SIGET - Sistema de Gestión de Torneos Deportivos

Modelos y Documentación del Software
Docente: Ing. Hely Suarez Marín
Integrantes:

Nelly Fabiola Cano Oviedo
Néstor Iván Granados Valenzuela
Fecha: Octubre / 2025

Problema

Los torneos deportivos académicos y aficionados suelen gestionar su información con hojas de cálculo, formularios dispersos o mensajería instantánea. A medida que crece el número de equipos, jugadores y partidos, también aumenta la **complejidad operativa**: programar encuentros sin choques de horario, asignar árbitros disponibles, registrar resultados de forma confiable y consultar estadísticas básicas.
Este manejo **manual y descentralizado** provoca errores de transcripción (marcadores mal digitados, equipos repetidos), **inconsistencia de datos** (jugadores duplicados, árbitros inhabilitados asignados por error) y **poca trazabilidad** (no queda claro quién cambió un resultado ni cuándo). La ausencia de un **modelo único** limita la generación de estadísticas confiables (partidos jugados, victorias, empates, derrotas) y dificulta la rendición de cuentas. En consecuencia, se afectan la **transparencia** del torneo y la **experiencia** de participantes y organizadores.

Solución

Se propone una **plataforma web simple y centralizada** de Gestión de Torneos Deportivos que estandarice:

1. **Registro de equipos y jugadores**,
2. **Programación de partidos** (fecha/hora/venue),
3. **Asignación de árbitros** disponibles,
4. **Registro de resultados**, y
5. **Estadísticas básicas** por equipo.
   El acceso se realiza con roles mínimos (Administrador, Árbitro, Espectador) para **delimitar responsabilidades** y reducir errores. El sistema define **reglas de negocio** que evitan inconsistencias (por ejemplo, no se puede programar un partido con el mismo equipo como local y visitante, o cerrar un partido played con goles negativos). Toda la información queda persistida en una **BD relacional** con **integridad referencial**.

Impacto

1. **Eficiencia y trazabilidad:** Disminuye tiempos de registro y elimina duplicidades. Cada cambio queda asociado a un usuario/fecha, reforzando el control.
2. **Transparencia del torneo:** Estadísticas generadas a partir de resultados **verificables** en la BD; se reducen conflictos por errores humanos.
3. **Escalabilidad académica:** La documentación (UML + BD normalizada) deja una base sólida para futuras extensiones (múltiples torneos, fases, sanciones, tablas de posiciones).

1. Resumen del Problema

El manejo manual y distribuido de información en torneos deportivos provoca **inconsistencias**, **pérdida de trazabilidad** y **dificulta estadísticas confiables**. Esto impacta la calidad organizativa y la experiencia de equipos y árbitros.

2. Posible solución (plataforma mínima de torneo)

Se propone un **MVP** que cubra el ciclo esencial: inscripción de equipos/jugadores → programación y arbitraje → registro de resultados → estadísticas. La solución se documenta completamente (UML + BD) para una implementación posterior.

Alcance (MVP)

- Registrar **Equipos** (nombre, ciudad/entrenador) y **Jugadores** (básicos y pertenencia a un equipo).
- Registrar **Árbitros** (licencia, contacto, estado activo).
- **Programar partidos** con fecha y hora, asegurando que **local ≠ visitante**.
- **Asignar árbitro** disponible.
- **Registrar resultados** (goles local/visitante) y marcar estado del partido como played.
- Generar **estadísticas básicas**: jugados, ganados, empatados y perdidos por equipo.

Actores y Funciones

Actor	Funciones principales
Administrador	Gestiona equipos, jugadores y árbitros; programa partidos; registra resultados; consultas estadísticas.
Árbitro	(Opcional) Confirma validez del resultado o reporta incidentes.
Espectador	Consultas estadísticas y resultados publicados.

