



Documentación Reto DWES

Juego De Consultas SQL “Hydrovia”

Michael Alberto, William Ruiz, Carla Lozano
19/12/2024 | Desarrollo Entorno Servidor | 2º DAW

Índice

1. Breve Descripción.....	3
2. Cómo Jugar	3
2.1 Inicio de Sesión	3
2.2 Opciones al Iniciar Sesión	4
2.3 Breve introducción de la historia	4
2.4 Selección de Dificultad.....	5
2.5 Progresión del Juego	5
2.6. Final del Juego	7
2. Historia.....	7
3. Estructura de Carpetas	8
4. Tecnologías Usadas	9
5. Lógica del proyecto	9
5.1 Arquitectura de software	9
5.2 Técnica por comparación de consultas.....	10
5.3 Gestión de Sesiones y Progreso	11
5.4 Estrategia de Guardado de Datos	11
6. Base de Datos.....	12
6.1 Tablas.....	12
6.2 Normalización Bases de Datos	13
6.3 Evaluación de formas normales	14
7. Diagramas	17
7.1 Diagrama de Casos de Uso.....	17
7.2 Diagrama de Flujo	20

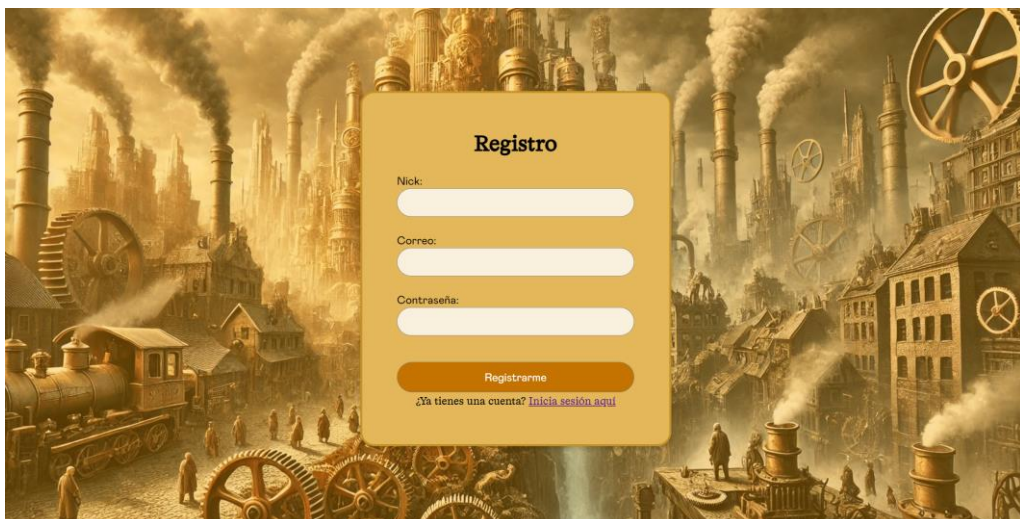
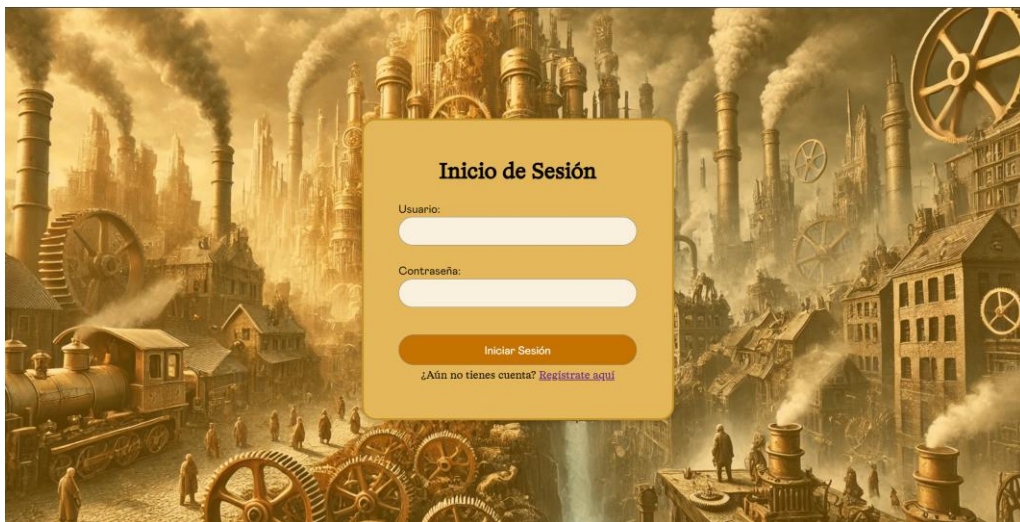
1. Breve Descripción

Hydrovia es un juego interactivo inspirado en la estética **Steampunk** que permite a los usuarios practicar consultas SQL mientras viven una historia emocionante. A lo largo de 12 capítulos, los jugadores deben resolver desafíos relacionados con la gestión de bases de datos para restaurar el equilibrio en un reino dividido por la falta de recursos. Con cada acierto, avanzan en la historia; pero los errores pueden hacerles perder puntos de vida.

