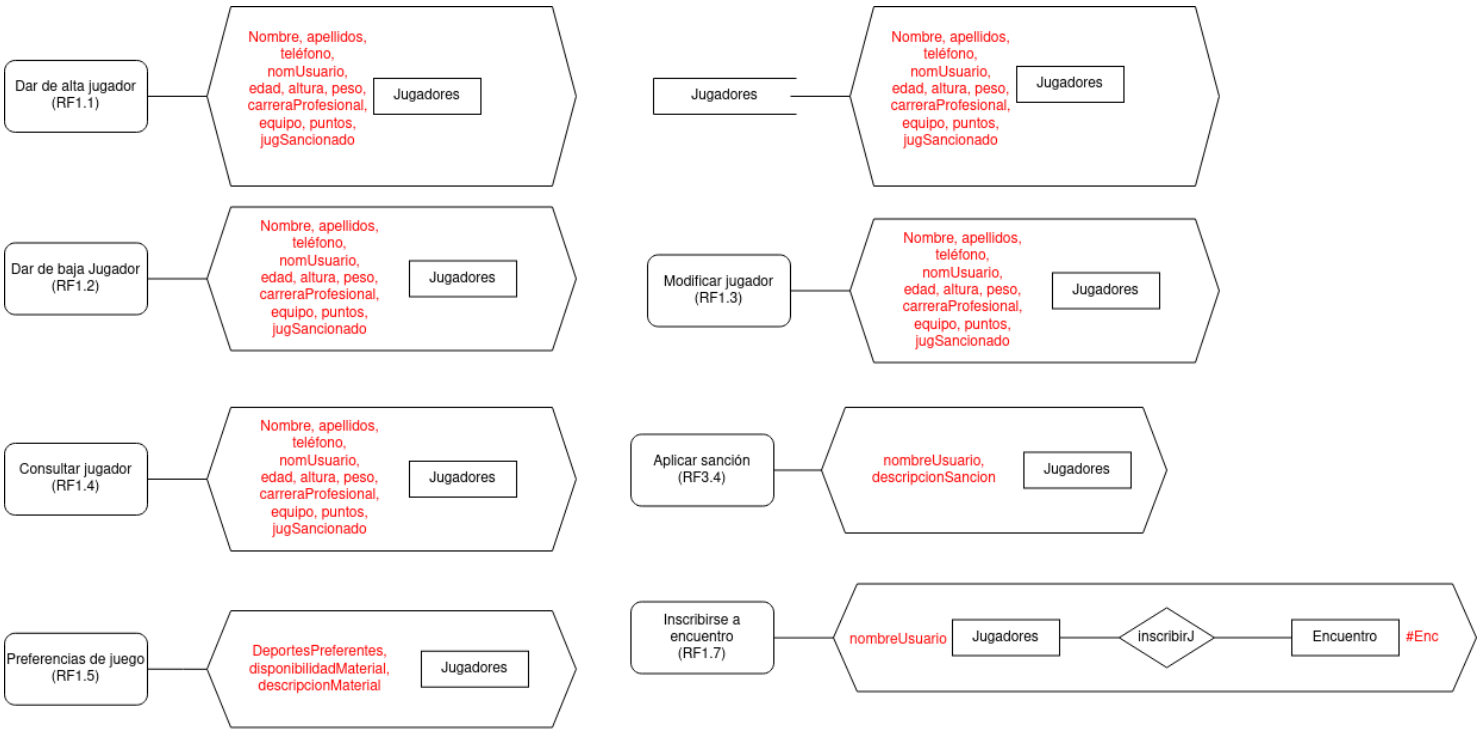
Descripción: “Si el premio que desea el jugador canjear tiene un valor más alto que la cantidad de puntos que tiene , no se realiza el canje de puntos y se devuelve un error”

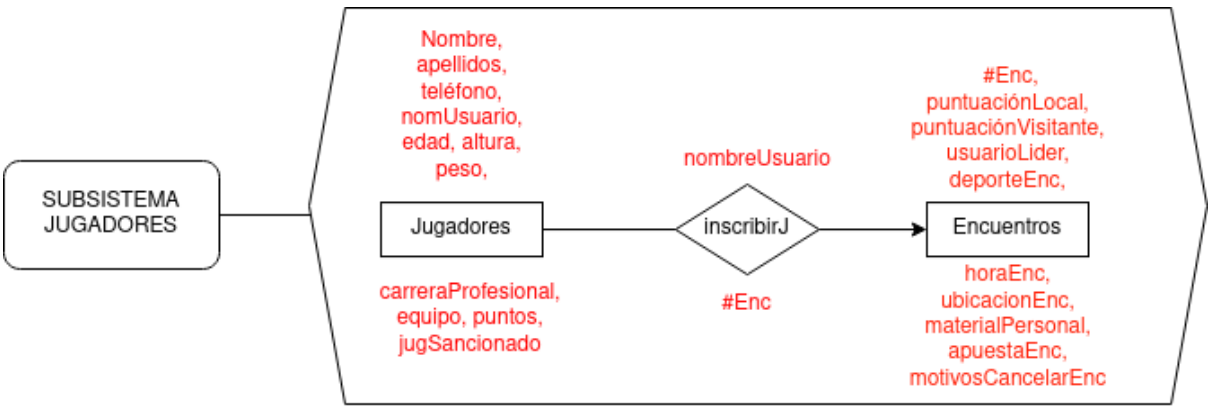


Para la mejor visualización de los DFD, visite el link https://drive.google.com/file/d/16YSs-jQJh6o3-bXKUx0VaPTifHYtF4oq/view?usp=sharing o la carpeta MATARRATAS.

