Sistema de Base de Datos Híbrido - Valorant Game Management

BASE DE DATOS AVANZADAS

Cabezas J.*, Callapa L.**, Ledezma D.***

Universidad Privada Boliviana

Campus La Paz

26 de junio de 2025

ÍNDICE

Resumen(Abstract)

Valorant genera una gran cantidad de información, por la cantidad de jugadores que tiene, Para dar una idea de cómo gestionarlo de manera correcta, se diseñó una arquitectura que combina bases de datos: PostgreSQL y MySQL para operaciones principales, MongoDB para análisis y Redis para acelerar el acceso. Se aplicó un modelo en forma de copo de nieve y un proceso ETL que transforma los datos en archivos CSV listos para análisis. Las pruebas de rendimiento confirmaron la escalabilidad, velocidad y confiabilidad del sistema. Además, se plantea como mejora futura integrar inteligencia artificial para predicciones y análisis en vivo durante las partidas.

1. Introducción

1.1. Contexto del proyecto

Valorant, un juego en línea que agarro popularidad en estos últimos años, gestiona una inmensa cantidad de información a escala mundial, y esta cifra continúa expandiéndose a medida que más personas se unen al juego. Al igual que en otros títulos en línea, se producen incontables sucesos al mismo tiempo en diversas partes del mundo. Valorant procesa y guarda una gran cantidad de datos de los jugadores, y también facilita transacciones financieras para la adquisición de aspectos de armas y otros objetos cosméticos. Aparte, los usuarios pueden optar por gastar dinero para acelerar el acceso a nuevos personajes y adquirir el pase de batalla disponible en el juego, que se renueva con cada cambio de temporada. Igualmente, al comenzar a jugar, es necesario que los jugadores se registren, facilitando información personal y detalles de su tarjeta o forma de pago, por si quieren comprar cosas en línea dentro del juego. Adicionalmente, hay páginas web oficiales conectadas a Valorant que exhiben el desempeño total de cada jugador por temporada, e incluso por partidas concretas, indicando las bajas, ayudas, defunciones, mapas usados y otros datos útiles para que el jugador examine y perfeccione su juego.

1.2. Justificación

En 2023, Valorant ya contaba con más de 20 millones de jugadores, y su popularidad siguió en aumento. La inmensa cantidad de datos que un videojuego multijugador de tal magnitud necesita administrar. Los servidores de Valorant deben funcionar casi sin pausa, asegurando la integridad y el acceso a los datos.

Aquí son muy importantes la distribución de los jugadores y tener respaldo de las transacciones Las consultas son clave para examinar las partidas y el desempeño de los jugadores, considerando varios parámetros en cada juego. Así asegurar respuestas más rápidas y tener respaldo en cualquier problema futuro

1.3. Planteamiento del problema

En los eSports actuales, como Valorant, uno de los principales desafíos es garantizar que la gestión financiera sea segura y efectiva, lo cual es fundamental tanto para los jugadores como para la industria. Si ocurren errores o retrasos, se puede perder la confianza, aparecen problemas legales y los jugadores deciden abandonar. Para evitar esto, es esencial contar con una base de datos robusta capaz de procesar gran cantidad de información de manera instantánea, con integridad y sin interrupciones.

Además, muchos jugadores profesionales carecen de un lugar oficial dentro del juego para revisar a fondo sus estadísticas, por lo que recurren a aplicaciones externas. Esto representa un riesgo para su privacidad y una oportunidad desperdiciada para mejorar su rendimiento en el entorno oficial del juego.

Otro inconveniente significativo es que el sistema actual para informar sobre errores, trampas y comportamientos inapropiados no opera de manera eficiente. No se reciben respuestas, no hay priorización ni seguimiento, lo que genera frustración entre los usuarios y disminuye la confianza en el sistema, perjudicando el ambiente competitivo. Por todas estas razones, es necesario desarrollar una solución que utilice una base de datos híbrida, fusionando tecnologías relacionales y no relacionales para gestionar el dinero de forma segura, centralizar todas las estadísticas de los jugadores y optimizar el sistema de reportes. La cuestión central del proyecto es: ¿Cómo se puede diseñar e implementar una base de datos híbrida que administre adecuadamente el dinero, las estadísticas y los reportes en un juego como Valorant, mejorando la experiencia del jugador y minimizando el uso de herramientas externas?

2. Objetivos

2.1. Objetivo General

Diseñar y crear un sistema de bases de datos mixto, combinando lo relacional y lo no relacional, para manejar con eficacia las operaciones financieras, los datos de la competencia y los informes de la comunidad en un juego online como Valorant, optimizando así la vivencia del jugador y disminuyendo la necesidad de programas ajenos.

2.2. Objetivos Específicos

- Diseñar una arquitectura de base de datos híbrida que combine motores relacionales para operaciones críticas y no relacionales para consultas analíticas masivas, asegurando escalabilidad, tolerancia a fallos y baja latencia.
- Integrar un sistema de estadísticas internas. Los jugadores podrán revisar datos detallados, como su puntería, rendimiento en cada mapa o qué personajes usan más.
- Crear una herramienta para manejar los reportes de errores, trampas y mal comportamiento. Esta herramienta organizará y priorizará los reportes, e informará a los usuarios sobre el estado de sus reportes.
- Incluir técnicas de replicación, partición y almacenamiento en caché para asegurar que el sistema funcione bien, incluso cuando haya muchos jugadores usándolo a la vez.
- Realizar pruebas para ver qué tan bien funciona el sistema, simulando situaciones con muchos usuarios para asegurar que todo funcione correctamente.

