Universidad Privada Boliviana

Proyecto Final Base de Datos Avanzada

Casa de Apuestas

Andres Sanchez - 69017

Adrian Sanchez - 69546

Alexia Marin - 60855

Docente: Paul Landaeta

La Paz – Bolivia – Junio de 2025

Resumen

Este proyecto nace como respuesta a una necesidad de un sistema para gestionar apuestas deportivas de los Juegos Olímpicos. Con este objetivo, se combinaron tecnologías relacionales, con PostgreSQL y MongoDB, lo que permitió manejar tanto las transacciones como la parte documental del sistema. Se desarrollaron funciones, procedimientos y triggers en PL/pgSQL, respetando los principios de integridad, es decir, atomicidad, consistencia, aislamiento y durabilidad (ACID). Para mejorar el rendimiento, se incorporó Redis como sistema de caché. El despliegue completo se realizó utilizando Docker, y finalmente en MongoDB, se optimizaron las consultas mediante operadores de agregación como $lookup, $unwind y $project, lo que permitió de forma ordenada acceder a la información.

1.-Introducción

Las plataformas de apuestas crearon la necesidad de contar con sistemas robustos y seguros que manejen grandes volúmenes de datos y transacciones en tiempo real para los usuarios.Este proyecto se enfoca en esa realidad, con el objetivo de ofrecer una solución que no solo cumpla con lo técnico, sino que también respete los principios de orden, claridad y eficiencia. A través de un enfoque distribuido, se plantea un sistema que combina lo mejor de las bases de datos relacionales y documentales, y que está preparado para responder con solidez en escenarios de alta demanda.

2.-Objetivos

2.1 Objetivo General

Diseñar e implementar un sistema de base de datos híbrido para la gestión eficiente de apuestas deportivas(Olimpiadas), asegurando la integridad, rendimiento y escalabilidad del sistema.

2.2 Objetivos Específicos

- Diseñar una base de datos relacional en PostgreSQL para gestionar usuarios, eventos y apuestas.
- Crear funciones y procedimientos almacenados que validen reglas del negocio y mantengan integridad.
- Implementar consultas optimizadas con EXPLAIN ANALYZE.
- Utilizar MongoDB como base documental con colecciones embebidas y referenciales.
- Integrar Redis como sistema de cache para consultas críticas.
- Emplear Docker para configurar entornos distribuidos.

3.-Requisitos

3.1 Requisitos Funcionales

Código	RF-01	**Tipo**	Funcional
Nombre	**Registro de eventos**		
Descripción	El sistema debe permitir que los usuarios dmin puedan crear nuevos eventos, colocando la información correspondiente.		

Código	RF-02	**Tipo**	Funcional
Nombre	**Auenticación de usuarios**		
Descripción	Cada usuario podra registrarse e iniciar sesión con su correo y contraseña para acceder a sus datos personales y realizar apuestas.		

Código	RF-03	**Tipo**	Funcional
Nombre	**Visualización de eventos**		
Descripción	Se deben poder visualizar los eventos disponibles,con sus mercados y cuotas.		

Código	RF-04	**Tipo**	Funcional
Nombre	**Registro de apuestas**		
Descripción	El sistema debe permitir que un usuario realice una apuesta en un mercado activo, teniendo en cuenta el saldo y registra la apuesta.		

Código	RF-05	**Tipo**	Funcional
Nombre	**Cancelación de Apuestas**		

Descripción	El sistema puede permitir que el usuario cancele su apuesta y tenga de vuelta el monto apostado.

Código	RF-06	**Tipo**	Funcional
Nombre	**Gestion de mercados deportivos**		
Descripción	El sistema debe poder crear y administrar los mercados de cada evento,con las cuotas para las apuestas entre otras opciones.		

Código	RF-07	**Tipo**	Funcional
Nombre	**Visualización de historial**		
Descripción	El sistema permite ver su historial de eventos apostados, apuestas ganadas y perdidas, etc.		

Código	RF-08	**Tipo**	Funcional
Nombre	**Registro y finalización de eventos**		
Descripción	En el sistema los administradores podrán finalizar un evento cuando termine, ingresar los resultados, asi poder procesar automáticamente las apuestas del evento.		

3.2 Requisitos No Funcionales

Código	RNF-08	**Tipo**	No Funcional
Nombre	**Encriptación de contraseñas**		
Descripción	Las contraseñas se guardan de forma segura, ya que el sistema realiza un cifrado de estas.		

Código	RNF-09	**Tipo**	No Funcional
Nombre	**Optimización de consultas**		
Descripción	La optimización tanto en PostgreSQL como en MongoDB, hacen que puedan funcionar de forma rápida.		

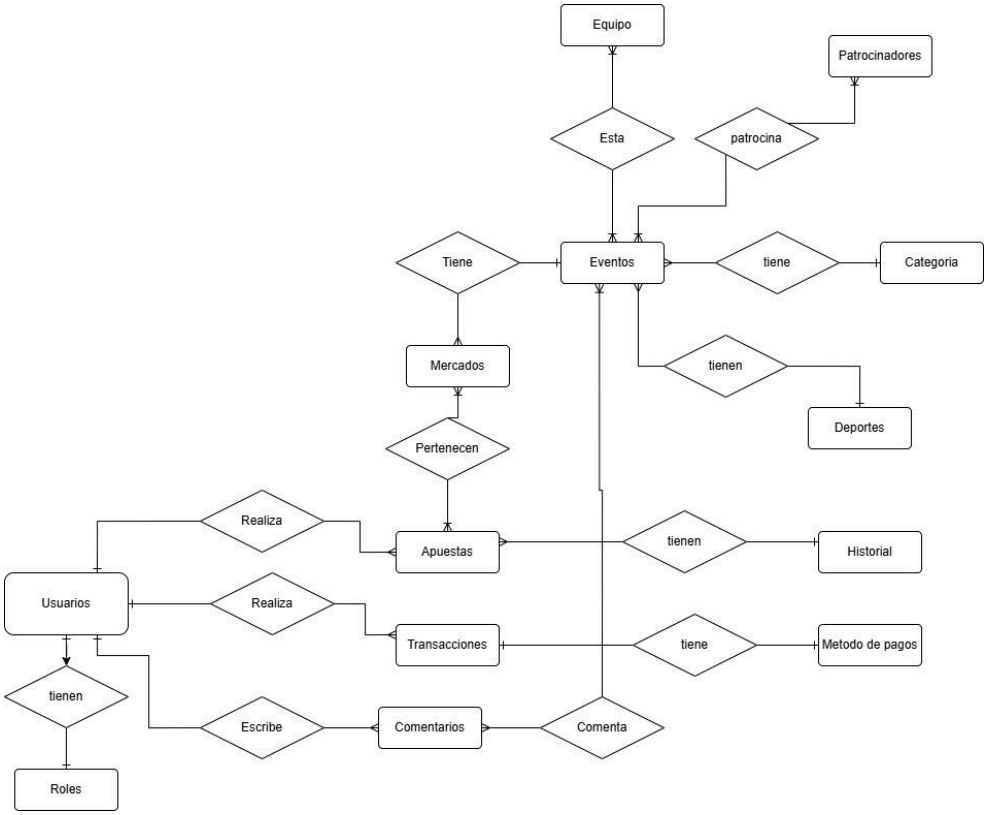
Código	RNF-10	**Tipo**	No Funcional
Nombre	**Nombres claros en las funciones y procedimientos**		
Descripción	El sistema mantiene nombres claros y fáciles de entender para poder seguir trabajando de forma colaborativa.		