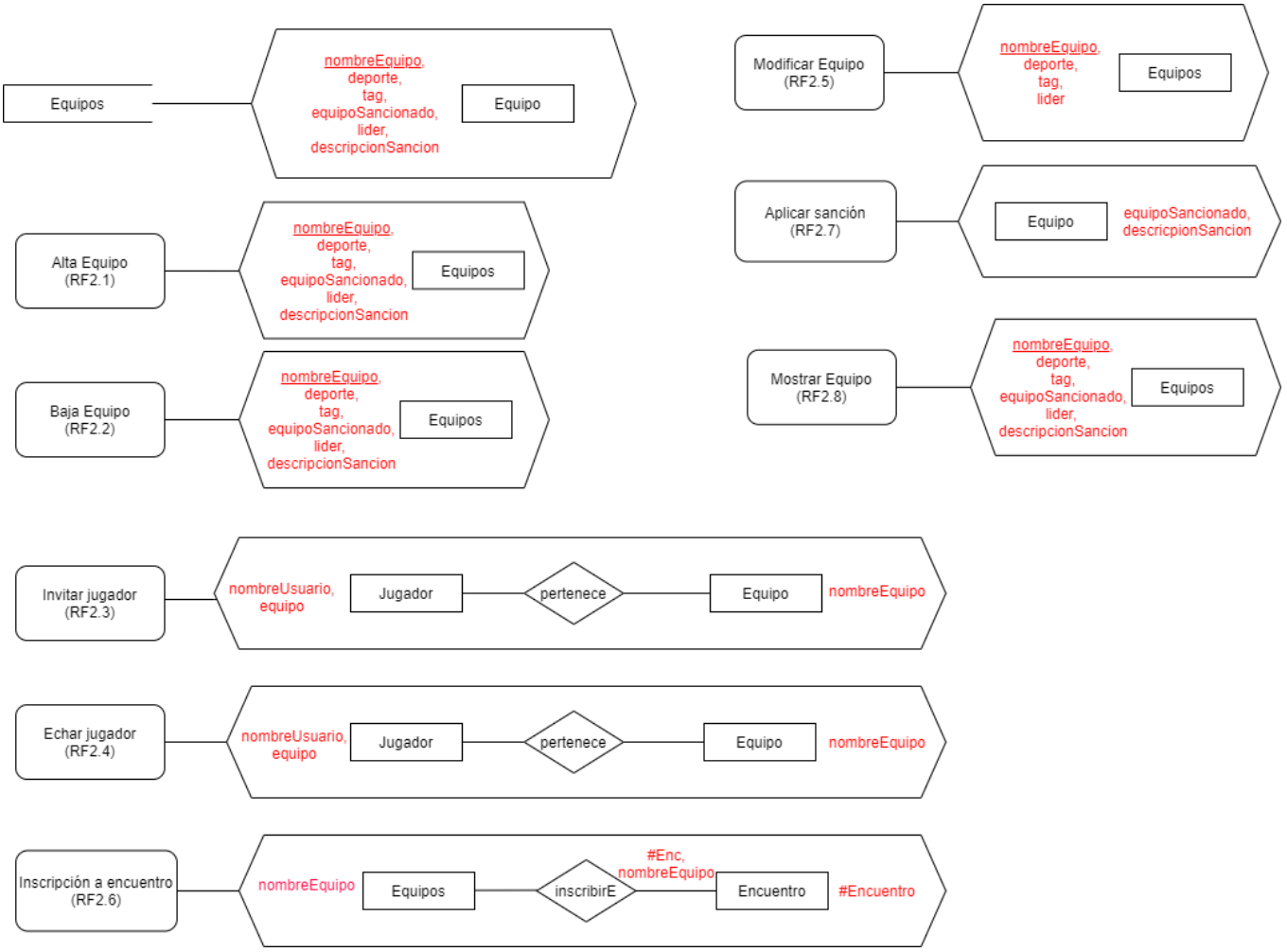
DISEÑO DE DATOS DE LOS SUBSISTEMAS

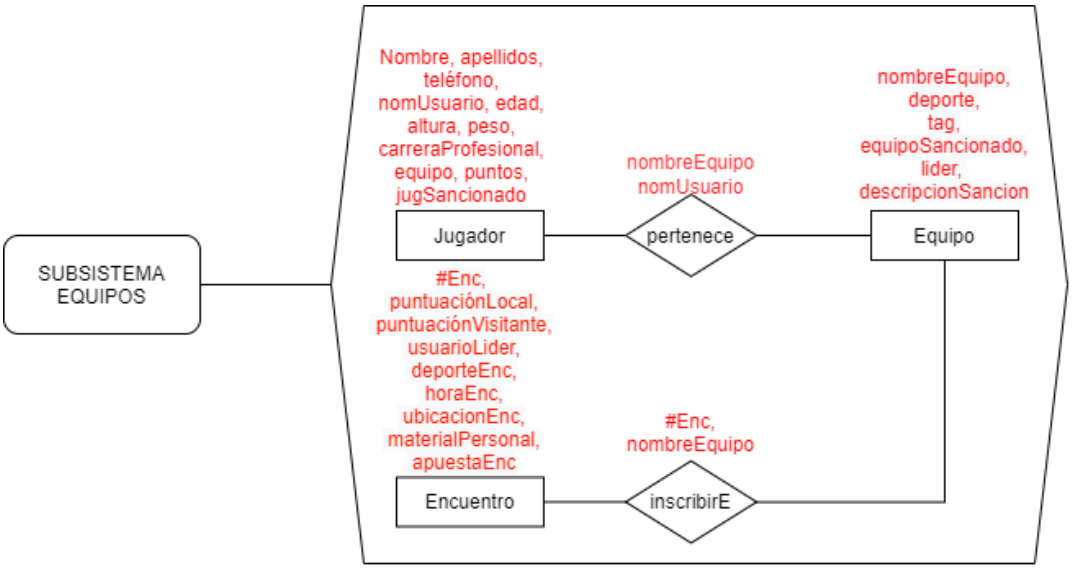
1- GESTIÓN DE JUGADORES: *(RAÚL R)*



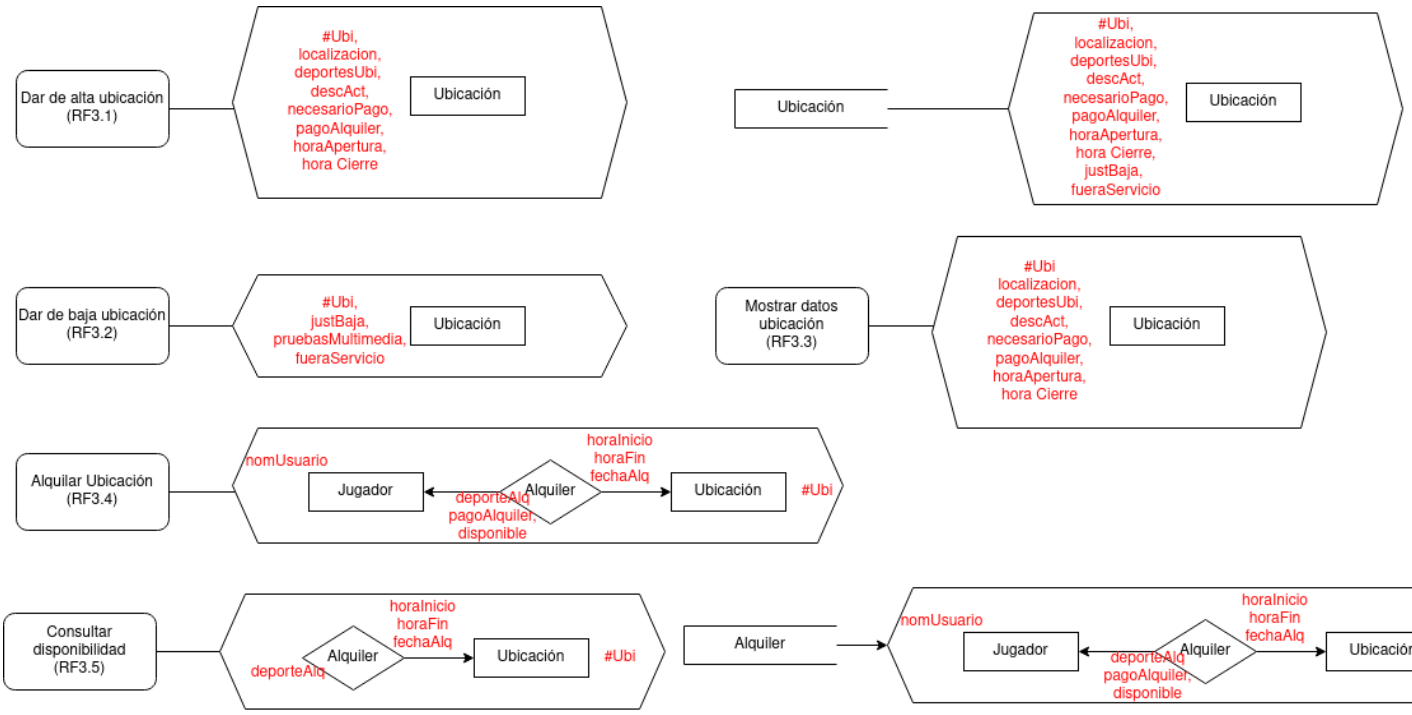


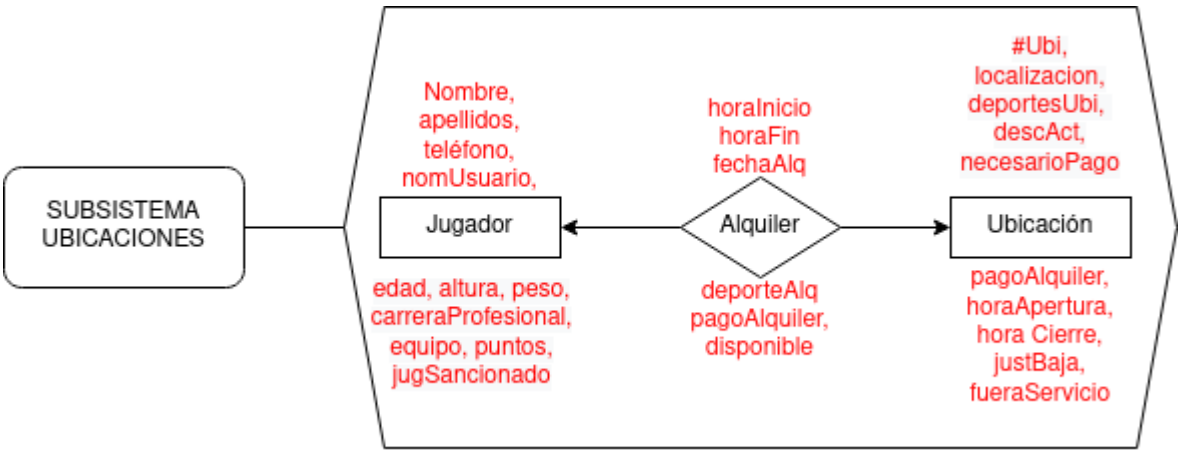
2- **GESTIÓN DE EQUIPOS:** *(ÁNGEL G)*





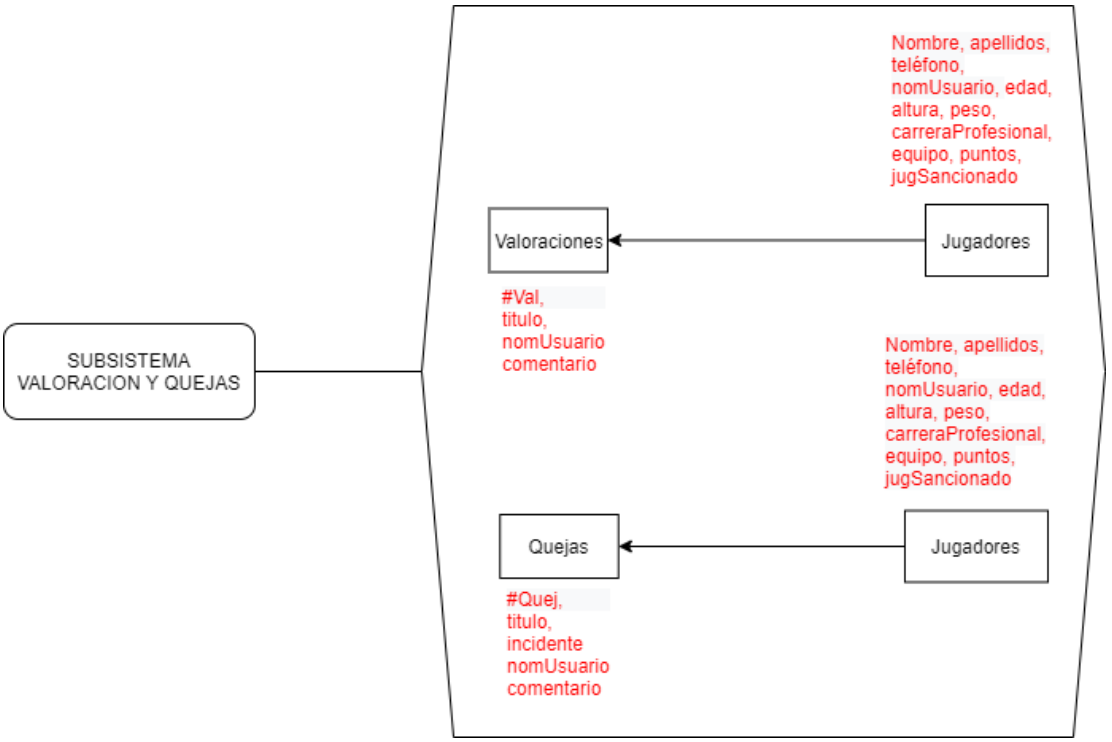
3- **GESTIÓN DE UBICACIONES:** *(CRISTIAN F)*



