Base de Datos de Liga de Futbol Profesional

Se desea guardar en primer lugar los datos de los jugadores. De cada jugador se quiere guardar el nombre1, nombre2, apellido1, apellido2, correo electrónico(s), municipio donde reside, fecha de nacimiento y posición en la que juega (portero, defensa, delantero, centrocampista, etc.). Cada jugador tiene un código de jugador que lo identifica de manera única. De cada uno de los equipos de la liga es necesario registrar el nombre del equipo, nombre del estadio en el que juega, el aforo que tiene el estadio, el año de fundación del equipo y el departamento que representa el equipo. Cada equipo también tiene un código que lo identifica de manera única. Un jugador solo puede pertenecer a un equipo. De cada partido que los equipos de la liga juegan hay que registrar la fecha en la que se juega el partido, los goles que ha metido el equipo de casa y los goles que ha metido el equipo de fuera. Cada partido tendrá un código numérico para identificar el partido. También se quiere llevar un recuento de los goles que hay en cada partido. Se quiere almacenar el minuto en el que se realiza el gol y la descripción del gol. Un partido tiene varios goles y un jugador puede anotar varios goles en un partido. Por último, se quiere almacenar, en la base de datos, los datos de los presidentes de los equipos de futbol (dpi, nombre1, nombre2, nombre3, apellido1, apellido2, fecha de nacimiento, correo electrónico(s), municipio donde reside, equipo del que es presidente y año en el que fue elegido presidente). Un equipo de futbol tan solo puede tener un presidente, y una persona solo puede ser presidente de un equipo de la liga.

Entidades y atributos

1. **Departamento**
    - codigo_departamento (PK)
    - nombre
2. **Municipio**
    - codigo_municipio (PK)
    - nombre
    - codigo_departamento (FK)
3. **Jugador**
    - codigo_jugador (PK)
    - nombre1, nombre2, apellido1, apellido2

- fecha_nacimiento
- posicion
- codigo_municipio (FK)
- codigo_equipo (FK)

4. **Equipo**
    - codigo_equipo (PK)
    - nombre
    - estadio
    - aforo
    - fundacion
    - codigo_departamento (FK)
5. **Partido**
    - codigo_partido (PK)
    - fecha
    - goles_casa
    - goles_fuera
    - codigo_equipo_casa (FK)
    - codigo_equipo_fuera (FK)

6. **Gol**
    - id_gol (PK)
    - codigo_partido (FK)
    - codigo_jugador (FK)
    - minuto
    - descripcion

7. **Presidente**
    - dpi (PK)
    - nombre1, nombre2, nombre3
    - apellido1, apellido2
    - fecha_nacimiento
    - correo_electronico
    - codigo_municipio (FK)
    - codigo_equipo (FK) UNIQUE
    - anio_elegido
8. Correo Jugador
    - id_correo_jugador (PK)
    - correo
    - codigo_jugador (FK)
9. Correo Presidente
    - id_correo_presidente (PK)
    - dpi (FK)
    - correo_electronico

Relaciones

- Un **departamento** tiene varios **municipios** (1:*).
- Un **municipio** pertenece a un **departamento** (1:1).
- Un **equipo** representa a un **departamento** (N:1).
- Un **equipo** tiene muchos **jugadores** (1:*).
- Un **equipo** tiene un **presidente** (1:1).
- Un **jugador** reside en un **municipio** (N:1).
- Un **presidente** reside en un **municipio** (N:1).
- Un **partido** involucra dos **equipos** (casa y fuera).
- Un **partido** tiene muchos **goles** (1:*).

- Un **jugador** puede anotar varios **goles** en diferentes partidos (1:*).
- Un **jugador** puede tener varios **correos** (1:*).
- Un **presidente** puede tener varios **correos** (1:*).

Realizando el análisis de las Entidades y Relaciones, todo está normalizado en **3FN**:

- No hay atributos multivaluados ni repetidos.
- Cada tabla depende de su clave primaria.
- No hay dependencias transitivas.