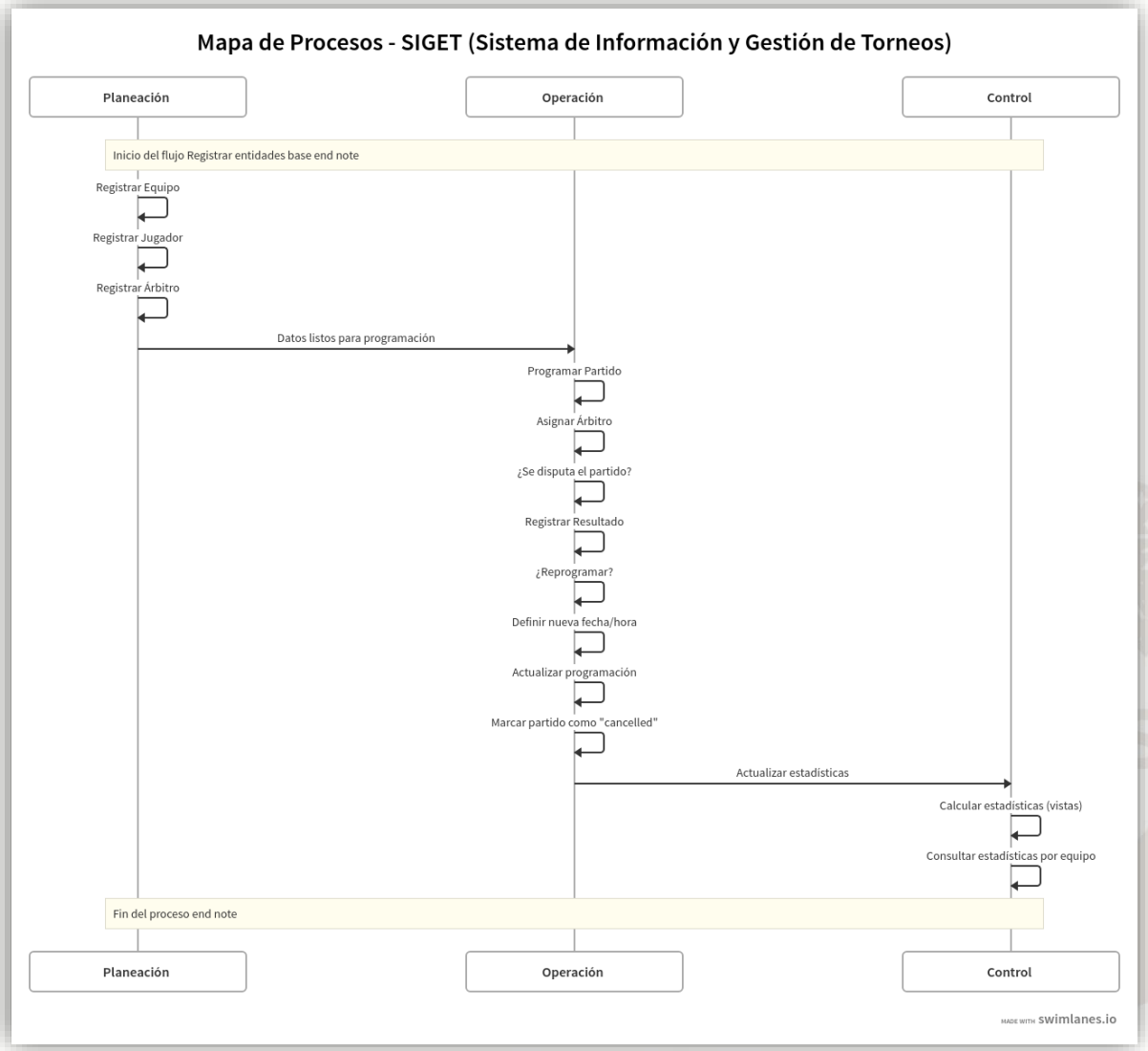
Reglas clave del sistema

- Un partido **no puede** tener el mismo equipo como local y visitante.

- Un partido **played** debe registrar **goles ≥ 0**.
- Un árbitro **inactivo** no puede asignarse.
- Las **estadísticas** se **derivan** de los partidos; no se almacenan acumulados (evita inconsistencias).

3. Descripción general del proceso

El organizador registra equipos/jugadores y árbitros. Luego programa cada partido, asigna un árbitro y, al finalizar el encuentro, registra el resultado. Las estadísticas se consultan en cualquier momento con base en los resultados **ya cerrados** (played).



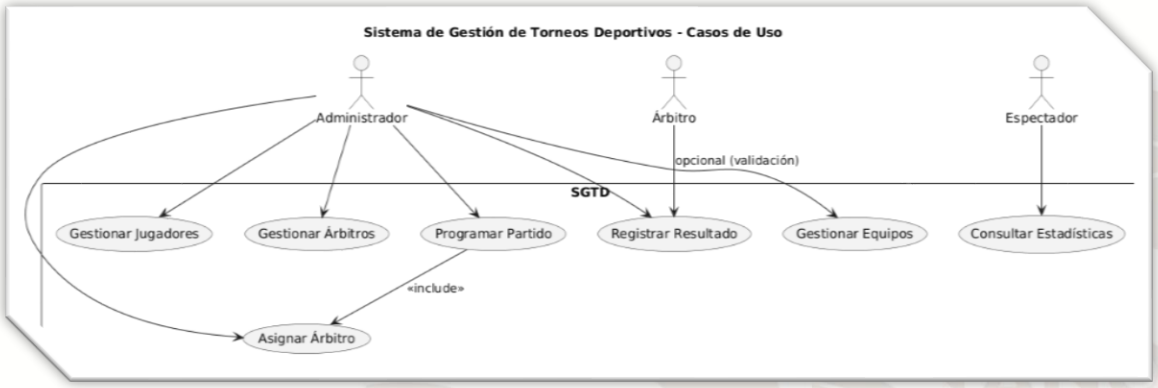
4. Revisión / Verificabilidad

Se contrastaron flujos básicos con prácticas reales: planillas de papel o mensajes dispersos suelen generar **conflictos** por falta de control de versiones y responsabilidades. El modelo propuesto corrige esto mediante **integridad referencial**, **reglas de negocio** y **vistas** para estadísticas.