4.- Diseño Conceptual

4.1 Modelo Entidad-Relación

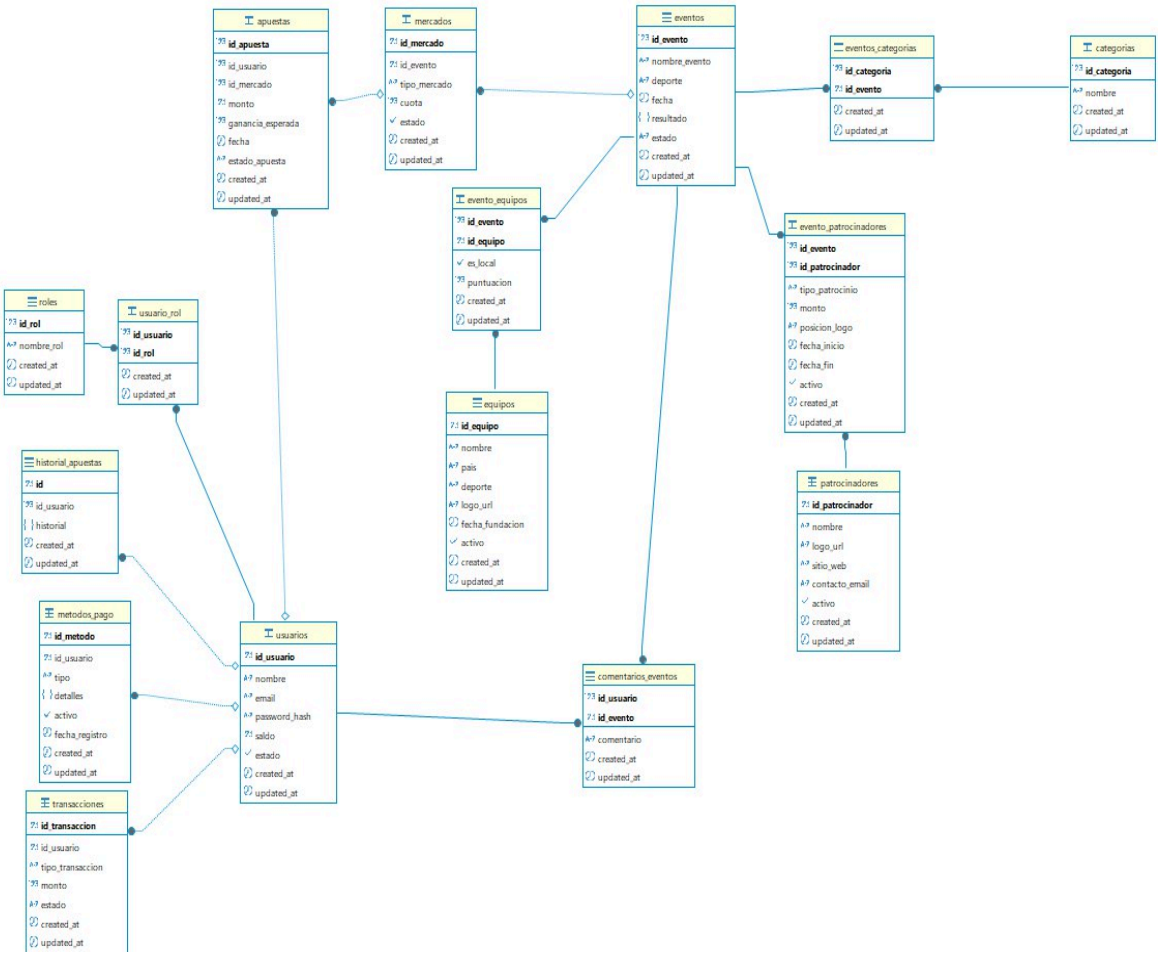
Nuestras entidades principales: usuarios, eventos, apuestas, comentarios, roles.

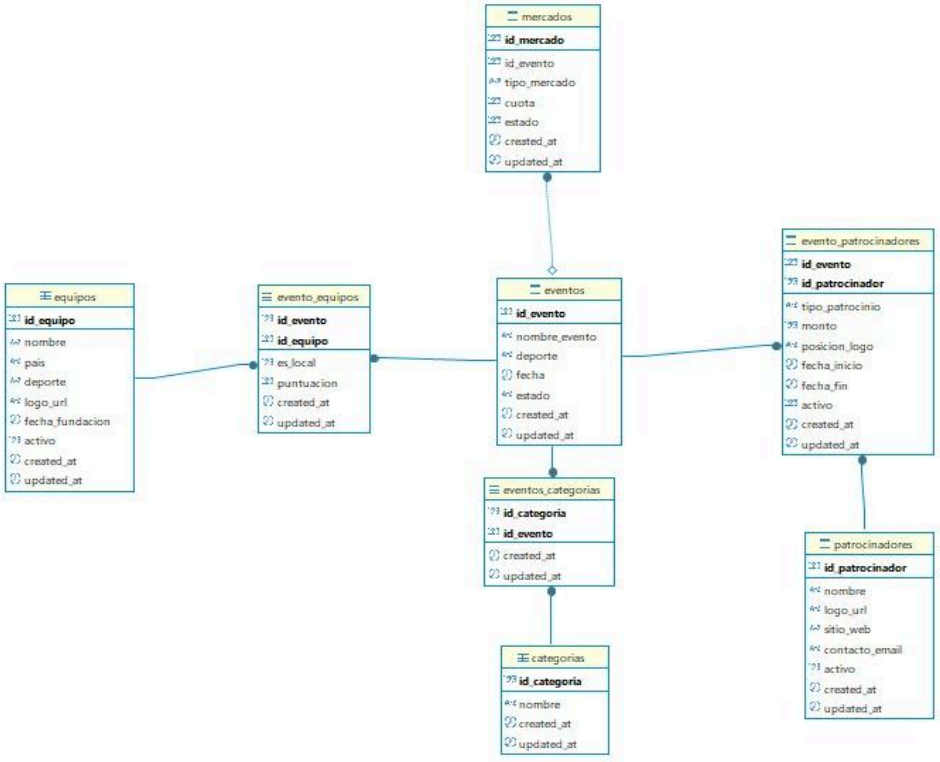
Las relaciones permiten gestionar apuestas hechas por usuarios sobre eventos, con control de sus saldos y diferentes apuestas que realicen.



4.2 Diagrama relacional (SQL)

El modelo relacional en PostgreSQL define PK en cada tabla y FK para garantizar integridad entre apuestas, eventos y usuarios.





4.3 Normalización

Organizamos los datos de forma eficiente para evitar redundancias y evitar líos a la hora de insertar, actualizar o eliminar información, así alcanzar la Tercera Forma Normal (3FN). Con esto pudimos evitar datos repetidos para que las tablas fueran más limpias y fáciles de manejar.

Primera Forma Normal (1FN)

- Revisamos que cada columna tuviera un solo valor.
    - Ejemplo: Si un usuario tenía varios métodos de pago o deportes favoritos, en lugar de ponerlos todos juntos en una misma celda, optamos por crear tablas separadas para cada uno.

Segunda Forma Normal (2FN)

- Acá ordenamos algunas tablas que usaban claves compuestas, moviendo datos que solo dependían de una parte de la clave a tablas nuevas.
    - Ejemplo: Las apuestas por evento tenían algunos datos que estaban juntos en una tabla así que los separamos, y los llevamos a las tablas de usuarios o mercados, según correspondía.

Tercera Forma Normal (3FN):

- Eliminamos dependencias innecesarias entre columnas.
    - Ejemplo: Se tenía información del país en la tabla de eventos, pero solo a través del nombre de la ciudad, decidimos crear una tabla de ciudades con su país relacionado para no repetir la misma información una y otra vez.

Así es como lo aplicamos en las tablas más importantes: usuarios, eventos, mercados, apuestas, transacciones y algunas más.

4.4 Diseño Documental (Referencial y Embebido)

Para la base de datos NoSQL, se decidió trabajar con MongoDB adoptando un enfoque híbrido que combina dos formas de estructurar los datos: documentos embebidos y referencias entre colecciones. Esta combinación nos permite lograr un buen equilibrio entre rendimiento y flexibilidad, dos aspectos clave en una aplicación que maneja múltiples entidades y relaciones dinámicas.

Elegir entre un modelo embebido o uno referencial no fue al azar. Se tuvieron en cuenta varios factores, entre ellos:

- Qué tan seguido se accede a la información.
- Si existe una relación jerárquica clara o una dependencia fuerte entre los datos.
- La necesidad de que ciertos datos puedan utilizarse de forma independiente o compartida.
- El tamaño y el crecimiento esperado de la información relacionada (por ejemplo, listas largas de actividades o registros históricos).

Modelo Embebido

Colección Preferencias: Esta colección guarda configuraciones propias de cada usuario: el idioma que prefiere, si quiere recibir notificaciones, qué deporte sigue más o cómo le gusta ver la interfaz. Toda esta información es exclusiva del usuario y no tiene sentido re utilizarla en otro contexto, así que lo más práctico fue mantenerla embebida en el documento del usuario o en una colección específica que conserve esa estructura.