2. Cómo Jugar

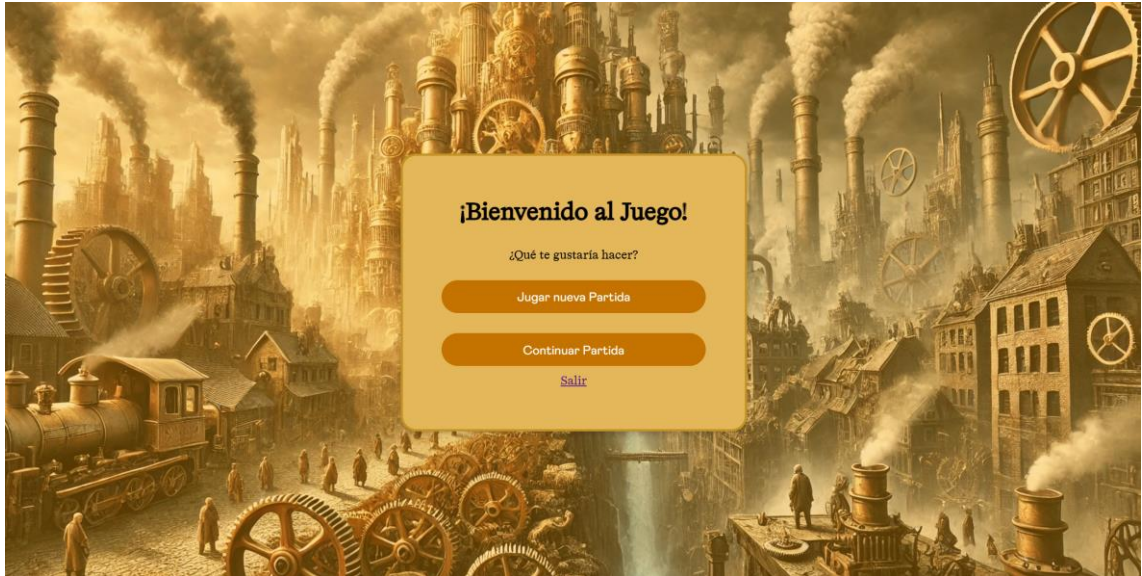
2.1 Inicio de Sesión

- Al acceder al juego, encontrarás un formulario de inicio de sesión.
- Si no tienes cuenta, puedes registrarte desde la opción "**Regístrate aquí**".



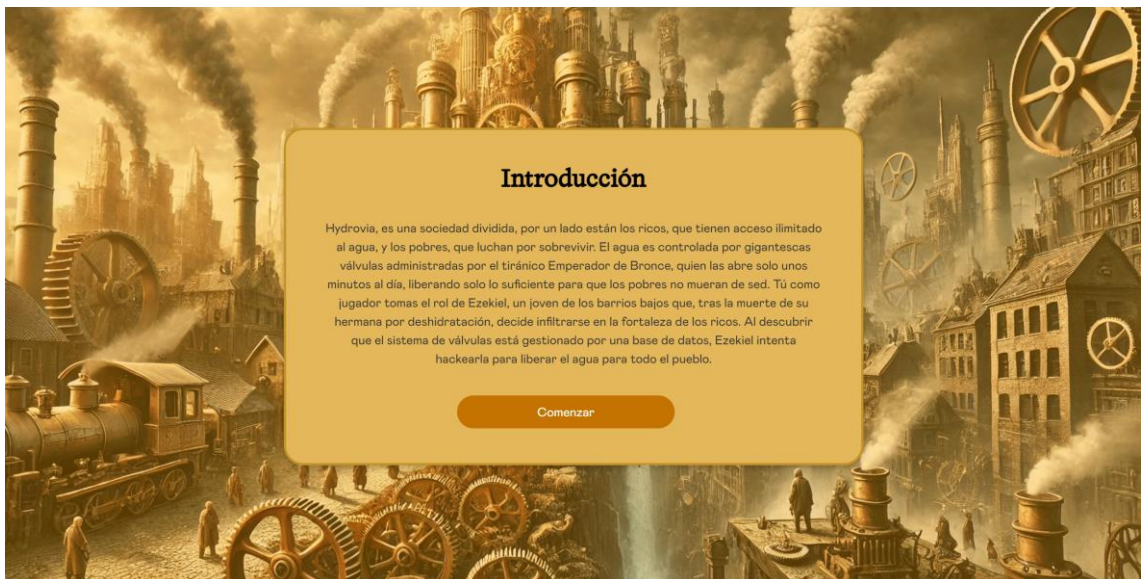
2.2 Opciones al Iniciar Sesión

- **Jugar Nueva Partida:** Comienza desde el capítulo 0, la introducción.
- **Continuar Partida:** Retoma tu progreso anterior gracias al sistema de guardado de sesiones.



2.3 Breve introducción de la historia

- Se dota al usuario de contexto para comprender de qué trata la historia antes de comenzar a jugar.



2.4 Selección de Dificultad

- Elige entre **Fácil, Medio y Difícil**.
- La dificultad afecta la complejidad de las consultas SQL y la cantidad de puntos de vida perdidos por error:
 - Fácil: Pierdes **10 puntos**.
 - Medio: Pierdes **20 puntos**.
 - Difícil: Pierdes **30 puntos**.
- Todos los jugadores comienzan con **100 puntos de vida**.