5. Diagramas UML realizados

5.1 Casos de Uso

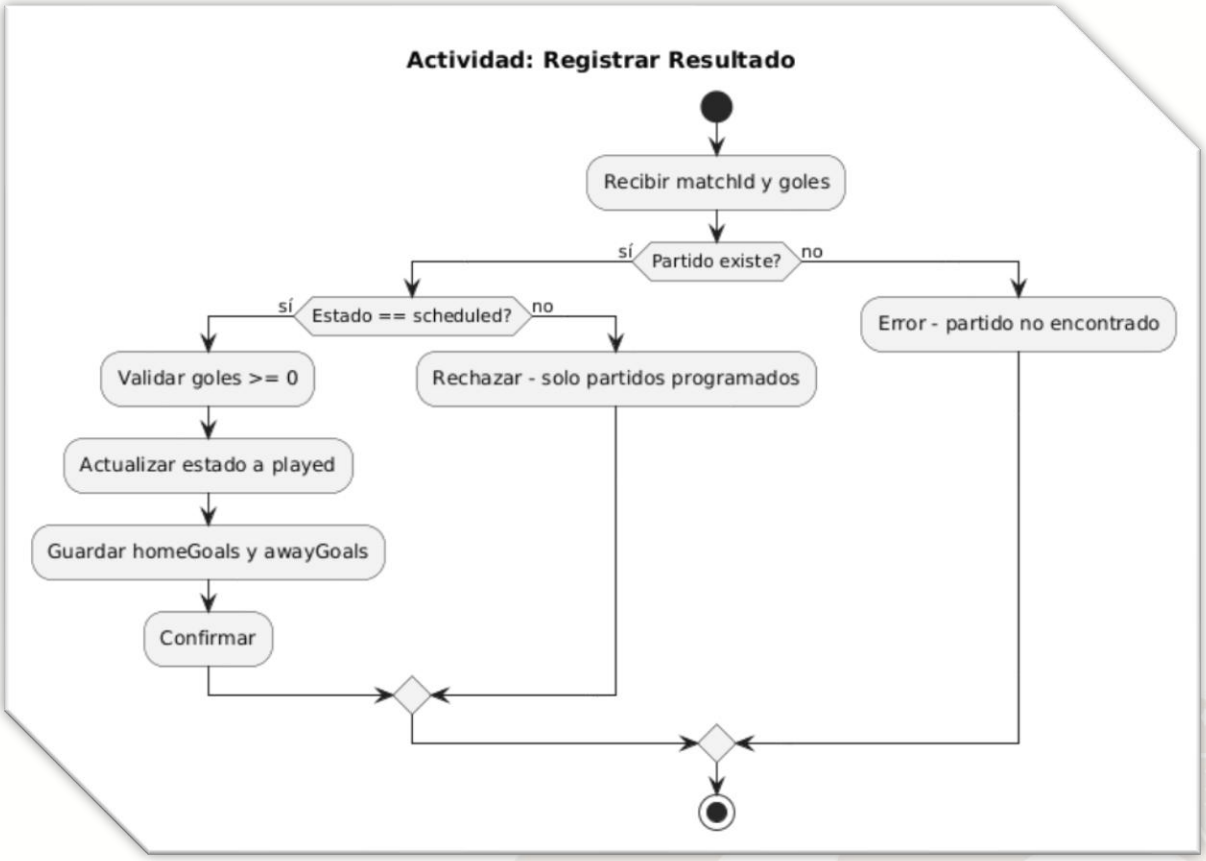
- **Propósito.** Mostrar, desde la perspectiva del usuario, **qué** funcionalidades ofrece el sistema, sin entrar en detalles técnicos.
- **Qué representa aquí.** Tres actores: **Administrador** (gestión y operación), **Árbitro** (opcional para validación/observaciones) y **Espectador** (consulta). Casos clave: Gestionar Equipos, Gestionar Jugadores, Gestionar Árbitros, Programar Partido, Asignar Árbitro, Registrar Resultado y Consultar Estadísticas. Programar Partido **incluye** Asignar Árbitro.
- **Cómo leerlo.** Las líneas conectan actores con casos que pueden ejecutar; estereotipos include/extend clarifican dependencias.
- **Razón de diseño.** Enfocado en un **MVP** claro para calificar académicamente; evita casos fuera de alcance (inscripciones online, sanciones).
- **Relación con BD / reglas.** Cada caso justifica entidades y restricciones: Registrar Resultado exige status='played' y home_goals/away_goals ≥ 0.



5.2 Actividades (Registrar Resultado)

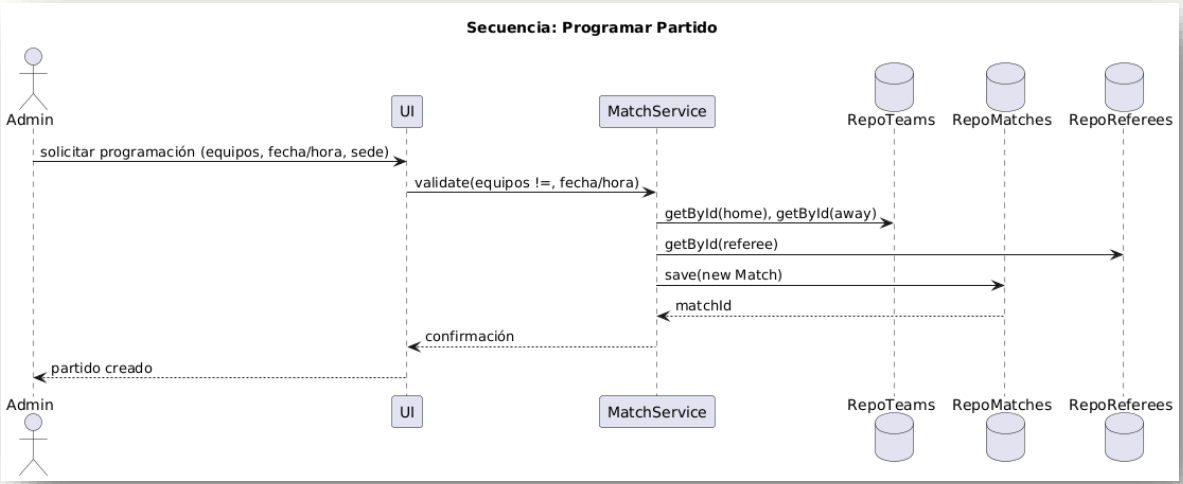
- **Propósito.** Describir el **flujo operativo** paso a paso con decisiones explícitas.
- **Qué representa.** Inicia con matchId y goles; verifica existencia del partido y que su estado sea **scheduled**; valida goles ≥ 0; actualiza a **played** y guarda marcadores. Si el partido no existe o ya no está en scheduled, se rechaza.
- **Cómo leerlo.** Actividades (rectángulos), decisiones (rombos), flujo (flechas). Los bucles y finales están claramente definidos.

- **Razón de diseño.** Poner foco en la **validación previa** al cierre del partido para garantizar consistencia del torneo.
- **Relación con BD / reglas.** Cambia matches.status y home_goals/away_goals; el CHECK evita negativos, la app fuerza completar goles para played.



5.3 Secuencia (Programar Partido)

- **Propósito.** Representar el **orden temporal** de los mensajes entre UI, Servicio y Repositorios al **programar** un encuentro.
- **Qué representa.** La UI envía solicitud con equipos/fecha/hora/árbitro; el Servicio valida que local ≠ visitante, consulta existencia de equipos y árbitro, y persiste el partido.
- **Cómo leerlo.** Mensajes verticales en el tiempo; llamadas a repositorios claramente separadas.
- **Razón de diseño.** Mantener **coherencia** entre validaciones y persistencia; menor acoplamiento (Servicios vs Repos).
- **Relación con BD / reglas.** Inserta en matches, aplica la regla home_team_id <> away_team_id.