Ventajas: Acceso directo y rápido, sin necesidad de hacer consultas adicionales. Además, es información que rara vez cambia y su tamaño es limitado, así que no representa una carga para el documento.

Ejemplos típicos de datos embebidos:

- Idioma preferido
- Activación o desactivación de notificaciones
- Deporte favorito
- Estilo visual (tema claro/oscuro, tamaño de fuente, etc.)

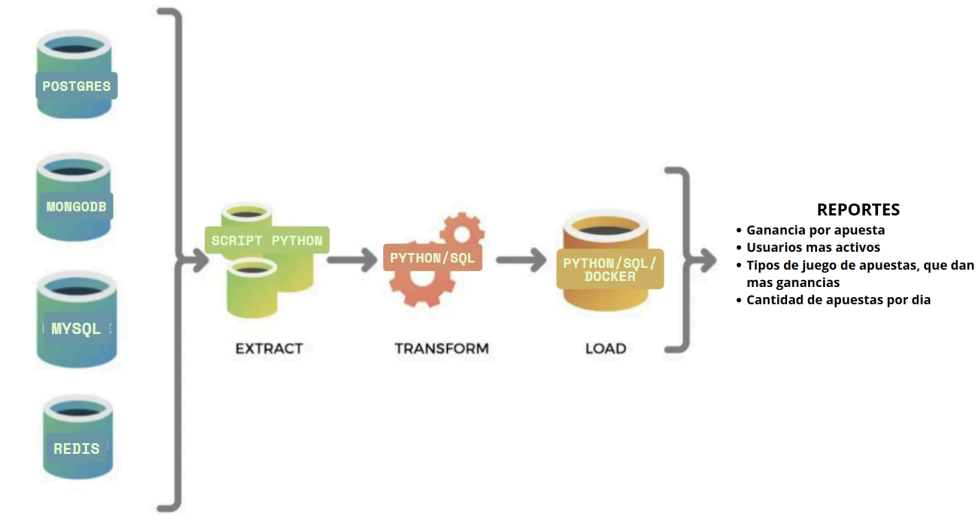
Modelo Referencial

En cambio, cuando hablamos de entidades que pueden crecer, relacionarse con varios usuarios o que deben actualizarse de forma independiente, lo mejor fue aplicar el modelo referencial. Algunas colecciones que siguen esta lógica son:

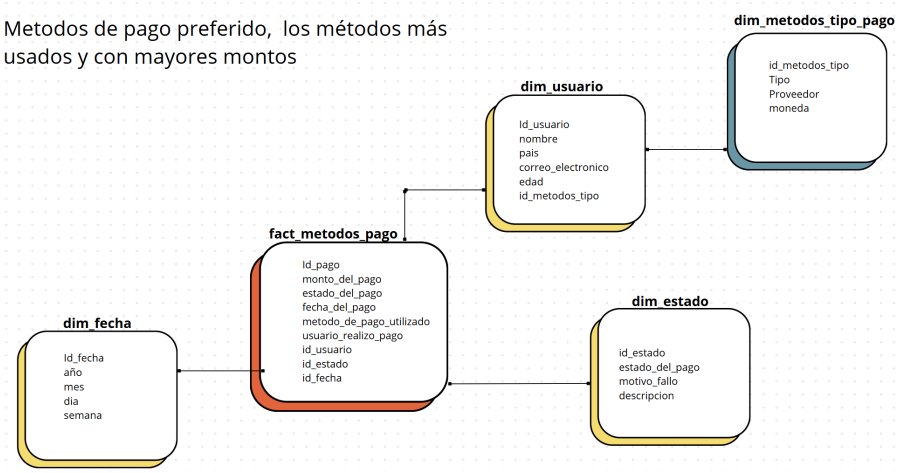
- **notificaciones**
  Cada usuario puede recibir múltiples mensajes, alertas o promociones. Al mantenerlos como documentos separados y referenciarlos, evitamos duplicar información del usuario y facilitamos actualizaciones masivas si es necesario.
- **reportes**
  Permiten a los usuarios denunciar actividades dentro de la plataforma. Mantenerlos por separado hace más fácil su gestión y seguimiento, especialmente para el personal que revisa estos casos.
- **mensajes_soporte**
  Agrupa los mensajes enviados por los usuarios al equipo de soporte. La independencia de esta colección permite llevar un historial claro de cada caso sin mezclarlo con los datos principales del usuario.
- **actividades_usuario**
  Aquí se registran acciones como inicios de sesión, apuestas realizadas o consultas frecuentes. Esta información es clave para generar estadísticas o detectar comportamientos, por lo que necesita mantenerse bien organizada y fácilmente consultable de forma histórica.
- **recompensas_diarias**
  Relaciona las recompensas otorgadas a los usuarios por su actividad diaria. Al mantenerlas como referencias, evitamos sobrecargar el documento del usuario y podemos hacer análisis más eficientes.

Diseño documental de la Base de Datos (Referencial y Embebido).

4.5 Diagrama de Flujo ETL



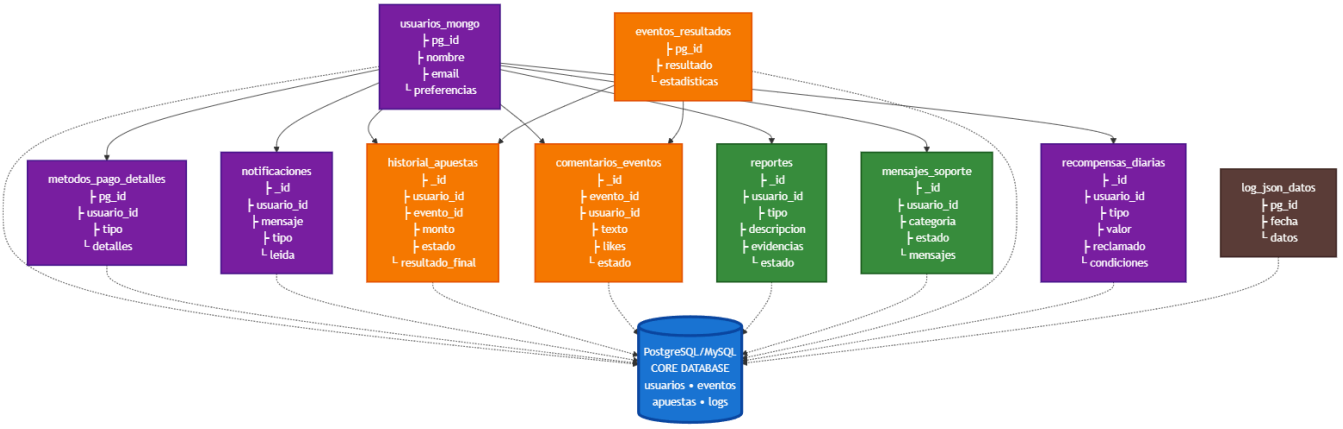
4.6 Diseño Snowflake



5.- Diseño de la base de datos

5.1 Modelo NoSQL

MongoDB:



En el modelo NoSQL implementado con **MongoDB**, se diseñaron colecciones que almacenan documentos en formato JSON, priorizando flexibilidad y eficiencia en la consulta de datos. A continuación, se describe la estructura general de cada colección, señalando sus campos clave y relaciones implícitas con otras colecciones:

Esta estructura modular permite escalar fácilmente el sistema y aplicar filtros, agregaciones y relaciones entre colecciones sin comprometer el rendimiento. Las decisiones entre modelos embebidos y referenciales se tomaron basándose en la frecuencia de lectura, tamaño del documento, y necesidad de independencia en los registros.

6.- Arquitectura y justificación técnica

6.1. PostgreSQL Cluster (Master-Slave)

PostgreSQL es un sistema de gestión de bases de datos relacional avanzado, conocido por su robustez, fiabilidad y cumplimiento estricto de ACID, lo que garantiza la integridad y consistencia de los datos. Su capacidad para soportar replicación Master-Slave permite no solo la alta disponibilidad del servicio, sino también el balanceo de cargas, ya que las consultas de lectura pueden ser distribuidas a réplicas esclavas sin afectar al nodo principal. Esto es fundamental en aplicaciones críticas donde la pérdida de datos o inconsistencias no son tolerables.