2.5 Progresión del Juego

- El juego consta de **12 capítulos**.
- En cada capítulo, recibirás una breve descripción de la historia relacionada con ese capítulo y el objetivo de la consulta.
- Se te presentarán tres posibles consultas SQL como opciones (radios).
- Solo una consulta es correcta. Si seleccionas y aceptas la consulta correcta, avanzas al siguiente capítulo.
- Si necesitas ayuda al responder, puedes consultar la base de datos ficticia sobre el juego, que aparece el botón "Consultar Base de Datos".

 Nivel: difícil  100

Capítulo 1: Sed de Venganza

Ezekiel escucha rumores sobre la ubicación de la sala de control del agua. La base de datos está protegida dentro de la Torre de las Mareas, la fortaleza del Emperador de Bronce. Ezekiel encuentra un mapa incompleto en el mercado negro. Consulta la base de datos mapas para buscar las secciones faltantes que lo lleven a la entrada secreta.

Elige la consulta correcta

- ☐ SELECT nombre, descripcion FROM mapas WHERE descripcion = '%entrada secreta';
- ☐ SELECT nombre, descripcion FROM mapas WHERE fortaleza LIKE Torre;
- ☐ SELECT COUNT(*) AS total_mapas FROM mapas WHERE fortaleza = 'Torre de las Mareas';

Aceptar



 Nivel: difícil  100

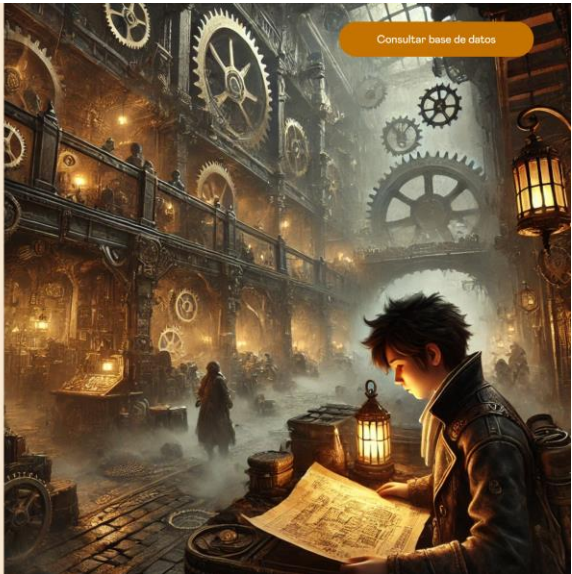
Capítulo 1: Sed de Venganza



Ezekiel escucha rumores sobre la ubicación de la sala de control del agua. La base de datos está protegida dentro de la Torre de las Mareas, la fortaleza del Emperador de Bronce. Ezekiel encuentra un mapa incompleto en el mercado negro. Consulta la base de datos mapas para buscar las secciones faltantes que lo lleven a la entrada secreta.

Elige la consulta correcta

¡Respuesta correcta!

Continuar a Capítulo 2



 Nivel: difícil  100

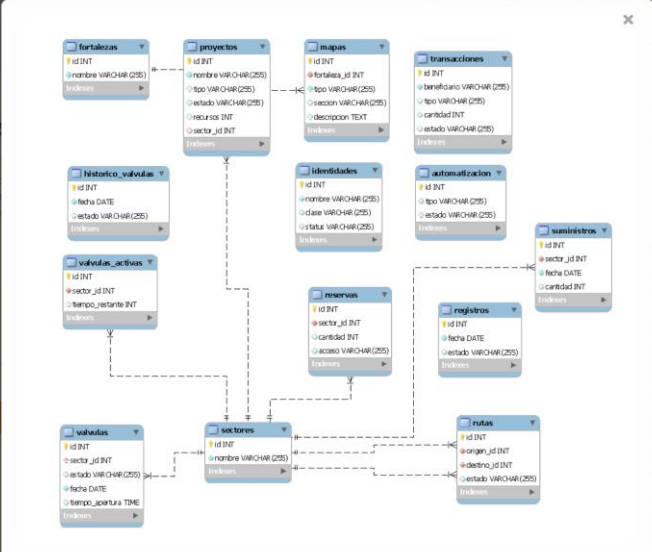
Capítulo 1: Sed de Venganza

Ezekiel escucha rumores sobre la ubicación de la sala de control del agua. La base de datos está protegida dentro de la Torre de las Mareas, la fortaleza del Emperador de Bronce. Ezekiel encuentra un mapa incompleto en el mercado negro. Consulta la base de datos mapas para buscar las secciones faltantes que lo lleven a la entrada secreta.

Elige la consulta correcta

- ☐ SELECT nombre, descripcion FROM mapas WHERE descripcion = '%entrada secreta';
- ☐ SELECT nombre, descripcion FROM mapas WHERE fortaleza LIKE Torre;
- ☐ SELECT COUNT(*) AS total_mapas FROM mapas WHERE fortaleza = 'Torre de las Mareas';

Consultar base de datos



2.6. Final del Juego

- Si completas el capítulo 12 con puntos de vida restantes, habrás cumplido el objetivo: **Restaurar el sistema y salvar a la población.**
 - Si pierdes todos los puntos, deberás reiniciar desde el principio.
-

2. Historia

Hydrovia es un reino sumido en la desigualdad: los ricos disfrutan de abundancia, mientras que los pobres luchan por conseguir lo más básico, como el acceso al agua potable. Esta disparidad se debe a un sistema de distribución de recursos completamente corrupto, gestionado por un complejo sistema de válvulas y datos que los líderes de Hydrovia controlan para mantener el poder.