3. Requisitos del Sistema

3.1. Requisitos Funcionales

- **Gestión de jugadores**: El sistema debe permitir registrar, actualizar y consultar información de los jugadores, incluyendo, nivel, rango y país.
- **Gestión de partidas**: Debe registrar partidas jugadas, incluyendo mapa, fecha, duración, equipos participantes y estadísticas individuales por jugador.
- **Gestión de equipos**: El sistema debe permitir la creación y consulta de equipos conformados por cinco jugadores.
- **Registro de estadísticas**: Debe almacenar estadísticas detalladas como kills, muertes, asistencias, y desempeño por agente y por ronda.
- **Gestión de transacciones**: El sistema debe permitir registrar compras y recargas de VP (valorant points), asociadas a métodos de pago y tipos de transacción.
- **Manejo de incentivos y acceso cada día:** La plataforma dará y hará visibles bonificaciones por entrar al juego a diario.

- **Identificación y guardado de fallos al pagar:** Es importante guardar los errores que surjan al mover dinero, para revisarlos después.
- **Manejo de quejas sobre otros jugadores:** Se debe poder reportar malas conductas de otros usuarios, con razón y pruebas.
- **Acceso a estadísticas agregadas**: Debe ofrecer resúmenes de desempeño, historial de agentes y consultas agregadas sobre el comportamiento de los jugadores.

3.2. Requisitos no Funcionales

- **Rendimiento**: Lo ideal es que el sistema conteste a las preguntas más frecuentes en la mayoría de las situaciones, tardando menos de 2 segundos si la carga es la habitual.
- **Escalabilidad**: Tiene que poder trabajar con cantidades enormes de datos mediante el uso de fragmentación, particiones y bases distribuidas.
- **Disponibilidad**: El sistema debe estar disponible la mayor parte del tiempo, considerando estrategias de respaldo y recuperación.
- **Mantenibilidad**: El código y la arquitectura del sistema han de ser fáciles de entender y estar en módulos, lo que hará más sencillas las mejoras.
- **Usabilidad**: Debe ser fácil de usar para usuarios con conocimientos técnicos y administradores, ayudando a consultar datos e interpretar cifras.
- **Consistencia de datos**: Debe asegurar la integridad referencial en operaciones críticas como registros de partidas o transacciones.
- **Tolerancia a fallos**: Debe registrar adecuadamente los errores que puedan surgir, especialmente en transacciones monetarias, permitiendo su análisis posterior.

4. Diseño Conceptual

4.1. Modelo Conceptual Entidad-Relación

- **Modelo de la base de datos de PostgreSQL**
  Diagrama Modelo Entidad Relación PostgreSQL
- **Modelo de la base de datos de MySQL**

Diagrama Modelo Entidad Relación MySQL

4.2. Modelo Relacional (Tablas, campos, PK, FK)

- Modelo relacional de PostgreSQL

  Diagrama Relacional PostgreSQL
- Modelo relacional de MySQL

  Diagrama Relacional MySQL

4.3. Normalización hasta 3FN

Primera Forma Normal (1FN): Ambas bases de datos, la de PostgreSQL y la de MySQL, cumplen con la 1FN al eliminar datos repetidos y asegurar que cada atributo contenga valores atómicos. Por ejemplo, en la tabla players no se repiten datos de jugadores, y en purchases los campos item_name y item_type se mantienen como valores indivisibles, evitando listas o grupos.
Segunda Forma Normal (2FN): Todas las dependencias funcionales dependen de la clave primaria. En la base de Valorant, la tabla player_stats depende totalmente de player_id y de match_id, mientras que en la base de transacciones, wallet_transactions se basa en wallet_id, transaction_id y en vp_package_id.
Tercera Forma Normal (3FN): No existen dependencias transitivas no clave en ninguna de las bases. En Postgres, por ejemplo atributos como country y regions en players se gestionan en tablas independientes (country y regions), evitando que dependan transitivamente de otros atributos. En la base de transacciones, description en payment_methods se separa si depende de method_name lo cual asegura que no haya redundancias.

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

Colecciones embebidas: Son las de player_feedback, player_agents_performance, player_issues, player_settings, y player_reports, y esto es porque contienen datos que se relacionan directamente con un documento principal el cual es el player y se almacenan dentro del mismo documento. Esto reduce la necesidad de consultas externas, mejora el rendimiento al acceder a datos relacionados como en el caso de la consulta de retroalimentación de un jugador.
Colecciones referenciales: En este caso las colecciones serían las de leaderboard_snapshots, daily_login_rewards, match_rounds, match_messages, y player_stats son referenciales porque almacenan datos que pueden ser

compartidos o consultados independientemente, como estadísticas globales o recompensas diarias. Se usan referencias que son las ids para vincularlas con player, permitiendo flexibilidad y escalabilidad, ya que un mismo dato como en el caso de la obtención del líder, puede relacionarse con múltiples jugadores sin duplicación.
Propósito: El diseño embebido sirve para optimizar consultas frecuentes y mantener datos relacionados juntos, mientras que el referencial lo que permite es manejar relaciones más dinámicas y reducir redundancia lo cual permite la escalabilidad y consultas específicas en MongoDB.

4.5. Diagrama de flujo del ETL

Diagrama de flujo ETL

4.6. Diseño de la Base de Datos Snowflake

El diseño del Snowflake se hizo para satisfacer cuatro reportes clave los cuales son: desempeño promedio de jugadores por agente, jugadores con más transacciones exitosas, rendimiento de equipos en partidas específicas y análisis de compras por método de pago. Estos reportes se eligieron por su capacidad para optimizar y facilitar consultas analíticas y manejar relaciones complejas entre datos.

