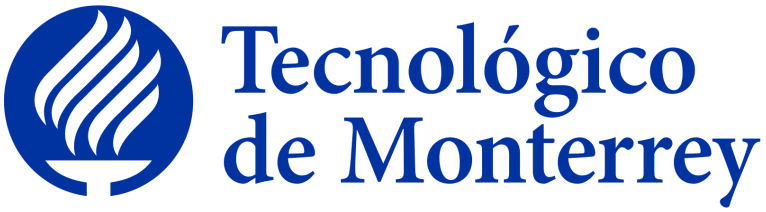


ITESM Campus Santa Fe

Nombre del bloque:

TC2005B: Construcción de software y toma de decisiones (Gpo. 401)

Nombre del entregable:

5. Normalización de la base de datos del reto apoyándose en recursos de IA generativa

Justificación - Versión 3.0

Equipo BotRunners:

Diego Córdova Rodríguez, A01781166

Lorena Estefanía Chewtat Torres, A01785378

Eder Jezrael Cantero Moreno, A01785888

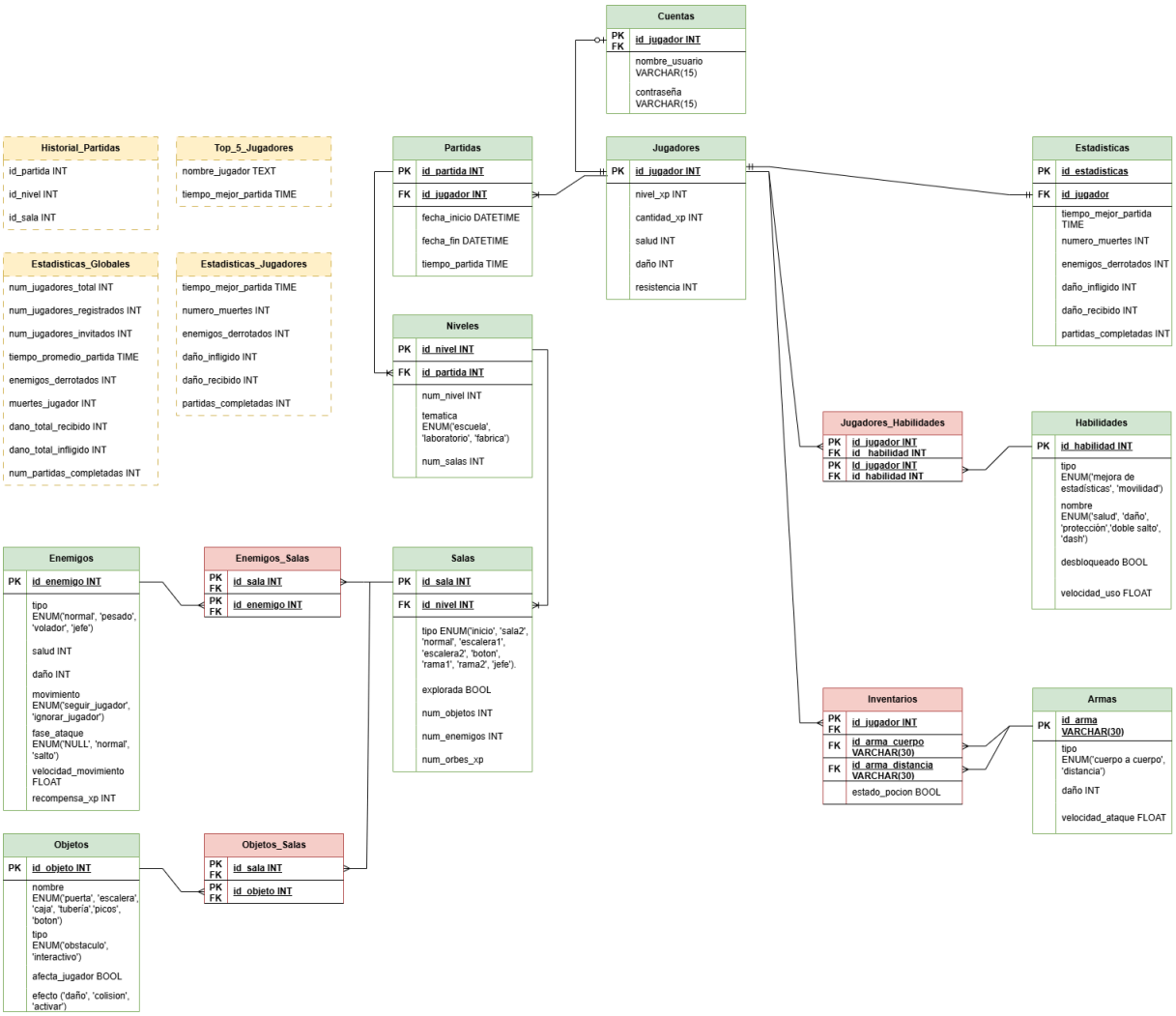
Profesor:

Esteban Castillo Juarez

Fecha de entrega:

11 de abril de 2025

Diagrama Entidad-Relación: Base de Datos - Overclocked - Versión 3.0



Acceso al archivo .drawio con el Diagrama:

https://drive.google.com/file/d/14BnVYPJ7p_k7qtHA1E8RiwaqcbGkKEMs/view?usp=sharing

Justificación del diagrama

El modelo ER de nuestra base de datos ha pasado por diversos cambios a lo largo del desarrollo de nuestro videojuego. Los principales cambios que podemos observar en comparación con las versiones anteriores es la implementación de vistas para la visualización de los datos sin alterar las tablas al hacer las llamadas con la API. De igual forma, agregamos la tabla Cuentas como extensión de la tabla Jugadores, para aquellos usuarios que se hayan registrado con un nombre de usuario y contraseña.