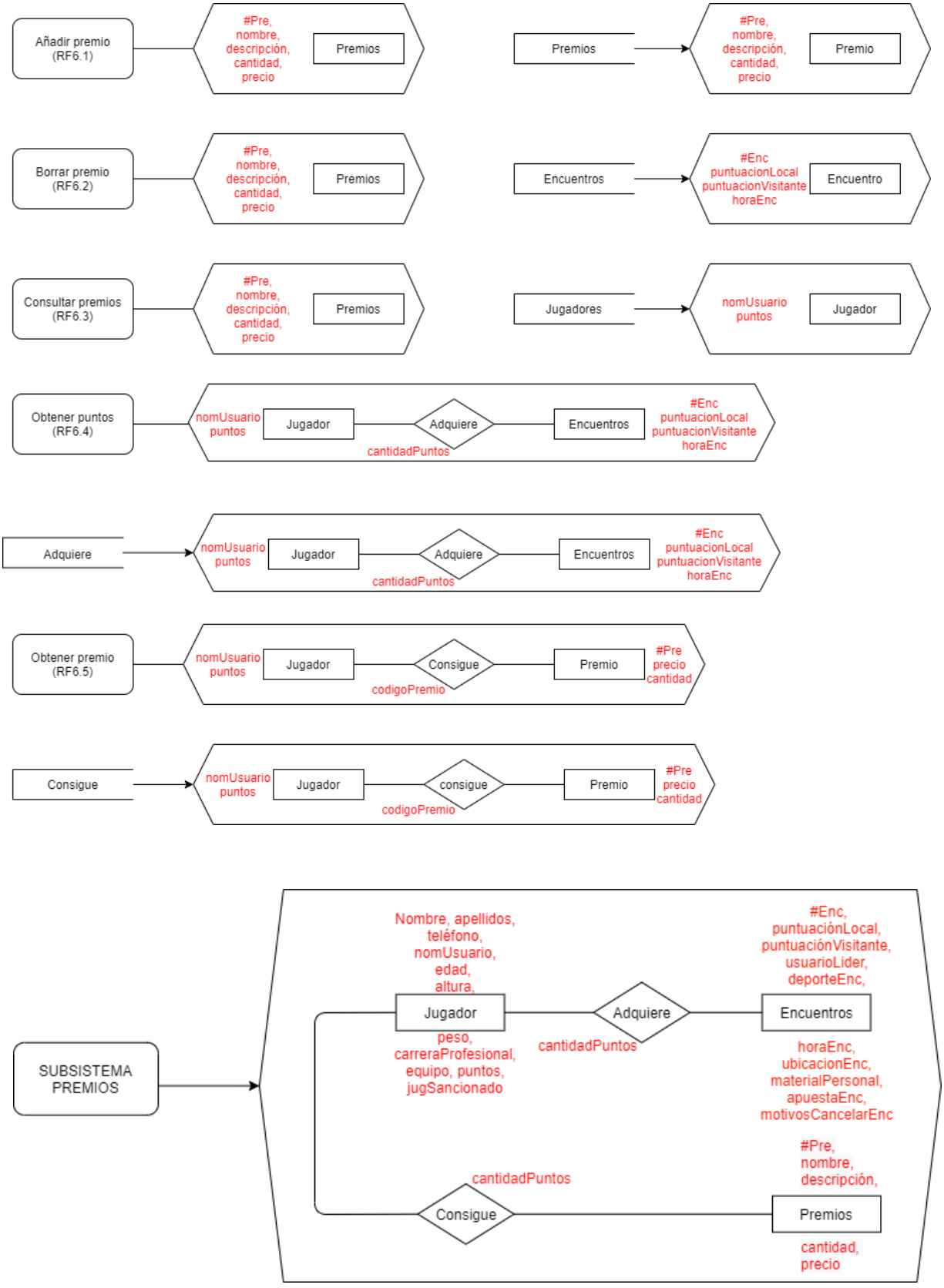


4- GESTIÓN DE ENCUENTROS: *(SANTI M)*

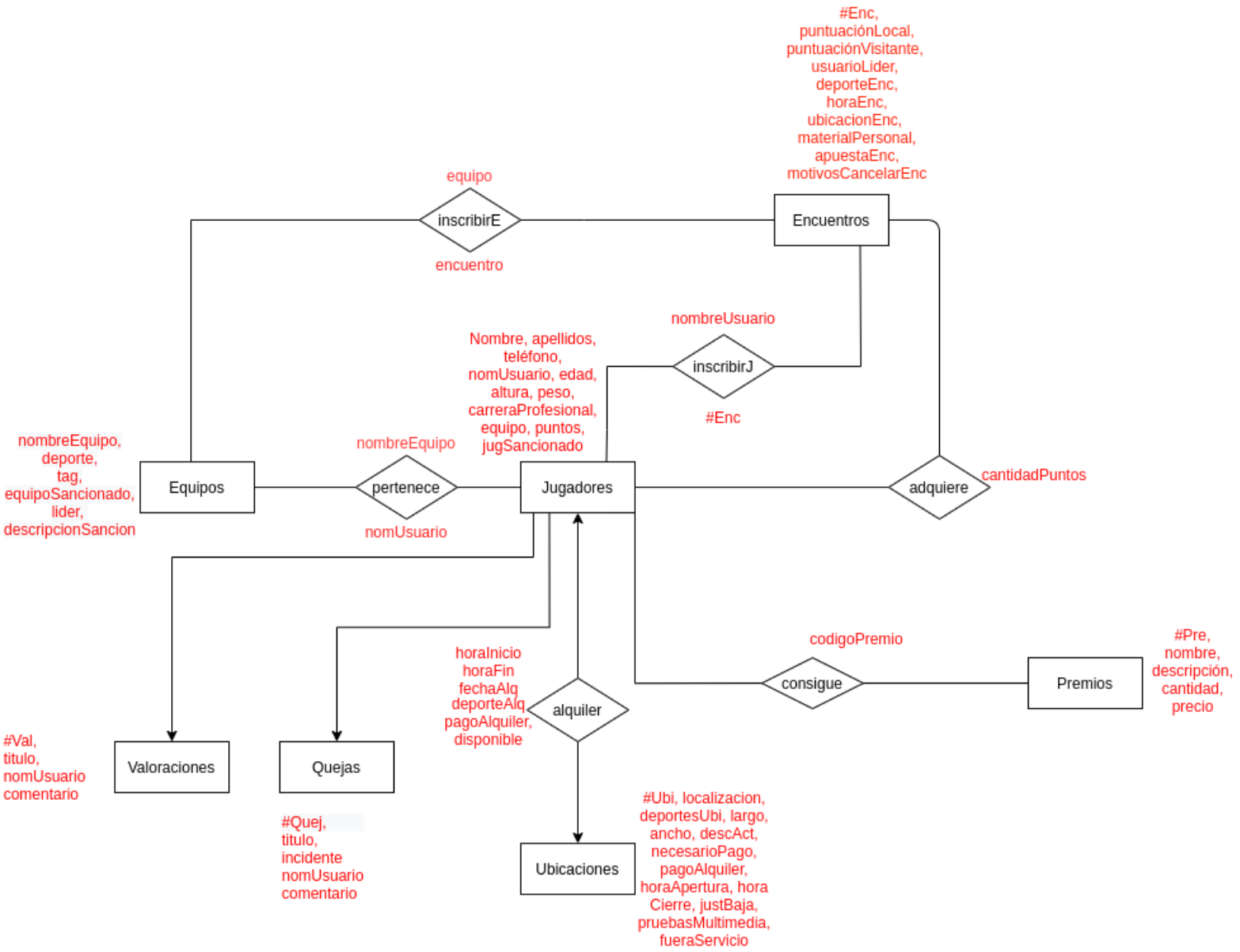
5- GESTIÓN DE VALORACIONES: *(FRAN J)*



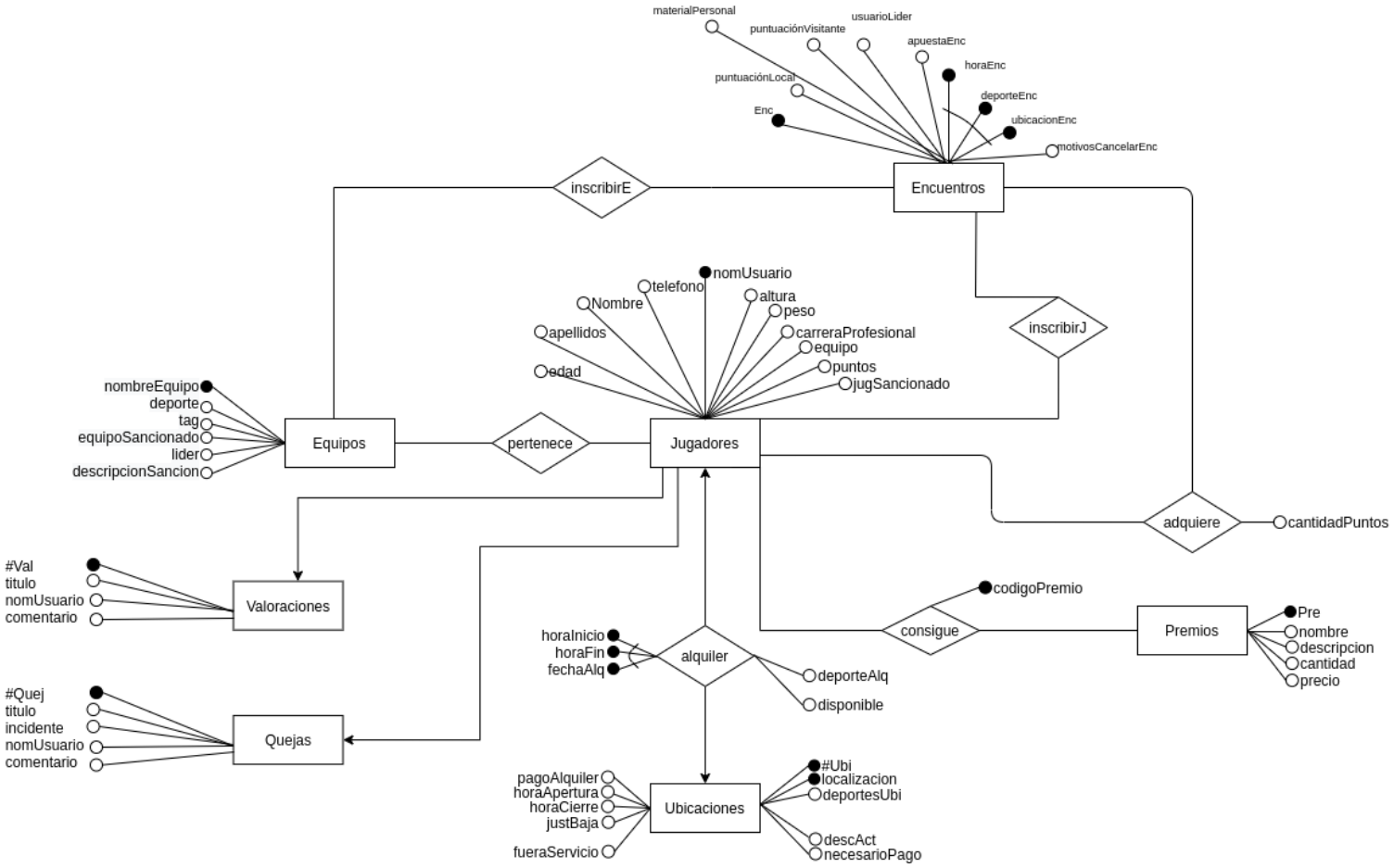
6- **GESTIÓN DE PREMIOS:** *(RAÚL C)*



ESQUEMA EXTERNO ARMAZÓN (DFD0)



ESQUEMA ENTIDAD RELACIÓN DEL SISTEMA



PASO A TABLAS (NORMALIZADAS)

1- Jugadores: Nombre, apellidos, teléfono, nomUsuario, edad, altura, peso, carreraProfesional, equipo, puntos, jugSancionado, deportesPreferentes, disponibilidadMaterial, descripcionMaterial, descripcionSancion

2- Equipos: nombreEquipo, deporte, tag, equipoSancionado, lider, puntos, descripcionSancion

3- Ubicacion: #Ubi(CP), localizacion(CC), deportesUbi, descAct, necesarioPago, pagoAlquiler, horaApertura, hora Cierre, justBaja, fueraServicio

4- Encuentros: #Enc(CP), ubicacionEncuentro (CE: Ubicacion), horaEnc, deporteEnc (CC), puntuaciónLocal, puntuaciónVisitante, usuarioLider, materialPersonal, motivosCancelarEnc, apuestaEnc

5- Premios: #Pre, nombre, descripcion, cantidad, precio

6- Valoraciones: #Val, nomUsuario, titulo, comentario

7- Quejas: #Quej, nomUsuario, titulo, incidente, comentario

8- Pertenece: equipo (CE: nomEquipo), jugador (CE: nomUsuario)

9- inscripcionE: #Enc (CE: #encuentro), equipo(CE: nomEquipo)

10- InscripcionJ: nomUsuario (CE: nomUsuario), #Enc (CE: Encuentro)

11- Alquiler: Usuario (CE: nomUsuario), fechaAlq, horaInicio, horaFin, #Ubi (CE: Ubicacion), deporteAlq

12- Consigue: nomUsuario (CE: nomUsuario), codigoPremio, #Pre (CE: #Pre) .

13- Adquiere: nomUsuario (CE: nomUsuario), #Enc (CE: Encuentro), cantidadPuntos.

NORMALIZACIÓN DE TABLAS

1- Jugadores: Nombre, apellidos, teléfono, nomUsuario, edad, altura, peso, carreraProfesional, equipo, puntos, jugSancionado, deportesPreferentes, disponibilidadMaterial, descripcionMaterial, descripcionSancion

Al analizar la tabla de jugadores y buscar la existencia tanto de dependencias parciales, como transitivas, como de tipo Atributo no primo → Atributo primo, nos damos cuenta de que no existen tales dependencias y por lo tanto la tabla se encuentra ya normalizada. Esto es lo mismo que afirmar que dicha tabla se encuentra ya en la forma normal de Boyce-Codd.

2- Equipos: nombreEquipo, deporte, tag, equipoSancionado, lider, puntos, descripcionSancion

2NF→ No existen dependencias parciales, ya que la clave primaria es un solo atributo.