- **Rol:** Base de datos relacional principal para almacenamiento estructurado, con alta consistencia y soporte para transacciones complejas.

- **Justificación:** PostgreSQL es robusto, confiable además de la implementación Master-Slave para alta disponibilidad y redundancia. Se usa para gestionar datos críticos donde la integridad es primordial.

- **Configuración:** Un nodo primario (master) con un nodo réplica (slave) para tolerancia a fallos y consultas distribuidas.

6.2. MySQL

MySQL es otro sistema de bases de datos relacional muy popular, reconocido por su simplicidad, velocidad y amplia compatibilidad con diversas aplicaciones, especialmente en entornos web y legados. Su arquitectura optimizada para lecturas rápidas lo hace ideal para escenarios donde la velocidad es prioritaria y la carga de trabajo puede incluir consultas frecuentes. Además, MySQL contribuye a diversificar el ecosistema de bases de datos del sistema, ofreciendo una alternativa que puede ser útil en integraciones con tecnologías existentes o requerimientos específicos.

- **Rol:** Segunda base relacional para la distribución de los datos menos críticos que no llegan a tener un impacto directo en el flujo financiero de la aplicación

- **Justificación:** MySQL presenta ventajas en rendimiento para cargas de trabajo con consultas simples y lectura intensiva, lo que optimiza recursos para datos secundarios y menos críticos

6.3. MongoDB Sharded Cluster

MongoDB es una base de datos NoSQL orientada a documentos, diseñada para manejar grandes volúmenes de datos no estructurados o semi-estructurados, como logs, eventos o datos JSON. Su arquitectura de sharding permite la escalabilidad horizontal, distribuyendo los datos entre múltiples shards para mejorar la capacidad de almacenamiento y el rendimiento, adaptándose muy bien a cargas variables y sistemas que requieren alta disponibilidad.

- **Rol:** Base de datos NoSQL distribuida, diseñada para manejar grandes volúmenes de datos no estructurados o semi-estructurados, como logs, historiales o datos JSON.

- **Justificación:** MongoDB permite escalabilidad horizontal mediante sharding, lo que mejora la capacidad para manejar cargas variables y gran cantidad de datos. El cluster incluye configuradores, shards y routers (mongos) para la distribución y consulta eficiente de datos.

6.4. ETL Database (PostgreSQL)

La base de datos dedicada para los procesos ETL está separada de las bases de datos operativas para evitar impacto en el rendimiento y permitir una mejor gestión de los datos destinados a análisis y reportes. Los procesos ETL extraen datos de las bases PostgreSQL, MySQL y MongoDB, los transforman y cargan en esta base dedicada para su posterior explotación.

- **Rol:** Base de datos dedicada para procesos ETL (Extract, Transform, Load) que integran y transforman datos de las otras bases para análisis y reportes.

- **Justificación:** Separar el almacenamiento ETL mejora el rendimiento del sistema general y facilita la gestión de procesos de integración sin impactar las bases de datos operativas.

6.5. Redis Cache

Redis es un sistema de caché en memoria altamente rápido que soporta diversas estructuras de datos, ofreciendo tiempos de respuesta extremadamente bajos para consultas frecuentes. Se usa para almacenar temporalmente datos críticos que requieren acceso rápido, aliviando la carga de las bases de datos principales y mejorando la experiencia del usuario final.

- **Rol:** Caché en memoria para acelerar el acceso a datos frecuentes y reducir la latencia en consultas.

- **Justificación:** Redis es rápido, con soporte para estructuras de datos avanzadas y persistencia opcional. La caché distribuye la carga y mejora el tiempo de respuesta del sistema.

6.6 Flujo de Datos y Comunicación

El sistema implementa replicación Master-Slave en PostgreSQL para garantizar alta disponibilidad y consistencia, permitiendo que el nodo primario maneje las escrituras mientras las réplicas atienden las lecturas. Esta configuración mejora la resiliencia ante fallos y optimiza el rendimiento.

Las bases de datos relacionales (PostgreSQL y MySQL) y el cluster MongoDB están interconectados para facilitar consultas combinadas y sincronización parcial de datos, ofreciendo un ecosistema heterogéneo pero coherente que puede manejar distintos tipos de información de manera eficiente.

Por otro lado los procesos ETL extraen datos de todas las bases (PostgreSQL, MySQL y MongoDB) y los cargan en una base dedicada para análisis, lo que permite transformar y preparar los datos sin afectar las bases operativas.

Finalmente, Redis actúa como una capa de cache en memoria, recibiendo datos de todas las bases para acelerar las respuestas a consultas frecuentes y disminuir la latencia, optimizando así el rendimiento global del sistema. Las aplicaciones acceden a este conjunto distribuido de bases y caches para balancear la carga y garantizar una experiencia ágil y confiable para el usuario final.

7.-Implementación

7.1 Describir los Sp, View, Triggers, Functions, Indices,etc

Funciones

autenticar_usuario: Esta función permite verificar si un usuario existe y si su contraseña es válida. Se utiliza crypt para comparar la contraseña ingresada con el hash almacenado en la base de datos.

get_saldo_usuario: Obtiene el saldo actual de un usuario en la plataforma. Si no se encuentra saldo, devuelve cero por defecto usando COALESCE.

get_eventos_activos: Devuelve una lista de los eventos que aún no han ocurrido y que no están cancelados. Sirve para mostrar al usuario los eventos en los que aún puede apostar.

get_mercados_por_evento: A partir del ID de un evento, devuelve los mercados disponibles (por ejemplo: ganador, empate, etc.), siempre y cuando estén activos.

get_apuestas_por_usuario: Devuelve todas las apuestas que ha realizado un usuario, incluyendo información del evento, monto apostado y ganancia esperada.

get_comentarios_evento: Muestra todos los comentarios que los usuarios han dejado en un evento determinado, incluyendo la fecha en la que se hicieron.

Procedimientos

sp_insertar_usuario: Inserta un nuevo usuario en la base de datos con su contraseña cifrada. Tiene manejo de errores mediante EXCEPTION y ROLLBACK.

sp_eliminar_apuesta: Permite eliminar una apuesta si está activa. Si no existe o no está activa, lanza una excepción indicando el error.

sp_registrar_evento: Registra un nuevo evento deportivo en estado "programado". Incluye manejo de errores por si ocurre algún fallo al insertar.

sp_cancelar_evento: Cancela un evento modificando su estado y también cancela todas las apuestas asociadas al mismo. Incluye control de errores.

sp_retirar_saldo: Resta un monto del saldo de un usuario. Verifica que el monto sea positivo y que el usuario tenga suficiente saldo antes de realizar la operación.

sp_registrar_apuesta: Registra una apuesta para un mercado, revisando que el evento esté activo y que el usuario tenga saldo suficiente. Si todo es correcto, se descuenta el saldo y se inserta la apuesta.

Triggers

tr_log_creacion_apuesta: Cada vez que se crea una nueva apuesta, este trigger registra automáticamente los datos en una tabla de logs en formato JSON.

tr_log_cambio_saldo: Detecta cuando el saldo de un usuario cambia y registra el saldo anterior y el nuevo, junto con la fecha del cambio.

tr_log_apuesta_cancelada: Si una apuesta es cancelada (y antes no lo estaba), se registra esta acción en el log con los detalles de la apuesta.

tr_validar_saldo_no_negativo: Este trigger impide que se actualice el saldo de un usuario a un valor negativo, lanzando una excepción si ocurre.

tr_log_resultado_evento: Cada vez que cambia el resultado de un evento, se registra en los logs el valor anterior y el nuevo.

tr_calcular_ganancia_esperada: Antes de insertar una nueva apuesta, este trigger calcula automáticamente la ganancia esperada multiplicando el monto por la cuota del mercado.

Vistas
vista_apuestas_activas: Muestra todas las apuestas que están en estado "activa", junto con datos del usuario, evento, monto y ganancia.

vista_top_usuarios_apuestas: Muestra los usuarios que más apuestan, indicando cuántas apuestas han hecho y el monto total apostado.

vista_eventos_finalizados: Lista todos los eventos que ya se han llevado a cabo (aunque en tu código falta completar el WHERE para verificar si la fecha ya pasó).