El modelado de la base de datos del videojuego Overclocked cuenta con 10 principales tablas que representan los aspectos clave para el funcionamiento del juego. Por otro lado, se cuenta con 4 tablas intermedias, cuyo trabajo reside en evitar la duplicidad de información para aquellas tablas relacionadas de forma muchos a muchos y 4 vistas que nos ayudarán a poder mostrar la información sin correr el riesgo de alterar nuestra base de datos. De esta forma, se puede tener una base de datos más organizada que sea funcional al momento de trabajar en MySQL y conectarla con nuestro videojuego a través de la API.

A continuación se presenta la justificación de cada una de las tablas propuestas en el esquema relacional del videojuego, aclarando el uso de cada una dentro del mismo.

Justificación de tablas principales

Jugadores

Esta tabla contiene la información de cada uno de los jugadores del videojuego. Cada registro de esta tabla contiene un único jugador y el siguiente conjunto de datos:

- **Nivel de xp:** Nivel actual de experiencia del jugador en el juego.
- **Cantidad de experiencia:** Determina el progreso del nivel actual de experiencia en el juego.
- **Salud:** Cantidad de vida que el jugador posee en el juego.
- **Daño:** Cantidad de daño que inflige el jugador a enemigos.
- **Resistencia:** Esta cifra disminuye la cantidad de daño que infligen los enemigos sobre el jugador.

Esta tabla cuenta con las siguientes relaciones:

- **Relación con Cuentas:** Un jugador puede tener una cuenta (es decir, jugadores registrados), pero a la vez también pueden haber jugadores que no tengan una cuenta (jugadores no registrados), por lo cual es una relación 1 a 1 opcional. Esta separación permite que haya una diferencia entre los jugadores registrados (los que tienen una cuenta) y los jugadores invitados.
- **Relación con tabla Estadísticas:** Cada jugador posee un solo registro de estadísticas en el juego (relación 1 a 1).
- **Relación con tabla Partidas:** Un jugador puede participar en diferentes partidas, pero las partidas solo pueden ser jugadas por un jugador (relación 1 a muchos). Esto se debe a la naturaleza de cada partida, pues estas son generadas de forma aleatoria, por lo que cada una es diferente a las anteriores. Además, el juego no es multijugador, por lo que cada partida puede ser jugada por un único jugador. Debido a esto, cada vez que el jugador inicia una nueva partida, se crea un nuevo registro de esta, pues es prácticamente imposible que exista una partida idéntica a otra.
    - **Ejemplo:** Si un jugador con id 1 inicia la partida con id 101, el mismo jugador podrá después jugar otra partida con id 102; sin embargo, esta partida no podrá ser jugada por otro jugador, pues este nuevo jugador no tendría una partida aleatoria, sino que estaría en una que ya existía previamente.
- **Relación con Inventarios:** Un jugador puede contar con diferentes elementos en su inventario, a la vez que muchos inventarios pueden pertenecer a muchos jugadores (relación muchos a muchos). Debido a esto, la tabla intermedia Inventarios se encarga de ligar cada jugador con sus armas (cuerpo a cuerpo y distancia) y poción de curación.
- **Relación con Habilidades:** Un jugador puede tener muchas habilidades, así como muchas habilidades pueden ser poseídas por muchos jugadores (relación muchos a muchos). De esta forma, la tabla intermedia Jugadores_Habilidades se encarga de contener los registros de las habilidades que posee cada jugador, ya sean mejoras de estadísticas o mejoras de movimiento (doble salto o esquivar (dash)).

Cuentas:

Anteriormente, la gestión del tipo de cuenta se manejaba dentro de la tabla jugadores, pero después de revisar si la base de datos cumplía correctamente con las reglas de normalización, establecimos que se necesitaría una tabla Cuentas para poder gestionar de mejor forma el acceso y autenticación de los jugadores en el videojuego, separando la lógica del jugador con la lógica del inicio de sesión. Así podemos distinguir los jugadores registrados de los invitados, recopilando estadísticas de las sesiones de juego de ambos. La diferencia entre ambos es que si los jugadores invitados abandonan el juego (salen de la página en el navegador) y regresan, no tendrán forma de recuperar sus estadísticas o armas adquiridas.

Esta tabla contiene las credenciales básicas de las cuentas de los jugadores, que son las siguientes:

- **Nombre de usuario:** Nombre único para identificar al jugador.
- **Contraseña:** Clave necesaria para que el jugador pueda iniciar sesión y recuperar sus datos entre sesiones de juego.

Esta tabla cuenta únicamente con una relación:

- **Relación con tabla Jugador:** Debido a que permitimos jugadores invitados o jugadores registrados, la relación con la tabla jugador es de uno a uno opcional. Si no tienes una cuenta registrada y solamente quieres jugar como invitado, no se te asignará una cuenta, y lo contrario sucede con las cuentas registradas.

Estadísticas:

Esta tabla contiene las estadísticas recopiladas de cada jugador durante las partidas del juego. Cada registro de esta tabla contiene un solo conjunto de estadísticas, que son las siguientes:

- **Tiempo_mejor_partida:** Indica el mejor tiempo en el que el jugador ha completado una partida. El atributo tiempo_partida en la tabla Partidas es diferente a este ya que indica el tiempo que el jugador se mantuvo jugando esa partida específica, pero no es necesariamente la mejor que ha tenido.

- **Numero_muertes:** Indica el número total de muertes que el jugador tuvo durante todas sus partidas.
- **Enemigos_derrotados:** Indica el número total de enemigos derrotados durante todos los niveles de todas las partidas del jugador.
- **Daño_infligido:** Indica el número total de daño infligido por el jugador a los diferentes tipos de enemigos.
- **Partidas_completadas:** Indica el número total de partidas completadas por el jugador a lo largo de todas sus sesiones de juego.

Esta tabla únicamente cuenta con una relación:

- **Relación con tabla Jugador:** Cada jugador posee un solo registro de estadísticas en el juego (relación 1 a 1), debido a que cada partida va a tener resultados distintos por la aleatoriedad de los niveles.