3NF→ Existiría una dependencia transitiva en el caso de que líder fuera asociado a un único equipo (se arreglaría marcándola como CC); sin embargo un jugador puede ser líder de varios equipos. Por lo tanto no existen más posibles dependencias transitivas por lo que se cumple la 3NF.

BCNF→ No existen dependencias del tipo: Atributo no primo → Atributo primo. Por lo que se cumple BCNF.

3- Encuentros:#Enc, puntuaciónLocal, puntuaciónVisitante, usuarioLider, deporteEnc, horaEnc, ubicacionEncuentro, materialPersonal, apuestaEnc

2FN->No existen dependen parciales, ya que la clave primaria es solo un atributo.

3FN->No se cumple. horaEnc, deporteEnc y ubicacionEncuentro tienen dependencias transitivas, pues:

horaEnc, deporteEnc,ubicacionEncuentro----------> cualquier atributo

Para solucionar esto hacemos a estos atributos una claves candidata.

3-Encuentros: #Enc(CP), horaEnc,deporteEnc, ubicacionEncuentro(CE: Ubicacion) (CC), puntuaciónLocal, puntuaciónVisitante, usuarioLider, apuestaEnc, materialPersonal.

FNBC -> No hay ningún atributo primo que dependa de un no primo, por lo tanto se cumple FNBC.

4- Ubicacion: #Ubi, localizacion, deportesUbi, descAct, necesarioPago, pagoAlquiler, horaApertura, hora Cierre, justBaja, fueraServicio

2FN -> No existen dependencias parciales, al ser la clave primaria un solo atributo.

3FN -> No se cumple. Localización tiene dependencias transitivas, pues:

localizacion ----------> cualquier atributo no primo

Solucionaremos esto haciéndola clave candidata.

4- Ubicacion: #Ubi(CP), localizacion(CC), deportesUbi, descAct, necesarioPago, pagoAlquiler, horaApertura, hora Cierre, justBaja, fueraServicio

FNBC -> No hay ningún atributo primo que dependa de un no primo. Se satisface.

5-Premios - #Pre, nombre ,descripción , cantidad , precio

2FN -> No existen dependencias parciales, al ser la clave primaria un solo atributo.

3FN -> No existen dependencias transitivas, pues el único identificador válido es el código del premio.

FNBC -> No hay ningún atributo primo que dependa de uno no primo. Se satisface.

6-Valoraciones: : #Val, nomUsuario, titulo, comentario

2FN -> No existen dependencias parciales, al ser la clave primaria un solo atributo.

3FN -> Se cumple. No existen dependencias transitivas ya que el único identificador válido es el código de valoración.

FNBC -> No hay ningún atributo primo que dependa de un no primo. Se satisface.

7-Quejas: #Quej, nomUsuario, titulo, incidente, comentario

2FN -> No existen dependencias parciales, al ser la clave primaria un solo atributo.

3FN -> Se cumple. No existen dependencias transitivas ya que el único identificador válido es el código de queja.

FNBC -> No hay ningún atributo primo que dependa de un no primo. Se satisface.

8- Pertenece: equipo (CE: nomEquipo), jugador (CE: nomUsuario)

2FN→ No existen dependencias parciales, ya que no hay atributos no primarios. Se cumple la 2FN.

3FN→ No existen dependencias transitivas, ya que no hay atributos no primos. Se cumple la 3FN.

FNBC→ No existen dependencias del tipo: Atributo no primo → Atributo primo ya que no hay atributos no primos. Por lo tanto se cumple FNBC.

9- inscripcionE: #Enc (CE: #encuentro), equipo(CE: nomEquipo)

2FN→ No existen dependencias parciales, ya que no hay atributos no primarios. Se cumple la 2FN.

3FN→ No existen dependencias transitivas, ya que no hay atributos no primos. Se cumple la 3FN.

FNBC→ No existen dependencias del tipo: Atributo no primo → Atributo primo ya que no hay atributos no primos. Por lo tanto se cumple FNBC.

10- InscripcionJ: nomUsuario (CE: nomUsuario), #Enc (CE: Encuentro)

Al analizar la tabla de InscripcionJ nos damos cuenta que ocurre lo mismo que con la tabla de Jugadores. Es decir, esta tabla no presenta dependencias ya que se encuentra normalizada o, como es lo mismo, en la forma normal de Boyce-Codd.

11- Alquiler: Usuario (CE: nomUsuario), fechaAlq, horaInicio, horaFin, deporteAlq, #Ubi (CE: Ubicacion)

Detectamos que deporteAlq no ha de formar parte de la clave, ya que en una sola fecha, solo podrá haber un deporte, que no requerirá conocerlo para identificar el alquiler.

11- Alquiler: Usuario (CE: nomUsuario), fechaAlq, horaInicio, horaFin, #Ubi (CE: Ubicacion), deporteAlq

2FN -> No hay dependencias parciales, ya que para conocer el deporteAlq necesitamos toda la clave candidata.

3FN -> No existen dependencias transitivas, ya que solo hay un atributo no primo.

FNBC -> No hay ningún atributo primo que dependa de uno no primo. Se satisface.

12-Consigue - nomUsuario, codigoPremio, #Pre

2FN -> No existen dependencias parciales, pues no hay atributos no primos que dependan de parte de la clave.

3FN -> No existen dependencias transitivas, pues no hay atributos no primos.

FNBC -> No hay ningún atributo primo que dependa de uno no primo. Se satisface.

13-Adquiere -nomUsuario, #Enc, cantidadPuntos

2FN -> No existen dependencias parciales, pues no podemos obtener el atributo con parte de la clave.

3FN -> No existen dependencias transitivas, pues solo hay un atributo no primo.

FNBC -> No hay ningún atributo primo que dependa de uno no primo. Se satisface.

CREACIÓN DE TABLAS SQL

```
CREATE TABLE jugadores(
   nombre VARCHAR2(20) NOT NULL,
   apellidos VARCHAR2(40) NOT NULL,
   teléfono VARCHAR2(20) NOT NULL,
   nombreUsuario VARCHAR(15) NOT NULL PRIMARY KEY,
   edad INT NOT NULL,
   altura FLOAT NOT NULL,
   peso FLOAT NOT NULL,
   carreraProfesional VARCHAR(30) NOT NULL,
   equipo VARCHAR2(20) DEFAULT NULL,
   puntos INT DEFAULT 0,
   jugSancionado NUMBER(1) DEFAULT 0,
   deportesPreferentes VARCHAR2(100),
   disponibilidadMaterial NUMBER(1) NOT NULL,
   descripcionMaterial VARCHAR2(500),
   descripcionSancion VARCHAR2(500)
);

CREATE TABLE equipos(
   nombreEquipo VARCHAR2(20) NOT NULL PRIMARY KEY,
   deporte VARCHAR2(100),
   tag VARCHAR2(3) NOT NULL,
   equipoSancionado NUMBER(1) DEFAULT 0,
   lider VARCHAR2(20) NOT NULL,
   descripcionSancion VARCHAR2(500)
```

```
CREATE TABLE quejas(
   codQuej VARCHAR2(5) NOT NULL PRIMARY KEY,
   titulo VARCHAR2(50) NOT NULL ,
   incidente VARCHAR2(100) NOT NULL,
   nomUsuario VARCHAR2(50) NOT NULL,
   comentario VARCHAR2(500) NOT NULL
);

CREATE TABLE pertenece(
   equipo VARCHAR2(20) NOT NULL,
   jugador VARCHAR(15) NOT NULL,
   PRIMARY KEY(equipo, jugador),
   FOREIGN KEY (jugador) REFERENCES
jugadores(nombreUsuario),
   FOREIGN KEY (equipo) REFERENCES
equipos(nombreEquipo)
);


CREATE TABLE inscripcionE(
   codEnc VARCHAR2 (5) NOT NULL,
   equipo NOT NULL,
   PRIMARY KEY(equipo, codEnc),
   FOREIGN KEY (codEnc) REFERENCES
encuentros(codEnc),
```

```
);


CREATE TABLE encuentros(
      codEnc VARCHAR2 (5) NOT NULL PRIMARY KEY,
           puntuacionLocal INT DEFAULT 0 CHECK
(puntuacionLocal>=0),
           puntuacionVisitante INT DEFAULT 0 CHECK
(puntuacionVisitante>=0),
      usuarioLider VARCHAR(15) NOT NULL,
      deporteEnc VARCHAR2(20) NOT NULL CHECK
(deporteEnc='Fútbol' OR deporteEnc='Baloncesto' OR
deporteEnc='Pádel'),
      horaEnc DATE NOT NULL,
      ubicacionEncuentro VARCHAR2(100)  NOT NULL,
      material_Personal VARCHAR2(100) NOT NULL,
     apuestaEnc INT DEFAULT 0 CHECK (apuestaEnc>=0),
     FOREIGN KEY (usuarioLider) REFERENCES
jugadores(nombreUsuario),
    FOREIGN KEY (ubicacionEncuentro) REFERENCES
ubicacion(localizacion),
UNIQUE (ubicacionEncuentro,horaEnc,deporteEnc)
);

CREATE TABLE ubicacion(
   codUbi VARCHAR2(5) NOT NULL PRIMARY KEY,
   localizacion VARCHAR2(100) UNIQUE NOT NULL,
   deportesUbi VARCHAR2(20) NOT NULL,
   descAct VARCHAR2(500),
   necesarioPago NUMBER(1) DEFAULT 0 CHECK
(necesarioPago=0 OR necesarioPago=1) NOT NULL,
   pagoAlquiler FLOAT DEFAULT 0 NOT NULL
CHECK(pagoAlquiler>=0),
   horaApertura DATE NOT NULL,
   horaCierre DATE NOT NULL,
   justBaja VARCHAR2(500),
   fueraServicio INT  DEFAULT 0 CHECK (fueraServicio=0 OR
fueraServicio=1)  NOT NULL);
CREATE TABLE premios(
   codPre VARCHAR2(8) NOT NULL PRIMARY KEY,
   nombre VARCHAR2(50) NOT NULL,
   descripcion VARCHAR2(300) NOT NULL,
   cantidad INT NOT NULL,
   precio INT NOT NULL
);

CREATE TABLE valoraciones(
   codVal VARCHAR2(5) NOT NULL PRIMARY KEY,
   titulo VARCHAR2(50) NOT NULL,
   nomUsuario VARCHAR2(50) NOT NULL,
   comentario VARCHAR2(500) NOT NULL

);
```

```
   FOREIGN KEY (equipo) REFERENCES
equipos(nombreEquipo)
);
CREATE TABLE inscripciónJ(
   nombreUsuario VARCHAR2(15) NOT NULL,
   codEnc VARCHAR2 (5) NOT NULL,
   PRIMARY KEY(nombreUsuario, codEnc),
   FOREIGN KEY (codEnc) REFERENCES
encuentros(codEnc),
   FOREIGN KEY (nombreUsuario) REFERENCES
jugadores(nombreUsuario)
);

CREATE TABLE alquiler(
   codUbi VARCHAR2(5) NOT NULL,
   Usuario VARCHAR(15) NOT NULL,
   fechaAlq DATE NOT NULL,
   horaInicio DATE NOT NULL,
   horaFin DATE NOT NULL,
   deporteAlq VARCHAR2(20) NOT NULL,
   PRIMARY KEY(Usuario, fechaAlq, horaInicio, horaFin),
   FOREIGN KEY (Usuario) REFERENCES
jugadores(nombreUsuario),
   FOREIGN KEY (codUbi) REFERENCES ubicacion(codUbi)
);


CREATE TABLE consigue(
   nomUsuario VARCHAR2(15) NOT NULL REFERENCES
jugadores(nombreUsuario),
   codigoPremio VARCHAR2(16) NOT NULL,
   codPre VARCHAR(8) NOT NULL REFERENCES
premios(codPre),
   PRIMARY KEY(nomUsuario,codigoPremio,codPre)
);

CREATE TABLE adquiere (
      nomUsuario VARCHAR2(15) NOT NULL REFERENCES
jugadores(nombreUsuario),
      codEnc VARCHAR2(5) NOT NULL REFERENCES
encuentros(codEnc),
      cantidadPuntos INT NOT NULL,
      PRIMARY KEY( nomUsuario, codEnc)
);
```