Tú eres **Ezekiel**, un ingeniero brillante que, cansado de la injusticia, decide tomar el control de la situación. Armado con tu conocimiento de sistemas de datos y tecnología, te adentras en el corazón del sistema para manipular las válvulas y redistribuir el agua entre las distintas áreas del reino.



3. Estructura de Carpetas

A continuación, vamos a detallar la estructura de carpetas con la que hemos trabajado para el proyecto, basado en el MVC.

Hydrovia

- |
 - |—— index.php # Página principal y pantalla de login
- |
 - |—— **/controlador** # Lógica de controladores del juego
 - |—— controladorDeGuardadoSesion.php
 - |—— constructorDeCapitulo.php
 - |—— controladorRegistro.php
 - |—— FuncionesDatosJson.php
 - |—— ingresar.php
- |
 - |—— **/modelo** # Conexión a la base de datos y funciones relacionadas
 - |—— conexionPDO.php
- |
 - |—— **/vista** # Archivos para la interfaz de usuario
 - |—— capitulo.php # Vista controlar el capítulo y actualizar la sesión del usuario.
 - |—— capitulo0.php # Vista para el Capítulo 0 (introducción)
 - |—— derrota.php # Vista cuando el jugador pierde
 - |—— eleccionNiveles.php # Vista para seleccionar nivel dificultad
 - |—— formularioRegistro.php # Formulario para registrar una nueva cuenta
 - |—— inicioJuego.php # Vista principal al comenzar el juego
 - |—— juegoCompletado.php # Vista cuando el jugador completa el juego
 - |—— pruebaRestaurarDatos.php # Vista para probar restaurar datos de sesión
 - |—— /js # Subcarpeta para los archivos Javascript
 - |—— efectoTextoCap0.js
 - |—— modal.js
 - |—— /css # Subcarpeta para CSS
 - |—— estilos.css # Archivo de estilos CSS
 - |—— /img # Subcarpeta para Imágenes para la interfaz del juego
 - |—— fondo.png
 - |—— fondo2.png
 - |—— icono_salir.png
 - |—— imagenBD.png
 - |—— /imgCapitulos # Subcarpeta para Imágenes relativas a cada capítulo
 - |—— capitulo01.png
 - |—— capitulo02.png
 - |—— ... (hasta capitulo12.png)
 - |—— /fonts # Subcarpeta para fuentes personalizadas
 - |—— HALTimezone.ttf
 - |—— Mabry-Pro.ttf
 - |—— README.md # Documentación del proyecto

4. Tecnologías Usadas

- **Backend:** PHP
 - **Frontend:** HTML5, CSS3, JavaScript.
 - **Base de Datos:** MySQL
-

5. Lógica del proyecto

5.1 Arquitectura de software

Nuestro proyecto sigue el patrón de arquitectura de software **Modelo-Vista-Controlador (MVC)**, que se utiliza para separar la lógica de la aplicación en tres componentes principales: el **Modelo**, la **Vista** y el **Controlador**. Este enfoque ayuda a mantener el código modular, escalable y fácil de mantener, al mismo tiempo que facilita la evolución de las distintas partes de la aplicación.

- **Modelo (Model)**

El **Modelo** en nuestra aplicación es responsable de la gestión de los datos. Aquí se encuentra la parte encargada de interactuar con la base de datos.

Utilizamos el objeto **PDO** para realizar las conexiones con la base de datos y ejecutar consultas **SQL** de manera eficiente y segura. Los datos que se obtienen o se guardan en la base de datos son gestionados a través de este componente, que también se encarga de actualizar la sesión del usuario y restaurar los datos a medida que el jugador avanza en el juego.

- **Vista (View)**

La **Vista** es la encargada de presentar la información al usuario y de capturar las interacciones de éste. En este proyecto, la Vista está constituida por archivos PHP que generan también contenido HTML dinámico (incluyendo formularios, mensajes, imágenes, etc.) para mostrar al jugador los capítulos del juego, las opciones de dificultad, y los resultados de las acciones. Los archivos de Vista son interactivos, mostrando mensajes de error, avances, y proporcionan una interfaz atractiva para facilitar la experiencia de usuario, siguiendo con la estética Steampunk.

- **Controlador (Controller)**

El **Controlador** es el que maneja la lógica de negocio del juego. Recibe las acciones del usuario desde la Vista, las procesa, y, dependiendo de la interacción, actualiza el Modelo o la Vista. En este caso, el controlador se encarga de gestionar el inicio de sesión, la selección de dificultad, el progreso a través de los capítulos, y la verificación de las respuestas del jugador. Además, gestiona el sistema de puntuación y vida del jugador, asegurándose de que las reglas del juego se apliquen correctamente. El controlador se comunica con el Modelo para recuperar datos o almacenar el progreso del usuario.