Habilidades

Esta tabla contiene los registros de las habilidades que pueden ser obtenidas en el videojuego al momento en el que el jugador sube de nivel, ya sean mejoras de salud, daño, protección o habilidades de movimiento, como es el doble salto y esquivar (dash). Algunos de los atributos más importantes de esta tabla son los siguientes:

- **Tipo:** Declara si la habilidad es una mejora de salud, daño o protección del jugador; o si de lo contrario, se trata de una mejora de habilidad.
- **Nombre:** Especifica qué habilidad se está eligiendo (Salud, daño, protección, doble salto o dash).
- **Desbloqueada:** Es un valor Verdadero o Falso que indica si la habilidad ha sido desbloqueada por el jugador.
- **Velocidad_uso:** Indica el tiempo que el jugador debe esperar para reutilizar la habilidad tras cada uso.

Esta tabla únicamente cuenta con una relación:

- **Relación Jugadores_Habilidades:** A través de esta tabla intermedia, se establece la relación entre muchos jugadores y sus habilidades obtenidas (relación muchos a muchos).
- A diferencia del modelo anterior, para poder hacer que un mismo usuario cuente con dos habilidades diferentes en nuestra base de datos, tanto **id_jugador** e **id_habilidad** fungen como primary keys y foreign keys respectivamente.

Armas

Esta tabla establece los registros de las armas existentes en el videojuego, ya sean armas cuerpo a cuerpo (brazo de robot o llave de metal) o a distancia (Pistola láser de disparo rápido o Pistola láser de disparo lento). Algunos de los atributos más importantes de la tabla son los siguientes:

- **ID_Arma (Nombre):** Actualmente y con el fin de tener una mayor eficiencia al guardar nuestros datos en la BD, el nombre funge como ID, esto para optimizar el proceso de obtener las armas del jugador almacenadas en la BD y hacerlo de forma más directa. Solo existen 4 armas dentro del juego y cada una tiene un nombre único, razón por la que se decidió que este sea su identificador.
- **Tipo:** Establece si es un arma cuerpo a cuerpo o a distancia.
- **Daño:** Establece la cantidad de daño que inflige el arma al ser utilizada.
- **Velocidad_ataque:** Tiempo entre usos del arma.

Esta tabla únicamente cuenta con la siguiente relación:

- **Relación con Jugador (Inventarios):** Esta tabla se relaciona con los jugadores a través de una tabla intermedia Inventarios, donde muchos jugadores pueden poseer muchos inventarios, así como muchos inventarios son poseídos por muchos jugadores (relación muchos a muchos). De esta forma, en los inventarios existe una llave foránea para el arma cuerpo a cuerpo y a distancia que posee el jugador.
- **Estado poción:** Respecto a la justificación anterior, se agrega un estado booleano a la poción que indica si ya fue usada
- **Curación poción:** Indica con un valor entero la cantidad de salud que puede curar.

Partidas

Esta tabla almacena información sobre cada partida que se inicia dentro del juego. Cada partida es única, por lo cual esta información sirve para registrar datos clave de cada partida, como los siguientes:

- **Fecha_inicio:** Indica la fecha y hora en la cual se inició esa partida
- **Fecha_fin:** Indica la fecha y hora en la cual terminó la partida
- **Tiempo_partida:** Tiempo total que el jugador invierte dentro de la partida en formato hora-minuto-segundo. Es diferente al tiempo_mejor_partida almacenado en la tabla Estadísticas de los Jugadores, ya que en esta tabla solo se guarda un registro del tiempo que lleva el jugador dentro de la partida. En el caso de que este sea mayor al mejor tiempo del jugador, se puede hacer un JOIN entre las tablas Partidas, Jugadores y Estadísticas, actualizando el valor.

Esta tabla cuenta con las siguientes relaciones:

- **Relación con Jugador:** Cada jugador puede tener múltiples partidas, pero cada partida pertenece a un solo jugador y es generada aleatoriamente (relación de 1 a muchos). Esto significa que cada partida tiene datos e información única, por lo que no puede ser reutilizada por ningún otro jugador.
- **Relación con Niveles:** Cada partida contiene muchos niveles (3), y estos niveles sólo pueden pertenecer a una partida (relación de 1 a muchos) por la misma razón de aleatoriedad en los niveles. Como cada nivel tiene diferentes atributos generados aleatoriamente, estos no se pueden repetir entre partidas.

Niveles

Esta tabla mantiene los registros de todas las partidas que han sido generadas de forma aleatoria para el juego. Cada vez que un jugador inicia una nueva partida aleatoria, se genera un nuevo registro, pues ninguna será idéntica a las anteriores en la base de datos. Esta tabla posee los siguientes atributos clave:

- **Num_nivel:** Indica el número del nivel dentro de la partida, sea 1, 2 o 3.
- **Temática:** Indica la temática del nivel en el juego. Por ejemplo, el nivel 1 tiene una temática de escuela, el nivel 2 de laboratorio y el 3 de fábrica. Esto ha sido definido así desde la planificación y documento de diseño de nuestro juego (GDD).

- **Num_salas:** Indica la cantidad de salas con las que cuenta cada nivel.

Esta tabla cuenta con las siguientes relaciones:

- **Relación con Jugadores:** Cada partida pertenece a un único jugador, mientras que el jugador puede participar en muchas partidas (relación 1 a muchos). Esto es especificado a detalle en la descripción de la tabla Jugadores: Las partidas son generadas de forma aleatoria, por lo que cada una es diferente a las anteriores. Además, el juego no es multijugador, por lo cada partida solo puede ser jugada por un único jugador.
- **Relación con Salas:** Una partida contiene muchas salas, mientras que muchas salas están en una sola partida (relación 1 a muchos). De forma similar que con la tabla Jugadores, las salas son generadas de forma aleatoria, por lo que no se pueden repetir en diferentes niveles, pues siempre habrán elementos distintos entre cada uno.