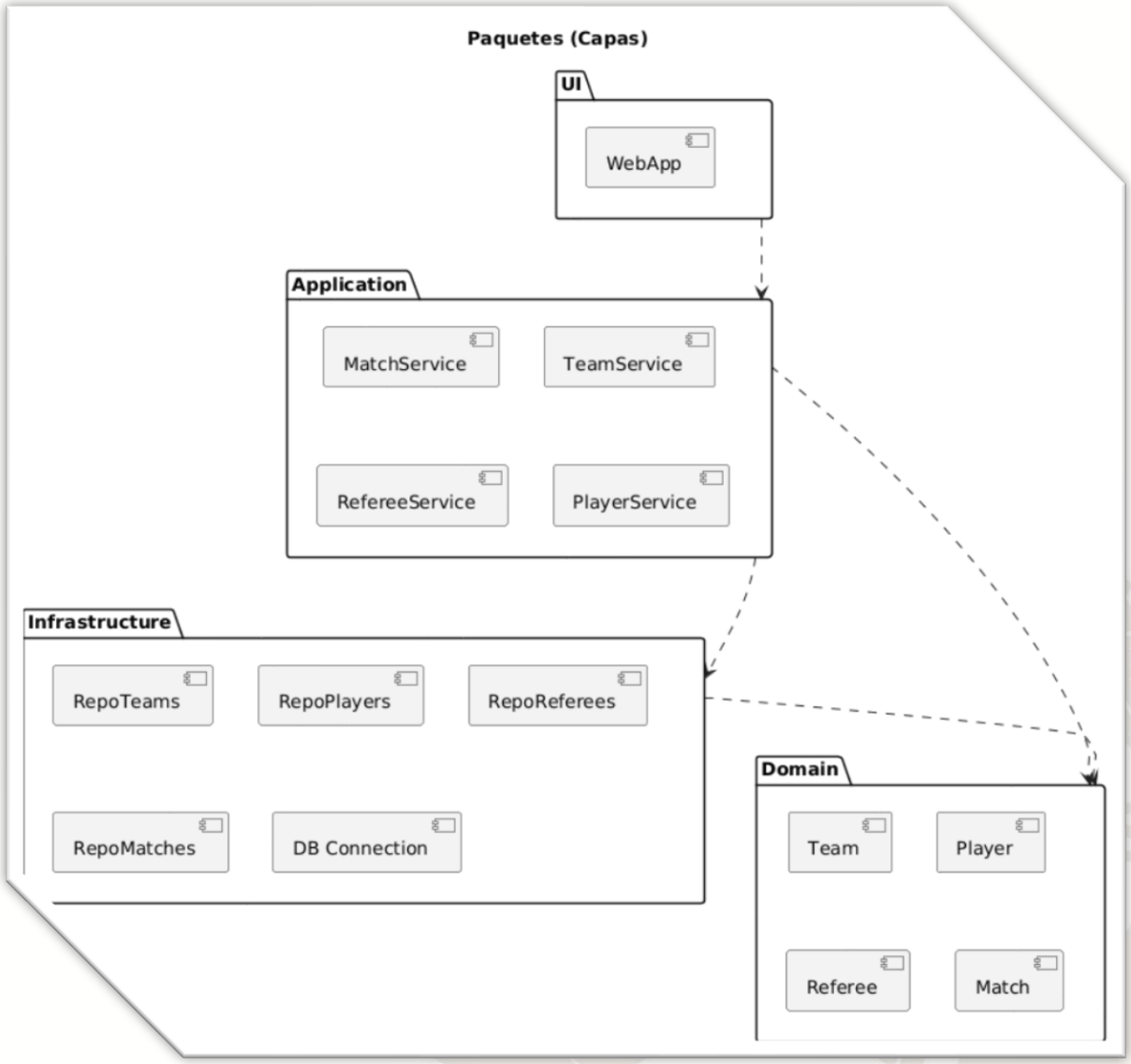
5.4 Comunicación (Registrar Resultado)

- **Propósito.** Mostrar la **colaboración** entre objetos con **numeración** de mensajes, equivalente a la secuencia pero desde la **topología** de relaciones.
- **Qué representa.** El Admin entrega los goles en la UI → Servicio valida → Repositorio consulta/actualiza el partido.
- **Cómo leerlo.** Se siguen los números de mensajes (1, 2, 3…) para reconstruir el hilo de interacción.
- **Razón de diseño.** Útil en auditorías o informes impresos donde la **red de colaboración** es más informativa que el eje temporal.
- **Relación con BD / reglas.** Cambios en matches y transición scheduled → played.