7.2 Transacciones ACID

Durante el desarrollo del sistema, se trabajó con transacciones que cumplen con las propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad). La idea fue simple: asegurarnos de que todo funcione bien, incluso cuando las operaciones son complejas o se ejecutan al mismo tiempo por varios usuarios. Esto fue especialmente importante en cosas como mover saldos, registrar apuestas o cancelar eventos.

Atomicidad

Cuando se hace una operación que implica varios pasos (como retirar saldo y actualizar el estado de una apuesta), todo eso se ejecuta como un solo bloque. Si algo falla en el camino, se revierte todo. Nada queda a medias.

Por ejemplo:

- En el procedimiento sp_retirar_saldo, si el usuario no existe o no tiene suficiente saldo, se cancela todo y no se toca nada.

- En sp_cancelar_evento, se cambia el estado del evento y se cancelan todas las apuestas relacionadas, pero todo dentro de una misma transacción. O se hace todo, o no se hace nada.

Esto evita que el sistema termine en un estado raro o incompleto.

Consistencia

El sistema tiene reglas que se validan antes de ejecutar cualquier operación. Esto nos asegura que la base de datos siempre mantenga datos válidos y coherentes.

Algunos ejemplos:

- No se puede retirar un monto negativo (porque no tendría sentido).

- Tampoco se puede eliminar una apuesta que ya fue procesada o finalizada.

Estas validaciones están en los procedimientos almacenados, y ayudan a evitar errores que podrían romper la lógica del sistema.

Aislamiento

Como el sistema puede ser usado por muchos usuarios al mismo tiempo, es clave que una operación no afecte a otra. Las transacciones ayudan a eso: cada operación corre de forma aislada, sin que lo que hace un usuario interfiera con lo que hace otro.

Esto evita problemas como que dos personas intenten modificar el mismo dato al mismo tiempo y uno de los cambios se pierda.

Durabilidad

Una vez que se confirma una transacción (es decir, se hace el COMMIT), los cambios se guardan para siempre. Aunque el sistema se caiga justo después, lo que ya se hizo no se pierde.

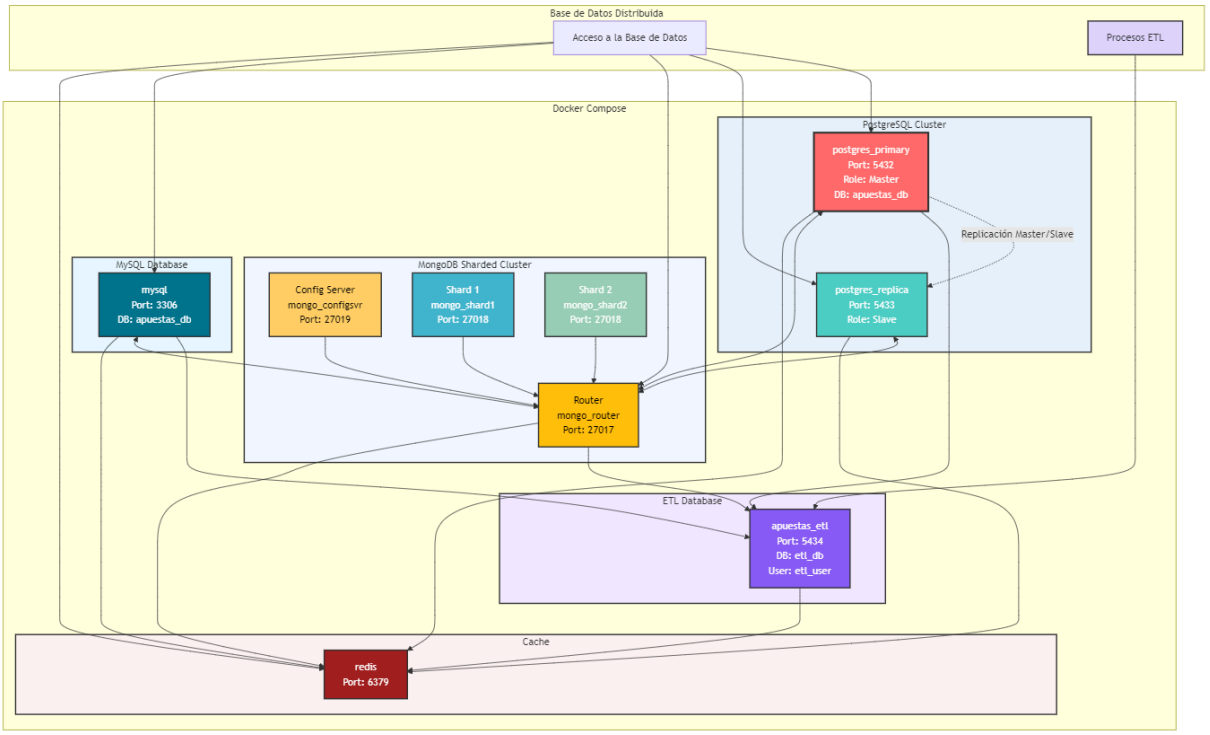
PostgreSQL se encarga de esto automáticamente, usando sus propios mecanismos de escritura y recuperación. Básicamente, una vez que algo queda guardado, está garantizado.

7.3 Backups, Roles y Permisos

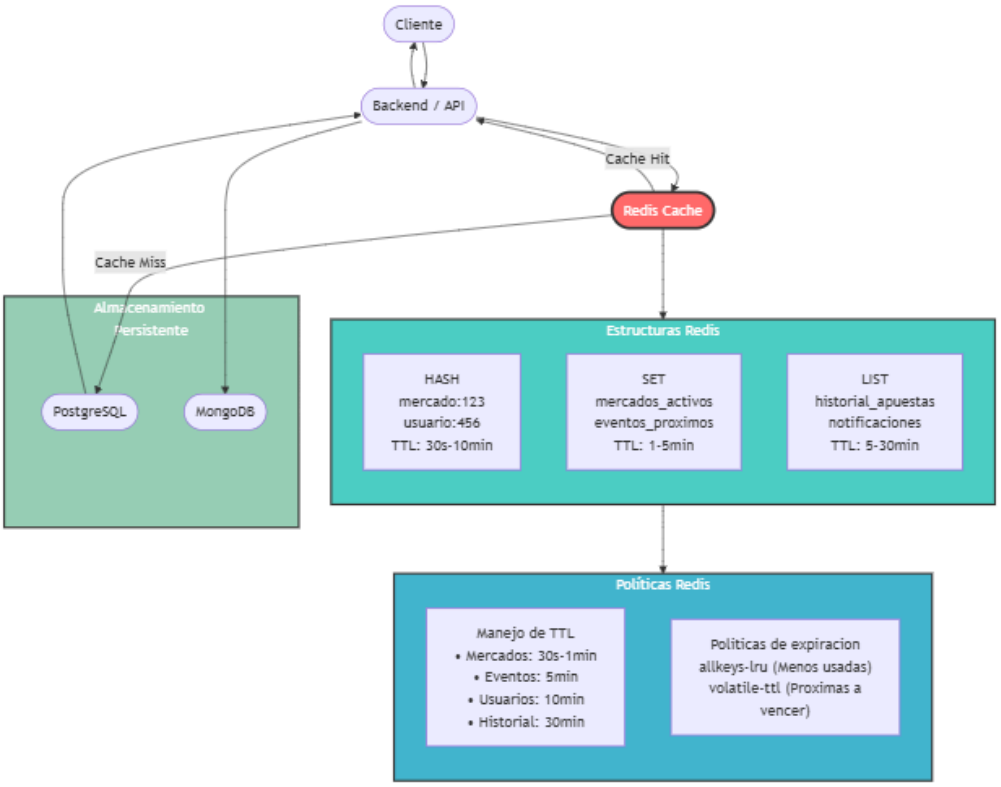
Se tiene definidos tres roles principales: admin, apostador y cajero, que establece una política de control de acceso clara y segura en la base de datos. Cada rol cuenta con permisos específicos que limitan las acciones de los usuarios, garantizando la separación de responsabilidades, asi manteniendo la integridad y seguridad del sistema. Esto es muy útil en un entorno financiero o de apuestas, debido

a que tienen acceso a información sensible y la ejecución de transacciones deben estar estrictamente controlados.
Para la implementación, en PostgreSQL se creó una estructura detallada de roles y permisos: el rol admin tiene control total, el rol apostador tiene permisos de lectura e inserción en las tablas relacionadas con apuestas, y el rol cajero puede gestionar usuarios, métodos de pago y transacciones con permisos de lectura, inserción y actualización. En MySQL, debido a que no cuenta con un sistema de roles tan granular, la gestión de permisos se reforzó mediante un diseño relacional cuidadoso, con claves y relaciones que apoyan la lógica de negocio y segregan responsabilidades centrandose mas en los eventos deportivos
Este diseño de un sistema de backups y restauración, nos garantiza la disponibilidad y recuperación ante fallos.