Salas

Esta tabla almacena la información del contenido de cada habitación de los niveles, específicamente sus objetos (interactuables u obstáculos) y los enemigos. Esta tabla posee los siguientes atributos clave:

- **Tipo:** Es el tipo de sala que se encuentra en un nivel. Los tipos permitidos son: 'inicio', 'sala2', 'normal', 'escalera1', 'escalera2', 'boton', 'rama1', 'rama2', 'jefe').
- **Explorada:** Es un dato booleano que indica si la sala ha sido explorada o no por el jugador. Este dato es importante ya que nos ayuda dentro del juego a actualizar el minimapa en la interfaz, así como a tener un registro de las salas que han sido (o no) exploradas por el jugador.
- **Num_objetos:** Número de objetos presentes en la sala, sean interactuables (botones, escaleras o puertas) u obstáculos (picos que dañan al jugador, cajas o tuberías).
- **Num_enemigos:** Número de enemigos presentes en la sala, sean de tipo normal, pesado, voladores, o se trate del jefe final del nivel.
- **Num_orbes_xp:** Nos da el número de orbes de experiencia generados de forma natural en una sala (también pueden ser obtenidos después de derrotar enemigos, pero no forman parte de la estructura de la sala).

Esta tabla cuenta con las siguientes relaciones:

- **Relación con Niveles:** Una partida contiene muchas salas, mientras que muchas salas están en una sola partida (relación 1 a muchos). Las salas son generadas de forma aleatoria, por lo que no se pueden repetir en diferentes niveles, pues siempre habrán elementos distintos entre cada uno.
- **Relación con Enemigos:** Esta tabla se relaciona con los enemigos que contiene a través de una tabla intermedia, pues así como muchos enemigos pertenecen a muchos niveles, muchos niveles poseen muchos enemigos (relación muchos a muchos).
- **Relación con Objetos:** Esta tabla se relaciona con los objetos que contiene a través de una tabla intermedia, pues así como muchos objetos pertenecen a muchos niveles, muchos niveles poseen muchos objetos (relación muchos a muchos).

Enemigos

Esta tabla almacena la información de los diferentes tipos de enemigos que se encuentran en el juego, ya sean enemigos normales (3 tipos) o jefes finales. Esta tabla contiene los siguientes atributos necesarios para identificar a cada enemigo:

- **Tipo:** Conjunto fijo de los tipos de enemigos, que pueden ser normales, pesados, voladores o jefes finales de los niveles.
- **Salud:** Cantidad de vida que el enemigo contiene en el juego.
- **Daño:** Cantidad de daño que inflige el enemigo al jugador.
- **Movimiento:** Conjunto fijo de los tipos de movimiento que pueden tener los enemigos, ya sea seguir al jugador o ignorar al jugador.
- **Fase_ataque:** Conjunto fijo de las fases de ataque en la que se encuentra el jefe, que puede ser fase normal o fase de salto, en donde el jefe podrá saltar hacia la dirección del jugador. Debido a que las fases de ataque solo son un atributo de los jefes finales, si se trata de otro tipo de enemigo, este atributo puede tener un valor nulo.
- **Velocidad_movimiento:** Indica la velocidad de movimiento que el enemigo posee.
- **Recompensa_xp:** Cantidad de experiencia que el enemigo suelta al morir.

Esta tabla únicamente cuenta con la siguiente relación:

- **Relación con Salas:** Varias salas pueden tener varios enemigos, así como varios enemigos pueden estar en varias salas (relación muchos a muchos). Debido a la naturaleza de esta relación, se utiliza una tabla intermedia.

Objetos

Esta tabla almacena todos los registros de objetos que pueden existir en las salas de los niveles generados de forma aleatoria en el videojuego. Algunos de los atributos clave de esta tabla son los siguientes:

- **Nombre:** Establece el nombre del objeto, ya sea una puerta, escalera, caja, tubería, picos que dañan al jugador o botones.
- **Tipo:** Determina si el objeto es considerado como un obstáculo para el jugador o es interactivo, como las puertas, escaleras o botones.
- **Afecta_jugador:** Valor Verdadero o Falso para determinar si el objeto inflige alguna cantidad de daño al jugador, como son los picos.
- **Efecto:** Determina el tipo de efecto que ejerce el objeto sobre el jugador, como es daño, colisión (es un objeto sólido inamovible) o activar (en caso de poder tener un estado activado/desactivado, como los botones).

Esta tabla únicamente cuenta con la siguiente relación:

- **Relación con Salas:** La relación de todos los objetos que se pueden llegar a encontrar dentro de una sala se da a través de la tabla intermedia Objetos_Salas, donde se establece la relación de las salas con todos los objetos que contiene. Así como muchas salas pueden tener muchos objetos, muchos objetos se pueden encontrar en muchas salas (relación muchos a muchos).

Justificación de tablas intermedias

Jugadores_Habilidades

Esta tabla intermedia almacena los registros de las habilidades que poseen los jugadores. De esta manera, se puede llevar un registro de las mejoras de estadísticas y de movilidad (doble salto o esquivar (dash)) que han sido desbloqueadas por los diferentes jugadores del juego.

Respecto a la versión anterior, ahora id_jugador e id_habilidad tienen la propiedad de ambas ser llaves primarias y foráneas, para que un jugador pueda tener múltiples habilidades y se referencien correctamente los datos.

Esta tabla cuenta con las siguientes relaciones:

- **Relación entre Habilidades y Jugadores:** Al ser una tabla intermedia, la llave primaria es el id del jugador que contiene las habilidades. Así como muchos jugadores pueden desbloquear muchas habilidades, muchas habilidades pueden ser desbloqueadas por muchos jugadores (relación muchos a muchos).

Inventarios

Esta tabla intermedia almacena las distintas armas que el jugador posee. De esta manera, se puede llevar un registro de las armas desbloqueadas dentro del juego y cuales se están usando para derrotar a los enemigos. Además esta tabla también sirve para que el progreso de las armas desbloqueables no se pierda. Si el jugador pierde pero ha desbloqueado un arma, esta se quedará en su inventario. También, en el inventario se encuentra un valor booleano que representa si ya se usó o no la poción:

- **Estado_pocion:** Valor verdadero o falso que indica si se ha utilizado la poción de curación durante ese nivel, debido a que solo se puede utilizar una vez por nivel.