5.2 Técnica por comparación de consultas

Uno de los aspectos interesantes de la lógica del juego es cómo se gestionan las consultas SQL que deben ser respondidas por el jugador. Para cada capítulo, se presentan tres consultas SQL como opciones, y el jugador debe seleccionar la correcta. La comparación de la respuesta seleccionada por el jugador con la consulta correcta se realiza mediante un proceso de validación utilizando **consultas y arrays**.

Proceso de Validación:

1. **Tres opciones de consulta:** En cada capítulo, se muestran tres consultas SQL. Una de ellas es la correcta y las otras dos son incorrectas.
2. **Guardar las consultas correctas:** La consulta correcta esta guardada en la base de datos como un **string**.
3. **Comparación:** Cuando se carga el capítulo, de forma automática se hace una petición a la base de datos para poder extraer la respuesta correcta según el capítulo y nivel de dificultad actual del capítulo, si es la primera vez que se juega el capítulo por defecto estaría en el 1, esta respuesta es almacenada en una array, además en simultaneo se hace otra petición a la base de datos por las opciones de respuesta erróneas (extraerá 2) y las guardará en un array, ambos arrays se fusionan y son usados para generar y rellenar las opciones de respuesta que tendrá el capítulo. Al jugar y fallar en la respuesta las respuestas se vuelven a mezclar y varían en su posición para que sea más complicado atinar con la correcta. Si el jugador acierta saldrá un botón de continuar con el capítulo siguiente.
4. **Progreso del jugador:** Si la respuesta es correcta, el jugador avanza al siguiente capítulo. Si la respuesta es incorrecta, se pierde una cantidad determinada de puntos de vida según la dificultad seleccionada, y el jugador debe intentar nuevamente.

Uso de JSON

El uso de **JSON** para el guardado de los datos de la sesión asegura que los datos del jugador se validen de forma precisa y consistente. Al tratar los datos como cadenas codificadas en JSON, evitamos problemas con los inserts y reducimos mucho el uso de consultas SQL ya que el JSON es un objeto con todos los datos que necesitamos para gestionar la partida del usuario y al ser solo un objeto podemos extraerlo de la base de datos con una única consulta y actualizarlo con otra.

Además, esta técnica protege la **base de datos** al evitar que el usuario interactúe directamente con ella. Si el jugador pudiera hacer consultas directas, como múltiples **delete**, podría dañar la base de datos. Al manejar casi todo a través de JSON, garantizamos que las interacciones sean seguras y controladas.

5.3 Gestión de Sesiones y Progreso

El sistema de **sesiones** se utiliza para rastrear el progreso del jugador a lo largo del juego. Al iniciar sesión, el jugador puede continuar desde el capítulo donde dejó el juego, gracias al uso de variables de sesión que almacenan:

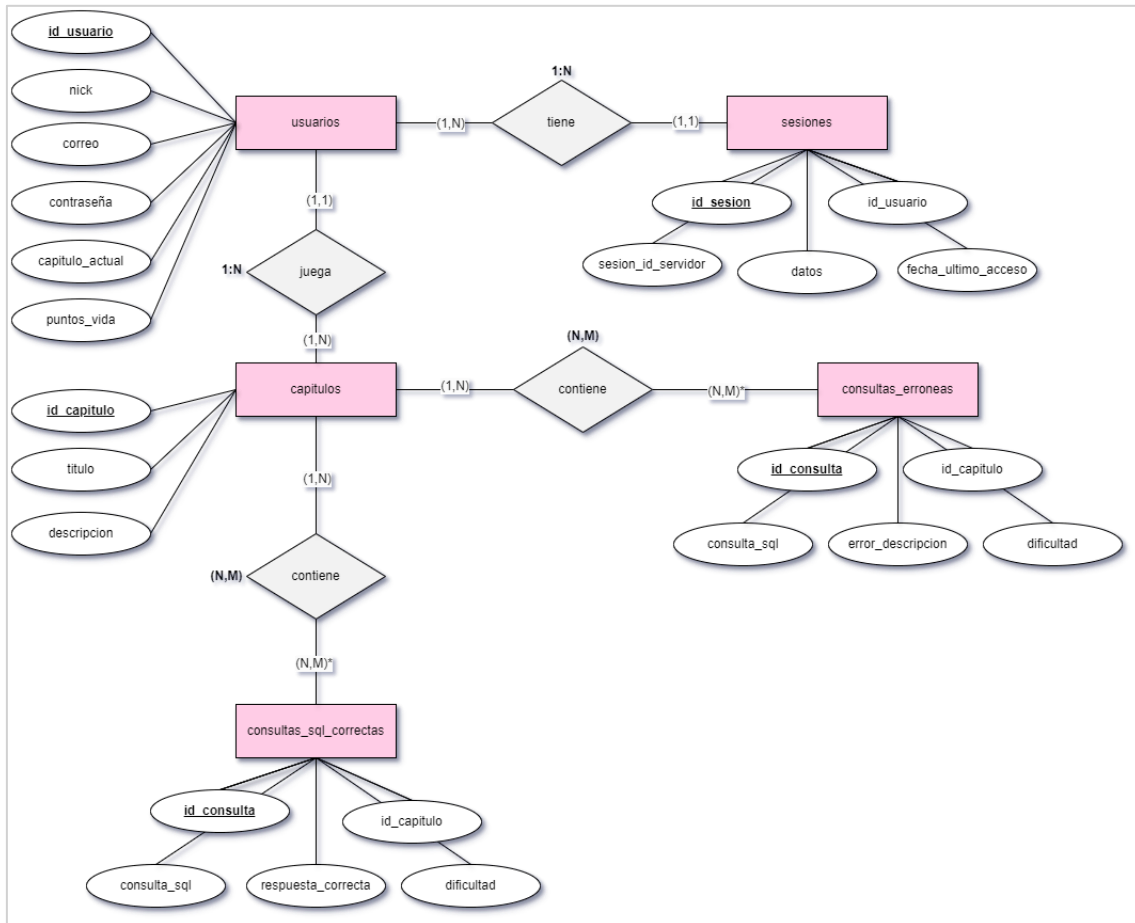
- **Puntos de vida:** Dependiendo de la dificultad seleccionada, los puntos de vida del jugador se restan cuando responde incorrectamente.
- **Capítulo actual:** Almacena el capítulo que el jugador está jugando actualmente. De este modo, puede continuar el juego sin perder el progreso.
- **Dificultad seleccionada:** La dificultad afecta la complejidad de las consultas SQL y la cantidad de puntos de vida perdidos por error.