Creación de la Base de Datos y sus Tablas respectivas

Para realizar la creación de la base de datos (utilizando sqlDeveloper) y las distintas tablas que la componen se utilizarán los scripts:

Crear esquema (opcional)

```
CREATE USER liga_futbol IDENTIFIED BY liga123;

GRANT CONNECT, RESOURCE TO liga_futbol;

ALTER USER liga_futbol DEFAULT TABLESPACE USERS;
```

Usar el esquema

```
ALTER SESSION SET CURRENT_SCHEMA = liga_futbol;
```

Tabla Departamento

```
CREATE TABLE Departamento (
   codigo_departamento NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
   nombre VARCHAR2(100) NOT NULL
);
```

Tabla Municipio

```
CREATE TABLE Municipio (
   codigo_municipio NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
   nombre VARCHAR2(100) NOT NULL,
   codigo_departamento NUMBER NOT NULL,
   CONSTRAINT fk_municipio_departamento FOREIGN KEY (codigo_departamento)
      REFERENCES Departamento(codigo_departamento)
);
```

Tabla Equipo

```
CREATE TABLE Equipo (
codigo_equipo NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
   nombre VARCHAR2(100) NOT NULL,
   estadio VARCHAR2(100) NOT NULL,
   aforo NUMBER NOT NULL,
   fundacion NUMBER(4) NOT NULL,
   codigo_departamento NUMBER NOT NULL,
   CONSTRAINT fk_equipo_departamento FOREIGN KEY (codigo_departamento)
      REFERENCES Departamento(codigo_departamento)
);
```

Tabla Presidente

```
CREATE TABLE Presidente (
   dpi VARCHAR2(15) PRIMARY KEY,
```

```
    nombre1 VARCHAR2(50) NOT NULL,
    nombre2 VARCHAR2(50),
    nombre3 VARCHAR2(50),
    apellido1 VARCHAR2(50) NOT NULL,
    apellido2 VARCHAR2(50),
    fecha_nacimiento DATE NOT NULL,
    codigo_municipio NUMBER NOT NULL,
    anio_elegido NUMBER(4) NOT NULL,
    codigo_equipo NUMBER UNIQUE,
    CONSTRAINT fk_presidente_equipo FOREIGN KEY (codigo_equipo)
        REFERENCES Equipo(codigo_equipo),
    CONSTRAINT fk_presidente_municipio FOREIGN KEY (codigo_municipio)
        REFERENCES Municipio(codigo_municipio)
);
```

Tabla CorreoPresidente

```
CREATE TABLE CorreoPresidente (
    id_correo_presidente NUMBER GENERATED BY DEFAULT AS IDENTITY
PRIMARY KEY,
    dpi VARCHAR2(15) NOT NULL,
    correo VARCHAR2(100) NOT NULL,
    CONSTRAINT fk_correo_presidente FOREIGN KEY (dpi)
        REFERENCES Presidente(dpi)
);
```

Tabla Jugador

```
CREATE TABLE Jugador (
    codigo_jugador NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    nombre1 VARCHAR2(50) NOT NULL,
    nombre2 VARCHAR2(50),
    apellido1 VARCHAR2(50) NOT NULL,
    apellido2 VARCHAR2(50),
    fecha_nacimiento DATE NOT NULL,
    posicion VARCHAR2(20) CHECK (posicion IN ('Portero','Defensa','Delantero','Centrocampista')),
    codigo_municipio NUMBER NOT NULL,
    codigo_equipo NUMBER NOT NULL,
    CONSTRAINT fk_jugador_equipo FOREIGN KEY (codigo_equipo)
        REFERENCES Equipo(codigo_equipo),
    CONSTRAINT fk_jugador_municipio FOREIGN KEY (codigo_municipio)
        REFERENCES Municipio(codigo_municipio)
);
```

Tabla CorreoJugador

```
CREATE TABLE CorreoJugador (
    id_correo_jugador NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    codigo_jugador NUMBER NOT NULL,
    correo VARCHAR2(100) NOT NULL,
    CONSTRAINT fk_correo_jugador FOREIGN KEY (codigo_jugador)
```

```
    REFERENCES Jugador(codigo_jugador)
);
```

Tabla Partido

```
CREATE TABLE Partido (
  codigo_partido NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  fecha DATE NOT NULL,
  goles_casa NUMBER DEFAULT 0,
  goles_fuera NUMBER DEFAULT 0,
  codigo_equipo_casa NUMBER NOT NULL,
  codigo_equipo_fuera NUMBER NOT NULL,
  CONSTRAINT fk_partido_equipo_casa FOREIGN KEY (codigo_equipo_casa)
    REFERENCES Equipo(codigo_equipo),
  CONSTRAINT fk_partido_equipo_fuera FOREIGN KEY (codigo_equipo_fuera)
    REFERENCES Equipo(codigo_equipo)
);
```