Esta tabla cuenta con las siguientes relaciones:

- **Relación entre Armas y Jugador:** Al ser una tabla intermedia, la llave primaria es el id del jugador que posee la poción de curación y las diferentes armas. Como muchos jugadores pueden desbloquear las distintas armas del juego, muchas armas pueden ser desbloqueadas por muchos jugadores (relación muchos a muchos).

Enemigos_Salas

Esta tabla intermedia almacena los enemigos que pertenecen a las diferentes salas de cada uno de los niveles del videojuego. Así, se lleva un registro de todos los tipos de enemigos, sean normales, pesados, voladores o jefes que pertenecen a las salas.

Respecto a la versión anterior, ahora id_sala e id_enemigo tienen la propiedad de ser llaves primarias y foráneas. Esto permite representar correctamente la relación muchos a muchos entre salas y enemigos, garantizando la unicidad de cada par y la integridad de las referencias de los datos.

Como parte de la retroalimentación de la entrega anterior, se sugirió agregar más atributos en esta tabla intermedia; sin embargo, tras revisarlo con el profesor, se concluyó que no existen atributos adicionales de valor o estadísticas relevantes para el videojuego.

Esta tabla cuenta con las siguientes relaciones:

- **Relación entre Enemigos y Salas:** Al ser una tabla intermedia, la llave primaria es el id de la sala en donde se encuentran los enemigos, siendo el elemento clave para la relación. Mientras que muchos enemigos pertenecen a muchas salas, muchas salas contienen muchos enemigos (relación muchos a muchos).
- Se asignan como foreign keys y primary keys el id de la sala y el id del enemigo, para que muchas salas puedan tener muchos enemigos en nuestra base de datos.

Objetos_Salas

Esta tabla intermedia almacena los objetos que se encuentran en las diferentes salas de cada uno de los niveles del juego. De esta forma, se lleva un registro de la cantidad y tipo de objetos que se encuentran en la sala, por ejemplo, puertas, escaleras, cajas, tuberías, picos o el botón para abrir la sala final.

Respecto a la versión anterior, ahora id_sala e id_objeto tienen la propiedad de ser llaves primarias y foráneas. Esto permite representar correctamente la relación muchos a muchos entre salas y objetos, garantizando la unicidad de cada par y la integridad de las referencias de los datos.

Como parte de la retroalimentación de la entrega anterior, se sugirió agregar más atributos en esta tabla intermedia; sin embargo, tras revisarlo con el profesor, se concluyó que no existen atributos adicionales de valor o estadísticas relevantes para el videojuego.

Esta tabla cuenta con las siguientes relaciones:

- **Relación entre Objetos y Salas:** Al ser una tabla intermedia, la llave primaria es el id de la sala en donde se encuentran los objetos. Debido a que hay distintos tipos de objetos, muchos de estos pueden pertenecer a muchas salas, y muchas salas pueden contener muchos objetos (relación muchos a muchos).
- Se asignan como llaves primarias y foráneas id_objeto e id_sala para que muchas salas puedan tener muchos objetos.

Justificación de las vistas:

El uso de vistas en nuestra base de datos nos permite almacenar consultas de manera virtual, dándonos así la posibilidad de evitar realizar la misma consulta en repetidas ocasiones, mostrando los datos relevantes de la partida sin necesidad de hacer uso de múltiples uniones (JOINS).

Historial_Partidas:

La vista "Historial de Partidas" permite rastrear la progresión y el recorrido de los jugadores dentro de una partida. De esta forma, podemos realizar las siguientes acciones:

- Facilita la reconstrucción del trayecto de un jugador a través de los niveles y salas, lo cual puede ser útil para analizar patrones de juego.
- Permite a los jugadores revisar en qué salas han estado y en qué niveles han jugado, proporcionando información valiosa para mejorar su desempeño.

Estadísticas_Globales:

Parte de la retroalimentación de la primera versión fue agregar estadísticas propias del videojuego como un todo, de forma adicional a las estadísticas propias de los jugadores. Así, la vista de estadísticas globales nos permite establecer métricas generales para el rendimiento de los jugadores en la totalidad del videojuego:

- Nos permite diferenciar entre jugadores invitados y registrados.
- Proporciona métricas importantes como el tiempo promedio de duración de una partida, los enemigos derrotados, muertes, daño infligido, recibido y el número de partidas completadas.
- Facilita la implementación de tableros para visualizar las estadísticas del juego como un todo.

De esta manera, como desarrolladores, podemos ingresar a estas estadísticas para detectar patrones de comportamiento de los jugadores en sus diferentes partidas. Esto nos permite tomar decisiones que impacten de forma positiva el funcionamiento del juego. Por ejemplo, si el número total de muertes de los jugadores es muy alto en comparación a las partidas jugadas en total, podemos reducir el número de enemigos por sala, su salud o daño.

Esto también le da la libertad a los jugadores de ver cuál es el promedio de tiempo de aquellos jugadores que tienen un mejor desempeño en el juego. De esta forma, pueden analizar cuántos enemigos deben derrotar para subir de nivel y ser más capaces de completar el juego en un menor tiempo.

Top_5_Jugadores:

Basándonos en uno de los requerimientos de nuestro videojuego, mismo que es necesario para poder mostrar el top 5 de los jugadores de forma más específica, se decidió implementar esta vista para ver los jugadores que han tenido los menores tiempos en las partidas del videojuego. A través de esta vista, podemos realizar lo siguiente:

- Filtrar entre los mejores 5 tiempos de todas las partidas de todos los jugadores.
- Podemos implementar un tablero de los mejores jugadores dentro de la sección de estadísticas del juego.
- Ver estos datos fomenta la competencia entre los jugadores y los impulsa a esforzarse para completar el juego en menos tiempo, teniendo la posibilidad de llegar a estar en esta tabla. Como desarrolladores, esta vista nos permite hacer ajustes a la dificultad del videojuego en caso de ver que los jugadores completan el juego en mucho o poco tiempo.