INSERCIÓN DE TUPLAS EJEMPLO

Para la comprobación del funcionamiento de estas sentencias, insertamos tres tuplas de prueba en cada una de las tablas:

```
insert into Jugadores (nombre, apellidos, teléfono, nombreUsuario, edad, altura, peso, carreraProfesional, equipo, deportesPreferentes, disponibilidadMaterial, descripcionMaterial) values('Pepe', 'Castro Moreno', 619619619, 'Pepe88', 20, 180,  90, 'No tiene', 'No tiene', 'Fútbol', 1, 'Balón de futbol' );
insert into Jugadores (nombre, apellidos, teléfono, nombreUsuario, edad, altura, peso, carreraProfesional, equipo, deportesPreferentes, disponibilidadMaterial, descripcionMaterial) values('Willy', 'Fernandez Jimenez', 607607607, 'Willy777', 18, 190,  80, 'No tiene', 'No tiene',  'Baloncesto', 1, 'Balón de baloncesto' );
insert into Jugadores (nombre, apellidos, teléfono, nombreUsuario, edad, altura, peso, carreraProfesional, equipo, deportesPreferentes, disponibilidadMaterial) values('Julian', 'Muñoz Castro', 629629629, 'Julian1234', 35, 150,  60, 'Jugador del Numancia en ACB', 'No tiene actualmente',  'Baloncesto', 0);

insert into equipos(nombreEquipo,tag , deporte, lider) values ('betis', 'bts', 'Baloncesto', 'Julian1234');
insert into equipos(nombreEquipo,tag , deporte, lider) values ('alaves', 'alv', 'Fútbol', 'Pepe88');
insert into equipos(nombreEquipo,tag , deporte, lider) values ('numanciafc', 'nmc', 'Baloncesto', 'Willy777');

insert into ubicacion (codUbi,localizacion,deportesUbi,necesarioPago,pagoAlquiler,horaApertura,horaCierre) values ('10001','Estadio Los Carmenes,SN,GRANADA','Fútbol','1','5',sysdate,sysdate);
insert into ubicacion (codUbi,localizacion,deportesUbi,horaApertura,horaCierre) values ('10002','Poligono industrial Motril','Baloncesto',sysdate,sysdate);
insert into ubicacion (codUbi,localizacion,deportesUbi,necesarioPago,pagoAlquiler,horaApertura,horaCierre) values ('10003','Polideportivo Maracena, SN, Maracena','Baloncesto','1','7',sysdate,sysdate);
```

```
insert into
encuentros(codEnc,puntuacionLocal,puntuacionVisitante,usuarioLider,deporteEnc,horaEnc,
ubicacionEncuentro,material_Personal,apuestaEnc,motivosCancelarEnc)
values('12345',2,0,'Pepe88','Fútbol',sysdate,'Estadio Los Carmenes,SN,GRANADA','Un
balón de futbol, botellas de agua, un fisio y un arbitro',2000,'Se han puesto todos malos');
insert into
encuentros(codEnc,puntuacionLocal,puntuacionVisitante,usuarioLider,deporteEnc,horaEnc,
ubicacionEncuentro,material_Personal,apuestaEnc,motivosCancelarEnc)
values('23456',100,80,'Willy777','Baloncesto',sysdate,'Poligono industrial Motril','Un balón de
baloncesto y redes de canasta',1000,'Ninguno');
insert into
encuentros(codEnc,usuarioLider,deporteEnc,horaEnc,ubicacionEncuentro,material_Persona
l,apuestaEnc,motivosCancelarEnc)
values('34567','Julian1234','Baloncesto',sysdate,'Polideportivo Maracena, SN,
Maracena','Un balón de futbol ',1000,'Ninguno');

insert into premios(codPre,nombre,descripcion,cantidad,precio) values(10101010,'PS5','La
ultima consola de Sony, con un SSD super rapido y un mando con tecnologia
DualSense',3,2000000);
insert into premios(codPre,nombre,descripcion,cantidad,precio) values(10101011,'Camiseta
Futbol PSG','Camiseta del famoso equipo frances con el numero 7 a la espalda y el nombre
de MBAPPE',5,250000);
insert into premios(codPre,nombre,descripcion,cantidad,precio) values(10101012,'NVIDIA
GeForce RTX 3070','Tarjeta grafica para que tu ordenador vuele en terminos de velocidad
de procesamiento de graficos',2,1500000);

insert into valoraciones (codVal,titulo,nomUsuario,comentario) values('12345','Mejor
jugador,mejor persona', 'Pepe88', 'Muy amable...');
insert into valoraciones (codVal,titulo,nomUsuario,comentario) values('12347','Me alegro de
jugar con el', 'Willy777', 'Un placer de persona');
insert into valoraciones (codVal,titulo,nomUsuario,comentario) values('12346','muy amable',
'Julian1234', 'Muy respetuoso');

insert into quejas (codQuej,titulo,incidente,nomUsuario,comentario) values('12ABC','Falto al
respeto','LESION', 'Pepe88', 'Todo el partido insultando');
```

```
insert into quejas (codQuej,titulo,incidente,nomUsuario,comentario) values('12AB5','Jugo muy sucio', 'LESION','Julian1234', 'Casi me lesiona');
insert into quejas (codQuej,titulo,incidente,nomUsuario,comentario) values('12ABF','Falto al respeto','LESION', 'Willy777', 'Todo el partido insultando');
insert into pertenece(equipo, jugador) values( 'alaves', 'Pepe88');
insert into pertenece(equipo, jugador) values( 'numanciafc', 'Willy777');
insert into pertenece(equipo, jugador) values( 'betis', 'Julian1234');
insert into inscripcionE(codEnc, equipo) values('12345', 'betis');
insert into inscripcionE(codEnc, equipo) values('23456', 'numanciafc');
insert into inscripcionE(codEnc, equipo) values('34567', 'alaves');
insert into inscripciónJ(nombreUsuario, codEnc) values('Pepe88', '12345');
insert into inscripciónJ(nombreUsuario, codEnc) values('Julian1234', '23456');
 insert into inscripciónJ(nombreUsuario, codEnc) values('Willy777', '34567');
insert into alquiler(codUbi,Usuario,fechaAlq,horaInicio,horaFin,deporteAlq) values ('10002','Willy777',sysdate,sysdate,sysdate,'Baloncesto');
insert into alquiler(codUbi,Usuario,fechaAlq,horaInicio,horaFin,deporteAlq) values ('10001','Pepe88',sysdate,sysdate,sysdate,'Fútbol');
insert into alquiler(codUbi,Usuario,fechaAlq,horaInicio,horaFin,deporteAlq) values ('10003','Julian1234',sysdate,sysdate,sysdate,'Baloncesto');
insert into consigue(nomUsuario,codigoPremio,codPre)
values('Pepe88','ABCD1234','10101011');
insert into consigue(nomUsuario,codigoPremio,codPre)
values('Julian1234','ABCD1235','10101012');
insert into consigue(nomUsuario,codigoPremio,codPre)
values('Willy777','ABCD1236','10101010');

insert into adquiere(nomUsuario,codEnc,cantidadPuntos)
values('Willy777','34567',2000000);
insert into adquiere(nomUsuario,codEnc,cantidadPuntos)
values('Julian1234','23456',2000000);
insert into adquiere(nomUsuario,codEnc,cantidadPuntos) values('Pepe88','12345',2000000);
```

Para la mejor visualización de los Esquemas, visite el link https://drive.google.com/file/d/1yGcxisui3KQWAkJUV8on0AsXTxaOOdIT/view?usp=sharing o la carpeta MATARRATAS.

DESARROLLO DEL SUBSISTEMA

Para el desarrollo de las transacciones hemos reutilizado lo aprendido en el Seminario 2, creando estas en Java, realizando la conexión con la base de datos Oracle de la escuela.

La interfaz sigue siendo textual, pues sería más trabajoso realizar una adecuada.

Subsistema Jugadores

Para el subsistema de jugadores, hemos realizado un cambio en los requisitos funcionales. Dicho cambio se basa en la eliminación del **RF 1.5 (preferencias de juego)**, el cual, consistía en leer y escribir las preferencias de juego de los usuarios. Una vez comencé a escribir el código, me di cuenta de que dicho requisito funcional estaba dentro del **RF 1.3 (modificar datos del jugador)**. Esto es así debido a que, tras iniciar los campos correspondientes a las preferencias de juego con el **RF 1.1 (alta jugador)**, se le permite al usuario (haciendo uso del **RF 1.3**) modificar dichos campos en cualquier momento. Por lo tanto, la mera existencia de un requisito para la escritura de dichos campos era innecesaria.

Tras esto, hemos implementado las transacciones oportunas para el correcto funcionamiento del subsistema de jugadores. Más concretamente he creado las transacciones de **Alta jugador (RF 1.1)**, **Baja jugador (RF 1.2)**, **Modificar datos del jugador (RF 1.3)**, **Mostrar datos del jugador (RF 1.4)**, **Aplicar sanción (RF 1.5)** y **Inscribirse al encuentro (RF 1.6)**. Algunas de estas funciones modifican o alteran la tabla ‘jugadores’ e ‘inscripciónJ’ de nuestra base de datos. Por ello, en las transacciones correspondientes he implementado que, al final de la función, pregunte al usuario si quiere confirmar los cambios o no. Si dicho usuario responde “Si”, se hace commit para avanzar, pero si responde que “No”, hacemos un rollback al savepoint establecido siempre al principio de cada función.

- **RF 1.1 ‘Alta jugador’**

Esta funcionalidad pedirá todos los atributos necesarios para dar de alta a un jugador, tendiendo en cuenta también, que se volverá a pedir que introduzca un atributo en concreto, si lo introducimos incorrectamente. Destacar que algunos de estos atributos se inicializan por defecto, sin preguntar al usuario que introduzca nada. Además de esto, se realiza una operación ‘SELECT’. Dicha operación la empleo para verificar que, al introducir el nombre de usuario (como es la clave primaria de la tabla de jugadores) comprobar si dicho atributo introducido ya se encuentra en la tabla o no. Por último, se realiza un ‘INSERT’ para almacenar todos los datos introducidos del jugador en la tabla de Jugadores.