7.4 Base de datos Distribuidas



7.5 REDIS(CACHE)



En la figura se muestra que cuando un cliente realiza una consulta, el backend primero revisa si los datos están en Redis (Cache Hit). Si no están (Cache Miss), accede a las bases de datos persistentes: PostgreSQL o MongoDB.
Se utilizan diferentes estructuras de Redis según el tipo de dato:

- HASH para mercados y usuarios (TTL: 30s–10min),
- SET para mercados activos y eventos próximos (TTL: 1–5min),
- LIST para historial de apuestas y notificaciones (TTL: 5–30min).

y con políticas de expiración como allkeys-lru que elimina los menos usados y volatile-ttl que prioriza los que están por vencer. Así nos permite mantener el sistema rápido, eficiente y útil para gran volumen de consultas.

7.6 Consultas MongoDB (lookup, unwind, etc.)

- **Usuarios con más notificaciones**
  Nos muestra quiénes son los usuarios que más notificaciones han recibido en el sistema. Básicamente, los más “populares” para el sistema.

- **Últimas notificaciones por tipo**
  Aquí sacamos las notificaciones más recientes que se han enviado, separadas por su tipo (como alerta, mensaje, recordatorio, etc.).
- **Conteo de reportes por estado**
  Nos da un resumen de cuántos reportes hay según su estado, por ejemplo: pendientes, resueltos o en proceso.
- **Actividades por día**
  Esta función nos muestra cuántas actividades se realizaron en cada día, útil para ver los días con más movimiento.
- **Usuarios más activos**
  Identifica a los usuarios que más interactúan con el sistema, ya sea apostando, comentando o entrando seguido.
- **Recompensas más comunes**
  Nos dice cuáles son las recompensas que más veces se han entregado, como para saber qué es lo que más gusta o se repite.
- **Soportes pendientes**
  Muestra todos los mensajes o tickets de soporte que todavía no se han respondido o resuelto.
- **Tipos de notificación más comunes**
  Nos ayuda a ver qué tipo de notificaciones se mandan más seguido, por ejemplo: recordatorios o mensajes informativos.
- **Preferencias más seleccionadas**
  Analiza cuáles son las configuraciones favoritas de los usuarios, como idioma o deporte preferido.
- **Tiempo promedio de respuesta en soporte**
  Calcula cuánto tiempo, en promedio, se demora el equipo de soporte en responder un mensaje.
- **Actividades por tipo**
  Agrupa todas las actividades según su tipo, como por ejemplo: apuestas, registros, o consultas.
- **Conversaciones de soporte por usuario**
  Muestra qué usuarios han tenido más conversaciones con el soporte, ideal para saber quién ha necesitado más ayuda.
- **Obtener notificaciones junto con datos del usuario**
  Aquí no solo ves la notificación, sino también información del usuario que la recibió, todo juntito.
- **Obtener reportes junto con información del usuario que lo reportó**
  En vez de solo mostrar el reporte, también te da los datos de la persona que lo hizo. Bien completo.
- **Listar recompensas diarias con datos del usuario**
  Muestra qué recompensas diarias se han entregado y a quién se le dio cada una.
- **Unir actividades del usuario con su información básica**
  Vincula lo que el usuario ha hecho con sus datos principales, como su nombre o email.
- **Obtener mensajes de soporte junto con el usuario que los envió**
  Muestra los mensajes de soporte, pero también quién los escribió. Para tener contexto completo.
- **Ver historial de apuestas junto con datos del evento**
  Te permite ver qué apostó el usuario, cuándo, y en qué evento fue la apuesta. Muy útil para hacer seguimiento.

- **Ver métodos de pago con detalles del usuario**
  Lista todos los métodos de pago que usaron los usuarios, y también quién es el dueño de cada uno.
- **Mostrar actividades del usuario junto a sus mensajes de soporte**
  Reúne en una sola vista lo que el usuario hizo en la plataforma y los mensajes que mandó al soporte.

7.1.1 Herramientas y tecnologías utilizadas

- Mongo, Postgres, Readis, MySQL, DataGrep, DBeaver

Docker Compose – Configuración técnico

El sistema distribuido del proyecto, se utilizó Docker Compose, donde la configuración levanta simultáneamente los distintos servicios involucrados en el sistema, tanto relacionales como no relacionales.

docker-compose.yml:

1. PostgreSQL: Base de datos relacional para la gestión de usuarios, eventos y apuestas.
2. MongoDB :
    a. configsvr: Servidor de configuración.
    b. shard1, shard2: Fragmentos de datos distribuidos.
    c. mongos: Router que dirige las consultas a los shards correctos.
3. Redis: Sistema de caché utilizado para optimizar consultas críticas.
4. DBeaver: Herramienta cliente conectada por red para visualización de datos.

Ahora para levantar el docker:

1. Clona el repositorio

```
git clone <https://github.com/Andrezuu/proyect-bdd-avanzada.git>
cd proyect-bdd-avanzada
```

2. Copia el archivo de configuración

```
cp .env.example .env
# Edita las variables necesarias (puerto, usuario, contraseña, etc.)
```

3. Levanta todos los servicios Este paso levantará:

- PostgreSQL
- MongoDB
- Redis

Ejecuta en la raíz del proyecto:

```
docker-compose up -d
```

Espera a que los contenedores estén 100% iniciados (puedes verificar con docker ps).

4. Inicializa la base de datos

```
./init.bat
# Ejecuta todos los scripts relacionales automáticamente
```

5. Instalar dependencias para MongoDB (solo si usarás los scripts Python)

```
pip install pymongo faker psycopg2-binary mysql-connector-python
```

6. Cargar datos falsos (opcional)

Puedes poblar la base de datos con datos de prueba ejecutando:

```
python scripts/init_data.py
```

7. Backup y Restore

- Para hacer un backup de PostgreSQL:

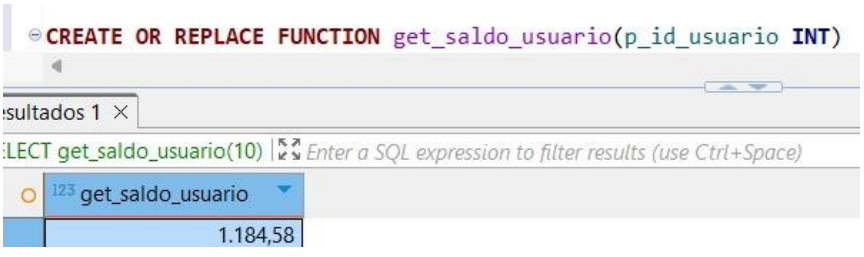
```
node scripts/backup.js
```

- Para restaurar un backup:

```
node scripts/restore.js
```

8.- Pruebas y evaluación

FUNCTION
funtion get_saldo_usuario



PROCEDURES
1.sp_retirar_saldo

```
CREATE OR REPLACE PROCEDURE sp_retirar_saldo(p_id_usuario INT, p_monto NUMERIC)
```

usuarios 1 ×

SELECT id_usuario, nombre, saldo FROM usuarios WHERE | Enter a SQL expression to filter results (use Ctrl+Space)

	id_usuario	nombre	saldo
1	2	Belén Blanca Bautista	379,63

2.sp_registrar_evento,

```
CREATE OR REPLACE PROCEDURE sp_registrar_evento(
    p_nombre varchar(45),
```

entos 1 ×

ECT * FROM eventos WHERE deporte = 'Fútbol' | Enter a SQL expression to filter results (use Ctrl+Space)