Estadisticas_Jugadores:

Esta vista nos permite obtener las estadísticas de los jugadores de forma similar que con la tabla Estadisticas. La diferencia es que esta vista tiene el propósito de ser mostrada en el menú de estadísticas personales del videojuego, con nombres de columnas que no revelan información sensible a los jugadores. De esta manera, esta tabla sirve únicamente para que cada jugador pueda visualizar sus estadísticas personales. A través de esta vista, podemos realizar lo siguiente:

- Mostrar las estadísticas de los jugadores sin revelar información importante de la base de datos, como el identificador (id) de las estadísticas o jugadores.
- Proporciona un formato más simple sobre las columnas de información de la tabla de estadísticas. Por ejemplo, la columna “tiempo_mejor_partida” pasa a ser “Mejor Tiempo”.

Normalización de la base de datos del reto apoyándose en recursos de IA generativa

La normalización de las bases de datos es un proceso mediante el cual se pueden organizar todos los datos contenidos en estas, incluyendo la creación de tablas y las relaciones entre ellas. Esto evita la redundancia en la base de datos, así como elimina dependencias incoherentes, garantizando que sea más fácil de mantener a futuro (Learn Microsoft, 2024).

Para normalizar una base de datos, existen las reglas de normalización, llamadas "Formas Normales". Para nuestro videojuego, estaremos enfocándonos en las primeras 3, es decir, hasta llegar a la tercera forma normal (3FN). A continuación se presentan las 3 primeras formas normales y sus características, así como los elementos que fueron normalizados en nuestra base de datos, en base a lo aprendido en clase y las recomendaciones de la IA generativa ChatGPT.

Primera forma normal (1FN)

Características:

- Todos los atributos son atómicos, es decir, en cada celda hay un solo valor
- Cada tabla debe tener una llave primaria.
- No deben existir filas o columnas duplicadas
- La llave primaria no debe tener atributos nulos
- No debe existir variación en el número de columnas. Si algunas filas tienen 8 columnas, y otras 3, no se cumple esta forma normal (Chris, K., 2023).

Para cumplir con esta forma normal, desde la primera entrega del diagrama Entidad-Relación del videojuego, nos aseguramos de que todas las tablas tengan atributos atómicos. Además, cada tabla cuenta con una llave primaria definida, como id_jugador en Jugadores o claves compuestas como (id_jugador, id_habilidad) en Jugadores_Habilidades.

Durante el análisis de normalización, identificamos que la tabla Inventarios contiene 2 columnas para armas: id_arma_cuerpo e id_arma_distancia. Si bien podríamos optar por normalizar aún más esta tabla -como vimos en clase- y separar estas relaciones en una tabla

intermedia Inventario_Armas, decidimos mantener la estructura actual por las siguientes razones:

- Debido a la planificación inicial del videojuego, Overclocked no contará con armas adicionales.
- Cada jugador solo puede tener un total de 2 armas: una cuerpo a cuerpo y otra a distancia, por lo que esto ya está bien representado con la implementación actual.
- Esta estructura facilita la consulta rápida de armas al momento de implementar la API del juego, haciendo que sea más simple consultar y recuperar la información guardada de los jugadores.

Como se comentó en clase, es conveniente normalizar la base de datos de acuerdo al propósito del proyecto y la forma en la que será implementado. De esta forma, consideramos mantener este aspecto no normalizado completamente, pues el videojuego no lo requiere y su modificación haría que la recopilación de información sea más compleja.

La posibilidad de normalizar aún más este aspecto fue uno de los puntos recomendados por la IA generativa. A pesar de esto, decidimos priorizar la eficiencia y claridad de las consultas a la tabla Inventarios, pues el enfoque actual no compromete la integridad ni la coherencia de los datos. Además, ya no se agregarán más armas al videojuego.

Segunda forma normal (2FN)

Características:

- Se debe cumplir con la primera forma normal (1FN)
- No existe dependencia parcial: Todos los atributos no clave solo dependen de la llave primaria (Chris, K., 2023).

Para cumplir con esta forma normal, verificamos cada una de las tablas de la base de datos, asegurándonos de que no exista dependencia parcial alguna, es decir, que los atributos no clave dependan completamente de la llave primaria correspondiente.

Un ejemplo de esto es en la tabla jugadores, donde se tiene como llave primaria id_jugador y todos los atributos dependen únicamente del jugador, como ocurre con sus datos

del juego, como salud, daño, nivel de experiencia, entre otros. Estas son características únicas para cada jugador, y que no pertenecen a ninguna otra llave primaria.

Otro ejemplo es la tabla Estadísticas, donde almacenamos aquellos valores que a pesar de corresponder a un jugador, son organizados de mejor forma en una tabla aparte, donde se puede consultar aquellas cifras acumuladas por los jugadores a lo largo de sus partidas.

De igual forma, validamos que todos los campos adicionales en las demás tablas solo dependan de sus llaves primarias, como ocurre en Armas con sus atributos de tipo, daño y velocidad de ataque.

Utilizamos la IA generativa para validar que las tablas intermedias evitaran dependencias parciales, sin embargo, no tuvimos añadidos algunos. De igual manera, la IA nos ayudó a identificar las vistas Historial_Partidas, Top_5_Jugadores, Estadisticas_Globales y Estadisticas_Jugadores, por lo que se nos dijo que estas no debían ser consideradas para evaluar el cumplimiento de la segunda forma normal, pues solo se tratan de visualizaciones dentro del videojuego.

Tercera forma normal (3FN)

Características:

- Debe cumplir la regla 2FN
- No debe existir dependencia parcial transitiva: Un atributo no principal depende de otro no principal (Chris, K., 2023).

Para cumplir con esta forma normal, revisamos posibles casos de dependencia transitiva, donde un atributo no clave depende de otro no clave en lugar de la llave primaria. Para esto, evaluamos lo siguiente:

- Las entidades se encuentran separadas de forma independiente: Cada tabla representa un aspecto del juego y sus atributos dependen únicamente de la llave primaria. Por ejemplo, las salas contienen la información que dependen únicamente de su identificador, como es su tipo, número de obstáculos, número de enemigos, la bandera para saber si ha sido explorada, entre otros.

Al realizar este análisis, detectamos un único posible caso de dependencia transitiva. Esto ocurre en la tabla Jugadores, pues tipo_cuenta (registrado o invitado) depende de si la contraseña tiene un valor o no. Para corregir esto, creamos una nueva tabla Cuentas, donde se almacena el nombre del usuario y su contraseña. Así, únicamente los jugadores que tengan una cuenta permanecerán a esta, pero todos los jugadores (independientemente de si tienen cuenta o no) generarán estadísticas para el videojuego.

Al cuestionar aún más al modelo IA generativo, este nos comentó repetidas veces que ya se cumplía con la tercera forma normal. Se sugirió implementar una columna nombre_arma en Inventarios, sin embargo, esto fue descartado, pues este es un atributo que ya es contenido en la tabla Armas. De esta forma, esto agregaría una columna que no depende directamente del inventario, sino de las armas.

La IA también sugirió la posibilidad de crear una tabla Consumibles_Usados para almacenar los consumibles del jugador; sin embargo, el plan del juego no incluye agregar más consumibles al juego, por lo que se optó por mantener este campo en la tabla Inventarios. Además, estado_poción depende directamente de la llave primaria id_jugador, por lo que no rompe la tercera forma normal.

Con base en lo aprendido en clase y con el uso de un modelo de IA generativo (ChatGPT 4.0), los ajustes realizados a la base de datos mediante la aplicación de las tres primeras formas normales de normalización fueron mínimos. Esto se debe a que la estructura de nuestro modelo entidad-relación (ER) ya presentaba una base sólida y organizada desde su diseño inicial, respaldada por constantes revisiones y validaciones internas.

Gracias a esta planificación y a la implementación correcta de las reglas de normalización, se logró asegurar una estructura coherente y eficiente que evita la redundancia de datos, reduciendo la duplicación de los mismos y mejorando la integridad de la información. Además, este enfoque permite que las operaciones de inserción, actualización y eliminación se realicen de forma ordenada, minimizando errores y facilitando la escalabilidad del sistema en futuras iteraciones

Justificación de la normalización

Al corroborar nuestro modelo entidad-relación de la base de datos tanto de forma manual y con el apoyo de la IA generativa, llegamos a diversas conclusiones, principalmente derivado de que ya teníamos una estructura bastante sólida.

Tabla inventario:

La tabla Inventario, junto con la tabla armas, fueron sometidas a una revisión técnica para evaluar posibles mejoras en cuanto a rendimiento y estructura. Uno de los principales ajustes se centró en optimizar la manera en que se recuperaba la información sobre las armas que poseía cada jugador. Para ello, se realizó un cambio en el tipo de dato del ID, reemplazando el tipo anterior por VARCHAR, lo que permitió una recuperación más rápida y eficiente de los datos durante las consultas.

Este cambio, aunque puntual, contribuyó significativamente a mejorar el rendimiento general de las operaciones relacionadas con el inventario del jugador. De esta forma, podemos acceder directamente a las armas poseídas por el jugador con solo acceder a la tabla Inventarios.

Cabe destacar que no fue necesario realizar modificaciones estructurales al resto del modelo entidad-relación, ya que las demás tablas cumplían correctamente con su propósito y no presentaban redundancias ni conflictos que afectaran la funcionalidad del videojuego. Esta estabilidad en el diseño refleja una planificación sólida desde el inicio, enfocada en las necesidades reales del sistema.

Importancia de la normalización y uso de la IA generativa:

La normalización es algo fundamental en el desarrollo de las bases de datos relacionales, pues permite mantener un orden y mantener un orden lógico en los datos por medio de evitar redundancias, garantizando la integridad de los datos.

Desde una perspectiva a largo plazo, esto nos permitirá mantener de mejor forma la base de datos del videojuego. Conforme las necesidades del proyecto van evolucionando, la implementación correcta de las reglas de normalización volverá la escalabilidad una tarea mucho más sencilla. En este tipo de procesos, el uso de modelos de IA generativa se ha vuelto una herramienta bastante útil para complementar el trabajo de un administrador de base de datos, pues al estar estos modelos entrenados, pueden ayudar a identificar redundancias e inconsistencias de forma lógica.

A pesar de esto, los modelos generativos se deben utilizar como herramientas y nunca como reemplazo, pues las habilidades técnicas y la experiencia de un equipo especializado no son poseídas por una IA. La implementación de estas reglas de normalización y la depuración de la información generada representa una tarea igual o más importante que lo que pueda generar la herramienta, pues de nada sirve que te dé información si no se tienen los conocimientos necesarios para la implementación. Si todo se desarrollara únicamente con modelos generativos, tendríamos programas o bases de datos ineficientes e inconsistentes. Esto no aplica solo al área de la gestión de base de datos, sino que se extiende a la generación de código y desarrollo de software en general. En nuestro caso específico, el uso de esta herramienta IA permitió corroborar la estructura de nuestro modelo ER, pero solo como una herramienta de apoyo.

Restricciones de integridad

Las siguientes claves primarias garantizan que cada registro en nuestra base de datos cuente con un identificador único:

- **Jugadores:** (id_jugador INT PRIMARY KEY);
- **Cuentas:** (id_jugador INT PRIMARY KEY (también funciona como FOREIGN KEY de id_jugador en la tabla Jugadores)
- **Partidas:** (id_partida INT PRIMARY KEY);
- **Niveles:** (id_nivel INT PRIMARY KEY);
- **Enemigo:** (id_enemigo INT PRIMARY KEY);
- **Objetos:** (id_objeto INT PRIMARY KEY);
- **Salas:** (id_sala INT PRIMARY KEY);
- **Armas:** (id_arma INT PRIMARY KEY);
- **Estadísticas:** (id_estadísticas INT PRIMARY KEY);

Por otro lado, las siguientes claves foráneas garantizan la relación entre las tablas y se aseguran de que los datos sean correctos:

Cuentas: Relación con el jugador al que pertenece.

- id_jugador INT, FOREIGN KEY (id_jugador) REFERENCES jugadores (id_jugador)

Niveles: Relación con la partida a la que pertenece.

- id_partida INT, FOREIGN KEY (id_partida) REFERENCES partidas(id_partida);

Salas: Relación al nivel al que pertenece

- id_nivel INT, FOREIGN KEY (id_nivel) REFERENCES niveles(id_nivel);

Enemigos_Salas: Relación entre el las salas y los enemigos que contienen.

- id_sala INT, FOREIGN KEY (id_sala) REFERENCES salas(id_sala);
- id_enemigo INT, FOREIGN KEY (id_enemigo) REFERENCES enemigos(id_enemigo);

Objetos_Salas: Relación entre las salas y los objetos que contienen.

- id_sala INT, FOREIGN KEY (id_sala) REFERENCES salas(id_sala);

- id_objeto INT, FOREIGN KEY (id_objeto) REFERENCES objetos(id_objeto);

Inventarios: Relación los diferentes jugadores y el inventario de cada uno.

- id_jugador INT, FOREIGN KEY (id_jugador) REFERENCES jugadores(id_jugador);

Jugadores_Habilidades: Relación entre los jugadores y las habilidades que poseen.

- id_jugador INT, FOREIGN KEY (id_jugador) REFERENCES jugadores(id_jugador);
- id_habilidad INT, FOREIGN KEY (id_habilidad) REFERENCES habilidades(id_habilidad);

Estadísticas: Relación entre los jugadores y sus estadísticas únicas por cada uno.

- id_jugador INT, FOREIGN KEY (id_jugador) REFERENCES jugadores(id_jugador);

Se debe utilizar la restricción PRIMARY KEY AUTO_INCREMENT para aumentar que exista una secuencia lógica en los registros de la base de datos. A continuación se muestra un ejemplo para la id de los jugadores en la tabla ‘jugadores’:

- id_jugador INT PRIMARY KEY AUTO_INCREMENT.

De igual manera, se deben usar restricciones NOT NULL para aquellos valores que no pueden ser nulos dentro de la base de datos, forzando a que tengan un valor determinado. Esto ocurre para los valores de la tabla ‘cuentas’, donde el usuario debe tener un nombre de usuario que no sea NULL y sea único dentro de la base de datos, con la restricción UNIQUE:

- nombre_usuario VARCHAR(15) NOT NULL UNIQUE;

Esto asegura que cada usuario tenga un nombre único en la base de datos, evitando duplicidad de la información.

Por otro lado, cuando un jugador está registrado tiene un registro en la tabla ‘cuentas’. Debido a esto, la tabla cuentas no debe permitir datos nulos para el id del usuario, su nombre y contraseña (NOT NULL).

- tipo_cuenta ENUM(‘registrado’, ‘invitado’) NOT NULL;

Para aquellos atributos que tengan como tipo de dato DATETIME, como las fechas de inicio y fin de las partidas, se debe usar la restricción CURRENT_TIMESTAMP como valor

por defecto, automatizando el proceso de introducir fechas en la tabla. El uso de estas restricciones permiten que MySQL registre de forma automática las fechas y horas de inicio y fin de cada partida, evitando que estos datos se tengan que insertar manualmente.

- fecha_inicio DATETIME DEFAULT CURRENT_TIMESTAMP;
- fecha_fin DATETIME DEFAULT CURRENT_TIMESTAMP;

También se deben utilizar las restricciones ON DELETE CASCADE y ON UPDATE CASCADE para las llaves foráneas, asegurando la integridad de la base de datos para que no existan datos huérfanos.

ON DELETE CASCADE: Si un jugador es eliminado (como puede pasar con los jugadores invitados cuando dejen de jugar), sus partidas y datos asociados deben ser eliminados automáticamente, así como los niveles de las partidas que jugó y las salas dentro de estos, evitando datos huérfanos. Esto evita que existan partidas sin un jugador asociado dentro de la base de datos. De la misma manera, si una partida es eliminada, también se eliminan los niveles y salas relacionadas en cascada, garantizando consistencia en la tabla de datos.

ON UPDATE CASCADE: En el caso de que la id de un jugador se cambia dentro de la tabla ‘jugadores’, se deben actualizar todas las referencias a esta id en las tablas relacionadas, como lo son ‘partidas’ o ‘estadísticas’. Esto evita que haya referencias inválidas en la tabla de datos si los identificadores llegan a cambiar en algún momento.

- **Ejemplo en la tabla ‘partidas’:** FOREIGN KEY (id_jugador) REFERENCES jugadores(id_jugador) ON DELETE CASCADE ON UPDATE CASCADE;
- **Ejemplo en la tabla ‘niveles’:** FOREIGN KEY (id_partida) REFERENCES partidas(id_partida) ON DELETE CASCADE ON UPDATE CASCADE;
- **Ejemplo en la tabla ‘salas’:** FOREIGN KEY (id_nivel) REFERENCES niveles(id_nivel) ON DELETE CASCADE ON UPDATE CASCADE;

De esta forma, si se elimina un jugador, todas sus partidas, así como los niveles y salas asociados se eliminan. Por otro lado, si la id de un jugador cambia dentro de la tabla ‘jugadores’, todas las referencias a esta id en las demás tablas se actualizarán de forma automática, evitando referencias inválidas.

Referencias

Chris, K. (2023, 7 de agosto). *Normalización de base de datos: formas normales 1nf 2nf 3nf ejemplos de tablas*. freeCodeCamp. https://www.freecodecamp.org/espanol/news/normalizacion-de-base-de-datos-formas-normales-1nf-2nf-3nf-ejemplos-de-tablas/#primera-forma

Learn Microsoft. (2024, 6 de junio). *Fundamentos de la normalización de bases de datos*. https://learn.microsoft.com/es-es/office/troubleshoot/access/database-normalization-description