- **RF 1.2 ‘Baja jugador’**

En esta funcionalidad, lo primero que se realiza es pedir al jugador que introduzca su nombre de usuario. Una vez introducido, comprobamos la existencia del mismo con un ‘SELECT’. Si el usuario no existe, se vuelve a pedir que lo escribamos correctamente. Pero si existe, realizamos el ‘DELETE’ correspondiente del jugador. Antes de finalizar, como hemos comentado anteriormente, pregunta al usuario si está seguro de dar de baja al jugador. Si este es el caso se realiza un commit y se finaliza la función. Pero si el usuario decide lo contrario, el programa realiza un rollback al savepoint colocado al principio de la función.

- **RF 1.3 ‘Modificar datos del jugador’**

En dicha función, lo primero que nos piden al igual que en muchas otras, es introducir nuestro nombre de usuario, y una vez más, comprobamos con la ayuda de un ‘SELECT’ que el usuario existe en nuestra base de datos. Tras esto, se le despliega al usuario un menú, en el que puede elegir el campo que desea modificar (modificará tantos campos como quiera hasta que seleccione la opción de salida). En cada uno de los campos correspondientes, tras pedir al usuario que introduzca el cambio (siempre comprobando que lo hace correctamente), se realiza un ‘UPDATE’ para reflejar esa modificación en nuestra base de datos. Finalmente, tal y como se explicó en la transacción anterior, se le pregunta al usuario si está seguro de confirmar todos los cambios que ha realizado. De ser así se realiza el commit oportuno y finaliza. Pero si este no es el caso, volvemos al principio de la función con un rollback.

- **RF 1.4 ‘Mostrar datos del jugador’**

Esta funcionalidad simplemente pide al jugador que introduzca su nombre de usuario. Tras esto, comprueba con un 'SELECT' que dicho nombre de usuario existe en la tabla de jugadores. Si no existe lo notifica, pero si existe, realiza un 'SELECT' para luego mostrar la gran mayoría de los atributos de dicho jugador asociado al nombre de usuario introducido.

- **RF 1.5 ‘Aplicar sanción’**

Dicha funcionalidad, pide al usuario que introduzca el nombre de usuario al cual se le quiere aplicar la sanción. Una vez comprobada la existencia del usuario (con el ‘SELECT’ correspondiente), se pide introducir la descripción de la sanción en cuestión. Una vez escrita, se realizan 2 ‘UPDATE’, uno para activar el campo de ‘jugSancionado’ y ponerlo a 1, y otro para guardar la descripción de la sanción.

- **RF 1.6 ‘Inscribirse al encuentro’**

Inscribirse al encuentro es una función a las que se le pasa un nombre de usuario y un código de encuentro. Tras comprobar que dichos parámetros existen en nuestra base de datos con los correspondientes ‘SELECT’, procedemos a realizar un ‘INSERT’ en la tabla de inscripcionJ donde asociamos el jugador al código del encuentro pasado por parámetro. Tras esto, preguntamos al usuario si está seguro de inscribirse al encuentro. Y como ya sabemos, si dice que si, hacemos un commit, sino, hacemos un rollback al principio de la función.

Disparadores

- **Disparador RF 1.2 deleteJugador**

La funcionalidad de este disparador se basa en que, antes de realizar un DELETE de un jugador, se borren las tuplas correspondiente donde esté presente ese jugador en las tablas.

```
CREATE OR REPLACE TRIGGER deleteJugador
  BEFORE
  DELETE ON JUGADORES
  FOR EACH ROW

  DECLARE
    equipoBorrar varchar2(20);
BEGIN
    DELETE FROM inscripciónJ WHERE nombreUsuario=:old.nombreUsuario;
    DELETE FROM adquiere WHERE nomusuario=:old.nombreUsuario;
    DELETE FROM consigue WHERE nomusuario=:old.nombreUsuario;
    DELETE FROM encuentros WHERE usuarioLider=:old.nombreUsuario;
    DELETE FROM alquiler WHERE usuario=:old.nombreUsuario;
    DELETE FROM pertenece WHERE jugador=:old.nombreUsuario;
    SELECT nombreequipo into equipoBorrar FROM equipos WHERE lider=:old.nombreUsuario;
    DELETE FROM inscripcionE WHERE equipo=equipoBorrar;
    DELETE FROM equipos WHERE nombreequipo=equipoBorrar;
    DELETE FROM valoraciones WHERE nomusuario=:old.nombreUsuario;
    DELETE FROM quejas WHERE nomusuario=:old.nombreUsuario;
END;
```

- **Disparador RF 1.6 controlSancion**

La funcionalidad de este disparador es comprobar, antes de realizar un INSERT en la tabla de inscripcionJ (o lo que es lo mismo antes de que un jugador se inscriba en un encuentro), que el jugador esté o no sancionado. Si lo está, se mostrará el mensaje de error que se ve a continuación.

```
CREATE OR REPLACE TRIGGER controlSancion
  BEFORE
  UPDATE OR INSERT on inscripciónJ
  FOR EACH ROW
  DECLARE
    sancionado number(1,0);

BEGIN
    SELECT jugSancionado INTO sancionado FROM jugadores WHERE nombreUsuario=:new.nombreUsuario;
        IF(sancionado = 1) THEN
                raise_application_error(-20600, ' El usuario está sancionado y no puede inscribirse al encuentro.');
        END IF;
END;
```

Subsistema Equipos

En el subsistema de equipos los requisitos funcionales no han sido necesario modificarlos para la implementación, en este sistema hemos hecho una especie de verificación de “permisos” por ejemplo al modificar equipo (RF 2.5), agregar un jugador al equipo (RF 2.3), expulsarlo (RF 2.4), eliminar equipo (RF 2.2) o inscribir a un encuentro (RF 2.6), de tal forma que estas accione solo podrán realizarse si el usuario activo que la realiza es el líder del equipo; sin embargo no se ha implementado la garantización de las operaciones que solo puede realizar un admin como aplicar sanción.

Además para algunas de las operaciones que más peso tienen al realizarlas como modificar equipo, eliminar equipo o expulsar jugador (ya que eliminan datos) ha sido añadida la confirmación de la acción con un savepoint antes de la operación y un rollback para cancelarla.

Al entrar en el menú del subsistema de equipos se pedirá el nombre de usuario con el que se desea “iniciar sesión”.

-**RF2.1 (Crear equipo)**

Esta funcionalidad pedirá los atributos necesarios para crear el equipo, pedirá cada atributo hasta que se introduzca un dato válido, además realiza una operación SELECT para verificar que la clave primaria (el nombre del equipo no está siendo usado), tras esto se realizan 2 INSERT un para almacenar los datos introducidos del equipo y además se incluirá al líder (usuario que crea el equipo) en la tabla pertenece para así tenerlo en cuenta como miembro del equipo.

-**RF2.2 (Baja equipo)**

Primero se pedirá el nombre del equipo que se desea dar de baja, y se comprobará que el líder del equipo a eliminar sea el mismo usuario que lo solicita con una operación SELECT la cual también servirá para saber si el equipo existe previamente, tras esto se hará una transacción compuesta de un DELETE la cuál antes de producirse se llamará al disparador deletePertenece (explicado en los disparadores del subsistema). Y se borrarán todos los datos almacenados de este equipo en las tablas equipos, pertenece e inscripcionE. Además se pedirá confirmación de la acción con el uso de savepoint y rollback antes de realizar las operaciones.

-**RF2.3 (Agregar jugador)**

Esta funcionalidad al igual que la anterior pedirá el nombre del equipo al que agregar el jugador y comprobará que el usuario que realiza la transacción es el líder del equipo mediante un SELECT, tras esto se pedirá el nombre del jugador a agregar al equipo y se comprobará su existencia con un SELECT, tras esto se producirá un INSERT a la tabla pertenece.

-**RF2.4 (Echar jugador)**

Servirá para expulsar a un jugador de un equipo, esta operación solo la puede realizar el líder y se comprobaría de la misma forma que en las dos funcionalidades anteriores, se pedirá el nombre del equipo y el nombre del jugador a expulsar se comprobará la existencia de los dos con un SELECT y se producirá un DELETE sobre la tabla pertenece con los atributos equipo y jugador correspondientes. Además se pedirá confirmación de la acción con el uso de savepoint y rollback antes de realizar las operaciones.

-**RF2.5 (Modificar equipo)**

En esta funcionalidad se ha eliminado la posibilidad de modificar la clave primaria del equipo (nombreEquipo) como se tenía pensado con anterioridad. Al igual que las anteriores se confirmará mediante un SELECT la existencia del equipo y que el editor sea el líder del mismo. Al igual que en crear se pedirá el dato correspondiente hasta que sea válido. Tras recopilar los datos correspondientes se realizará un UPDATE sobre el equipo elegido. Además se pedirá confirmación de la acción con el uso de savepoint y rollback antes de realizar las operaciones.

-**RF2.6 (Inscribir encuentro)**

Al igual que en las demás funcionalidades también se requiere que el que pida la realización de la operación sea el líder del equipo lo cual se realiza con un SELECT y se seguirá pidiendo hasta que se ingrese un equipo hasta que no se introduzca nada o se pida un equipo existente del cual el usuario sea líder. Tras esto se pedirá un código de encuentro y se comprobará que sea existente con un SELECT sobre la tabla encuentros.

Tras esto se procede a inscribir al equipo en el encuentro realizando la operación INSERT en la tabla inscripcionE y se comprobará que sea una inserción valida con el disparador controlInscripcion definido su funcionamiento más abajo en la sección de disparadores.

Además se realiza un SELECT para al terminar de realizar la operación poder mostrar los equipos inscritos a ese encuentro hasta el momento.

-**RF2.7 (Aplicar sanción)**

En esta funcionalidad pide un nombre de equipo a sancionar y se comprueba que exista con un SELECT, tras esto se pide una descripción de sanción válida y se realiza un UPDATE el cual pone el bit de equipoSancionado a 1 y añade la descripción de la sanción. Añadir también que se ha obviado la parte de que solamente un administrador del sistema pueda hacer esta operación para que sea más fácil probarla de tal forma que la puede realizar cualquier usuario (aunque no sea correcto realisticamente.)

-**RF2.8 (Mostrar Equipo)**

Esta funcionalidad simplemente pide un nombre de equipo y mediante un SELECT sobre la tabla equipos, de esta forma se consiguen los datos necesarios a

mostrar (además de comprobar su existencia) y a continuación se realiza otro SELECT sobre la tabla pertenece para mostrar los miembros que pertenecen a tal equipo.

-DISPARADORES (RF2.2 y RF 2.6)

Disparador RF 2.2 deletePertenece

```
38  create or replace trigger deletePertenece
39    BEFORE DELETE on equipos
40    FOR EACH ROW
41
42  BEGIN
43      DELETE FROM pertenece WHERE equipo=:old.nombreEquipo;
44      DELETE FROM inscripcionE WHERE equipo=:old.nombreEquipo;
45  END;
```

La función de este disparador que antes de realizar un DELETE de un equipo se borren las tuplas correspondiente donde esté presente ese equipo en las tablas pertenece (borra la asociación jugadores con el equipo) e inscripcionE.