5.4 Estrategia de Guardado de Datos

Se utiliza un sistema de guardado de datos prácticamente en tiempo real, donde la información de progreso del jugador se guarda en la base de datos después de cada interacción importante. Esto permite que el jugador no pierda su avance incluso si cierra la aplicación o se desconecta. Al restaurar la sesión al iniciar nuevamente, el jugador podrá retomar el juego desde donde lo dejó.

6. Base de Datos

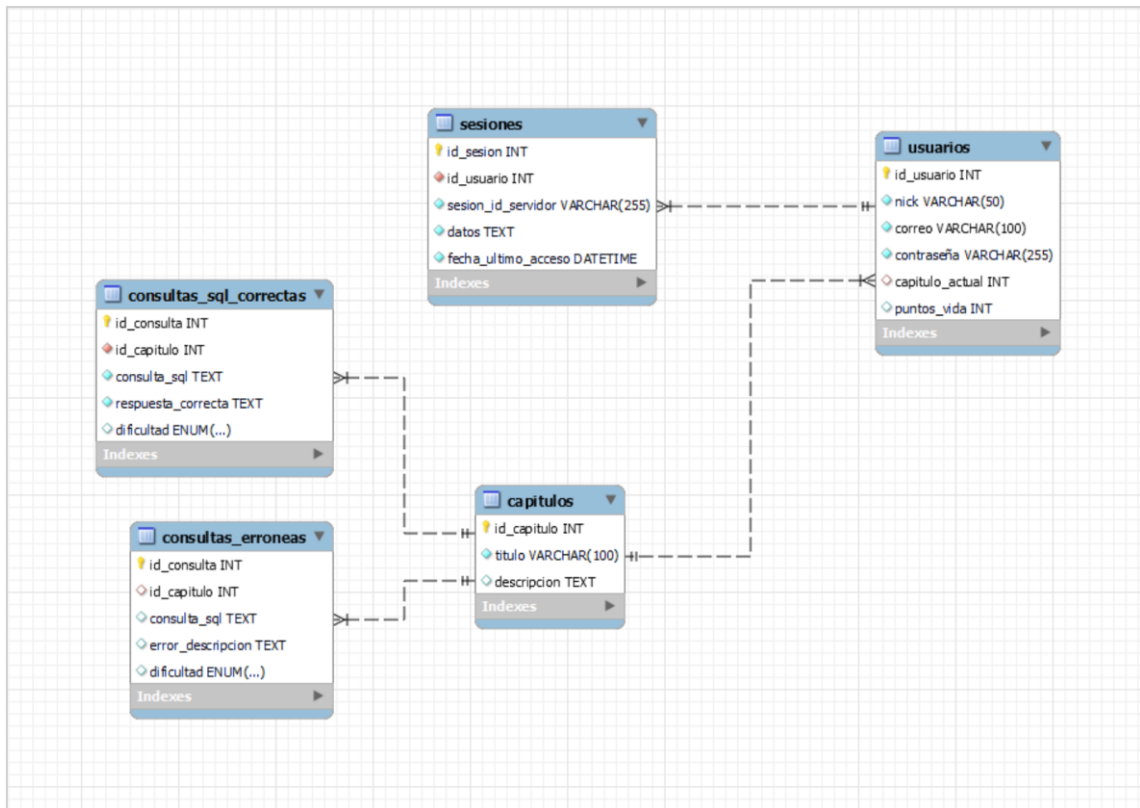
A continuación mostramos el Modelo ER gráfico:



6.1 Tablas

La base de datos `bd_reto` se utiliza para almacenar la información del juego y consta de las siguientes tablas:

- **usuarios:** Almacena la información de los usuarios registrados, incluyendo nombre, correo, contraseña, progreso (capítulo actual) y puntos de vida.
- **capitulos:** Contiene los detalles de los capítulos del juego, como el título y la descripción de cada uno.
- **sesiones:** Registra información sobre las sesiones activas de los usuarios, permitiendo continuar el juego desde donde lo dejaron.
- **consultas_sql_correctas:** Guarda las consultas SQL correctas para cada capítulo y nivel de dificultad.
- **consultas_erroneas:** Almacena las consultas erróneas y sus respectivas descripciones, que ayudan al jugador a identificar sus errores.