El esquema incluye dos tablas de hechos: Fact_Player_Stats y Fact_Transactions. Fact_Player_Stats almacena métricas de rendimiento que son las de kills, deaths, assists, damage y headshots las cuales están vinculadas a jugadores, agentes, equipos y partidos, esto permite el análisis de desempeño y rendimiento de equipos. Fact_Transactions registra datos de transacciones (amount, status) relacionadas con jugadores, paquetes VP, métodos de pago y tipos de transacción, facilitando el reporte de compras y transacciones con éxito.

Las tablas dimensionales las cuales son: Dim_Matches, Dim_Agents, Dim_Teams, Dim_Players, Dim_VP_Packages, Dim_Payment_Methods y Dim_Transaction_Types se desnormalizan en un modelo de copo de nieve para reducir redundancia y mejorar el rendimiento. Por ejemplo, Dim_Players se crece y se conecta con referencias a Dim_Country y Dim_Regions, mientras que Dim_Teams incluye múltiples jugadores, reflejando las relaciones jerárquicas. Esto permite consultas eficientes al descomponer datos en niveles más detallados, como para el desempeño por agente o el análisis por método de pago.

El proceso comenzó identificando las necesidades de los reportes, definiendo las tablas de hechos con métricas clave y creando dimensiones que soportan filtros y agrupaciones. Las relaciones se modelaron con claves foráneas para conectar hechos con dimensiones lo cual permite la escalabilidad y también flexibilidad.

5. Diseño de la Base de Datos

5.1. Modelo NoSQL

Modelo NoSQL

5.2. Estructura de documentos y colecciones

5.2.1. Documentos Embebidos

- **Historial de recompensas diarias de inicio de sesión**

  Documentos individuales que mantienen un registro exhaustivo de las recompensas diarias que se proporcionan a los jugadores en función de su actividad de inicio de sesión. Cada registro incluye la fecha y los Puntos de Victoria (PV) otorgados, lo que permite una revisión rápida de la regularidad del jugador y la implementación de beneficios.
- **Capturas diarias de tablas de clasificación**

  Documentos que muestran la posición de los jugadores en un día específico, abarcando su clasificación, eliminaciones, muertes y juegos realizados. Este formato permite conservar el historial del ranking para análisis futuros o elaboración de informes históricos sin necesidad de cálculos complejos en tiempo real.
- **Mensajes en partidas (Match Messages)**

  Cada documento representa una partida específica y alberga una lista de los mensajes intercambiados entre los jugadores durante esa partida, indicando quién envía cada mensaje y la hora precisa. Resulta útil para la moderación, el examen de la comunicación durante el juego y posibles informes sobre la conducta.
- **Rondas de una partida (Match Rounds)**

  Documentos que retienen información detallada sobre cada ronda en una partida, incluyendo el número de la ronda, el equipo que ganó, el equipo que perdió, la

ubicación del spike, si este fue colocado y la duración de la ronda. Esta disposición, que refleja la mecánica del juego Valorant, facilita el acceso rápido a los resultados por ronda.

- **Reportes contra jugadores (Reported Players)**

  Por cada jugador que recibe un reporte, se guarda un archivo con una lista de los reportes recibidos, incluyendo quién reportó, el motivo (como trampas o comportamiento tóxico), la fecha y la evidencia (clips, registros de chat). Facilita la gestión de sanciones y las revisiones disciplinarias.

- **Configuraciones personales de jugadores (Player Settings)**

  Documentos que registran las preferencias de un jugador, tales como el idioma, la calidad gráfica, el modo de pantalla completa y las configuraciones de notificación. La estructura jerárquica permite realizar ajustes personalizados para mejorar la experiencia de juego.

5.2.2. Colecciones referenciales

- **Rendimiento de jugadores por agente y partidas**

  Aquí se guarda cómo le fue a cada jugador usando un agente específico en cada partida: cuántas kills hizo, cuántas veces murió y cuántas asistencias dio. Esta info es súper útil para ver qué tan bueno es alguien con un agente, comparar jugadores entre sí y hacer análisis más pro del juego.

- **Retroalimentación y comentarios de jugadores**

  Es el espacio donde los jugadores pueden dejar sus opiniones sobre el juego. Pueden hablar del lag, si los mapas están bien diseñados, o cualquier cosa que les moleste o les guste. También se guarda la fecha y el tipo de comentario. Sirve para que el equipo del juego sepa en qué mejorar.

- **Estadísticas detalladas por ronda y jugador**

  Aquí se anota lo que hace cada jugador en cada ronda: kills, muertes, headshots, daño causado, etc. Es como tener una lupa para analizar el rendimiento en detalle. Muy útil para entrenar, hacer reportes o ver cómo mejorar estrategias.

- **Tickets de soporte y comunicación (Support Tickets)**

Esto es para cuando un jugador tiene algún problema con el juego. Se guarda qué tipo de problema tuvo, en qué estado está (si ya se solucionó o no) y todas las conversaciones que tuvo con el equipo de soporte. Básicamente, es el sistema para ayudar a los jugadores cuando algo falla.

