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 un volumen masivo de datos sobre estadísticas, transacciones y reportes, destacando la necesidad de un sistema seguro y eficiente para garantizar la integridad financiera, ofrecer estadísticas en el juego y mejorar la gestión de denuncias, reduciendo la dependencia de herramientas externas. Para abordar esto, se diseñó una base de datos híbrida que integra PostgreSQL y MySQL para operaciones críticas, MongoDB para datos analíticos, y Redis para caching, complementada con un modelo en copo de nieve y un proceso ETL que transforma y carga datos en CSV.

La implementación aseguró alta disponibilidad, escalabilidad y bajo tiempo de respuesta, validada mediante pruebas de carga. Los procedimientos almacenados facilitaron consultas avanzadas, cumpliendo con los objetivos de optimizar la experiencia del jugador y soportar un entorno de esports global. Las conclusiones confirman la efectividad de la arquitectura híbrida, recomendando la integración de inteligencia artificial para predicciones en tiempo real y análisis de streaming para enriquecer estadísticas 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

Para el año 2023, Valorant ya había superado los 20 millones de jugadores, y esa cifra no ha parado de crecer. Esto nos da una perspectiva clara del volumen masivo de información que un videojuego multijugador de renombre debe gestionar. Los servidores de Valorant necesitan operar prácticamente sin interrupción, y la integridad y accesibilidad de los datos deben estar garantizadas. Aquí es donde entra en juego el "sharding," que ayuda a manejar las diversas regiones del mundo. Además, los sistemas maestro-esclavo son cruciales para asegurar que la información esté disponible en todo momento. Las consultas son esenciales para analizar las partidas y el rendimiento de los jugadores, considerando diversas métricas en cada juego. Asimismo, para garantizar tiempos

de respuesta rápidos del servidor, vitales para el emparejamiento y la carga de partidas, se utilizan índices y vistas para optimizar el rendimiento en operaciones críticas.

1.3. Planteamiento del problema

En los eSports actuales, como Valorant, un gran reto es asegurar que el manejo del dinero sea seguro y eficaz, algo clave para los jugadores y para el negocio. Si hay fallos o tardanzas, se pierde la confianza, surgen problemas legales y los jugadores se van. Para que esto no ocurra, se necesita una base de datos sólida que pueda manejar mucha información al instante, con integridad y sin caídas.

También, muchos jugadores profesionales no tienen un sitio oficial dentro del juego para ver sus estadísticas a fondo, así que usan programas de terceros. Esto es un riesgo para su privacidad y una oportunidad perdida para que mejoren su juego dentro del entorno oficial del juego.

Otro problema importante es que el sistema actual para reportar fallos, trampas y malas conductas no funciona bien. No hay respuestas, ni priorización, ni seguimiento, lo que frustra a la gente y resta confianza en el sistema, dañando el ambiente competitivo. Por todo esto, es necesario crear una solución basada en una base de datos híbrida, que combine tecnologías relacionales y no relacionales para manejar el dinero de forma segura, tener todas las estadísticas de los jugadores en un solo lugar y mejorar el sistema de reportes. La pregunta principal del proyecto es: ¿Cómo crear e implementar una base de datos híbrida que gestione bien el dinero, las estadísticas y los reportes en un juego como Valorant, mejorando la experiencia del jugador y reduciendo 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.
- Implementar un sistema interno de estadísticas competitivas que permita a los jugadores consultar métricas avanzadas (como precisión, desempeño por mapa o uso de agentes) directamente dentro del juego.
- Desarrollar un módulo optimizado de gestión de reportes que clasifique, priorice y dé seguimiento a denuncias de bugs, trampas y conductas tóxicas, ofreciendo retroalimentación al usuario.
- Integrar mecanismos de replicación, particionamiento y caching que aseguren la disponibilidad, consistencia y rendimiento del sistema bajo alta demanda.
- Evaluar el desempeño del sistema propuesto mediante pruebas de carga y simulación

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

Esta base de datos NoSQL se ideó precisamente para la gestión completa de datos sobre partidas de Valorant. Se ha estructurado minuciosamente para guardar y facilitar la búsqueda veloz de información clave, como el rendimiento individual y colectivo de los jugadores, las dinámicas dentro del juego, los reportes de los usuarios, los ajustes personales de cada jugador, los comentarios detallados y las peticiones de soporte técnico. La configuración principal de este diseño está optimizada para potenciar las funcionalidades propias de los sistemas de esports, abarcando desde un análisis de datos exhaustivo hasta la provisión de una ayuda técnica ágil y eficaz.