Tabla Gol

```
CREATE TABLE Gol (
  id_gol NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  codigo_partido NUMBER NOT NULL,
  codigo_jugador NUMBER NOT NULL,
  minuto NUMBER NOT NULL,
  descripcion VARCHAR2(255),
  CONSTRAINT fk_gol_partido FOREIGN KEY (codigo_partido)
```

```
    REFERENCES Partido(codigo_partido),
  CONSTRAINT fk_gol_jugador FOREIGN KEY (codigo_jugador)
    REFERENCES Jugador(codigo_jugador) );
```

Procedimientos de Administración y Mantenimiento

1. Procedimientos Almacenados

Procedimiento para insertar un nuevo equipo

```
CREATE OR REPLACE PROCEDURE InsertarEquipo(
  p_nombre IN VARCHAR2,
  p_estadio IN VARCHAR2,
  p_aforo IN NUMBER,
  p_fundacion IN NUMBER,
  p_codigo_departamento IN NUMBER
) AS
BEGIN
  INSERT INTO Equipo (nombre, estadio, aforo, fundacion, codigo_departamento)
  VALUES (p_nombre, p_estadio, p_aforo, p_fundacion, p_codigo_departamento);
  COMMIT;
  DBMS_OUTPUT.PUT_LINE('Equipo insertado correctamente');
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE('Error al insertar equipo: ' || SQLERRM);
END;
```

Procedimiento para transferir un jugador entre equipos

```
CREATE OR REPLACE PROCEDURE TransferirJugador(
   p_codigo_jugador IN NUMBER,
   p_codigo_equipo_destino IN NUMBER
) AS
   v_equipo_actual NUMBER;
BEGIN
   -- Verificar si el jugador existe
   SELECT codigo_equipo INTO v_equipo_actual
   FROM Jugador
   WHERE codigo_jugador = p_codigo_jugador;

   IF v_equipo_actual = p_codigo_equipo_destino THEN
      DBMS_OUTPUT.PUT_LINE('El jugador ya pertenece a este equipo');
      RETURN;
   END IF;
```

Actualizar el equipo del jugador

```
   UPDATE Jugador
   SET codigo_equipo = p_codigo_equipo_destino
   WHERE codigo_jugador = p_codigo_jugador;

   COMMIT;
   DBMS_OUTPUT.PUT_LINE('Jugador transferido correctamente del equipo ' ||
                        v_equipo_actual || ' al equipo ' || p_codigo_equipo_destino);
EXCEPTION
```

```
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Jugador no encontrado');
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error en transferencia: ' || SQLERRM);
END;
```

Procedimiento para registrar un partido completo

```
CREATE OR REPLACE PROCEDURE RegistrarPartido(
    p_fecha IN DATE,
    p_equipo_casa IN NUMBER,
    p_equipo_fuera IN NUMBER
) AS
    v_partido_id NUMBER;
BEGIN
    -- Insertar el partido
    INSERT INTO Partido (fecha, codigo_equipo_casa, codigo_equipo_fuera)
    VALUES (p_fecha, p_equipo_casa, p_equipo_fuera)
    RETURNING codigo_partido INTO v_partido_id;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Partido registrado con ID: ' || v_partido_id);
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error al registrar partido: ' || SQLERRM);
END;
```

Procedimiento para obtener estadísticas de un equipo

```
CREATE OR REPLACE PROCEDURE EstadisticasEquipo(
   p_codigo_equipo IN NUMBER
) AS
   v_total_jugadores NUMBER;
   v_promedio_edad NUMBER;
   v_porteros NUMBER;
   v_defensas NUMBER;
   v_centrocampistas NUMBER;
   v_delanteros NUMBER;
BEGIN
   -- Total de jugadores
   SELECT COUNT(*) INTO v_total_jugadores
   FROM Jugador
   WHERE codigo_equipo = p_codigo_equipo;
```

Promedio de edad

```
   SELECT AVG(EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR
FROM fecha_nacimiento))
   INTO v_promedio_edad
   FROM Jugador
   WHERE codigo_equipo = p_codigo_equipo;
```

Jugadores por posición

```
 SELECT COUNT(*) INTO v_porteros
FROM Jugador
WHERE codigo_equipo = p_codigo_equipo AND posicion = 'Portero';
```

```
SELECT COUNT(*) INTO v_defensas
 FROM Jugador
WHERE codigo_equipo = p_codigo_equipo AND posicion = 'Defensa';

 SELECT COUNT(*) INTO v_centrocampistas
FROM Jugador
WHERE codigo_equipo = p_codigo_equipo AND posicion = 'Centrocampista';

 SELECT COUNT(*) INTO v_delanteros
FROM Jugador
WHERE codigo_equipo = p_codigo_equipo AND posicion = 'Delantero';
```

Mostrar resultados

```
 DBMS_OUTPUT.PUT_LINE('=== ESTADÍSTICAS DEL EQUIPO ===');
 DBMS_OUTPUT.PUT_LINE('Total jugadores: ' || v_total_jugadores);
 DBMS_OUTPUT.PUT_LINE('Promedio edad: ' || ROUND(v_promedio_edad, 1));
 DBMS_OUTPUT.PUT_LINE('Porteros: ' || v_porteros);
DBMS_OUTPUT.PUT_LINE('Defensas: ' || v_defensas);
DBMS_OUTPUT.PUT_LINE('Centrocampistas: ' || v_centrocampistas);
 DBMS_OUTPUT.PUT_LINE('Delanteros: ' || v_delanteros);

EXCEPTION
  WHEN NO_DATA_FOUND THEN
 DBMS_OUTPUT.PUT_LINE('Equipo no encontrado');
 WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Error al obtener estadísticas: ' || SQLERRM);
```

```
END;
```

Procedimiento para limpiar datos antiguos

```
CREATE OR REPLACE PROCEDURE LimpiarDatosAntiguos(
p_fecha_limite IN DATE
) AS
  v_partidos_eliminados NUMBER := 0;
 v_goles_eliminados NUMBER := 0;
BEGIN
```

Eliminar goles de partidos antiguos

```
DELETE FROM Gol
WHERE codigo_partido IN (
SELECT codigo_partido
 FROM Partido
 WHERE fecha < p_fecha_limite
 );
 v_goles_eliminados := SQL%ROWCOUNT;
```

Eliminar partidos antiguos

```
DELETE FROM Partido
 WHERE fecha < p_fecha_limite;
 v_partidos_eliminados := SQL%ROWCOUNT;

COMMIT;
DBMS_OUTPUT.PUT_LINE('Limpieza completada:');
```

```
DBMS_OUTPUT.PUT_LINE('- Partidos eliminados: ' || v_partidos_eliminados);
DBMS_OUTPUT.PUT_LINE('- Goles eliminados: ' || v_goles_eliminados);


EXCEPTION
WHEN OTHERS THEN
ROLLBACK;
DBMS_OUTPUT.PUT_LINE('Error en limpieza: ' || SQLERRM);
END;
```

2. Triggers

Trigger para validar la edad del jugador (mínimo 16 años)

```
CREATE OR REPLACE TRIGGER TR_ValidarEdadJugador
BEFORE INSERT OR UPDATE ON Jugador
FOR EACH ROW
DECLARE
   v_edad NUMBER;
BEGIN
   v_edad := EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM
:NEW.fecha_nacimiento);

   IF v_edad < 16 THEN
      RAISE_APPLICATION_ERROR(-20001, 'El jugador debe tener al menos 16
años');
   END IF;
END;
```

Trigger para validar que un equipo no juegue contra sí mismo

```
CREATE OR REPLACE TRIGGER TR_ValidarEquiposPartido
BEFORE INSERT OR UPDATE ON Partido
FOR EACH ROW
BEGIN
   IF :NEW.codigo_equipo_casa = :NEW.codigo_equipo_fuera THEN
      RAISE_APPLICATION_ERROR(-20002, 'Un equipo no puede jugar contra sí
mismo');
   END IF;
END;
```

Trigger para auditar cambios en la tabla Equipo

```
CREATE TABLE AuditoriaEquipo (
   id_auditoria NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY
KEY,
   codigo_equipo NUMBER,
   accion VARCHAR2(10),
   usuario VARCHAR2(50),
   fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   datos_anteriores VARCHAR2(1000)
);

CREATE OR REPLACE TRIGGER TR_AuditoriaEquipo
AFTER INSERT OR UPDATE OR DELETE ON Equipo
FOR EACH ROW
BEGIN
   IF INSERTING THEN
      INSERT INTO AuditoriaEquipo (codigo_equipo, accion, usuario,
datos_anteriores)
      VALUES (:NEW.codigo_equipo, 'INSERT', USER,
           'Nuevo equipo: ' || :NEW.nombre);
   ELSIF UPDATING THEN
      INSERT INTO AuditoriaEquipo (codigo_equipo, accion, usuario,
datos_anteriores)
      VALUES (:OLD.codigo_equipo, 'UPDATE', USER,
           'Cambios: ' || :OLD.nombre || ' -> ' || :NEW.nombre);
   ELSIF DELETING THEN
```

```
        INSERT INTO AuditoriaEquipo (codigo_equipo, accion, usuario,
datos_anteriores)
        VALUES (:OLD.codigo_equipo, 'DELETE', USER,
            'Eliminado: ' || :OLD.nombre);
    END IF;
END;
```

Trigger para mantener consistencia en goles

```
CREATE OR REPLACE TRIGGER TR_ActualizarGolesPartido
AFTER INSERT OR UPDATE OR DELETE ON Gol
DECLARE
    CURSOR c_partidos IS
        SELECT DISTINCT codigo_partido FROM Gol;
BEGIN
    FOR rec IN c_partidos LOOP
        UPDATE Partido p
        SET goles_casa = (
            SELECT COUNT(*)
            FROM Gol g
            JOIN Jugador j ON g.codigo_jugador = j.codigo_jugador
            WHERE g.codigo_partido = p.codigo_partido
            AND j.codigo_equipo = p.codigo_equipo_casa
        ),
        goles_fuera = (
            SELECT COUNT(*)
            FROM Gol g
            JOIN Jugador j ON g.codigo_jugador = j.codigo_jugador
            WHERE g.codigo_partido = p.codigo_partido
            AND j.codigo_equipo = p.codigo_equipo_fuera
        )
        WHERE p.codigo_partido = rec.codigo_partido;
    END LOOP;
END;
```

Trigger para validar que un presidente tenga al menos 30 años

```
CREATE OR REPLACE TRIGGER TR_ValidarEdadPresidente
BEFORE INSERT OR UPDATE ON Presidente
FOR EACH ROW
DECLARE
   v_edad NUMBER;
BEGIN
   v_edad := EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM
:NEW.fecha_nacimiento);

   IF v_edad < 30 THEN
      RAISE_APPLICATION_ERROR(-20003, 'El presidente debe tener al menos
30 años');
   END IF;
END;
```

Trigger para evitar duplicados de correos

```
CREATE OR REPLACE TRIGGER TR_ValidarCorreoUnico
BEFORE INSERT ON CorreoJugador
FOR EACH ROW
DECLARE
   v_count NUMBER;
BEGIN
   SELECT COUNT(*) INTO v_count
   FROM CorreoJugador
   WHERE correo = :NEW.correo;

   IF v_count > 0 THEN
      RAISE_APPLICATION_ERROR(-20004, 'El correo electrónico ya está
registrado');
   END IF;
END;
```

3. Funciones Útiles

Función para calcular la edad de una persona

```
CREATE OR REPLACE FUNCTION CalcularEdad(
    p_fecha_nacimiento IN DATE
) RETURN NUMBER AS
    v_edad NUMBER;
BEGIN
    v_edad := EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM
p_fecha_nacimiento);
    RETURN v_edad;
END;
```

Función para obtener el goleador de un equipo

```
CREATE OR REPLACE FUNCTION ObtenerGoleadorEquipo(
    p_codigo_equipo IN NUMBER
) RETURN VARCHAR2 AS
    v_goleador VARCHAR2(100);
    v_goles NUMBER;
BEGIN
    SELECT j.nombre1 || ' ' || j.apellido1, COUNT(*)
    INTO v_goleador, v_goles
    FROM Jugador j
    JOIN Gol g ON j.codigo_jugador = g.codigo_jugador
    WHERE j.codigo_equipo = p_codigo_equipo
```

```
    GROUP BY j.nombre1, j.apellido1
    ORDER BY COUNT(*) DESC
    FETCH FIRST 1 ROW ONLY;

    RETURN v_goleador || ' (' || v_goles || ' goles)';
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN 'Sin goles registrados';
    WHEN OTHERS THEN
        RETURN 'Error al calcular';
END;
```

Función para verificar aforo del estadio

```
CREATE OR REPLACE FUNCTION VerificarAforo(
    p_codigo_equipo IN NUMBER,
    p_espectadores IN NUMBER
) RETURN VARCHAR2 AS
    v_aforo_maximo NUMBER;
BEGIN
    SELECT aforo INTO v_aforo_maximo
    FROM Equipo
    WHERE codigo_equipo = p_codigo_equipo;

    IF p_espectadores > v_aforo_maximo THEN
        RETURN 'EXCEDIDO - Aforo máximo: ' || v_aforo_maximo;
```

```
    ELSE
        RETURN 'DENTRO DEL LÍMITE - Capacidad: ' ||
            ROUND((p_espectadores / v_aforo_maximo) * 100, 1) || '%';
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN 'Equipo no encontrado';
END;
```