6. Arquitectura y Justificación Técnica

6.1. Arquitectura general del sistema

Arquitectura General del Sistema

6.2. Tecnologías seleccionadas y justificación

- **PostgreSQL:** Lo seleccionamos como nuestra base de datos relacional principal para tareas vitales: registro de partidas, estadísticas de jugadores y operaciones económicas. Su solidez en la integridad referencial, junto a su habilidad con SP, triggers y vistas, lo hacen ideal para asegurar la coherencia y eficiencia en las operaciones ACID.
- **MySQL:** La empleamos para la gestión de datos económicos, abarcando las transacciones de puntos de Valorant (VP) y los fondos de los usuarios. Su función de réplica maestro-esclavo y su soporte para transacciones ACID aseguran alta disponibilidad y escalabilidad en la gestión de pagos.
- **MongoDB:** La elegimos como base de datos NoSQL para el almacenamiento de datos semiestructurados como las configuraciones de jugadores, informes y datos estadísticos. Su estructura basada en documentos permite realizar consultas rápidas y flexibles, optimizando el análisis del rendimiento y la gestión de informes.
- **Redis:** La implementamos como sistema de caché para acelerar las consultas más críticas, como los saldos de cuentas, los paquetes de VP y las clasificaciones. Su arquitectura en memoria y su compatibilidad con TTL (Time To Live) mejoran la latencia, crucial para un rendimiento óptimo en momentos de alta demanda.
- **Docker Compose:** Lo usamos para el despliegue y la organización de la arquitectura distribuida, que comprende PostgreSQL, MySQL, MongoDB y Redis. Facilita la escalabilidad, la persistencia de los datos y la simulación de entornos distribuidos, como el sharding y la réplica. Para satisfacer la creciente demanda mundial y reducir los tiempos de espera en Valorant, optamos por la fragmentación de la base de datos. Esta estrategia, que implica la división horizontal con PostgreSQL y la distribución de datos según la ubicación geográfica (como LA y EU), facilita un acceso más rápido a la información para los jugadores cercanos. Así, el juego responde de forma más ágil y la carga se distribuye entre los servidores, previniendo saturaciones y permitiendo que el sistema crezca sin problemas a medida que se suman usuarios.
  Por otro lado, utilizamos MySQL con una arquitectura maestro-esclavo: un servidor central se encarga de registrar los cambios (compras o actualizaciones de cuentas), mientras que los servidores secundarios atienden las consultas y generan informes. De

esta manera, el sistema está siempre disponible, las operaciones importantes se mantienen coherentes y el rendimiento mejora al permitir que múltiples usuarios consulten datos sin afectar al servidor principal. Además, esta configuración ofrece respaldo y resiliencia ante fallos, garantizando que el juego siga funcionando incluso si el servidor maestro falla.

7. Implementación

7.1. SP, Views, Triggers, Functions y Índices

7.1.1. SP's

Postgres:

- **Assign players to team:** Para crear los equipos, el sistema junta a cinco jugadores únicos, verificando que cada identificación sea válida y figure en la base de datos, gracias a las relaciones entre tablas.
- **Delete player:** Esta función permite borrar jugadores sin riesgo, confirmando primero que existan y que no estén participando en ningún equipo en ese momento. Después, elimina automáticamente toda la información vinculada, protegiendo la consistencia de la información, algo crucial para cumplir con las leyes de privacidad y mantener ordenada la información del juego.
- **Insert player:** Cuando se registra un jugador nuevo, se verifica que ni su correo electrónico ni su nombre de usuario estén ya en uso antes de guardar la información en las tablas de jugadores y datos adicionales, todo al mismo tiempo.
- **Registrar partida:** Este es un proceso detallado que guarda todos los datos de una partida, incluyendo el mapa, la duración, qué equipos ganaron y perdieron, y las estadísticas de cada jugador. Analiza listas de estadísticas repetidamente para guardar la información en las tablas de partidas, resultados y estadísticas de los jugadores, siendo la base del sistema para analizar el rendimiento y el historial de las partidas del juego.

MySQL:

- **Realizar compra vp:** Se encarga de todo el proceso de adquisición de Puntos Valorant (VP). Confirma el paquete seleccionado, consulta el saldo del usuario, inicia la operación con el método de pago elegido, actualiza el saldo con los VP adquiridos y sus bonificaciones, y guarda un registro detallado en wallet_transactions.

- **Refund transaction:** Tramita las devoluciones de operaciones de VP. Recupera los datos de la transacción original, reintegra los VP al saldo del usuario y marca la transacción como reembolsada. Ofrece una buena gestión de errores con rollback automático, siendo fundamental para la atención al cliente.

7.1.2. Views

Postgres:

- **Map player stats:** Se entrelazan los datos del jugador con detalles importantes del juego, tales como el nombre de usuario, la fecha del partido, el mapa en uso, el personaje escogido, la función del personaje y los números individuales del jugador. Esto ayuda a estudiar el juego para equilibrar personajes y mapas, sistema de sugerencias de personajes a la medida del jugador, generación de mapas de calor de cómo rinde cada uno en cada mapa.
- **Estadísticas jugadores:** Calcula medidas completas del rendimiento de cada jugador, todos los números acumulados y los promedios de KDA, ofreciendo una mirada completa de su trayectoria. Permite el sistema de clasificación automática de los jugadores, encontrar talentos para equipos profesionales, generar perfiles públicos para la comunidad, estudiar el progreso de los jugadores para sistemas de entrenamiento automático, encontrar jugadores que ya no están activos para intentar que vuelvan a jugar, y sirve como base de datos para algoritmos de aprendizaje automático que predicen el potencial competitivo de nuevos jugadores.

MySQL:

- **Errores transacciones:** Un registro detallado que une los errores en las transacciones con los datos del usuario afectado. Muestra el identificador del error, la transacción involucrada, el usuario y la descripción del fallo. Esta perspectiva facilita encontrar tendencias fraudulentas, señalar usuarios conflictivos, activar avisos automáticos al equipo de seguridad.
- **Transacciones completas:** Datos completos de las transacciones, incluyendo información del usuario, fechas, importes, formas de pago y tipos de transacción con nombres claros en vez de identificadores numéricos. Esto simplifica la creación de informes financieros para la dirección, paneles de control con

métricas clave del negocio, análisis de tendencias en formas de pago para negociar con proveedores, detección de irregularidades en los hábitos de compra.

7.1.3. Triggers

Postgres:

- **Validate email:** Verifica la unicidad del email antes de insertar o actualizar registros, previniendo duplicados en el sistema de usuarios. Asegura integridad referencial de cuentas de usuario, previene problemas de autenticación por emails duplicados, facilita recuperación de contraseñas.
- **Validate team:** Valida la formación de equipos verificando que todos los jugadores existan y sean únicos dentro del mismo equipo antes de insertar o actualizar. Garantiza equipos válidos para el matchmaking, previene errores en la lógica de juego por equipos, y mantiene la integridad de datos para algoritmos de balanceo de equipos.
- **Actualizar rangos al registrar match:** Ajusta automáticamente los rangos de todos los jugadores participantes después de registrar una partida, subiendo rango al equipo ganador y bajando al perdedor usando las funciones auxiliares subir_rango y bajar_rango. Automatiza el sistema de ranking competitivo, mantiene la progresión de jugadores actualizada en tiempo real.

MySQL:

- **Actualizar estado fallo:** se ejecuta después de actualizar transacciones, detectando cambios de estado a 'fallido' y registrando automáticamente errores cuando el método de pago es tarjeta de crédito. Permite auditoría automática de fallos de pago, generación de alertas para el equipo de soporte financiero, análisis de patrones de rechazo por proveedor de payment gateway, y construcción de perfiles de riesgo de usuarios para sistemas de prevención de fraude.
- **Complete transaction on wallet transactions:** Activado tras insertar registros en wallet_transactions que actualiza automáticamente el estado de la transacción padre', asegurando consistencia entre el procesamiento de VP y el estado transaccional. Garantiza integridad de datos en el flujo de compras.
- **Update wallet balance:** Actualiza automáticamente el balance del wallet cuando se registra una nueva wallet_transaction, sumando el monto de VP al saldo

existente. Mantiene sincronización en tiempo real entre transacciones y balances, elimina inconsistencias de datos por operaciones concurrentes, y asegura que los reportes de saldos sean siempre precisos para decisiones de negocio.

7.1.4. Functions

Postgres:

- **Siguiente rango y Bajar rango:** Componentes que dirigen el avance gradual del nivel de los participantes, explorando la estructura de niveles con 'rank_order' y modificando automáticamente el nivel con registro de fecha y hora. Dinamizan el esquema competitivo del juego, sostienen el impulso por mejorar, potencian los algoritmos para emparejamientos equilibrados y brindan datos de interacción para estudiar la fidelización de los jugadores.
- **Actualizar estadisticas jugador:** UPSERT (renueva datos actuales de un jugador en una partida específica o los añade si faltan), gestionando bajas, muertes y asistencias. Permite reparar datos de partidas, conexión con sistemas externos de rastreo, actualización de datos en tiempo real durante retransmisiones, y conserva flexibilidad para ajustes posteriores a la partida por parte de los administradores.
- **Average kills by rank and region:** Calcula el promedio de bajas por cruce de nivel y zona, proveyendo datos de rendimiento separados geográficamente. Simplifica el balanceo de servidores regionales, análisis de tendencias por zona para ajustes específicos, comparación de nivel entre diferentes mercados, y creación de perspectivas para tácticas de expansión geográfica del juego.
- **Get players performance metrics:** Devuelve una tabla completa de datos de rendimiento incluyendo KDA, ACS estimado y MMR calculado, ofreciendo un perfil analítico completo del jugador. Impulsa paneles de control de rendimiento individual, sistemas de entrenamiento automatizado, algoritmos de detección de cuentas smurf, análisis de evolución para programas de desarrollo de talentos y creación de certificaciones de habilidad para torneos.
- **Get player win rate:** Determina datos estadísticos de victoria incluyendo total de partidas, victorias, derrotas y porcentaje de victoria, examinando la participación del jugador en equipos ganadores y perdedores. Ofrece métricas cruciales para sistemas de clasificación, identificación de jugadores decisivos, análisis de

constancia de rendimiento y base para algoritmos de predicción de resultados en apuestas de esports.

- **Player yearly analysis:** Produce un informe anual completo del jugador incluyendo promedios de datos, mapa y agente más utilizados, ofreciendo una retrospectiva del año. Posibilita informes anuales personalizados tipo "Spotify Wrapped", análisis de evolución de preferencias de tendencias, identificación de especializaciones de jugadores y creación de contenido personalizado para redes sociales y marketing.

MySQL:

- **Tiene saldo suficiente**: Examina si un cliente tiene el capital necesario en su cuenta para llevar a cabo una compra, cotejando el importe requerido con el dinero que tiene. Permite confirmar las compras antes de hacer los cobros, evita fallos por falta de fondos al pagar, simplifica añadir productos al carrito con control al instante, y es útil para sugerir ofertas VP según el presupuesto del cliente.
- **Total gastado por usuario:** Determina el total gastado por un cliente juntando todas sus compras anteriores, mostrando cero si no hay compras hechas. Posibilita examinar el valor del cliente a lo largo del tiempo, agrupar a los clientes según cuánto gastan, crear programas de lealtad a la medida, identificar a los grandes compradores para ofertas exclusivas, y formar patrones que anticipen cómo comprarán los usuarios/jugadores.

7.1.5. Índices

Postgres:

- **Matches date:** Sobre fechas de partidas que acelera consultas con filtros temporales como reportes anuales, mensuales o por temporadas. Mejora significativamente las funciones de análisis temporal (player_yearly_analysis) y vistas que necesitan filtrar partidas por períodos específicos, reduciendo el tiempo de búsqueda en tablas grandes de historial de partidas.
- **Player stats match player:** Optimiza la relación entre jugadores y sus estadísticas de partida, acelerando consultas que calculan métricas de rendimiento

individual. Beneficia directamente a funciones como get_player_performance_metrics() y vistas de estadísticas que necesitan agrupar datos por jugador.

- **Players id username:** Optimizando tanto búsquedas por ID como por username. Acelera operaciones de autenticación, búsquedas de jugadores, y consultas que necesitan mostrar nombres de usuario en rankings o leaderboards.
- **Pps players match:** Para consultas de enfrentamientos directos entre jugadores específicos, optimizando análisis de rivalidades head-to-head. Permite generar rápidamente historiales de enfrentamientos, estadísticas de matchups, y análisis competitivos entre jugadores.

7.2. Transacciones ACID

- **Atomicidad:** Las transacciones garantizan que todas las operaciones se ejecuten completamente o no se ejecuten en absoluto. En realizar_compra_vp (MySQL), si falla cualquier paso del proceso de compra (validación de paquete, actualización de wallet, o registro de transacción), toda la operación se revierte mediante ROLLBACK automático.
- **Consistencia:** Los triggers y stored procedures mantienen la integridad referencial y reglas de negocio. El trigger trg_validate_email asegura que no existan emails duplicados, mientras que validate_team garantiza equipos válidos con 5 jugadores únicos.
- **Aislamiento:** El sistema maneja la concurrencia mediante niveles de aislamiento adecuados. Las transacciones de compra (realizar_compra_vp) están aisladas para prevenir condiciones de carrera cuando múltiples usuarios compran simultáneamente.
- **Durabilidad:** Una vez confirmadas, las transacciones persisten permanentemente en la base de datos. Los COMMIT explícitos en stored procedures como sp_delete_player y realizar_compra_vp garantizan que los cambios se escriban físicamente al storage. El sistema de triggers complementa la durabilidad al registrar automáticamente cambios críticos (estados de transacciones, actualizaciones de saldo) que quedan permanentemente almacenados.

7.3. Backups

7.3.1. Backup Automatizado

Para resguardar la información, el esquema que se ha puesto en marcha efectúa copias de seguridad automáticas cada día para las dos bases de datos, valiéndonos de tareas cron programadas para la medianoche. En MySQL, se usa mysqldump para crear ficheros SQL íntegros, mientras que en PostgreSQL se emplea pg_dump con un formato custom comprimido. Los dos scripts operan de manera similar: activan el comando de respaldo dentro del contenedor Docker, crean archivos con una marca de tiempo única y los transfieren al sistema anfitrión en carpetas estructuradas. La configuración a través de variables de entorno mantiene la seguridad de las credenciales, y la gestión de errores garantiza que cualquier problema durante el proceso quede registrado de forma correcta para su seguimiento operativo.

7.3.2. Restore Inteligente

Para reactivar la información, se optó por ubicar la copia de seguridad más actual según la fecha y llevar a cabo una recuperación total. El mecanismo es muy similar para las dos bases de datos: localizan el archivo de backup más nuevo, lo transfieren al espacio designado y lanzan la orden de recuperación pertinente. Con MySQL, hay que ajustar algunos parámetros extra para que los procedimientos almacenados funcionen bien, mientras que PostgreSQL emplea la función clean para prevenir problemas al restaurar. Gracias a esta automatización, se minimizan los errores humanos en momentos delicados, asegurando así que la recuperación de datos sea ágil y segura en caso de necesidad.

7.4. Base de Datos Distribuidas

7.4.1. Master Slave

Implementa replicación MySQL con un nodo maestro para todas las operaciones de escritura (transacciones VP, compras, actualizaciones de wallets) y dos nodos

esclavos para operaciones de lectura (reportes, consultas analíticas, dashboards). Esta arquitectura garantiza alta disponibilidad del sistema financiero, distribuye la carga de consultas entre múltiples nodos, y proporciona backup automático.

Diagrama Master Slave

7.4.2. Sharding

Particionamiento horizontal por región geográfica con dos instancias PostgreSQL independientes (LA y EU) que manejan datos específicos de jugadores según su ubicación.

Diagrama Sharding

7.5. Redis (Cache)

7.5.1. Arquitectura de Cache Implementada

Se utiliza Redis para acelerar las operaciones más críticas del negocio: consultas de paquetes VP, verificación de balances de wallets, métodos de pago y rankings de jugadores. Cada tipo de dato usa estructuras Redis específicas: Hash para paquetes VP permitiendo consultas rápidas por ID, String para balances con actualizaciones frecuentes, Set para métodos de pago como opciones fijas, y Sorted Set para rankings con ordenamiento automático por nivel y filtros por país usando Sets.

7.5.2. Estrategia de TTL y Perfomance

Los tiempos de vida (TTL) se configuran considerando qué tan importantes son los datos y con qué frecuencia cambian: paquetes VP (6 horas) buscando un equilibrio en la consistencia de los precios, saldos de billetera (6 horas) para asegurar la exactitud financiera, formas de pago (24 horas) debido a que no varían mucho, clasificaciones (24 horas) para reflejar las jerarquías más recientes, y jugadores por país (1 hora) para filtros geográficos que se actualizan constantemente. Esta implementación en MySQL/PostgreSQL disminuye considerablemente la presión sobre los procedimientos almacenados, por ejemplo, realizar_compra_vp, y las funciones, como tiene_saldo_suficiente.