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 (Daily Login Rewards)**
  Documentos por jugador que almacenan el registro histórico de recompensas diarias otorgadas en función de la actividad de inicio de sesión. Cada registro contiene la fecha y los puntos de victoria (VP) asignados. Esto facilita verificar rápidamente la continuidad de la actividad del jugador y aplicar incentivos.
- **Capturas diarias de tablas de clasificación (Leaderboard Snapshots)**
  Documentos que representan el ranking de jugadores para un día específico, con la posición, kills, muertes y partidas jugadas. El diseño permite almacenar el estado histórico del ranking para análisis posteriores o generación de reportes históricos sin cálculos complejos en tiempo real.
- **Mensajes en partidas (Match Messages)**

Cada documento representa una partida específica y contiene una lista de mensajes intercambiados entre jugadores durante la misma, con detalles de quién envió el mensaje y la marca temporal. Esto es útil para moderación, análisis de comunicación en el juego y posibles reportes de conducta.

- **Rondas de una partida (Match Rounds)**

  Documentos que contienen información detallada sobre cada ronda dentro de una partida: número de ronda, equipo ganador, equipo perdedor, sitio de plantado del spike, si se plantó el spike y duración de la ronda. Esta estructura refleja la mecánica de Valorant y permite consultas rápidas de resultados por ronda.

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

  Por cada jugador reportado, se guarda un documento con un listado de reportes recibidos, incluyendo quién reportó, la razón (por ejemplo, cheating o comportamiento tóxico), fecha y evidencia (clips, registros de chat). Facilita la gestión de sanciones y revisiones disciplinarias.

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

  Documentos que almacenan las preferencias del jugador, incluyendo idioma, calidad gráfica, modo pantalla completa y opciones de notificación. La estructura anidada facilita ajustes personalizados para mejorar la experiencia de juego.

5.2.2. Colecciones referenciales

- **Rendimiento de jugadores por agente y partidas (Player Performance)**

  Registra el rendimiento de cada jugador con un agente específico en cada partida, detallando kills, muertes y asistencias. Esta colección soporta análisis estadístico avanzado, generación de perfiles de jugador y comparaciones por agente.

- **Retroalimentación y comentarios de jugadores (Player Feedback)**

  Guarda comentarios enviados por jugadores respecto a distintos aspectos del juego, como desempeño del servidor o diseño de mapas, con tipo de feedback y fecha. Utilizado para mejoras continuas y atención al cliente.

- **Estadísticas detalladas por ronda y jugador (Player Round Stats)**

  Información precisa de desempeño por jugador en cada ronda, incluyendo bajas, muertes, headshots y daño realizado. Esencial para análisis tácticos, generación de informes y herramientas de entrenamiento.

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

  Gestiona incidencias abiertas por jugadores, registrando el tipo de problema, estado y un historial de mensajes entre jugador y equipo de soporte. Permite seguimiento eficiente de casos y mejora en la atención al cliente.