6.2 Normalización Bases de Datos

La base de datos que hemos diseñado para el reto cumple hasta las 3 primeras fases normales de normalización porque consideramos que de esa forma la base de datos es eficiente para almacenar el flujo del juego creado.

A continuación, vamos a detallar cada una de las tablas:

1. capitulos

- Contiene información sobre los capítulos del juego
- La clave primaria es: id_capitulo

2. usuarios

- Almacena información sobre los usuarios creados para el juego
- La clave primaria es: id_usuario
- La clave foránea es id_capitulo que hace referencia a la tabla capitulos (id_capitulo)

3. sesiones

- Representa a la información de las sesiones creadas para cada usuario en el juego
- La clave primaria es: id_sesion

- La clave foránea es id_usuario que hace referencia a la tabla usuarios (id_usuario)

4. consultas_sql_correctas

- Contiene consultas correctas para un capítulo
- La clave primaria es: id_consulta
- La clave foránea es id_capitulo que hace referencia a la tabla capitulos (id_capitulo)

5. consultas_erroneas

- Almacena consultas incorrectas con descripciones sobre el error
- La clave primaria es: id_consulta
- La clave foránea es id_capitulo que hace referencia a la tabla capítulos (id_capitulo)

6.3 Evaluación de formas normales

1FN (PRIMERA FORMA NORMAL)

Requiere:

- Que cada columna debe contener valores atómicos, es decir deben almacenar un dato único indivisible.
- No puede existir repeticiones de grupos de columnas.

Evaluación:

Tablas y claves primarias:

- Todas las tablas tienen una clave primaria definida (id_capitulo, id_usuario, id_sesion, id_consulta).
- No hay evidencia de celdas que contengan valores no atómicos. Por ejemplo:
 - En la tabla capítulos: titulo y descripcion son valores atómicos.
 - En usuarios, cada columna tiene un solo valor por celda (correo, contraseña, etc.).
- No se encuentran listas o combinaciones de datos en columnas.

Cumplimiento:

Todas las tablas cumplen con la **1FN** porque:

- Los valores son atómicos.
- Cada fila es única y tiene una clave primaria definida.

2FN (SEGUNDA FORMA NORMAL)

Requiere:

- Cumplir con la 1FN.
- Que todas las columnas no clave dependan completamente de la clave primaria. No puede haber dependencias parciales.

Evaluación:

Para cada tabla analizamos si las columnas no clave dependen completamente de la clave primaria.

1. Tabla capitulos:

- Clave primaria: id_capitulo.
- Las columnas titulo y descripcion dependen completamente de id_capitulo.
- No hay dependencias parciales.

2. Tabla usuarios:

- Clave primaria: id_usuario.
- Las columnas como nick, correo, contraseña, capitulo_actual y puntos_vida dependen completamente de id_usuario.
- **Aclaración:** Aunque capitulo_actual es una clave externa, depende completamente de id_usuario porque indica el progreso del usuario.

3. Tabla sesiones:

- Clave primaria: id_sesion.
- Las columnas id_usuario, sesion_id_servidor, datos y fecha_ultimo_acceso dependen completamente de id_sesion.

- **Aclaración:** La relación con usuarios a través de id_usuario no incumple la 2FN.

4. Tablas consultas_sql_correctas y consultas_erroneas:

- Ambas tienen como clave primaria id_consulta.
- Todas las columnas (id_capitulo, consulta_sql, respuesta_correcta en una; descripcion_error en la otra) dependen completamente de id_consulta.

Cumplimiento:

Todas las tablas cumplen con la **2FN** porque no hay dependencias parciales en ninguna.

3FN (TERCERA FORMA NORMAL)

Requiere:

- Cumplir con la 2FN.
- Que no haya **dependencias transitivas**, es decir, ninguna columna no clave debe depender de otra columna no clave.

Evaluación:

Para cada tabla, revisamos las dependencias funcionales que puedan ser transitivas.

1. Tabla capitulos:

- No hay columnas no clave que dependan de otras columnas no clave.
Por ejemplo, titulo y descripcion dependen únicamente de id_capitulo.

2. Tabla usuarios:

- capitulo_actual depende de id_usuario como clave externa.
- No hay evidencia de dependencias transitivas entre columnas no clave.

3. Tabla sesiones:

- Todas las columnas dependen únicamente de id_sesion.
- Aunque id_usuario es una clave externa, no genera dependencias transitivas entre otras columnas.

4. Tablas consultas_sql_correctas y consultas_erroneas:

- En ambas tablas, todas las columnas no clave dependen directamente de id_consulta.
- No hay dependencias transitivas observables.

Cumplimiento:

Todas las tablas cumplen con la **3FN** porque no se detectan dependencias transitivas.

En resumen, el diseño es eficiente y no presenta redundancias ni anomalías evidentes.

Las claves externas (id_capitulo en consultas_sql_correctas, id_capitulo en consultas_erroneas, id_usuario en sesiones) están bien definidas y no afectan la normalización.