7.6. Consultas en MongoDB

- **Total de rondas y duración promedio por partida:** Analiza match_rounds calculando rondas totales y duración promedio por partida. Fundamental para identificar partidas competitivas extensas, optimizar balanceo de mapas, ajustar tiempos de matchmaking y detectar patrones de engagement dónde partidas largas se correlacionan con mayor satisfacción del jugador.
- **Jugadores con más de 1 kill y daño > 100 en una ronda:** Filtra jugadores con rendimiento superior (>1 kill, >100 daño) enriqueciendo con datos del match. Identifica clutch players destacados, detecta talentos competitivos, ajusta matchmaking por skill excepcional.
- **Promedio de daño y kills por jugador con configuración gráfica:** Correlaciona estadísticas de rendimiento con configuraciones gráficas y de idioma del jugador. Optimiza requisitos del sistema, identifica configuraciones que proporcionan ventajas competitivas, personaliza recomendaciones técnicas y detecta problemas de rendimiento específicos por configuración.
- **Total de picks por agente:** Cuenta selecciones totales por agente en agent_performance_history ordenado por popularidad. Esencial para balanceo del meta, identificando agentes dominantes que requieren ajustes, agentes infrautilizados para buffs y métricas de adopción de personajes nuevos.
- **Top 3 agentes más usados:** Limita la consulta anterior a los tres agentes más populares para snapshot del meta actual. Ayuda a generar contenido de marketing destacando favoritos de la comunidad, análisis competitivo rápido para esports e identificación de la "trinity" del meta para balanceo prioritario.
- **Configuración del jugador con más daño promedio:** Identifica al jugador top en daño y revela sus configuraciones específicas (idioma, gráficos, pantalla completa). Crea guías de configuración basadas en jugadores élite, identifica setups óptimos para rendimiento competitivo y genera contenido educativo para aspirantes.
- **Total de mensajes por jugador por partida:** Analiza actividad de comunicación calculando mensajes totales y longitud promedio por jugador/partida.

Fundamental para sistemas anti-toxicidad, análisis de engagement social, detección de líderes que coordinan equipos y optimización de sistemas de chat.

- **Partidas con más de 1 mensaje y jugadores únicos comunicativos:** Analiza match_chat_logs contando mensajes totales y jugadores únicos que participaron en comunicación por partida, filtrando solo partidas con actividad de chat. Identifica partidas con alta interacción social, mide engagement comunicativo del lobby, detecta matches "silenciosos" vs "comunicativos" para análisis de experiencia de usuario y optimiza sistemas de matchmaking considerando preferencias de comunicación.
- **Total de VP ganados en recompensas por jugador:** Procesa daily_login_rewards sumando VP total obtenido por cada jugador y contando días con recompensas. Fundamental para análisis de monetización identificando jugadores que acumulan más rewards gratis, balanceo del sistema de recompensas para no impactar ventas, segmentación de jugadores por engagement diario y optimización de ofertas basadas en acumulación de VP gratuito
- **Fechas únicas y total de días con recompensa por jugador:** Calcula días únicos de login con recompensa. Mide consistencia de engagement, identifica jugadores en riesgo por login irregular, optimiza calendarios de eventos para maximizar participación y genera métricas de retención basadas en frecuencia de login diario.
- **Jugadores con mayor daño promedio por ronda (con idioma)**: Combina estadísticas de daño promedio con configuración de idioma del jugador desde user_settings. Analiza correlación entre configuración regional y skill level, identifica mercados geográficos con jugadores más competitivos, optimiza localización del juego según rendimiento.
- **Total de VP ganado y días logueados por jugador:** Proporciona dashboard limpio de actividad de usuarios, identifica jugadores más activos para programas VIP, analiza patrones de login para optimización de eventos diarios.
- **Promedio de mensajes por jugador con configuración detallada:** Cruza actividad de chat con configuraciones gráficas y de idioma del jugador. Identifica correlación entre configuraciones técnicas y participación social, optimiza sistemas de chat según capacidades del hardware.

- **Número de rondas ganadas por cada equipo:** Analiza victorias totales por equipo desde match_rounds para detectar desbalanceos competitivos. Fundamental para balanceo de sides en mapas (atacante vs defensor), identificación de mapas que favorecen ciertos equipos.
- **Jugadores con más asistencias totales por agente:** Analiza agent_performance_history sumando asistencias totales por combinación jugador-agente. Identifica especialistas en roles de soporte, detecta jugadores que maximizan teamwork con agentes específicos.
- **Conteo de tipos de issues en soporte (open vs closed vs pending):** Cuenta tickets de soporte agrupados por estado desde support_tickets. Fundamental para gestión operacional del equipo de soporte, identificación de cuellos de botella en resolución de tickets.
- **Jugadores con feedback de tipo "bug" y sus configuraciones:** Filtra reportes de bugs desde feedbacks y los enriquece con configuraciones técnicas del usuario. Identifica problemas específicos por configuración de hardware/software, prioriza fixes basados en configuraciones más afectadas.
- **Estadísticas promedio por ronda por agente:** Combina player_round_stats con agent_performance_history para calcular métricas promedio de kills y daño por agente. Esencial para balanceo de personajes, identificación de agentes sobrepowered que requieren nerfs, análisis de efectividad de agentes.
- **Jugadores con problemas de conexión o rendimiento:** Filtra tickets de soporte por issues técnicos específicos y correlaciona con configuraciones de usuario. Identifica patrones de problemas técnicos por configuración.
- **Recompensas acumuladas por jugadores:** Calcula VP total acumulado en recompensas diarias filtrando solo jugadores con configuración de idioma inglés. Analiza el engagement por región lingüística, optimiza estrategias de monetización segmentadas geográficamente.

7.7. Big Data, ETL, Diagrama Snowflake

En la base de datos de PostgreSQL las tablas que hacen uso del big data con 30000 datos son las de: matches, matches_result, player_player_stats, player_stats y la tabla de teams con 15000 datos. En cuanto a la base de datos de MySQL, las tablas que tienen big data es la de transactions

El proceso ETL diseñado para el proyecto se puede visualizar en el diagrama y se implementó para cargar datos de múltiples fuentes hacia un archivo CSV. El flujo tiene tres etapas:

- **Extracción:** Los datos se extraen de diversas fuentes, incluyendo PostgreSQL, MySQL, MongoDB y Redis, que contienen información importante como estadísticas de jugadores, transacciones y configuraciones. Esta etapa recolecta los datos en su forma original para después procesarlos.
- **Transformación:** Se eliminaron columnas vacías, se limpiaron las transacciones o pagos que estaban completadas, se removieron valores nulos y se eliminaron duplicados.
- **Carga (CSV):** Finalmente, los datos transformados se cargan directamente en un archivo CSV en lugar de una base de datos lo cual facilita la obtención de la información para los reportes. Esto simplifica el acceso y la manejabilidad de los datos procesados.