6. Arquitectura y Justificación Técnica

6.1. Arquitectura general del sistema

Arquitectura General del Sistema

6.2. Tecnologías seleccionadas y justificación

- **PostgreSQL:** Se eligió como nuestro sistema de base de datos relacional para llevar el control de funciones esenciales, como el registro de cada partida, las estadísticas de los jugadores y las operaciones financieras. Gracias a su firmeza en la integridad referencial y su compatibilidad con los procedimientos almacenados (SP), los activadores (triggers) y las vistas, resulta perfecto para asegurar la coherencia y un buen desempeño en las operaciones ACID.
- **MySQL:** Se usa para administrar la información financiera, incluyendo las operaciones con los puntos de Valorant (VP) y los monederos de los usuarios. Su capacidad para la réplica maestro-esclavo y su apoyo a las transacciones ACID aseguran una alta disponibilidad y la posibilidad de crecer en la gestión de pagos.
- **MongoDB:** Se optó por esta base de datos NoSQL para guardar datos semiestructurados como las configuraciones de los jugadores, los informes y las estadísticas analíticas. Su estructura basada en documentos permite hacer consultas de forma rápida y flexible, optimizando el estudio del rendimiento y la gestión de los informes**.**
- **Redis:** Se puso en marcha como un sistema de caché para agilizar las consultas más importantes, como los saldos de los monederos, los paquetes de VP y las clasificaciones. Su arquitectura en memoria y su compatibilidad con TTL (Time To Live) mejoran la latencia, lo cual es clave para un buen rendimiento en momentos de alta demanda.
- **Docker Compose:** Se usa para desplegar y organizar la arquitectura distribuida, que incluye PostgreSQL, MySQL, MongoDB y Redis. Facilita la expansión, la persistencia de los datos y la simulación de entornos distribuidos, como el sharding y la réplica.

  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.

7. Implementación

7.1. SP, Views, Triggers, Functions y Índices

7.1.1. SP's

Postgres:

- **Assign players to team:** Se trata de un método que forma grupos de 5 jugadores distintos, asegurándose de que cada ID sea diferente y existente en la base de datos, gracias a las claves foráneas. Incorpora un sistema de gestión de errores para casos de fallos de integridad referencial, clave para el emparejamiento y la organización de torneos.
- **Delete player:** Este método borra jugadores de forma segura, comprobando que existan y que no estén en ningún equipo activo. Luego, elimina en cascada todos los datos relacionados (estadísticas, datos personales) manteniendo la integridad referencial, algo vital para cumplir con las normas de privacidad y mantener limpios los datos del sistema.
- **Insert player:** El registro de jugadores nuevos valida que el email y el nombre de usuario no estén repetidos antes de meter los datos en las tablas players e info_players de forma atómica. Usa RETURNING para pillar el ID creado y relacionar ambas tablas, gestionando errores para evitar duplicados y asegurar la integridad referencial del sistema de usuarios.
- **Registrar partida:** Proceso complejo que registra partidas completas, incluyendo datos del mapa, duración, equipos ganador/perdedor y estadísticas individuales de los jugadores, usando parámetros JSON. Procesa arrays de estadísticas en bucle para meter datos en matches, matches_result y player_stats, siendo el centro del sistema de seguimiento del rendimiento e historial de partidas del juego.

MySQL:

- **Realizar compra vp:** Gestiona la compra completa de VP (Valorant Points), verificando el paquete elegido, obteniendo el monedero del usuario, creando la transacción con el método de pago indicado, actualizando el saldo del monedero con los VP comprados y los extras, y guardando el detalle en wallet_transactions. Tiene control de transacciones ACID con gestión de errores automática mediante rollback, algo fundamental para la monetización del sistema.
- **Refund transaction:** Procesa devoluciones de transacciones VP, recuperando la información de la transacción original, devolviendo los VP al monedero del usuario, y señalando la transacción como devuelta. Incluye una buena gestión de errores con rollback automático, siendo clave para la atención al cliente y la resolución de problemas comerciales.

7.1.2. Views

Postgres:

- **Map player stats:** Combina estadísticas de jugadores con información contextual de partidas, incluyendo username, fecha de match, mapa jugado, agente seleccionado, rol del agente y estadísticas individuales (kills/deaths/assists). Esto ayuda al análisis de meta del juego para balanceo de agentes y mapas, sistema de recomendaciones personalizadas de agentes por jugador, generación de heat maps de rendimiento por mapa, creación de perfiles de jugador para matchmaking inteligente, y desarrollo de algoritmos de predicción de rendimiento para competencias esports.
- **Estadísticas jugadores:** Calcula métricas de rendimiento consolidadas por jugador incluyendo partidas totales, estadísticas acumuladas y promedios de KDA, proporcionando una visión panorámica del desempeño histórico. Habilita el sistema de ranking y clasificación automática de jugadores, identificación de talentos para equipos profesionales, generación de perfiles públicos para la comunidad, análisis de progresión de jugadores para sistemas de coaching automatizado, detección de jugadores inactivos para campañas de retención, y funciona como base de datos para algoritmos de machine learning que predigan el potencial competitivo de nuevos jugadores.

MySQL:

- **Errores transacciones:** Auditoría que combina errores de transacciones con información del usuario afectado, mostrando ID del error, transacción relacionada, usuario y mensaje de error. Esta vista permite detectar patrones de fraude, identificar usuarios problemáticos, generar alertas automáticas para el equipo de seguridad, crear reportes de incidentes para mejorar la experiencia de pago, y realizar análisis forense de transacciones fallidas para optimizar los procesos de payment gateway.
- **Transacciones completas:** Información completa de transacciones incluyendo datos del usuario, fechas, montos, métodos de pago y tipos de transacción con nombres legibles en lugar de IDs. Facilita la generación de reportes financieros ejecutivos, dashboard de métricas de negocio, análisis de tendencias de métodos de pago para negociaciones con proveedores, detección de anomalías en patrones de compra, y sirve como base para sistemas de recomendación de productos VP basados en historial de compras.

7.1.3. Triggers

Postgres:

- **Validate email:** Verifica la unicidad del email antes de insertar o actualizar registros en info_players, 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, y mantiene la calidad de datos para sistemas de notificaciones y marketing.
- **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 mal formados, asegura fairness en competencias, 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, proporciona

motivación continua para el engagement, y alimenta algoritmos de matchmaking basados en skill rating.

MySQL:

- **Actualizar estado fallo:** se ejecuta después de actualizar transacciones, detectando cambios de estado a 'fallido' y registrando automáticamente errores en la tabla transaction_errors 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 a 'completed', asegurando consistencia entre el procesamiento de VP y el estado transaccional. Garantiza integridad de datos en el flujo de compras, automatiza la confirmación de transacciones exitosas, y proporciona base confiable para reportes financieros y conciliación contable.
- **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:** Funciones modulares que gestionan la progresión de ranking de jugadores, navegando por la jerarquía de rangos mediante rank_order y actualizando automáticamente el rango del jugador con timestamp. Automatizan el sistema competitivo del juego, mantienen motivación de progresión, alimentan algoritmos de matchmaking balanceado, y proporcionan métricas de engagement para análisis de retención de jugadores.
- **Actualizar estadisticas jugador:** UPSERT (actualiza estadísticas existentes de un jugador en una partida específica o las inserta si no existen), manejando kills,

deaths y assists. Permite corrección de datos de partidas, sincronización con sistemas externos de tracking, actualización de estadísticas en tiempo real durante streaming, y mantiene flexibilidad para ajustes post-partida por parte de administradores.

- **Average kills by rank and region:** Calcula el promedio de kills por combinación de rango y región, proporcionando métricas de rendimiento segmentadas geográficamente. Facilita balanceo de servidores regionales, análisis de meta por región para ajustes específicos, comparación de skill level entre diferentes mercados, y generación de insights para estrategias de expansión geográfica del juego.
- **Get players performance metrics:** Retorna una tabla completa de métricas de rendimiento incluyendo KDA, ACS estimado y MMR calculado, proporcionando un perfil analítico completo del jugador. Alimenta dashboards de rendimiento personal, sistemas de coaching automatizado, algoritmos de detección de smurf accounts, análisis de progresión para programas de development de talentos, y generación de certificaciones de skill para torneos.
- **Get player win rate:** Calcula estadísticas de victoria incluyendo total de partidas, wins, losses y porcentaje de victoria, analizando la participación del jugador en equipos ganadores y perdedores. Proporciona métricas clave para sistemas de ranking, identificación de jugadores clutch, análisis de consistencia de rendimiento, y base para algoritmos de predicción de resultados en apuestas esports.
- **Player yearly analysis:** Genera un reporte anual completo del jugador incluyendo promedios de estadísticas, mapa y agente más utilizados, proporcionando una retrospectiva del año. Habilita reportes anuales personalizados tipo "Spotify Wrapped", análisis de evolución de meta preferences, identificación de especializaciones de jugadores, y generación de contenido personalizado para redes sociales y marketing de retención.

MySQL:

- **Tiene saldo suficiente**: Valida si un usuario posee balance suficiente en su wallet para realizar una transacción específica, comparando el monto solicitado con el

saldo disponible. Permite validación previa de compras antes de procesar transacciones, previene errores de saldo insuficiente en el checkout, facilita implementación de carrito de compras con validación en tiempo real, y sirve como base para sistemas de recomendación de paquetes VP según capacidad de pago del usuario.

- **Total gastado por usuario:** Calcula el gasto acumulado total de un usuario sumando todas sus transacciones históricas, retornando cero si no tiene transacciones registradas. Habilita análisis de valor de vida del cliente (CLV), segmentación de usuarios por nivel de gasto, generación de programas de fidelización personalizados, detección de ballenas (high-spenders) para campañas VIP, y construcción de modelos predictivos de comportamiento de compra.

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, mejorando los tiempos de respuesta de dashboards y perfiles de usuario.
- **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, mejorando la experiencia de usuario en funciones sociales del juego.
- **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, fundamental para features de comparación y análisis de rendimiento personalizado.

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. En sp_refund_transaction, el reembolso de VP y la actualización del estado de transacción ocurren como una unidad atómica, previniendo inconsistencias donde el dinero se devuelva pero la transacción no se marque como reembolsada.
- **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. El SP sp_delete_player valida que un jugador no esté en equipos activos antes de eliminarlo, manteniendo la consistencia de las relaciones de datos y evitando referencias huérfanas.
- **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. Los triggers de actualización de wallet (update_wallet_balance) operan de forma aislada para evitar inconsistencias en saldos cuando ocurren múltiples transacciones concurrentes sobre el mismo wallet.
- **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

El sistema implementa backups diarios automáticos para ambas bases de datos usando cron jobs que se ejecutan a medianoche. MySQL utiliza mysqldump para generar archivos SQL completos, mientras PostgreSQL usa pg_dump con formato comprimido custom. Ambos scripts siguen el mismo patrón: ejecutan el comando de backup dentro del contenedor Docker, generan archivos con timestamp único, y los copian al sistema host en carpetas organizadas. La configuración mediante variables de entorno mantiene las credenciales seguras, y el manejo de errores asegura que cualquier fallo en el proceso se registre apropiadamente para monitoreo operacional.

7.3.2. Restore Inteligente

Se implementó la recuperación de datos identificando el backup más reciente por fecha y ejecutando la restauración completa. La lógica es prácticamente idéntica para ambos motores: buscan el archivo más nuevo, lo copian al contenedor y ejecutan el comando de restore correspondiente. MySQL requiere configuración adicional para manejar stored procedures correctamente, mientras PostgreSQL utiliza la opción clean para evitar conflictos durante la restauración. Esta automatización elimina errores manuales en situaciones críticas y garantiza que la recuperación de datos sea rápida y confiable cuando sea necesaria.

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 TTLs están ajustados según la criticidad y frecuencia de cambio: paquetes VP (6h) balanceando consistencia de precios, balances de wallet (6h) manteniendo precisión financiera, métodos de pago (24h) por su naturaleza estática, rankings (24h) para jerarquías actualizadas, y jugadores por país (1h) para filtros geográficos dinámicos. La implementación a MySQL/PostgreSQL reduce significativamente la carga en stored procedures como realizar_compra_vp y functions 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 intrincadas 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