Disparador RF 2.6 controlInscripcion

La función de este disparador consiste en comprobar antes de hacer un insert sobre la tabla inscripcionE que no haya ya dos equipos inscritos en el mismo encuentro, de lo contrario saltará un mensaje de error y no se realizará la inserción.

```
47  CREATE OR REPLACE TRIGGER controlInscripcion
48    BEFORE INSERT on inscripcionE
49      FOR EACH ROW
50      declare
51          cnt number;
52    BEGIN
53
54          SELECT COUNT(codEnc) INTO cnt from inscripcionE  where codEnc=:new.codEnc;
55      IF (  cnt >= 2) THEN
56        raise_application_error(-20600, ' Ya hay 2 participantes inscriptos en este encuentro.');
57      END IF;
58    END;
```

Subsistema Ubicaciones

En el caso del subsistema de ubicaciones no ha sido necesario alterar nada de los requisitos ya acordados.

Para el desarrollo de estos no se han implementado control de transacciones (rollbacks ni savepoints) puesto que la totalidad de las funciones son transacciones simples, puesto que no realizan más de una operación en los datos (INSERT o UPDATE) y solo es necesario hacer commit o no, con la previa pregunta al usuario antes de la realización de la operación.

-RF3.1 (Alta ubicación)

En la primera funcionalidad del subsistema, se han de preguntar todos los datos necesarios de una ubicación por terminal. Tras esto, se generará un código alfanumérico de 5 caracteres y se verificará que no existe una ubicación con este, con una operación SELECT a la tabla ubicación.

Tras esto, se pregunta si se quiere realizar la inserción y en caso afirmativo se realizará la operación INSERT y su correspondiente commit.

-RF3.2 (Baja ubicación)

Para el dado de baja de una ubicación, recordamos que no se realiza el borrado de esta en la base de datos, simplemente se almacena su estado de Fuera de servicio con la correspondiente justificación, puesto que podría establecerse de nuevo en el futuro con el arreglo de los posibles desperfectos que hayan ocasionado su baja del sistema.

Para ello se pedirá el código de la ubicación en cuestión y se realizará una operación SELECT en busca del código y ubicación en estado activo.

Posteriormente se pregunta el motivo de esto y se realiza la operación UPDATE de los atributos justBaja y fueraServicio.

-RF3.2’ (Rehabilitado ubicación)

Una funcionalidad muy simple ha sido añadida, y es la posibilidad de restablecer una ubicación como activa en el sistema. Para ello se realiza de nuevo una CONSULTA por código y verificando que la ubicación se encuentre inactiva. Tras esto se realiza el UPDATE eliminando la justificación de dado de baja y restableciendo el atributo fueraServicio a 0.

-RF3.3 (Mostrar ubicación)

Para mostrar una ubicación se realiza una CONSULTA por código y se obtiene el valor de fueraServicio, para garantizar una respuesta adecuada en el

caso de que se encuentre ACTIVA o INACTIVA. Posteriormente se hace una CONSULTA más amplia con los datos completos de la ubicación.

-RF3.4 (Consultar disponibilidad para alquiler de ubicación)

La funcionalidad más compleja es la siguiente, es requerido realizar en primer lugar una CONSULTA por código de ubicación y obtener los datos fueraServicio, horaApertura, horaCierre y deportesUbi. A continuación si la ubicación se encuentra ACTIVA se procede a obtener una fecha y horas adecuadas, que deben de estar dentro del tramo de apertura de la ubicación.

Además se pide el deporte a alquilar, y se comprueba que está dentro de la lista de deportes de esa ubicación.

A continuación se realiza una CONSULTA en la tabla Alquiler por código de ubicación, fecha, horas de inicio y fin y deporte.

En el caso de que no exista este alquiler, se vuelve a realizar otra CONSULTA en busca de alquileres ese mismo día, siendo necesario observarlos todos ellos y ver que las horas consultadas no se encuentran entre medias de otros alquileres. SE GARANTIZAN RS3.4 Y RS3.5

Por último, si el tramo se encuentra libre, se le da la opción de alquilar esta ubicación, con la previa lectura de usuario y su CONSULTA y la INSERCIÓN del nuevo alquiler con los datos consultados.

-RF3.5 (Alquiler de ubicación)

Para la adecuación de esta funcionalidad se aprovecha el código implementado para la anterior consulta de disponibilidad. Es por esto que solo se realiza la pregunta y CONSULTA del nombre de usuario para crear el alquiler, y la llamada a la funcionalidad de consultar disponibilidad, ya no siendo necesario volver a introducir el usuario ni preguntar si se quiere hacer el alquiler.

-DISPARADORES (RS3.3 y RS3.4)

Por último, mediante dos disparadores simples, se garantizan que en inserciones o modificaciones de las tablas ubicación y alquiler, no se pisen las horas de apertura y cierre ni de inicio y fin de alquiler, respectivamente.

```
CREATE OR REPLACE TRIGGER correccionHoraAlquiler
    BEFORE
    UPDATE OR INSERT ON alquiler
    FOR EACH ROW

BEGIN
    IF (TO_DATE(:new.horainicio,'HH24:MI')>= TO_DATE(:new.horafin,'HH24:MI')) THEN
        raise_application_error(-20600, ' Las horas de alquiler y finalizado no se pueden pisar');
    END IF;
END;

CREATE OR REPLACE TRIGGER correccionHoraApertura
    BEFORE
    UPDATE OR INSERT ON ubicacion
    FOR EACH ROW

BEGIN
    IF (TO_DATE(:new.horaapertura,'HH24:MI')>= TO_DATE(:new.horacierre,'HH24:MI')) THEN
        raise_application_error(-20600, ' Las horas de apertura y cierre no se pueden pisar');
    END IF;
END;
```

Subsistema Encuentros

En la realización de esta práctica hemos tenido que quitar el atributo motivosCancelarEnc ya que se iba utilizar para guardarlo cuando se borraba un encuentro, pero preferimos borrarlo directamente de la base de datos y los motivos pasarían a ser un condicionante para poder cancelar el encuentro.

Todas las transacciones que hemos implementado son para las operaciones DML(INSERT,DELETE,UPDATE). Específicamente en los **RF 4.1(Crear Encuentro)**, **RF 4.2(Cancelar encuentro)**, **RF 4.3(Modificar Encuentro)** , **RF 4.6(Establecer encuentro)**, como estas funciones modifican o alteran la tabla encuentros de nuestra base de datos, siempre al final de cada función preguntamos si queremos confirmar los cambios, si se responde “SI” se hace commit para avanzar y si respondemos que “NO” hacemos un rollback al savepoint establecido siempre al principio de cada método.

-RF4.1 (Crear Encuentro)

En esta funcionalidad se pedirán todos los datos necesarios para la creación de un encuentros, si cada dato introducido no cumple el formato esperado se seguirá pidiendo hasta que sí lo sea, el usuario que cree el encuentro será el usuario líder del mismo. La ubicación introducida debe existir en la base de datos del sistema por lo que se hace un SELECT para comprobar si existe dicha localización.

Una vez introducidos todos los datos se generará un código de encuentro(cadena alfanumérica de 5 caracteres) totalmente aleatorio, se comprobará si este ya existe, en caso afirmativo se volverá a crear otro hasta que no exista y una vez llegados a

este caso se preguntará si se quiere crear el encuentro, si se responde que no se hará un rollback hasta el savepoint creado antes de la entrada de datos y si se responde que sí, se hará la operación INSERT necesaria para crear el encuentro y su correspondiente commit.

-RF4.2 (Cancelar encuentro)

En esta funcionalidad se preguntará al usuario que introduzca el código del encuentro que desea cancelar, se buscará dicho código mediante una operación SELECT en la que además de buscar si existe, tiene que cumplir la condición de que el usuario líder de ese encuentro sea el mismo que el que solicita cancelarlo.

Luego se pedirá que se introduzca una opción sobre los motivos para cancelar el encuentro, por último tendremos que confirmar si queremos cancelar el encuentro, en caso afirmativo haremos un DELETE al encuentro identificado con dicho código de encuentro y su correspondiente commit, en caso contrario, haremos un rollback al savepoint creado antes de introducir los datos para cancelar el encuentro.

-RF4.3 (Modificar encuentro)

En esta funcionalidad se preguntará al usuario que introduzca el código del encuentro que desea modificar, se buscará dicho código mediante una operación SELECT en la que además de buscar si existe, tiene que cumplir la condición de que el usuario líder de ese encuentro sea el mismo que el que solicita modificarlo.

Después de esto se mostrará un menú en el que se podrá elegir el dato a modificar, cada uno tendrá sus respectivas restricciones respecto al formato(la de la ubicación sigue siendo la misma que en el RF4.1), cuando se introduzca bien el dato se realizará el UPDATE-SET con el nuevo valor al atributo correspondiente del encuentro identificado con el código de encuentro introducido anteriormente.

Una vez realizado esto podremos seguir modificando los datos que queramos hasta que elijamos la opción de no modificar más, es aquí donde se pedirá confirmar todos los cambios realizados, si la respuesta es afirmativa se realizará el commit correspondiente y si es negativa se hará un rollback al savepoint creado antes de empezar a modificar datos.

-RF4.4(Mostrar Encuentro)

Esta funcionalidad es la más corta, simplemente se pide introducir un código de encuentro que sí existe en la base de datos, se realizará un SELECT de todos los atributos del encuentro que se identifique con dicho código.

-RF4.5(Búsqueda de Encuentros)

Esta funcionalidad es muy similar a la anterior, elegiremos una opción sobre el filtro de búsqueda a realizar, se pedirá introducir el valor del dato y se hará un SELECT

que muestre todos los atributos de los encuentros que cumplan con los datos de búsqueda. Cabe destacar que la apuesta la busca en un rango de -100 y +100 al introducido para que no tenga que coincidir con el mismo.

-RF4.6 (Establecer resultado encuentro)

En la última funcionalidad del subsistema, se preguntará al usuario que introduzca el código del encuentro al que desea establecer el resultado, se buscará dicho código mediante una operación SELECT en la que además de buscar si existe, tiene que cumplir la condición de que el usuario líder de ese encuentro sea el mismo que el que solicita establecer el resultado.

Luego se pedirá que introduzca la puntuación del equipo local(del equipo del usuario líder) y la puntuación del equipo visitante, ambos tienen que ser mayores o iguales que 0, en caso contrario no se avanzaría.

Para acabar se pide confirmación para establecer los resultados, en caso afirmativo se realizaría un UPDATE-SET a los atributos puntuación local y puntuación visitante del encuentro identificado con el código introducido anteriormente, con su correspondiente commit. En caso contrario se hará un rollback al savepoint creado antes de introducir los datos.

-DISPARADORES (RF4.2)

Básicamente este disparador actuará después de borrar un encuentro de la base de datos, su función es la de borrar las tuplas de las tablas inscripcionE, inscripcionJ y adquiere en las que aparece el código de encuentro que habíamos borrado:

```
create or replace trigger deleteEncuentros
  BEFORE DELETE on encuentros
  FOR EACH ROW

BEGIN
    DELETE FROM inscripcionE WHERE codEnc=:old.codEnc;
    DELETE FROM adquiere WHERE codEnc=:old.codEnc;
    DELETE FROM  inscripciónJ WHERE codEnc=:old.codEnc;
END;
```

Subsistema Valoraciones y Quejas

En el caso del subsistema de valoraciones y quejas he cambiado la forma de mostrar las valoraciones para que pueda meter un nombre de usuario y filtrar esas valoraciones o sino mostrar todas las del sistema.

Para el desarrollo de estos requisitos no se han implementado control de transacciones (rollbacks ni savepoints) puesto que la totalidad de las funciones son transacciones simples, puesto que no realizan más de una operación en los datos

(INSERT o UPDATE) y solo es necesario hacer commit o no, con la previa pregunta al usuario antes de la realización de la operación.

-RF5.1 (Crear Valoración)

En la primera funcionalidad del subsistema, se han de preguntar todos los datos necesarios de una valoración por terminal. Tras esto, se generará un código alfanumérico de 5 caracteres y se verificará que existe un jugador que coincida con el nombre aportado en esta, con una operación SELECT a la tabla de jugador.

Tras esto, se pregunta si se quiere realizar la inserción y en caso afirmativo se realizará la operación INSERT y su correspondiente commit. En caso negativo, se preguntaran los datos de nuevo, haciendo el rollback antes de preguntar los datos.

-RF5.2 (Consultar Valoraciones)

Esta funcionalidad tiene dos opciones:

- Mostrar todas las valoraciones del sistema. Mostrando todos los datos de todas las valoraciones, haciendo uso del SELECT a valoraciones.

- Filtrar las valoraciones por un nombre de Usuario. Mostrando los datos de esas valoraciones de ese Usuario, haciendo uso del SELECT a valoraciones con la condición del nombre del usuario.

-RF5.3 (Modificar Valoración)

En esta funcionalidad el usuario ingresara por terminal el código de la valoración que quiere modificar y se verificará de que existe el código sino le volverá a preguntar. Una vez comprobado el código, se deberá ingresar el nuevo comentario para la valoración, una vez que cumpla los requisitos se hará un UPDATE al comentario de ese código.

-RF5.4 (Borrar Valoración)

Para borrar una valoración, se le pedirá introducir el código de la valoración que desea borrar, se comprobará que exista y si es así, se hará el DELETE de la tabla valoraciones según el código introducido.

Tras esto, se pregunta si se quiere realizar el borrado y en caso afirmativo se realizará la operación DELETE y su correspondiente commit. En caso negativo, se preguntará el código de nuevo, por el rollback antes de preguntar los datos.

-RF5.5 (Crear Queja)

En la última funcionalidad del subsistema es lo mismo que la funcionalidad crear Valoración(RF5.1), se han de preguntar todos los datos necesarios de una queja por terminal. Tras esto, se generará un código alfanumérico de 5 caracteres y se verificará que existe un jugador que coincida con el nombre aportado en esta, con una operación SELECT a la tabla de jugador.

Tras esto, se pregunta si se quiere realizar la inserción y en caso afirmativo se realizará la operación INSERT y su correspondiente commit. En caso negativo, se preguntaran los datos de nuevo, haciendo rollback antes de preguntar los datos.

-DISPARADORES

Por último, mediante los disparadores controlamos que a la hora de insertar una valoración no haya un título de una queja que sea igual sobre un mismo jugador y viceversa.

```
CREATE OR REPLACE TRIGGER evitarMismoTitulo
    BEFORE
    UPDATE OR INSERT ON valoraciones
    FOR EACH ROW
    DECLARE
    aux varchar2(50);
BEGIN
BEGIN
    SELECT titulo into aux from quejas where nomUsuario=:new.nomUsuario AND titulo=:new.titulo;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        aux:=NULL;
END;
    IF (aux=:new.titulo) THEN
            raise_application_error(-20600, ' El título coincide con una queja ya registrada a ese jugador.');
    END IF;
END;


CREATE OR REPLACE TRIGGER evitarMismoTitulo2
    BEFORE
    UPDATE OR INSERT ON quejas
    FOR EACH ROW
    DECLARE
    aux varchar2(50);
BEGIN

BEGIN
    SELECT titulo into aux from valoraciones where nomUsuario=:new.nomUsuario AND titulo=:new.titulo;
    EXCEPTION
    WHEN NO_DATA_FOUND THEN
        aux:=NULL;
END;
    IF (aux=:new.titulo) THEN
            raise_application_error(-20600, ' El título coincide con una valoracion ya registrada a ese jugador.');
    END IF;
END
```

Subsistema Premios

Por último,en el caso del subsistema de premios,tampoco ha sido necesario alterar nada de los requisitos ya acordados.

Para el desarrollo de estos se han implementado control de transacciones (rollbacks y savepoints) puesto que hay funciones que no realizan transacciones simples, es decir, realizan más de una operación DML(INSERT,DELETE,UPDATE) sobre los datos. Para las transacciones simples solo es necesario hacer commit o no, con la previa pregunta al usuario antes de la realización de la operación.

-RF6.1(Añadir premio)

Para esta primera funcionalidad del subsistema, se preguntarán los atributos de nombre, descripción, cantidad y precio del premio en cuestión, el cual se introducirán por el terminal. Una vez introducidos correctamente cumpliendo las restricciones semánticas, se genera un código único alfanumérico de 8 caracteres para identificar al premio puesto que se comprueba mediante un SELECT que no sea repetido.

Tras esto se pregunta si se desea confirmar la acción de añadir el premio, en el caso negativo, no se realiza nada y en caso afirmativo se realiza el INSERT de los datos y por último el commit.

-RF6.2 (Borrar premio)

En esta funcionalidad, se le pasará un código único de premio, con el cual realizará un DELETE de la tabla premios al que esté identificado por ese código. En este caso se lanzará el único disparador que tiene este subsistema, debido a que la tabla CONSIGUE utiliza como clave externa códigos de premios, por lo tanto este disparador ayuda a realizar el borrado en cascada. (Código del disparador al final).

Antes de realizar el DELETE establecemos un savepoint, puesto que si el disparador realiza su función estaremos haciendo varios DELETE. Se preguntará por último si se desea realizar el borrado, en donde si respondemos afirmativamente se realizará el commit, y el en el caso contrario, se realizará un rollback al savepoint establecido antes de realizar los DELETE.

-RF6.3 (Mostrar Lista Premios)

Esta es la funcionalidad más sencilla, pues solo realiza un SELECT de la tabla premios al completo y la muestra por pantalla.

-RF6.4 (Obtener puntos)

Esta es la funcionalidad más enrevesada puesto que toma muchas cosas en cuenta. Se le pasan el nombre de usuario y el código del encuentro del cual se quieran obtener los puntos.

En primer lugar se comprueba mediante un SELECT de la tabla ADQUIERE que no se hayan obtenido ya puntos por parte de ese usuario, de ese encuentro.

Si no se han obtenido ya, se pasa a consultar mediante otro SELECT de la tabla ENCUENTROS, el usuario líder, la hora del encuentro, la puntuación del equipo local y la puntuación del equipo visitante y se guardan esos datos. Dará fallo en caso de que no haya ningún encuentro identificado por ese código de encuentro.

Ahora se pasa a realizar una comprobación, la cual consiste en que el encuentro esté terminado, por lo que se mira que la hora y fecha actual, sea después de la hora del encuentro más 2 horas. Es decir, que si el encuentro era a las 18:00, hasta las 20:00 no se podrán obtener los puntos (dando así por hecho que el encuentro está terminado y los resultados anotados).

Se sigue ahora con la comprobación de en que equipo está el usuario, si local o visitante. Esto es posible puesto que el usuario líder del encuentro pertenece siempre al equipo local. Primero sacaremos el código de un equipo que participe en el encuentro con un SELECT de la tabla InscripciónE. Ahora con ese código de equipo se realiza otro SELECT de la tabla PERTENECE para obtener los jugadores de ese equipo. Se van comprobando uno a uno los jugadores, para saber si alguno de ellos coincide con el usuario que quiere obtener puntos y con el usuario líder. En caso de coincidir con el líder, sabemos que ese equipo es el local, y en caso contrario, será el equipo visitante. Y al comprobar con el usuario que quiere obtener puntos también, pues si está con el usuario líder estará en el equipo local y en el otro caso pues estará en el visitante.

Después de eso, comprobamos el resultado del encuentro gracias a la puntuación local y la puntuación visitante. En caso de ganar el equipo en el cual esté el usuario, ganará más puntos, si pierde, obtendrá muchos menos puntos y en caso de empate, obtendrá más que si perdiera y menos que si ganara.

Por lo que terminamos con un SELECT de los puntos del jugador, para sumarle la cantidad de puntos correspondientes.

Llegamos a la parte de realizar las sentencias DML y puesto que vamos a realizar un INSERT en la tabla ADQUIERE que guarde esta recepción de puntos y un UPDATE de los puntos del jugador. Creamos un Savepoint antes de ello, y mediante una pregunta de confirmación de la acción, se realiza un rollback al savepoint si la respuesta es negativa, y un commit si la respuesta es afirmativa.

-RF6.5 (Canjear Premio)

Para esta última funcionalidad, pasaremos el nombre del usuario que va a recibir el premio y el código del premio que se quiere obtener.

Se empieza realizando una consulta con SELECT de la tabla PREMIOS para obtener el precio y la cantidad de existencias disponibles de ese premio. En caso de no quedar existencias no se podrá canjear.

Se consultarán a continuación los puntos del jugador con un SELECT, y si los puntos que tienen son menos del precio del premio a canjear, no se podrá entonces obtener el premio.

Luego de esto, se genera un código único alfanumérico de 16 caracteres, puesto que se comprueba que no está repetido con un SELECT en la tabla CONSIGUE. Se resta en uno la cantidad del stock del premio, se le quitan los puntos que valga el premio al jugador y se proceden a realizar las operaciones DML.

Antes de realizarlas crearemos un savepoint. Estas operaciones consisten en un UPDATE de PREMIOS actualizando la cantidad del premio, otro UPDATE de JUGADORES, restando los puntos del precio del premio a los puntos del usuario que lo ha canjeado, y finalmente, un INSERT a CONSIGUE guardando el usuario, el premio, y el código obtenido (el cual se debería usar en la tienda correspondiente).Finalmente, una pregunta de confirmación, con un rollback al savepoint en caso negativo, y el commit en el caso afirmativo.

DISPARADOR RF6.2

Como se ha explicado en el RF6.2, este disparador, borrará cualquier fila de la tabla CONSIGUE, antes de que se borre el premio de la tabla PREMIOS identificado por el mismo código de premio.

```
CREATE OR REPLACE TRIGGER borradoPremio
        BEFORE
        DELETE ON premios
        FOR EACH ROW

BEGIN
        DELETE FROM consigue  WHERE codPre=:old.codpre;
END;
```

MOTIVACIÓN DE LA ELECCIÓN DE SOFTWARE

Tras la realización del seminario 2 descubrimos una manera no muy compleja de realizar el control de transacciones para un sistema de información.

Es por esto que hemos diseñado la totalidad de este en JAVA, desarrollando cada funcionalidad en funciones, dentro de clases que representan subsistemas completos. A su vez, se ha realizado gestión de paquetes y creado una clase principal que muestre el menú completo de nuestra aplicación de encuentros deportivos, siendo esta textual para ahorrar trabajo que no incumbe a la asignatura.

El control de transacciones se ha realizado entonces utilizando la libreria OJDBC8, e implementando la base de datos en ORACLE SQL utilizando el servidor de la escuela.

Para concluir la práctica, comentar que ha sido muy llevadera y entretenida, con lo que hemos aprendido muchísimo con respecto a este campo.