Diagrama SnowFlake

7.8. Docker Compose

7.8.1. Raíz del proyecto

Configuración principal que despliega la arquitectura base del sistema con PostgreSQL y MySQL en contenedores separados. Incluye inicialización automática de esquemas, persistencia de datos mediante volúmenes nombrados, y configuración de puertos estándar para desarrollo local y pruebas de integración.

7.8.2. Cache

Despliega Redis como servicio de caché, autenticación por password y configuración optimizada para alta disponibilidad. Fundamental para el sistema ya que acelera consultas críticas de paquetes VP, balances de wallets y rankings de jugadores.

7.8.3. Master Slaves

Configura replicación maestro-esclavo MySQL con un nodo principal para escrituras y dos réplicas para lecturas, incluyendo monitoring con Prometheus exporter. Proporciona alta disponibilidad para el sistema financiero, distribución de carga de consultas transaccionales y backup automático.

7.8.4. Mongo

Despliega MongoDB con autenticación habilitada para almacenamiento de datos NoSQL como logs de chat, configuraciones de usuario, reportes de feedback y análisis de comportamiento. Complementa la arquitectura relacional con capacidades de almacenamiento flexible para datos semi-estructurados.

7.8.5. Sharding

Implementa arquitectura de sharding horizontal con dos instancias PostgreSQL separadas (LA y EU) simulando distribución geográfica. Cada instancia maneja datos específicos por región, mejorando latencia y escalabilidad para usuarios distribuidos globalmente.

8. Pruebas y Evaluación de Resultados

8.1. Demostración de SP, Backups, Explain (Que respondan objetivos)

- PostgreSQL:
  SP_registrar_partida
- MySQL:
  SP_realizar_compra_vp
  SP_refund_transaction
- MongoDB:
  Jugadores con mayor daño promedio por ronda (con nombre de jugador)
  Conteo de tipos de issues en soporte
- Backup:
  Ofuscamiento
  Backup
- MasterSlave
- Sharding_LA_EU
- Cache
- Índices
  Índice player (id, username)
  Índice matches (matches)

8.2. Resultados en relación a los objetivos

Para ajustarse a la creciente demanda mundial y disminuir los tiempos de espera en Valorant, se optó por fragmentar la base de datos. Esta estrategia, que involucra la división horizontal con PostgreSQL y el reparto de datos según la ubicación geográfica (como LA y EU), facilita un acceso más veloz a la información para los jugadores cercanos. Así, el juego responde más rápido y la carga se reparte

entre los servidores, evitando saturaciones y permitiendo que el sistema crezca sin problemas a medida que se suman usuarios.

Por otro lado, se usa MySQL con una estructura maestro-esclavo: un servidor central se encarga de registrar los cambios (compras o actualizaciones de cuentas), mientras que los servidores secundarios atienden las consultas y generan informes. De esta forma, el sistema siempre está disponible, las operaciones importantes se mantienen coherentes y el rendimiento mejora al permitir que múltiples usuarios consulten datos sin afectar al servidor principal. Además, esta configuración ofrece respaldo y resistencia a fallos, asegurando que el juego siga funcionando incluso si el servidor maestro falla.

9. Conclusiones y Recomendaciones

9.1. Conclusiones del proyecto

El estudio evidenció que una base de datos combinada puede manejar las exigencias de un juego como Valorant, fusionando tecnologías relacionales y no relacionales con eficacia. La puesta en práctica mejoró el manejo de operaciones económicas, datos de jugadores e informes de la comunidad, garantizando integridad, adaptabilidad y desempeño. Las pruebas efectuadas ratifican que el sistema satisface las necesidades de un entorno de esports mundial, brindando una solución sólida que optimiza la experiencia del usuario y respalda la toma de decisiones clave para la evolución del juego.

9.2. Recomendaciones para mejoras futuras

Desde la perspectiva del desarrollo del proyecto, se recomienda profundizar en pruebas de estrés para evaluar con mayor precisión el comportamiento del sistema bajo cargas reales. Además, integrar mecanismos automatizados de respaldo y monitoreo continuo ayudaría a reforzar la confiabilidad del sistema en producción. También sería beneficioso explorar soluciones de seguridad más avanzadas, tanto cifrado de datos en reposo como en tránsito, manejo de roles y permisos, especialmente para las transacciones financieras y la información sensible de los jugadores.

En cuanto a la materia se sugiere dedicar más tiempo a la configuración de ETL, tanto práctica como teórica. Esto podría permitir comprender de mejor manera cómo implementarlo en el proyecto y conocer más acerca de este.

10. Referencias

- “Get started.” Docker Documentation. Published May 30, 2025. https://docs.docker.com/get-started/
- What is MongoDB? - Database Manual - MongoDB Docs. https://www.mongodb.com/docs/manual/

- MySQL :: MySQL 8.4 Reference Manual. https://dev.mysql.com/doc/refman/8.4/en/
- PostgreSQL 17.5 documentation. PostgreSQL Documentation. Published May 8, 2025. https://www.postgresql.org/docs/17/index.html
- Docs. Docs. https://redis.io/docs/latest/
- VALORANT. Published June 24, 2025. https://playvalorant.com/en-us/

11. Anexos

11.1. Diagramas adicionales

- Funcionamiento Básico del Cache