	id_evento	nombre_evento	deporte	fecha	
45	748	Cross-group neutral matrices - Fútbol	Fútbol	2025-08-05 08:32:28.000	{}
46	757	Open-source radical parallelism - Fútbol	Fútbol	2025-06-05 21:29:53.000	{"
47	759	Phased foreground core - Fútbol	Fútbol	2025-06-23 02:45:09.000	{}
48	760	Virtual dynamic parallelism - Fútbol	Fútbol	2025-06-14 02:59:23.000	{"
49	762	Expanded disintermediate process improvement - Fú	Fútbol	2025-06-03 20:00:06.000	{"
50	767	Persevering foreground hardware - Fútbol	Fútbol	2025-07-06 03:21:27.000	{}
51	777	Configurable composite extranet - Fútbol	Fútbol	2025-06-12 01:32:15.000	{"
52	780	Total attitude-oriented infrastructure - Fútbol	Fútbol	2025-07-31 18:34:06.000	{}
53	781	Cloned bi-directional extranet - Fútbol	Fútbol	2025-06-22 16:21:12.000	{"
54	791	Reverse-engineered contextually-based middleware	Fútbol	2025-06-05 07:26:51.000	{"
55	797	Focused analyzing hierarchy - Fútbol	Fútbol	2025-06-19 00:45:01.000	{"
56	799	Ergonomic bifurcated hub - Fútbol	Fútbol	2025-06-24 01:22:25.000	{}
57	801	Final Copa Libertadores	Fútbol	2025-11-15 20:00:00.000	[N
58	802	Final Copa Libertadores	Fútbol	2025-11-15 20:00:00.000	[N
59	804	Final Copa Libertadores	Fútbol	2025-11-15 20:00:00.000	[N
60	805	Final Copa UPB(DTI EN LA FINAL)	Fútbol	2025-11-15 20:00:00.000	[N

3.sp_eliminar_apuesta

4	6	pendiente

```
CREATE OR REPLACE PROCEDURE sp_eliminar_apuesta(p_id_apuesta INT
LANGUAGE plpgsql
```

uestas 1 ×

ECT id_apuesta, estado_apuesta FRO | Enter a SQL expression to filter results (use Ctrl+Spa

	id_apuesta	estado_apuesta

VISTAS

vista_detalle_eventos_simple

vista_detalle_eventos_simple 1 ×

select * from vista_detalle_eventos_simple | Enter a SQL expression to filter results (use Ctrl+Space)

	id_evento	nombre_evento	deporte	fecha	equipo	rol_equipo	categoria	patrocinador	tipo_mercado	cuota
1	1	Devolved leadingedge focus group - Fútbol	Fútbol	2025-06-08 16:12:37	La Rioja lure	Visitante	Tenis	[NULL]	1X2	2,81
2	1	Devolved leadingedge focus group - Fútbol	Fútbol	2025-06-08 16:12:37	La Rioja lure	Visitante	Tenis	[NULL]	Total Goles	1,72
3	1	Devolved leadingedge focus group - Fútbol	Fútbol	2025-06-08 16:12:37	La Rioja lure	Visitante	Tenis	[NULL]	Ganador	7,56
4	1	Devolved leadingedge focus group - Fútbol	Fútbol	2025-06-08 16:12:37	La Rioja lure	Visitante	Tenis	[NULL]	Más/Menos 2.5	1,73
5	1	Devolved leadingedge focus group - Fútbol	Fútbol	2025-06-08 16:12:37	Ourense Occaecati	Local	Tenis	[NULL]	1X2	2,81
6	1	Devolved leadingedge focus group - Fútbol	Fútbol	2025-06-08 16:12:37	Ourense Occaecati	Local	Tenis	[NULL]	Total Goles	1,72
7	1	Devolved leadingedge focus group - Fútbol	Fútbol	2025-06-08 16:12:37	Ourense Occaecati	Local	Tenis	[NULL]	Ganador	7,56
8	1	Devolved leadingedge focus group - Fútbol	Fútbol	2025-06-08 16:12:37	Ourense Occaecati	Local	Tenis	[NULL]	Más/Menos 2.5	1,73
9	2	Optimized multi-state firmware - Béisbol	Béisbol	2025-08-22 08:37:30	Zamora Aperiam	Local	Golf	Banca Privada EN S.Coop.	1X2	4,62

vistas_eventos_finalizados

```
CREATE OR REPLACE VIEW vista_eventos_finalizados AS
SELECT
  id_evento,
  nombre_evento
```

a_eventos_finalizados 1 ×

ECT * FROM vista_eventos_finalizados | Enter a SQL expression to filter results (use Ctrl+Space)

id_evento	nombre_evento	deporte	estado
54	Proactive context-sensitive solution - Fútbol	Fútbol	finalizado
55	Future-proofed maximized complexity - Tenis	Tenis	finalizado
57	Synergized tangible structure - Béisbol	Béisbol	finalizado
61	Customer-focused 6thgeneration implementation - Teni	Tenis	finalizado
64	Visionary discrete task-force - Béisbol	Béisbol	finalizado
69	Stand-alone hybrid contingency - Baloncesto	Baloncesto	finalizado

OPTIMIZATION

```
EXPLAIN ANALYZE
SELECT
  u.id_usuario,
  u.nombre AS nombre_usuario,
  COUNT(a.id_apuesta) AS total_apuestas,
  SUM(a.monto) AS monto_total_apostado
FROM usuarios u
JOIN apuestas a ON u.id_usuario = a.id_usuario
GROUP BY u.id_usuario, u.nombre
ORDER BY total_apuestas DESC;
```

ultados 1 ×

PLAIN ANALYZE SELECT u.id_usuario, u.r | Enter a SQL expression to filter results (use Ctrl+Space)

QUERY PLAN
Sort (cost=1546.01..1558.51 rows=5000 width=66) (actual time=51.430..51.661 rows=4985 loops=1)
Sort Key: (count(a.id_apuesta)) DESC
Sort Method: quicksort Memory: 615kB
-> HashAggregate (cost=1176.32..1238.82 rows=5000 width=66) (actual time=48.546..49.780 rows=4985 loops=1)
Group Key: u.id_usuario
Batches: 1 Memory Usage: 2513kB
-> Hash Join (cost=218.50..951.32 rows=30000 width=36) (actual time=10.703..35.947 rows=29998 loops=1)
Hash Cond: (a.id_usuario = u.id_usuario)
-> Seq Scan on apuestas a (cost=0.00..654.00 rows=30000 width=14) (actual time=0.216..10.700 rows=29998 loops=1
-> Hash (cost=156.00..156.00 rows=5000 width=26) (actual time=10.043..10.044 rows=5001 loops=1)
Buckets: 8192 Batches: 1 Memory Usage: 350kB
-> Seq Scan on usuarios u (cost=0.00..156.00 rows=5000 width=26) (actual time=0.140..7.604 rows=5001 loops=1)
Planning Time: 7.114 ms
Execution Time: 55.012 ms

```
EXPLAIN ANALYZE
SELECT
  u.id_usuario,
  u.nombre AS nombre_usuario,
  COUNT(a.id_apuesta) AS total_apuestas,
  SUM(a.monto) AS monto_total_apostado
FROM usuarios u
JOIN apuestas a ON u.id_usuario = a.id_usuario
```

ultados 1 ×

LAIN ANALYZE SELECT u.id_usuario, u.r | Enter a SQL expression to filter results (use Ctrl+Space)

QUERY PLAN
Sort (cost=1546.07..1558.58 rows=5001 width=66) (actual time=48.707..49.264 rows=4985 loops=1)
Sort Key: (count(a.id_apuesta)) DESC
Sort Method: quicksort Memory: 615kB
-> HashAggregate (cost=1176.30..1238.81 rows=5001 width=66) (actual time=44.295..47.298 rows=4985 loops=1)
Group Key: u.id_usuario
Batches: 1 Memory Usage: 2513kB
-> Hash Join (cost=218.52..951.31 rows=29998 width=36) (actual time=2.315..23.718 rows=29998 loops=1)
Hash Cond: (a.id_usuario = u.id_usuario)
-> Seq Scan on apuestas a (cost=0.00..653.98 rows=29998 width=14) (actual time=0.011..5.931 rows=29998 loops=1)
-> Hash (cost=156.01..156.01 rows=5001 width=26) (actual time=2.234..2.238 rows=5001 loops=1)
Buckets: 8192 Batches: 1 Memory Usage: 350kB
-> Seq Scan on usuarios u (cost=0.00..156.01 rows=5001 width=26) (actual time=0.010..1.352 rows=5001 loops=1)
Planning Time: 2.242 ms
Execution Time: 50.037 ms

CREATE INDEX idx_apuestas_id_usuario ON apuestas(id_usuario);
CREATE INDEX idx_usuarios_id_usuario ON usuarios(id_usuario);

TRIGGERS

trg_desactivar_metodos_pago

```
UPDATE usuarios SET estado = FALSE WHERE id_usuario = 1764;
SELECT * FROM metodos_pago WHERE id_usuario = 1764;
```

odos_pago 1

CT * FROM metodos_pago WHERE id_usuario = 1764 | Enter a SQL expression to filter results (use Ctrl+Space)

id_metodo	id_usuario	tipo	activo	created_at	updated_at
744	1.764	transferencia	[ ]	2025-06-26 15:53:29.021	2025-06-26 12:14:21.928
996	1.764	tarjeta_debito	[ ]	2025-06-26 15:53:29.021	2025-06-26 12:14:21.928
1.662	1.764	paypal	[ ]	2025-06-26 15:53:29.021	2025-06-26 12:14:21.928
223	1.764	tarjeta_credito	[ ]	2025-06-26 15:53:29.021	2025-06-26 12:14:21.928
529	1.764	paypal	[ ]	2025-06-26 15:53:29.021	2025-06-26 12:14:21.928

tr_log_resultado_evento

```
CREATE OR REPLACE FUNCTION validar_saldo_no_negativo()
```

usuarios 1

T UPDATE usuarios SET saldo = -10 WHERE | Enter a SQL expression to filter resu

```
SQL Error [P0001]: ERROR: No se permite saldo
negativo para el usuario 1
  Where: PL/pgSQL function
validar_saldo_no_negativo() line 4 at RAISE
```

METODOS MONGO

db.historial_apuestas.aggregate

	_id	nombre	total_ganadas	usuario_id
1	528	Desiderio Salcedo Nogueira	1	528
2	2658	Baldomero Frutos Contreras	1	2658
3	2459	Josefina Pinedo Torrecilla	1	2459
4	786	Mauricio Sanz-Chaves	2	786
5	2149	María Jesús Edelmira Uriarte Campos	1	2149
6	3221	Marita Amaya Baños	3	3221
7	2152	Trinidad Peñalver Mate	2	2152
8	1935	Odalys Gracia Miranda Linares	1	1935

1-20 of 21+

db.metodos_pago_detalles.aggregate

```
  }
}
])
```

	_id	nombre	total_metodos	usuario_id
1	528	Desiderio Salcedo Nogueira	2	528
2	626	Eligia Cortés Calderón	1	626
3	910	América Salvà Acedo	1	910
4	550	Cristóbal Losa-Busquets	1	550
5	1770	Emma Agustín Diego	2	1770
6	4047	Febe de Ledesma	1	4047
7	3271	Rafael Valverde Juan	4	3271
8	2868	Amílcar Amado Alcaraz Pera	2	2868

1-20 of 21+

9.- Conclusiones y Evaluación(corregir)

Este proyecto fue una buena oportunidad para poner en práctica todo lo que fuimos aprendiendo en la materia. Pudimos trabajar con bases de datos relacionales y no relacionales, y entender mejor cuando usar cada una. Usamos PostgreSQL, MongoDB, Redis y Docker, y eso nos ayudo a construir un sistema más completo, organizado y que responde bien en distintos escenarios.

Algo que nos sirvió mucho fue implementar transacciones ACID, procedimientos almacenados, colecciones embebidas y referenciales, y también consultas más complejas usando operadores de agregación. Todo eso nos ayudó a entender mejor cómo se procesan y organizan los datos en proyectos reales.

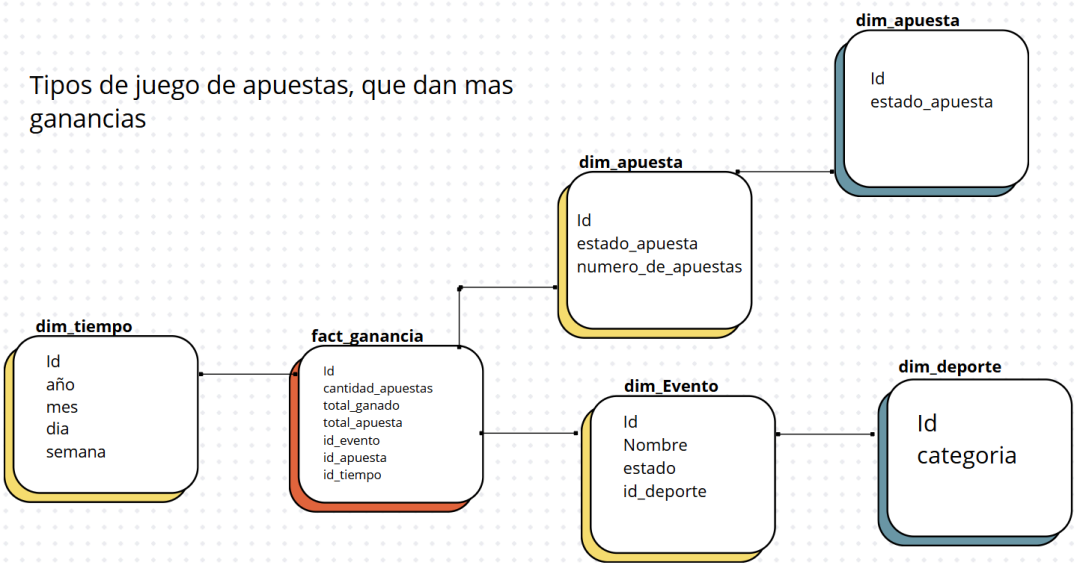
La materia en sí fue súper útil. No solo aprendimos la parte técnica, también practicamos cómo organizar el trabajo en equipo, planificar por etapas y tomar decisiones técnicas con criterio. Creemos que este tipo de experiencias con objetivos suma en nuestros skills para futuros proyectos.

10.- Referencias Bibliográficas

- SQL y BASES DE DATOS Desde Cero para PRINCIPIANTE
  - YouTube**:** https://www.youtube.com/watch?v=OuJerKzV5T0
- MongoDB Inc. *MongoDB Manual v7.0*.
  - Recuperado de: https://www.mongodb.com/docs/manual/
- Redis Labs. Redis Documentation.
  - Recuperado de: https://redis.io/docs/
- Establecer una Conexión de Postgres en DBeaver
  - Youtube: https://youtu.be/1zR7kTBoZg8?si=xK93yre7F9abH8iJ
- PostgreSQL. *CREATE TRIGGER — PostgreSQL Official Docs*.
  - Recuperado: https://www.postgresql.org/docs/current/sql-createtrigger.html
- PostgreSQL Global Development Group. (2024). *PostgreSQL Documentation 16*.
  - Recuperado de: https://www.postgresql.org/docs/
- Oracle Corporation. *MySQL 8.4 Reference Manual – Using Roles*.
  - Recuperado: https://dev.mysql.com/doc/refman/8.4/en/roles.html
- Landaeta, P. *BD-Avanzadas-2025* [Repositorio GitHub].
  - Recuperado: https://github.com/PaulLandaeta/BD-Avanzadas-2025.git
- pkdone. (s.f.). *Sharded MongoDB Docker Setup – docker-compose.yml* [Repositorio GitHub].
  - Recuperado:https://github.com/pkdone/sharded-mongodb-docker/blob/main/docker-compose.yml
- PhoenixNAP. *MongoDB Sharding Explained: How It Works and How to Set It Up*.
  - Recuperado: https://phoenixnap.com/kb/mongodb-sharding
- Bravo, A. V. (s.f.). *MySQL con Docker Compose* [GitBook].
  - Recuperado: https://avbravo-2.gitbook.io/docker/mysql-con-docker-compose

11.- Anexos

- Anexo 1: Desarrollo de 2do snowflake



- Anexo 2: Consulta para ver si todos los usuarios tienen método de pago

```sql
SELECT
    u.id_usuario,
    u.nombre,
    u.id_metodos_tipo
FROM
    apuestas_olimpiadas.dim_usuario u
LEFT JOIN
    apuestas_olimpiadas.dim_metodos_tipo_pago m
    ON u.id_metodos_tipo = m.id_metodos_tipo
WHERE
    m.id_metodos_tipo IS NULL;
```