5.5 Paquetes

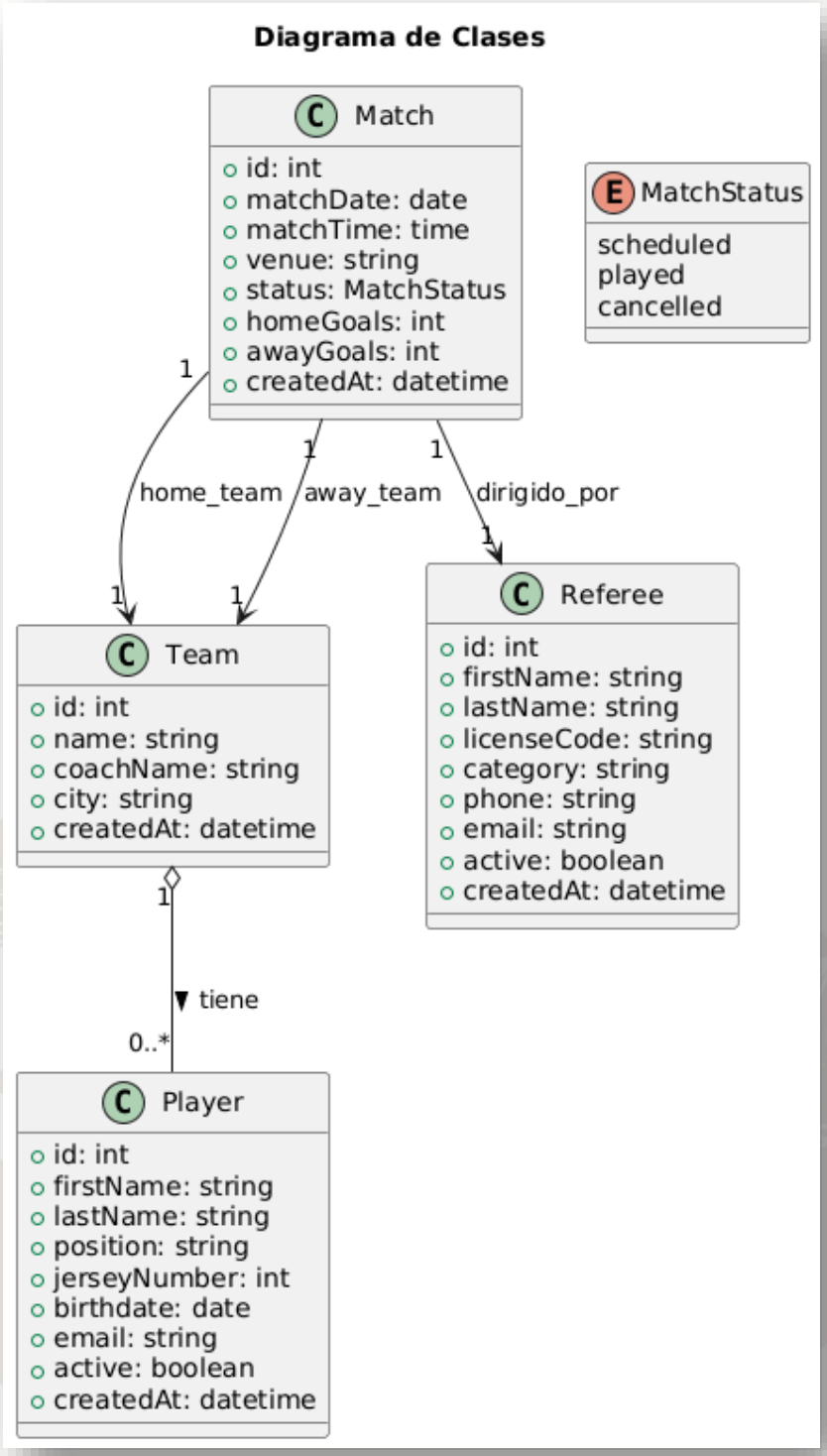
- **Propósito.** Organizar el sistema en **capas/modularidad**: UI, Application, Domain, Infrastructure.
- **Qué representa.** UI (formulario/tablas), Servicios de aplicación (validaciones y orquestación), Entidades de dominio (Team, Player, Match, Referee) y Repositorios/Conexión a BD.

- **Cómo leerlo.** Dependencias dirigidas: UI **usa** Application; Application **usa** Domain e **Infrastructure**.
- **Razón de diseño.** Mejora **mantenibilidad** y favorece pruebas unitarias al aislar lógica de acceso a datos.
- **Relación con BD / reglas.** Infrastructure encapsula DAOs/ORM; Domain expresa reglas como **invariantes** (equipos distintos).



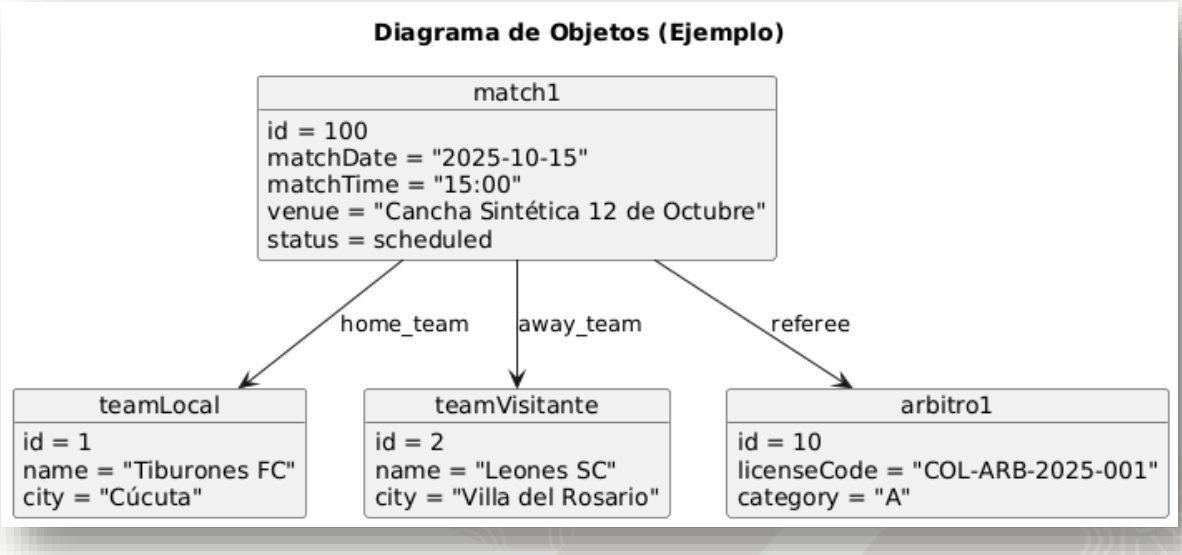
5.6 Clases

- **Propósito.** Describir la **estructura estática** del dominio con atributos y asociaciones cardinalizadas.
- **Qué representa.** Team (1) — (0..*) Player; Match → Team (local/visitante) y → Referee; MatchStatus como enumeración.
- **Cómo leerlo.** Atributos centrales (name, email, status), relaciones y multiplicidades.
- **Razón de diseño.** Mantener el modelo **mínimo** y **coherente** con el relacional.
- **Relación con BD / reglas.** Traduce a tablas teams, players, referees, matches y match_status.



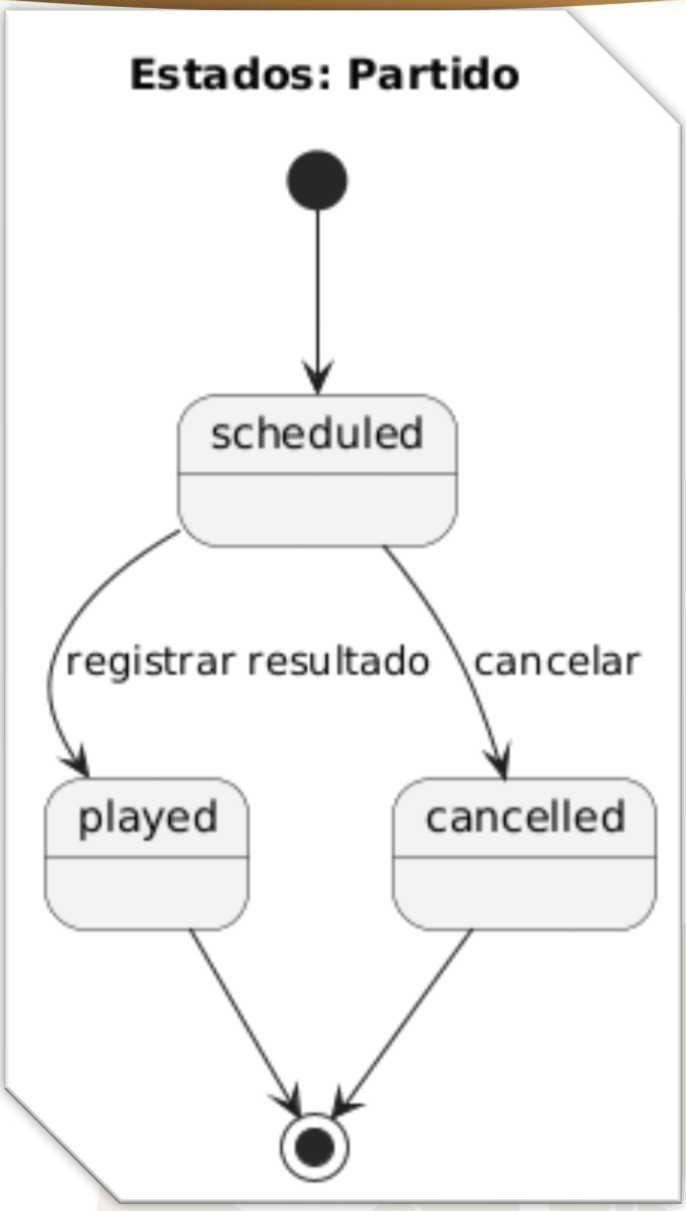
5.7 Objetos

- **Propósito.** Validar el modelo con **instancias concretas** (datos de ejemplo).
- **Qué representa.** Un Match scheduled entre Tiburones FC y Leones SC con Referee asignado, previo a registrar el resultado.
- **Cómo leerlo.** Objetos con valores y referencias explícitas (home_team/away_team/referee).
- **Razón de diseño.** Comprueba cardinalidades y coherencia antes de poblar la BD.
- **Relación con BD / reglas.** Refleja un registro real en matches y sus FKs.



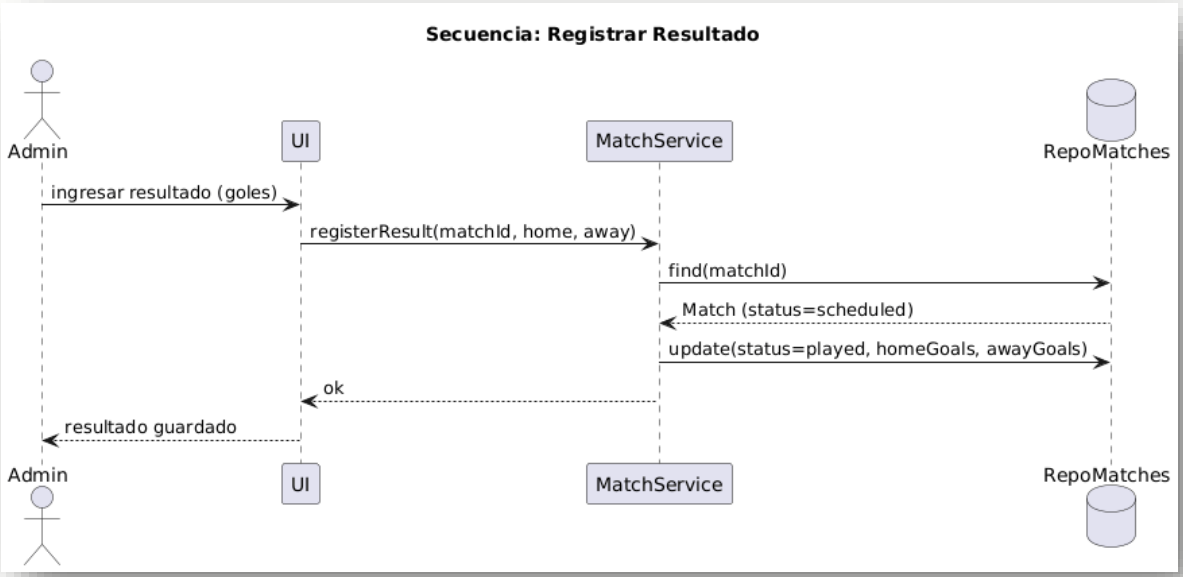
5.8 Estados (Partido)

- **Propósito.** Modelar el **ciclo de vida** del partido.
- **Qué representa.** Estados scheduled, played y cancelled con transiciones válidas.
- **Cómo leerlo.** Flechas con eventos (registrar resultado → played; cancelar → cancelled).
- **Razón de diseño.** Evitar estados ambiguos y reforzar **consistencia operacional**.
- **Relación con BD / reglas.** Persistido en matches.status (ENUM/CHK).



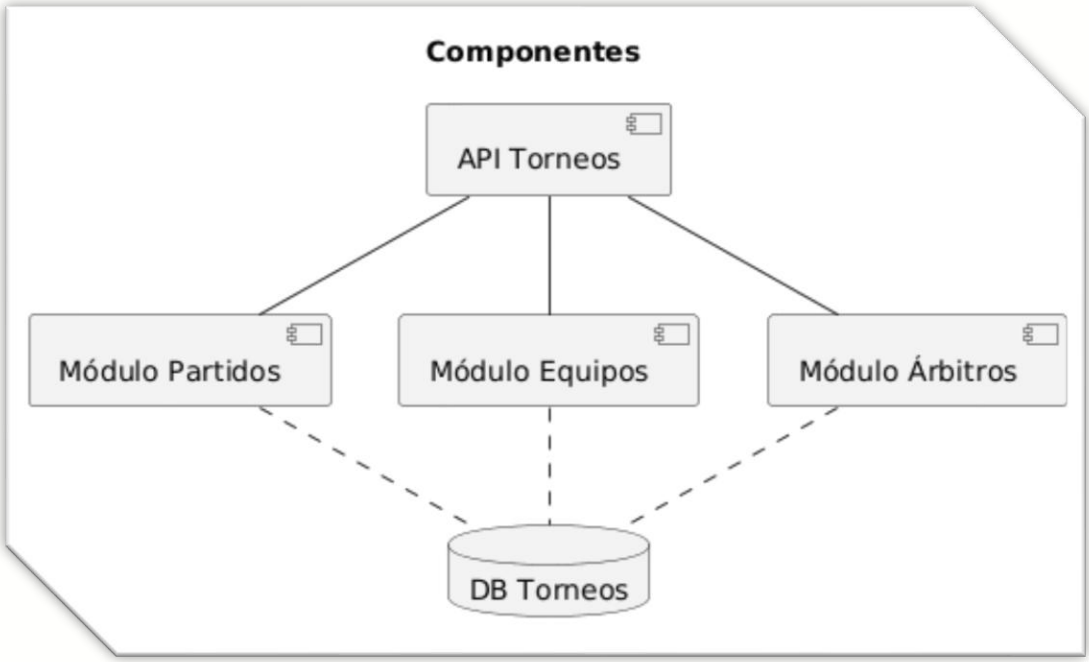
5.9 Secuencia (Registrar Resultado)

- **Propósito.** Registrar el marcador asegurando precondiciones.
- **Qué representa.** UI → Servicio registerResult() → Repo find/update.
- **Cómo leerlo.** Incluye validación de estado y actualización atómica.
- **Relación con BD / reglas.** Cambios en matches y recalculo de vistas estadísticas.



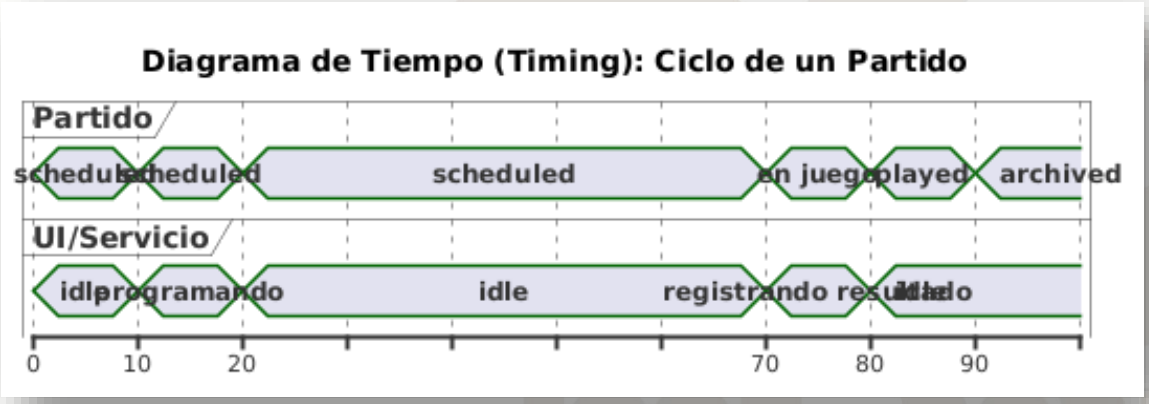
5.10 Componentes

- **Propósito.** Exponer la solución en **módulos desplegables**: API Torneos, Módulo Partidos, Módulo Equipos, Módulo Árbitros y BD.
- **Qué representa.** Dependencias de los módulos con la BD; API como fachada.
- **Cómo leerlo.** Cajas = componentes; enlaces = dependencias.
- **Relación con BD / reglas.** Cada componente mapea entidades/servicios a tablas.



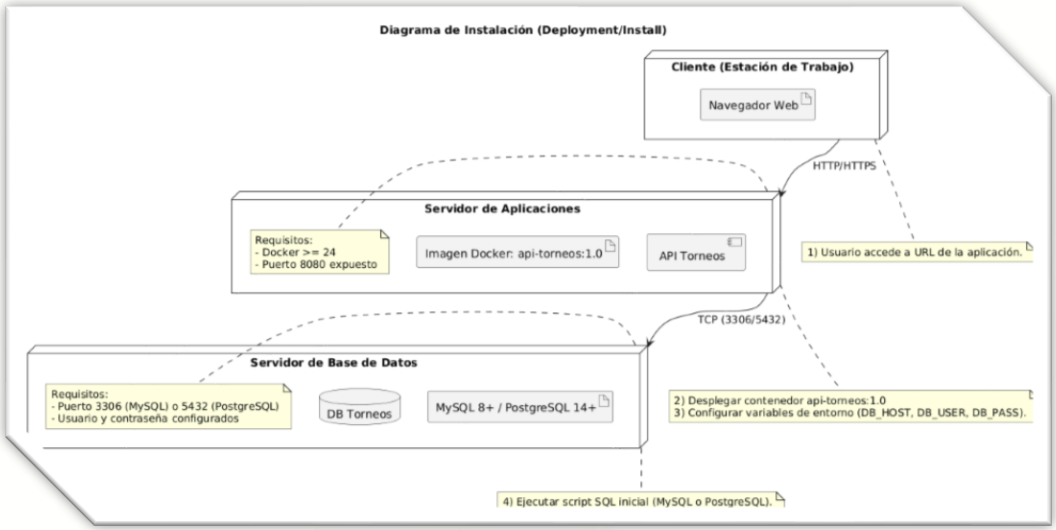
5.11 Tiempo (Timing)

- **Propósito.** Resumir **temporalmente** el ciclo de un partido y la interacción con la UI/Servicio.
- **Qué representa.** Transiciones de scheduled a played en la línea de tiempo, con hitos (programación, registro de resultado).
- **Cómo leerlo.** Ejes concisos por participante/estado.
- **Razón de diseño.** Identificar esperas y momentos críticos (ej. Registro tardío de resultados).



5.12 Instalación (Deployment/Install)

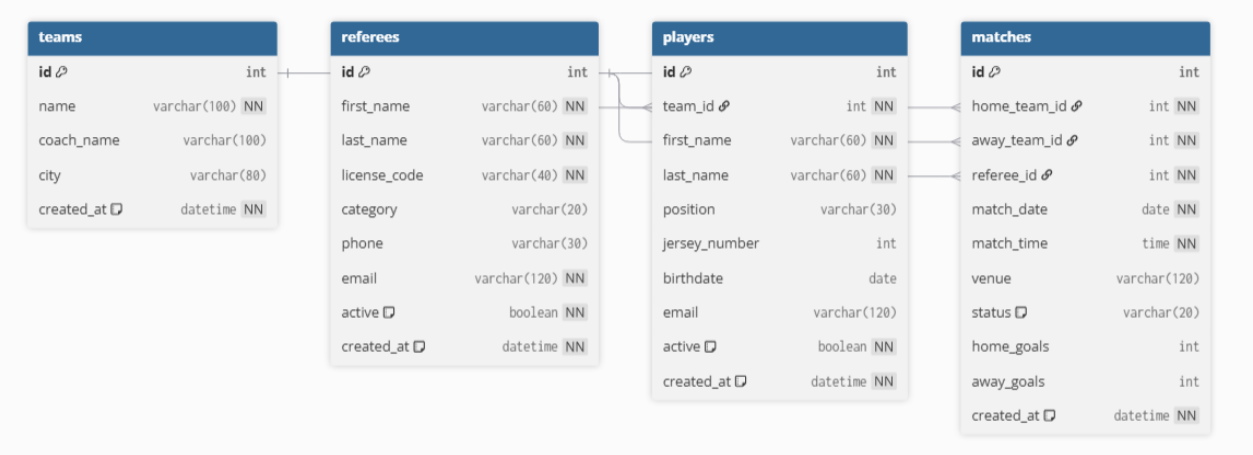
- **Propósito.** Mostrar **nodos físicos/lógicos** y **pasos mínimos** de instalación local.
- **Qué representa.** Cliente (navegador), Servidor de Aplicaciones (API/Servicios) y Servidor de BD (MySQL/PostgreSQL) con puertos y rerrequisitos (variables de entorno).
- **Cómo leerlo.** Nodos conectados por HTTP/HTTPS y TCP (3306/5432).
- **Relación con BD / reglas.** Alinea los scripts y la configuración de conexión.



6. Diagramas de Base de Datos (no UML – PlantUML ER/IE)

6.1 Modelo Entidad–Relación (ER)

Entidades: **teams**, **players**, **referees**, **matches**.
Relaciones: Team 1—N Player; Match N—1 Referee; Match → Team (local/visitante distintos).



6.2 Modelo Relacional

Traducción del ER a tablas con PK/FK/CHK/Índices:

- **teams**(id PK, name UNIQUE, coach_name, city, created_at)
- **players**(id PK, team_id FK→teams, first_name, last_name, position, jersey_number, birthdate, email, active, created_at)
- **referees**(id PK, first_name, last_name, license_code UNIQUE, category, phone, email UNIQUE, active, created_at)
- **matches**(id PK, home_team_id FK→teams, away_team_id FK→teams, referee_id FK→referees, match_date, match_time, venue, status, home_goals, away_goals, created_at; CHECK local≠visitante)

6.3 Diccionario de Datos (visual compacto)

Tarjetas por tabla con tipos, UNIQUE, NOT NULL, DEFAULT y FKs explícitas.

7. Normalización (hasta 3FN)

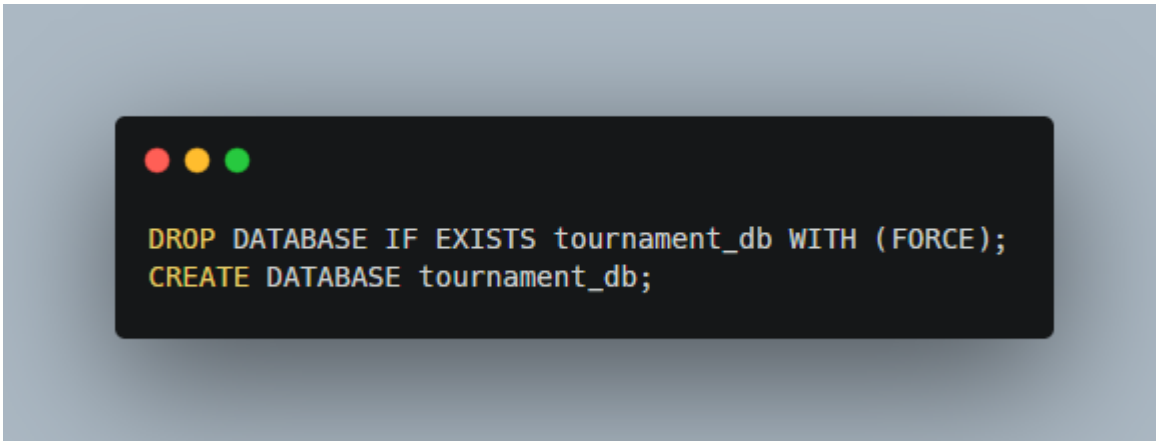
- **1FN:** todos los atributos son atómicos.
- **2FN:** PK simples (id) → no hay dependencias parciales.
- **3FN:** no hay dependencias transitivas entre atributos no clave; **estadísticas** se **derivan** desde matches (no se almacenan acumulados).

8. Vistas SQL

- **vw_team_matches:** une teams↔matches e identifica rol (home/away).
- **vw_team_stats:** agrega jugados/ganados/empates/perdidos a partir de partidos played.
  Nota: Las vistas **no** tienen FK; su dependencia con tablas base puede mostrarse con **líneas punteadas** en diagramas.

9. Guía breve de ejecución en PostgreSQL (DBeaver)

1. Conéctate a la base **postgres** y ejecuta:



```
DROP DATABASE IF EXISTS tournament_db WITH (FORCE);
CREATE DATABASE tournament_db;
```

2. Cambia el editor a **tournament_db** y ejecuta el resto del script (**sin \c**): creación de tablas, tipo match_status, índices y vistas.