7. Diagramas

7.1 Diagrama de Casos de Uso

Creación de Escenario

Para representar el escenario general de uso del sistema de juego elaborado primero se identifican los actores principales y luego se describen los casos de uso asociados a cada actor. Los actores en este contexto son principalmente el "Usuario" y el "Sistema de Juego".

Actores

- Usuario: Persona que interactúa con el juego.
- Sistema de juego: Entidad interna responsable de validar, registrar, y gestionar las partidas.

Casos de uso:

1. Iniciar Sesión

- **Actor:** Usuario
- **Descripción:** El usuario proporciona credenciales (nick y contraseña) para acceder al sistema.

2. Registrar Usuario

- **Actor:** Usuario
- **Descripción:** Permite crear una cuenta proporcionando nick, correo y contraseña.
- **Relación:** Incluye (include) el caso de uso "Iniciar Sesión" porque, tras registrarse, el usuario debe iniciar sesión para acceder al sistema.

3. Elegir Nueva Partida o Continuar

- **Actor:** Usuario
- **Descripción:** El usuario selecciona entre comenzar una nueva partida o continuar la partida desde donde dejó.
- **Relación:** Incluye (include) "Elegir Nivel de Dificultad" cuando se selecciona "Nueva Partida".

4. Elegir Nivel de Dificultad

- **Actor:** Usuario
- **Descripción:** El usuario selecciona un nivel entre Fácil, Medio o Difícil para empezar el juego.

5. Jugar Capítulo

- **Actor:** Usuario
- **Descripción:** Representa la interacción del jugador con cada capítulo del juego.
- **Relación:** Extiende (extends) "Perder Partida" si los puntos de vida llegan a cero.

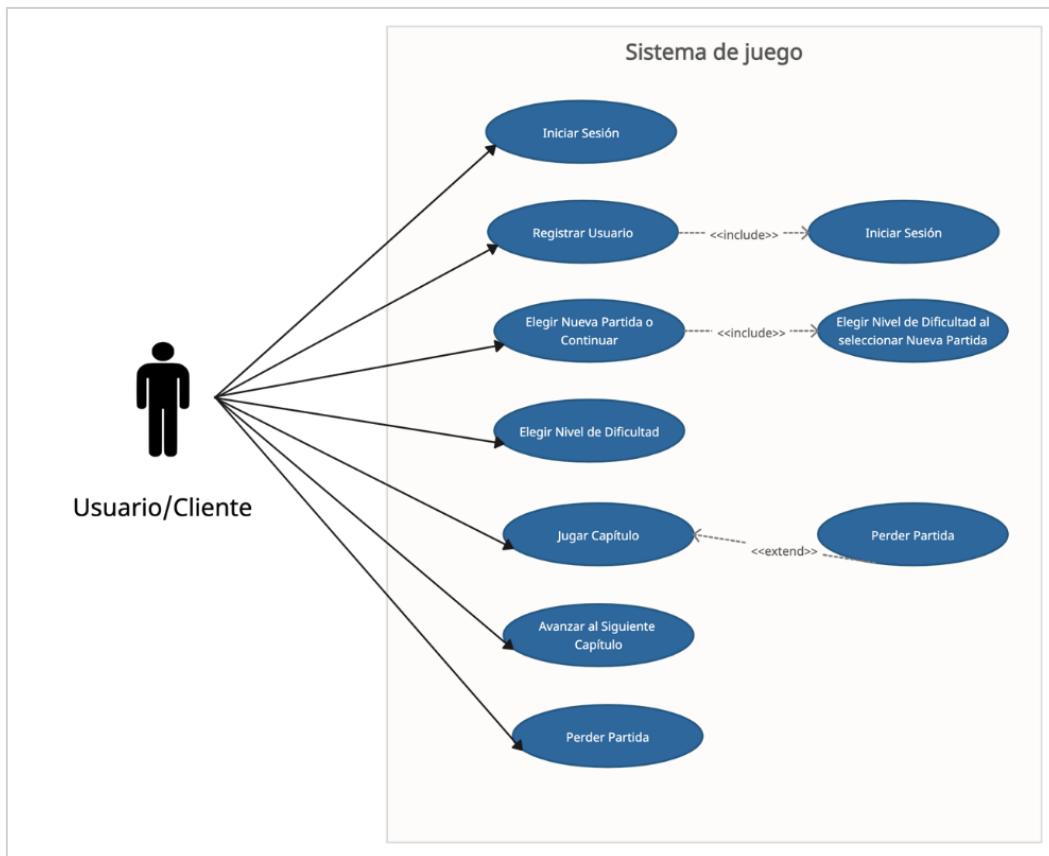
6. Avanzar al Siguiente Capítulo

- **Actor:** Sistema de Juego
- **Descripción:** El sistema evalúa si la respuesta a la consulta SQL es correcta y avanza al próximo capítulo.

7. Perder Partida

- **Actor:** Sistema de Juego
- **Descripción:** Si el jugador consume todos los puntos de vida, el sistema redirige automáticamente a la pantalla de inicio del juego.

A continuación, presentamos el diagrama de Casos de Uso del juego:



7.2 Diagrama de Flujo

A continuación, presentamos el diagrama de Flujo del juego:

