

Sitio de juegos online

Versión 3.0

Abril 2025

Nombre de los autores:

- Brandon Iza
- Rhommel Cardona

ÍNDICE

1	Introducción	3
2	Estudio inicial	4
3	Modelo Entidad Relación.....	5
4	Modelo Relacional	9
5	Creación de la base de datos.....	11
6	Inserción de datos en la base de datos	17
7	Consultas	19
8	Índices	30
9	Vistas.....	33
10	Triggers	39
11	Procedimientos y funciones.....	42
12	Copias de seguridad	47
12	Conclusión.....	48

1 Introducción

El presente documento constituye la culminación de un esfuerzo colaborativo y metódico en el diseño e implementación de una base de datos relacional para un Sitio de Juegos Online, desarrollado como parte integral del módulo de Bases de Datos del Ciclo Formativo de Grado Superior en Administración de Sistemas Informáticos en Red (ASIR). Este proyecto no solo ha sido un ejercicio académico, sino una simulación realista de los desafíos que implica gestionar sistemas de información complejos, desde el análisis de requisitos hasta la implementación de funcionalidades avanzadas.

A lo largo del curso, hemos integrado conocimientos teóricos y prácticos para abordar problemas como la normalización de datos, la optimización del rendimiento y la implementación de reglas de negocio específicas. El proceso iterativo de diseño —desde el Modelo Entidad-Relación (MER) inicial hasta su refinamiento final— refleja nuestra evolución en la comprensión de las relaciones entre entidades y la importancia de garantizar la integridad referencial. Herramientas como draw.io para el modelado gráfico y MariaDB para la implementación física han sido pilares fundamentales en este recorrido.

Además, este proyecto ha fomentado habilidades críticas como el trabajo en equipo, la resolución de conflictos técnicos (por ejemplo, garantizar el anonimato del creador de partidas) y la adaptación a retroalimentación constante. La generación asistida de datos mediante IA, la validación rigurosa de consultas y la creación de estructuras avanzadas (índices, triggers, procedimientos) subrayan nuestro compromiso con un enfoque profesional y detallista.

2 Estudio inicial

Sitio de juegos online por Internet

Un sitio de juegos online por Internet desea contar con una base de datos para gestionar los usuarios, juegos y partidas que se desarrollan en el mismo. El funcionamiento del sitio es el siguiente:

Cuando un usuario intenta entrar en este sitio, se le pedirá un login y un password. El sistema comprobará si el usuario tiene cuenta y en caso negativo se le pedirán los siguientes datos de alta antes de darle acceso: nombre, correo, nick (nombre de batalla), login y password. Se comprobará si ya existía con distinto login y password para darle un mensaje de error en caso afirmativo. Hay que tener en cuenta que el nick es único.

Una vez el usuario se ha dado de alta o ha entrado con su login y password correctos, puede visitar los distintos salones donde se están desarrollando las partidas. No se desea que quede constancia de dichos salones en la base de datos. Si un usuario quiere entrar en una partida o crear una nueva, tiene que tener un avatar¹ que será su representación en el mundo virtual. Un usuario podrá tener distintos avatares, pero cada avatar sólo pertenecerá a un usuario. De los avatares se almacenará el aspecto y el nivel y se identificará por el nick del propietario. Hay que tener en cuenta que cada avatar sólo sirve para un tipo de juego, mientras que en un juego puede haber registrados varios avatares. Los responsables del sitio quieren que quede constancia de esto en la base de datos. De los tipos de juegos se quiere almacenar un código identificador, nombre y descripción.

Los usuarios que tengan en casa el juego apropiado, podrán crear partidas de ese juego para que otros usuarios se unan a la partida o unirse a partidas existentes, siempre utilizando el avatar correspondiente. De las partidas se almacenará un código de partida, un password (opcional) para acceder a la partida, la fecha y hora de creación, el nombre de la partida y el estado (en curso o finalizada). Además hay que tener en cuenta que una partida sólo puede ser de un tipo de juego y un juego tener varias partidas. Se desea que quede constancia de esta restricción en la base de datos.

Las partidas se podrán dejar a medias para continuarlas otro día. Cuando un usuario crea una partida, puede dar un password de entrada para limitar el acceso. No quedará constancia de cual usuario es el creador de una partida. Los usuarios que se unen a una partida (a través de sus avatares) con password quedarán registrados de manera que si quieren abandonarla y unirse más tarde, no tengan que volver a introducir el password de dicha partida. Nunca se permitirá a los usuarios conectarse a partidas que se han dado por terminadas.

Un servicio de interés es que se pueda consultar los enfrentamientos que hay entre los distintos avatares en las partidas y el resultado de dicho enfrentamiento de cualquier partida en curso o terminada.

¹ Avatar: (DRAE) En la religión hindú, encarnación terrestre de alguna deidad.

3 Modelo Entidad Relación

3.1. Proceso de Diseño Conceptual: Una Evolución Iterativa

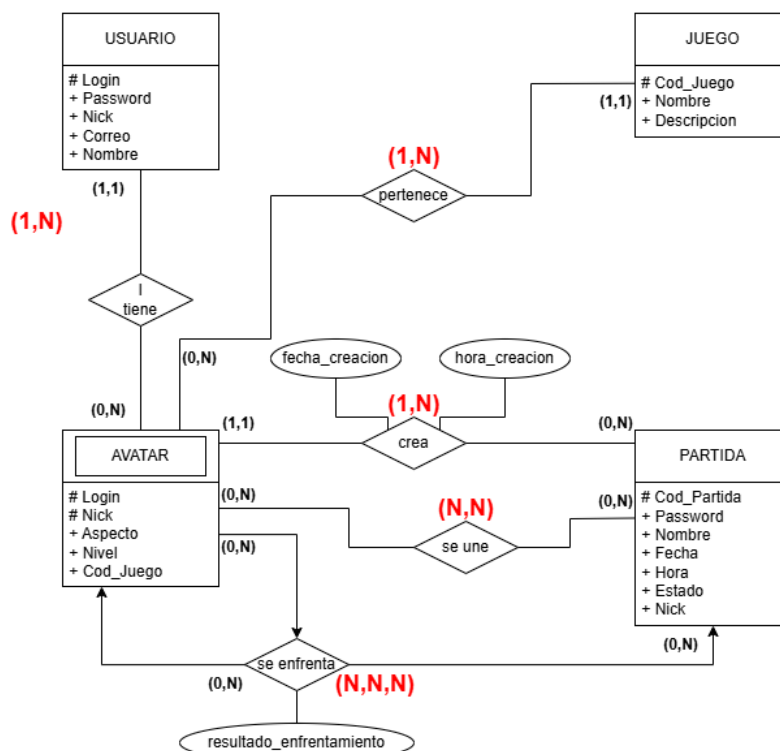
El diseño de la estructura de datos para el "Sitio de Juegos Online" fue un proceso iterativo y de aprendizaje continuo. Partimos de un análisis inicial del enunciado del proyecto para identificar las principales entidades y relaciones, plasmándolas en un primer Modelo Entidad-Relación (MER). A medida que avanzaba el curso y profundizábamos en los principios de modelado relacional y las particularidades del enunciado, fuimos refinando este modelo inicial a través de sucesivas versiones hasta alcanzar el diseño final implementado. Este enfoque iterativo nos permitió corregir errores conceptuales iniciales, optimizar la representación de las relaciones y asegurar una mayor coherencia y eficiencia en el modelo.

Para la creación y modificación de los diagramas MER en cada etapa, utilizamos la herramienta gráfica draw.io (<http://draw.io>).

3.2. Modelo Inicial (MER Inicial ,Tablas Blanco y Negro Ver Anexo 1)

Nuestra primera aproximación (representada en el **MER Inicial**, diagrama en blanco y negro, disponible en el Anexo 1) se centró en identificar las entidades más evidentes (USUARIO, JUEGO, AVATAR, PARTIDA) y las relaciones básicas descritas en el texto (tiene, pertenece, crea, se une, se enfrenta). Este modelo inicial, aunque útil como punto de partida, presentaba limitaciones significativas, como la presencia de relaciones Muchos-a-Muchos (N:M – se une) y una relación ternaria (N:N:N - se enfrenta), que no estaban resueltas de forma óptima para una implementación relacional directa. Además, la interpretación inicial de la relación crea desde JUEGO a PARTIDA necesitaba revisión.

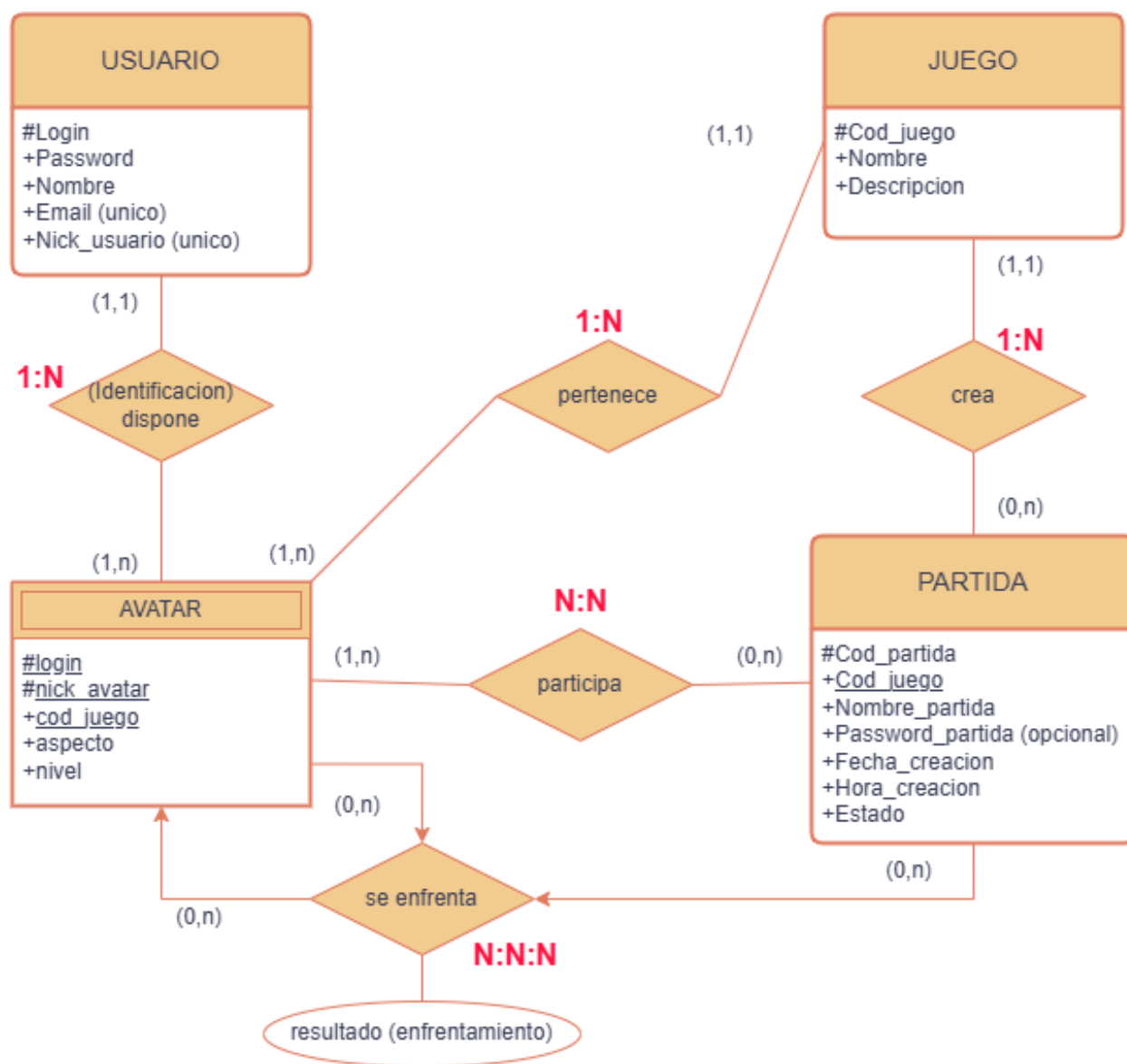
MODELO ENTIDAD-RELACION:



3.3. Segundo Modelo (MER Refinado - Tablas Amarillas, Ver Anexo 1)

En una segunda iteración (representada en el **MER Refinado**, diagrama con tablas amarillas, también en el Anexo 1), intentamos refinar algunas de las relaciones y la representación de atributos. Vinculamos la creación de partidas (crea) más directamente con el AVATAR (1:N) y asociamos atributos como fecha_creacion y hora_creacion a esta relación. También formalizamos la relación N:M se une entre AVATAR y PARTIDA. Sin embargo, este modelo aún conservaba la compleja relación ternaria N:N:N se enfrenta sin resolverla mediante tablas intermedias. Además, y crucialmente, la relación crea desde AVATAR (que identificaba al propietario del avatar como creador) entraba en **conflicto directo** con el requisito explícito del enunciado: *"No quedará constancia que cual usuario es el creador de una partida"*. Esta etapa fue valiosa para explorar alternativas, pero evidenció la necesidad de un rediseño más profundo.

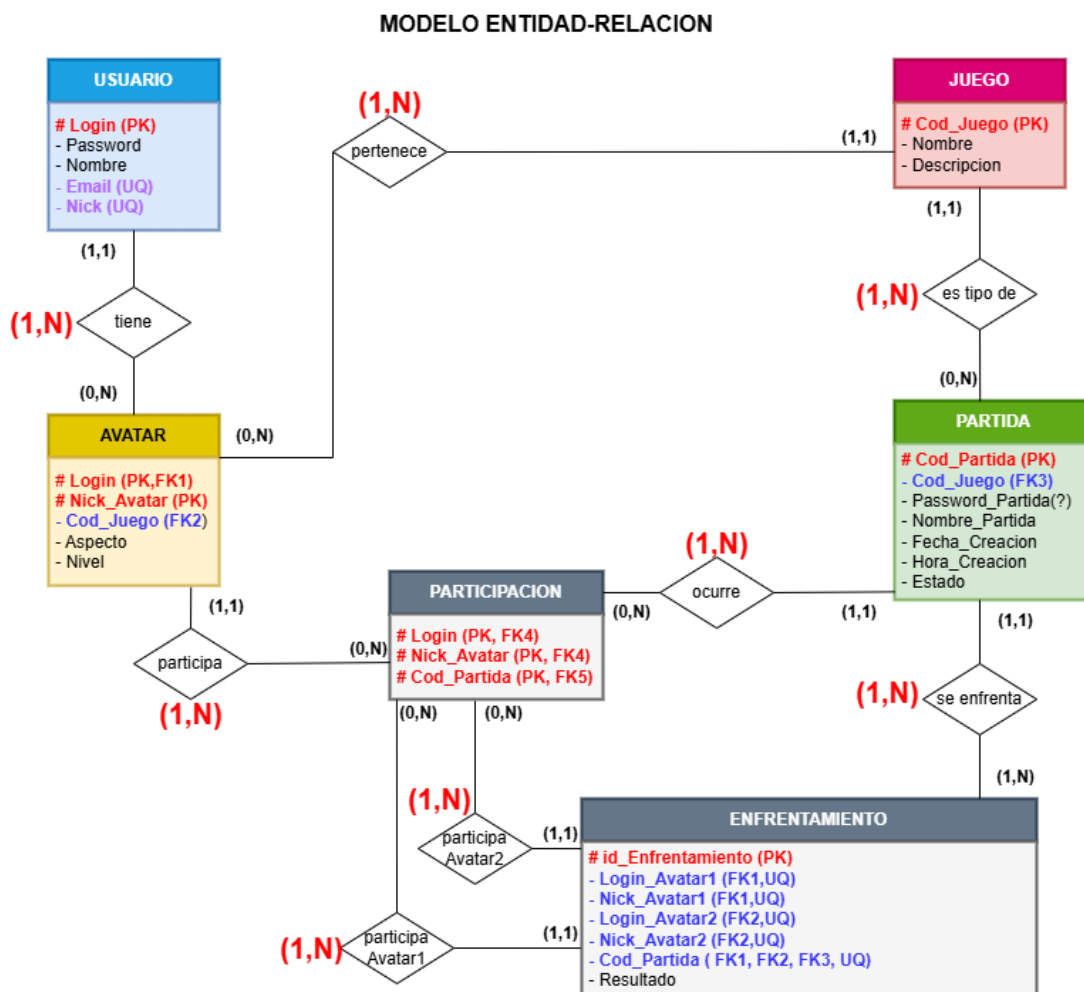
MODELO ENTIDAD-RELACION



3.4. Refinamiento Final y Modelo Definitivo (MER Final - Tablas de Colores, Ver Anexo 1)

La reflexión sobre los modelos anteriores, la comprensión clara de los requisitos (especialmente el anonimato del creador) y la aplicación de las técnicas correctas de modelado relacional nos llevaron al diseño definitivo (representado en el **MER Final**, diagrama con tablas de colores, detallado en el Anexo 1):

- **Cumplimiento del Requisito del Creador:** Se **eliminó** por completo la relación crea que vinculaba al creador con la partida. Los atributos Fecha_Creacion y Hora_Creacion se establecieron como propiedades intrínsecas de la entidad PARTIDA.
- **Resolución de N:M (Participación):** La relación N:M entre AVATAR y PARTIDA se resolvió de forma estándar mediante la **tabla asociativa PARTICIPACION**. Esta tabla actúa como puente, conteniendo las claves primarias de AVATAR (Login, Nick_Avatar) y PARTIDA (Cod_Partida), formando su propia PK compuesta y estableciendo las FKs correspondientes a las tablas originales.
- **Resolución de Enfrentamientos:** La compleja interacción de enfrentamientos se modeló creando la tabla **ENFRENTAMIENTO**. Esta tabla no se relaciona directamente con AVATAR, sino que se vincula **dos veces con PARTICIPACION** (mediante FKs compuestas que incluyen Login, Nick_Avatar y Cod_Partida) para identificar de forma única a los dos contendientes y asegurar que ambos pertenecen a la **misma partida**. También incluye una FK directa a PARTIDA.
- **Consolidación:** Se definieron formalmente todas las PKs (incluyendo las compuestas), FKs y UQs (Email y Nick en USUARIO), asegurando la coherencia global del modelo.



3.5. Diagrama MER Final y Evolución Documentada

El **ANEXO 1** de este documento contiene los tres diagramas MER mencionados (MER Inicial - Tablas Blanco y Negro; MER Refinado - Tablas Amarillas; y MER Final - Tablas de Colores). Esta presentación busca ilustrar la **evolución de nuestro diseño**, mostrando cómo, a través del análisis crítico y la aplicación de los conocimientos adquiridos, pudimos identificar y corregir las deficiencias de los modelos iniciales y llegar a una estructura final robusta, coherente y optimizada. El archivo fuente (MER-MR-sitiojuegosonline.drawio) de cada versión también se adjunta.

3.6. Entrega de anexos

La representación gráfica que combina el MER final y del MR , como la leyenda detallada, se encuentran en el **ANEXO 1**. El archivo fuente (MER-MR-Final-Sitio-Juegos-Online.drawio) se adjunta según lo solicitado

4 Modelo Relacional

En este punto se presenta el diagrama del Modelo relacional (MR). Al igual que el modelo MER se habrá de realizar con la misma herramienta gráfica comentada en el punto anterior (draw.io).

Se realizarán los comentarios oportunos y las consideraciones que se han tenido en cuenta a la hora de transformar el modelo MER a modelo MR.

4.1. Presentación del Modelo Relacional (MR)

En este punto, presentamos el **Modelo Relacional (MR)** de la base de datos sitiojuegosonline. Este modelo es la traducción directa del Modelo Entidad-Relación (MER) final a una estructura lógica compuesta por tablas, columnas y relaciones explícitas mediante claves primarias y foráneas, lista para ser implementada en un SGBD relacional como MariaDB. Al igual que el MER, este esquema se ha desarrollado utilizando draw.io para la representación visual combinada y refinado textualmente para asegurar la precisión.

4.2. Esquema del Modelo Relacional

El esquema define las siguientes seis tablas, con sus respectivas columnas, claves primarias (indicadas con #) y claves foráneas (indicadas con subrayado y un marcador FKX):

MODELO RELACIONAL

USUARIO (#Login, Password, Nombre, Email, Nick)

AVATAR (#Login, #Nick_Avatar, Cod_Juego, Aspecto, Nivel)
FK1 FK2

JUEGO (#Cod_Juego, Nombre, Descripcion)

PARTIDA (#Cod_Partida, Cod_Juego, Password_Partida, Nombre_Partida, Fecha_Creacion, Hora_Creacion, Estado)
FK3

PARTICIPACION (#Login, #Nick_Avatar, #Cod_Partida)
FK4 FK4 FK5

ENFRENTAMIENTO (#id_Enfrentamiento, Login_Avatar1, Nick_Avatar1, Login_Avatar2, Nick_Avatar2, Cod_Partida, Resultado)
FK6 FK6 FK7 FK7 FK6,FK7,FK8

PK = #Login, #Login#Nick, #Cod_Juego, #Cod_Partida, #Login#Nick_Avatar#Cod_Partida, #id_Enfrentamiento

FK1 = es clave foránea de USUARIO

FK2 = es clave foránea de JUEGO

FK3 = es clave foránea de JUEGO

FK4 = es clave foránea compuesta de AVATAR

FK5 = es clave foránea de PARTIDA

FK6 = es clave foránea compuesta de PARTICIPACION (Participante1)

FK7 = es clave foránea compuesta de PARTICIPACION (Participante2)

FK8 = es clave foránea de PARTIDA

4.3. Comentarios y Consideraciones del Modelo Relacional

- **Número de Tablas:** El modelo consta de un total de **seis tablas**, que corresponden a las entidades identificadas y a las tablas asociativas necesarias para resolver las relaciones N:M y

complejas del MER

(USUARIO, JUEGO, AVATAR, PARTIDA, PARTICIPACION, ENFRENTAMIENTO).

- **Claves Primarias (PK - #):** Cada tabla posee una clave primaria definida para identificar unívocamente cada fila. Se utilizan claves simples (USUARIO, JUEGO, PARTIDA, ENFRENTAMIENTO) y claves compuestas (AVATAR, PARTICIPACION) donde la combinación de atributos es necesaria para la identificación única, siguiendo las dependencias del MER.
- **Claves Foráneas (FK - subrayado y FKX):** Las relaciones entre las tablas se establecen mediante claves foráneas. En el esquema anterior, las columnas subrayadas indican una FK. Los marcadores FKX debajo de los atributos y la leyenda detallada clarifican cada una de estas relaciones.

4.4. Leyenda de Claves Foráneas (FK)

La siguiente leyenda describe cada una de las claves foráneas implementadas en el modelo, indicando la tabla y columna(s) de origen y destino:

- **FK1 = es clave foránea de USUARIO** (Relaciona AVATAR con USUARIO por Login)
- **FK2 = es clave foránea de JUEGO** (Relaciona AVATAR con JUEGO por Cod_Juego)
- **FK3 = es clave foránea de JUEGO** (Relaciona PARTIDA con JUEGO por Cod_Juego)
- **FK4 = es clave foránea compuesta de AVATAR** (Relaciona PARTICIPACION con AVATAR por Login, Nick_Avatar)
- **FK5 = es clave foránea de PARTIDA** (Relaciona PARTICIPACION con PARTIDA por Cod_Partida)
- **FK6 = es clave foránea compuesta de PARTICIPACION (Rol: Participante 1)** (Relaciona ENFRENTAMIENTO con PARTICIPACION para el primer contendiente)
- **FK7 = es clave foránea compuesta de PARTICIPACION (Rol: Participante 2)** (Relaciona ENFRENTAMIENTO con PARTICIPACION para el segundo contendiente)
- **FK8 = es clave foránea de PARTIDA** (Relaciona ENFRENTAMIENTO directamente con PARTIDA por Cod_Partida)

4.5. Entrega de Anexos

Tanto la representación gráfica que combina el MER final y este MR textual, como la leyenda detallada, se encuentran en el **ANEXO 1**. El archivo fuente (MER-MR-Final-Sitio-Juegos-Online.drawio) se adjunta según lo solicitado.

5 Creación de la base de datos

5.1. Proceso de Implementación Física

Una vez finalizado y validado el Modelo Relacional (detallado en la Sección 4 y el Anexo 1), procedimos a la implementación física de la base de datos sitiojuegosonline en el Sistema Gestor de Bases de Datos (SGBD) **MariaDB**. Para interactuar con el SGBD y ejecutar los comandos SQL necesarios, utilizamos la herramienta de administración gráfica **HeidiSQL**.

El proceso consistió en la ejecución de un script SQL cuidadosamente elaborado que contenía las sentencias necesarias para:

1. **Crear la Base de Datos:** Se utilizó el comando CREATE DATABASE sitiojuegosonline_brandoniza_rhommelcardona_final para crear el esquema inicial con el cotejamiento adecuado para soportar un amplio rango de caracteres.

```
CREATE DATABASE sitiojuegosonline_brandoniza_rhommelcardona_final
```

2. **Seleccionar la Base de Datos:** Se incluyó la sentencia USE sitiojuegosonline_brandoniza_rhommelcardona_final; para asegurar que las siguientes operaciones se realizaran en la base de datos correcta.

```
USE sitiojuegosonline_brandoniza_rhommelcardona_final
```

3. **Crear las Tablas:** Se ejecutaron las sentencias CREATE TABLE para cada una de las seis tablas definidas en nuestro Modelo Relacional: USUARIO, JUEGO, AVATAR, PARTIDA, PARTICIPACION y ENFRENTAMIENTO.

5.2. Detalles de la Creación de Tablas

Cada sentencia CREATE TABLE fue diseñada para reflejar fielmente el Modelo Relacional, especificando:

- **Nombres de Columnas:** Utilizando los nombres definidos en el MER/MR final.
- **Tipos de Datos:** Seleccionando los tipos más apropiados (VARCHAR, INT, TEXT, DATE, TIME) con sus longitudes correspondientes.
- **Restricciones de Nulabilidad:** Definiendo qué columnas eran obligatorias (NOT NULL) y cuáles opcionales (NULL).
- **Valores Predeterminados:** Estableciendo valores DEFAULT donde era pertinente (ej: Nivel = 1 en AVATAR, Estado = 'en curso' en PARTIDA).
- **Autoincremento:** Utilizando AUTO_INCREMENT para las claves primarias numéricas (Cod_Juego, Cod_Partida, id_Enfrentamiento) para facilitar la generación automática de IDs únicos.
- **Claves Primarias (PK):** Definiendo la PRIMARY KEY para cada tabla (simples y compuestas).
- **Restricciones Únicas (UQ):** Creando UNIQUE INDEX para las columnas Email y Nick en USUARIO, y para la combinación de participantes/partida en ENFRENTAMIENTO (uq_Enfrentamiento).
- **Claves Foráneas (FK):** Definiendo todas las FOREIGN KEY constraints necesarias para establecer las relaciones entre tablas, incluyendo las acciones referenciales ON UPDATE CASCADE y ON DELETE (usando CASCADE o RESTRICT según la lógica de negocio definida para cada relación).

Tabla Usuario:

#	Nombre	Tipo de datos	Longitud/Con...	Sin signo	Permitir ...	Rellen...	Predeterminado	Comentario	Collation
1	Login	VARCHAR	50	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predet...		utf8mb4_general_ci
2	Password	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...	Almacenar siempre ...	utf8mb4_general_ci
3	Nombre	VARCHAR	100	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_general_ci
4	Email	VARCHAR	100	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...		utf8mb4_general_ci
5	Nick	VARCHAR	50	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...		utf8mb4_general_ci

```

1 CREATE TABLE USUARIO (
2     Login VARCHAR(50) NOT NULL,
3     Password VARCHAR(255) NOT NULL COMMENT 'Almacenar siempre Hasheada!',
4     Nombre VARCHAR(100) NULL DEFAULT NULL,
5     Email VARCHAR(100) NOT NULL,
6     Nick VARCHAR(50) NOT NULL,
7     PRIMARY KEY ( Login ),
8     UNIQUE INDEX uq_Email ( Email ),
9     UNIQUE INDEX uq_Nick ( Nick )
10 )

```

Tabla Juego:

#	Nombre	Tipo de datos	Longitud/Con...	Sin signo	Permitir ...	Rellen...	Predeterminado	Comentario	Collation
1	Cod_Juego	INT	11	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	AUTO_INCREME...		
2	Nombre	VARCHAR	100	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...		utf8mb4_general_ci
3	Descripcion	TEXT		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_general_ci

```

1 CREATE TABLE JUEGO (
2     Cod_Juego INT(11) NOT NULL AUTO_INCREMENT,
3     Nombre VARCHAR(100) NOT NULL,
4     Descripcion TEXT NULL DEFAULT NULL,
5     PRIMARY KEY ( Cod_Juego )
6 )

```

Tabla Avatar:

<div> <div>Básico Opciones Índices (3) Llaves foráneas (2) Comprobar restricciones (0) Particiones</div> <div> <div> <div>+</div> Agregar <div>✕</div> Borrar <div>✕</div> Limpiar <div>▲</div> Subir <div>▼</div> Bajar </div> <div> <div>🔑</div> PRIMARY KEY <div>📍</div> idx_Nick_Avatar <div>📍</div> fk_Avatar_Juego </div> </div> </div>									
<div>Columnas: <div> <div>+</div> Agregar <div>✕</div> Borrar <div>▲</div> Subir <div>▼</div> Bajar </div> </div>									
#	Nombre	Tipo de datos	Longitud/Con...	Sin signo	Permitir ...	Rellen...	Predeterminado	Comentario	Collation
1	Login	VARCHAR	50	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predet...		utf8mb4_general_ci
2	Nick_Avatar	VARCHAR	50	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predet...		utf8mb4_general_ci
3	Aspecto	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_general_ci
4	Nivel	INT	11	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'1'		
5	Cod_Juego	INT	11	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...		

```

1 CREATE TABLE AVATAR (
2     Login VARCHAR(50) NOT NULL,
3     Nick_Avatar VARCHAR(50) NOT NULL,
4     Aspecto VARCHAR(255) NULL DEFAULT NULL,
5     Nivel INT(11) NOT NULL DEFAULT 1,
6     Cod_Juego INT(11) NOT NULL,
7     PRIMARY KEY ( Login , Nick_Avatar ),
8     INDEX idx_Nick_Avatar ( Nick_Avatar ),
9     CONSTRAINT fk_Avatar_Usuario
10    FOREIGN KEY ( Login )
11    REFERENCES USUARIO ( Login )
12    ON UPDATE CASCADE ON DELETE CASCADE,
13
14    CONSTRAINT fk_Avatar_Juego
15    FOREIGN KEY ( Cod_Juego )
16    REFERENCES JUEGO ( Cod_Juego )
17    ON UPDATE CASCADE ON DELETE RESTRICT
18 )

```

Tabla Partida:

<div> <div>Básico Opciones Índices (2) Llaves foráneas (1) Comprobar restricciones (0) Particiones</div> <div> <div>+</div> Agregar <div>✕</div> Borrar <div>✕</div> Limpiar <div>▲</div> Subir <div>▼</div> Bajar </div> <div> <div>🔑</div> PRIMARY KEY <div>📍</div> fk_Partida_Juego </div> </div>
--

```

1 CREATE TABLE PARTIDA (
2     Cod_Partida INT(11) NOT NULL AUTO_INCREMENT,
3     Cod_Juego INT(11) NOT NULL,
4     Nombre_Partida VARCHAR(100) NULL DEFAULT NULL,
5     Password_Partida VARCHAR(50) NULL DEFAULT NULL,
6     Fecha_Creacion DATE NOT NULL,
7     Hora_Creacion TIME NOT NULL,
8     Estado VARCHAR(20) NOT NULL DEFAULT 'en curso'
9     COMMENT 'Ej: en curso, finalizada, pausada',
10
11     PRIMARY KEY ( Cod_Partida ),
12     CONSTRAINT fk_Partida_Juego
13     FOREIGN KEY ( Cod_Juego )
14     REFERENCES JUEGO ( Cod_Juego )
15     ON UPDATE CASCADE ON DELETE RESTRICT
16 )

```

Tabla Participacion:

Básico

Opciones

Índices (2)

Llaves foráneas (2)

Comprobar restricciones (0)

Particiones

Código CREATE

Código ALTER

+

 Agregar

✖

 Borrar

✖

 Limpiar

▲

 Subir

▼

 Bajar

Nombre

🔑

 PRIMARY KEY

🔑

 fk_Participacion_Partida

Columnas:

+

 Agregar

✖

 Borrar

▲

 Subir

▼

 Bajar

#	Nombre	Tipo de datos	Longitud/Con...	Sin signo	Permitir ...	Rellen...	Predeterminado	Comentario	Collation
1	Login	VARCHAR	50	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predet...		utf8mb4_general_ci
2	Nick_Avatar	VARCHAR	50	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predet...		utf8mb4_general_ci
3	Cod_Partida	INT	11	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Sin valor predet...		

```

1 CREATE TABLE PARTICIPACION (
2     Login VARCHAR(50) NOT NULL,
3     Nick_Avatar VARCHAR(50) NOT NULL,
4     Cod_Partida INT(11) NOT NULL,
5     PRIMARY KEY ( Login , Nick_Avatar , Cod_Partida ),
6     CONSTRAINT fk_Participacion_Avatar
7     FOREIGN KEY ( Login , Nick_Avatar )
8     REFERENCES AVATAR ( Login , Nick_Avatar )
9     ON UPDATE CASCADE ON DELETE CASCADE,
10
11     CONSTRAINT fk_Participacion_Partida
12     FOREIGN KEY ( Cod_Partida )
13     REFERENCES PARTIDA ( Cod_Partida )
14     ON UPDATE CASCADE ON DELETE CASCADE
15 )

```

Tabla Enfrentamiento:

Básico

Opciones

Índices (5)

Llaves foráneas (3)

Comprobar restricciones (0)

Particiones

Código CREATE

Código ALTER

Agregar

Borrar

Limpiar

Subir

Bajar

Nombre

PRIMARY KEY

uq_Enfrentamiento

fk_Enfrentamiento_Participacion1

fk_Enfrentamiento_Participacion2

fk_Enfrentamiento_Partida

Columnas:

Agregar

Borrar

Subir

Bajar

#	Nombre	Tipo de datos	Longitud/Con...	Sin signo	Permitir ...	Rellen...	Predeterminado	Comentario	Collation
1	id_Enfrentamiento	INT	11	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	AUTO_INCREME...		
2	Login_Avatar1	VARCHAR	50	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...	Participante 1 - Login	utf8mb4_general_ci
3	Nick_Avatar1	VARCHAR	50	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...	Participante 1 - Avat...	utf8mb4_general_ci
4	Login_Avatar2	VARCHAR	50	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...	Participante 2 - Login	utf8mb4_general_ci
5	Nick_Avatar2	VARCHAR	50	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...	Participante 2 - Avat...	utf8mb4_general_ci
6	Cod_Partida	INT	11	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Sin valor predeter...	Partida donde ocurre	
7	Resultado	VARCHAR	20	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL	Ej: Victoria A1, Victo...	utf8mb4_general_ci

```

1 CREATE TABLE ENFRENTAMIENTO (
2   id_Enfrentamiento INT(11) NOT NULL AUTO_INCREMENT,
3   Login_Avatar1 VARCHAR(50) NOT NULL COMMENT 'Participante 1 - Login',
4   Nick_Avatar1 VARCHAR(50) NOT NULL COMMENT 'Participante 1 - Avatar Nick',
5   Login_Avatar2 VARCHAR(50) NOT NULL COMMENT 'Participante 2 - Login',
6   Nick_Avatar2 VARCHAR(50) NOT NULL COMMENT 'Participante 2 - Avatar Nick',
7   Cod_Partida INT(11) NOT NULL COMMENT 'Partida donde ocurre',
8   Resultado VARCHAR(20) NULL DEFAULT NULL COMMENT 'Ej: Victoria A1, Victoria A2, Empate',
9   PRIMARY KEY ( id_Enfrentamiento ),
10  UNIQUE INDEX uq_Enfrentamiento ( Login_Avatar1 , Nick_Avatar1 , Login_Avatar2 , Nick_Avatar2 , Cod_Partida )
11  COMMENT 'Evita duplicados exactos del mismo enfrentamiento',
12
13  CONSTRAINT fk_Enfrentamiento_Participacion1
14  FOREIGN KEY ( Login_Avatar1 , Nick_Avatar1 , Cod_Partida )
15  REFERENCES PARTICIPACION ( Login , Nick_Avatar , Cod_Partida )
16  ON UPDATE CASCADE ON DELETE CASCADE,
17
18  CONSTRAINT fk_Enfrentamiento_Participacion2
19  FOREIGN KEY ( Login_Avatar2 , Nick_Avatar2 , Cod_Partida )
20  REFERENCES PARTICIPACION ( Login , Nick_Avatar , Cod_Partida )
21  ON UPDATE CASCADE ON DELETE CASCADE,
22
23  CONSTRAINT fk_Enfrentamiento_Partida
24  FOREIGN KEY ( Cod_Partida )
25  REFERENCES PARTIDA ( Cod_Partida )
26  ON UPDATE CASCADE ON DELETE CASCADE
27 )

```

5.3. Entrega de anexos

La implementación física de este modelo mediante sentencias CREATE TABLE se detalla en el **ANEXO 2**.

6 Inserción de datos en la base de datos

6.1. Necesidad y Estrategia de Población de Datos

Tras la correcta implementación de la estructura física de la base de datos sitiojuegosonline_brandoniza_rhommelcardona_final (Sección 5 y Anexo 2), el siguiente paso fundamental fue poblarla con un volumen significativo de datos de prueba. Esta etapa es indispensable para:

- Validar que el esquema de tablas, tipos de datos y relaciones definidos aceptan información según lo diseñado.
- Verificar rigurosamente el funcionamiento de todas las restricciones implementadas (Claves Primarias, Claves Foráneas, Restricciones de Unicidad).
- Probar la lógica y efectividad de los Triggers creados para la tabla ENFRENTAMIENTO, que implementan reglas de negocio específicas.
- Disponer de un conjunto de datos suficientemente amplio y realista para ejecutar las consultas de la Sección 7 y obtener resultados representativos que permitan evaluar la funcionalidad y potencial rendimiento del sistema.

La inserción manual de cientos de registros interrelacionados a través de seis tablas habría sido extremadamente laboriosa, lenta y altamente susceptible a errores de inconsistencia.

6.2. Estrategia: Generación Asistida por Google AI Studio y Validación Iterativa

Para abordar eficientemente la necesidad de datos voluminosos y coherentes, decidimos emplear una **estrategia de generación asistida**. Específicamente, utilizamos las capacidades de generación de texto de la plataforma **Google AI Studio (impulsada por modelos como Gemini)** como herramienta principal para crear los conjuntos de datos iniciales para cada tabla.

El proceso, sin embargo, no fue una simple generación automática, sino un **ciclo iterativo y supervisado** centrado en garantizar la máxima coherencia con nuestro **Modelo Entidad-Relación (MER) final**:

1. **Definición Precisa del Formato:** Instruimos a Google AI Studio sobre el formato CSV exacto requerido para la importación vía LOAD DATA INFILE en HeidiSQL (separador ;, delimitador ", escape \, terminador \r\n, codificación UTF-8, encabezado). Se especificó el orden exacto de las columnas para cada tabla.
2. **Generación Secuencial por Dependencias:** Solicitamos la generación de ~100 registros por tabla, respetando estrictamente el orden impuesto por las claves foráneas: JUEGO y USUARIO primero, seguidos por AVATAR, luego PARTIDA, después PARTICIPACION, y finalmente ENFRENTAMIENTO.
3. **Validación y Refinamiento Continuo:** Cada archivo CSV generado por la IA fue sometido a validación antes de la importación definitiva:
 - *Revisión Visual:* Inspección inicial para detectar anomalías obvias.

- *Importación de Prueba:* Intento de carga en HeidiSQL con INSERT (puede arrojar errores) para que la base de datos detectara violaciones de PK, UQ o FK.
- *Corrección Asistida:* Cuando se detectaban errores (como las referencias incorrectas a Cod_Juego en PARTIDA o la falta de participantes comunes para ENFRENTAMIENTO), se informaba del error específico a Google AI Studio, proporcionando la información necesaria (ej: lista de IDs válidos) y solicitando la **regeneración de los datos corregidos**. Este bucle de generación-validación-corrección se repitió hasta obtener archivos CSV consistentes que se importaron sin errores de restricción.
- *Verificación Funcional:* Se realizaron pruebas adicionales (intentos de inserción de duplicados, referencias inválidas, violaciones de triggers) para confirmar activamente el correcto funcionamiento de todas las restricciones y reglas de negocio implementadas.

6.3. Resultado de la Inserción

Este método iterativo, combinando la capacidad de generación de volumen de la IA con nuestra supervisión y la validación proporcionada por el SGBD y HeidiSQL, nos permitió poblar exitosamente la base de datos sitiojuegosonline con un conjunto significativo y coherente de datos ficticios:

- USUARIO: 105 registros.
- JUEGO: 93 registros.
- AVATAR: 107 registros (incluyendo el bot).
- PARTIDA: 100 registros.
- PARTICIPACION: 102 registros.
- ENFRENTAMIENTO: 100 registros(excluyendo el VS BOT).

Disponer de esta base de datos poblada fue crucial para proceder con confianza a las siguientes fases del proyecto, especialmente la ejecución y análisis de consultas.

6.4. Entrega de anexos

Los script de las sentencias INSERT se detallan en el **ANEXO 3** tanto las sentencias individuales como un script con el total y el orden establecido para poblar la base de datos de forma correcta.

7 Consultas

7.1. Propósito y Metodología de Validación

Habiendo completado la implementación física de la base de datos sitiojuegosonline_brandoniza_rhommelcardona_final (detallada en la Sección 5 y el Anexo 2) y tras poblarla con un conjunto representativo de datos de prueba (descrito en la Sección 6 y el Anexo 3), el paso lógico siguiente fue validar su funcionalidad y capacidad para responder a requerimientos de información mediante la ejecución de consultas SQL.

Esta fase tuvo un doble objetivo fundamental:

- **Validación Funcional:** Confirmar que el esquema relacional diseñado, con sus tablas, relaciones (Claves Foráneas) y restricciones (Claves Primarias, Únicas), permite recuperar la información necesaria de manera coherente y precisa.
- **Demostración Técnica:** Ilustrar la aplicación práctica de diversas construcciones del lenguaje SQL aprendidas durante el módulo, incluyendo consultas simples con WHERE, funciones de agregación (COUNT), agrupación (GROUP BY), la combinación de datos de múltiples tablas mediante JOINS (INNER y LEFT/RIGHT) y el uso de subconsultas.

Para ello, seleccionamos un conjunto de 10 consultas significativas, inspiradas en los ejercicios y análisis realizados en la documentación previa del proyecto (referida como "2ª Parte Proyecto Integrador"). Estas consultas fueron cuidadosamente **adaptadas** a la estructura final de tablas (USUARIO, JUEGO, AVATAR, PARTIDA, PARTICIPACION, ENFRENTAMIENTO) y nombres de columnas de nuestra base de datos implementada en MariaDB.

7.2. Ejecución, Evidencias y Referencia a Anexos

Cada una de las 10 consultas SQL adaptadas fue ejecutada utilizando la interfaz gráfica de HeidiSQL conectada a nuestra base de datos poblada. Durante este proceso, se verificó la correcta sintaxis de cada consulta y se analizaron los resultados obtenidos para asegurar que fueran lógicos y consistentes con los datos de prueba insertados.

Los scripts SQL definitivos para cada una de estas 10 consultas, incluyendo comentarios que explican su propósito y la lógica implementada, se encuentran detallados en el **ANEXO 4** de este documento. Dicho anexo también incluye evidencias visuales (como capturas de pantalla) de la ejecución y/o de los resultados obtenidos para cada consulta, demostrando así su correcto funcionamiento sobre la base de datos sitiojuegosonline_brandoniza_rhommelcardona_final.

7.3 Consulta 1

```
1  /*Muestra los usuarios cuyo Nick empieza por "B"*/
2
3  SELECT Nombre, Email, Nick
4  FROM USUARIO
5  WHERE Nick LIKE 'B%';
6
7  /*• Explicación:
8  1. SELECT Nombre, Email, Nick: Especifica las columnas
9     que queremos mostrar de la tabla USUARIO.
10 2. FROM USUARIO: Indica que los datos se obtendrán de
11   la tabla USUARIO.
12 3. WHERE Nick LIKE 'B%': Filtra las filas para incluir
13   solo aquellas donde el valor de la columna Nick comience
14   con 'B'. El símbolo % actúa como comodín para cualquier
15   secuencia de caracteres que siga a la 'B'. La comparación
16   LIKE con el cotejamiento _ci buscará tanto 'B' como 'b'.
17  */
```

USUARIO (5r × 3c)				
#	Nombre	Email	Nick	
1	Beatriz Sanz	bsanz002@mail.test	Bea_S	
2	Benito Jurado	bjurado065@mail.test	BeniJur	
3	Berta Moya	bmoya028@mail.test	BerM	
4	Blanca Teruel	bteruel084@mail.test	BlanTer	
5	Bruno Cortez	bcortez047@mail.test	BruCor	

7.4 Consulta 2

```

1  /*Muestra los juegos que contengan "estrategia" en la descripcion*/
2
3  SELECT Cod_Juego, Nombre
4  FROM JUEGO
5  WHERE Descripcion LIKE '%estrategia%';
6
7  /*• Explicación:
8  1. SELECT Cod_Juego, Nombre: Selecciona las columnas deseadas de la
9     tabla JUEGO.
10 2. FROM JUEGO: Especifica la tabla JUEGO.
11 3. WHERE Descripcion LIKE '%estrategia%': Filtra las filas. El LIKE
12     con comodines % al principio y al final busca la subcadena
13     "estrategia" en cualquier parte dentro del texto de la columna
14     Descripcion.
15 */

```

JUEGO (3r × 2c)			
#	Cod_Juego		Nombre
1	1		Ajedrez Cosmico
2	5		Conquista Galactica RTS
3	10		Constructores de Imperios 4X

7.5 Consulta 3

```

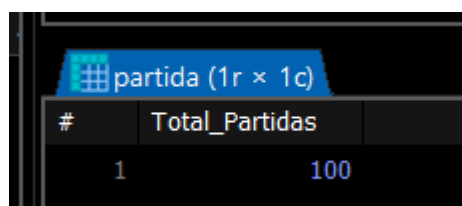
1  /*Muestra las partidas en las que ha participado el Usuario "user001"*/
2
3  SELECT DISTINCT p.Cod_Partida, p.Nombre_Partida, p.Estado
4  FROM USUARIO u
5  INNER JOIN AVATAR a ON u.Login = a.Login
6  INNER JOIN PARTICIPACION pa ON a.Login = pa.Login AND a.Nick_Avatar = pa.Nick_Avatar
7  INNER JOIN PARTIDA p ON pa.Cod_Partida = p.Cod_Partida
8  WHERE u.Login = 'user001';
9
10 /*• Explicación:
11 1. Se necesita unir (INNER JOIN) varias tablas para conectar al usuario
12    con sus partidas:
13    - USUARIO (alias u) con AVATAR (alias a) usando la clave común Login.
14    - AVATAR (a) con PARTICIPACION (alias pa) usando la clave compuesta
15      del avatar (Login, Nick_Avatar).
16    - PARTICIPACION (pa) con PARTIDA (alias p) usando Cod_Partida.
17 2. WHERE u.Login = 'user001': Filtra para considerar solo las participaciones
18    originadas por el usuario 'user001'.
19 3. SELECT DISTINCT p.Cod_Partida, p.Nombre_Partida, p.Estado: Muestra los
20    detalles de las partidas encontradas. DISTINCT asegura que cada partida
21    aparezca solo una vez, incluso si el usuario participó con diferentes avatares
22    en la misma partida (aunque nuestra estructura actual lo impide, es una buena
23    práctica).
24 */

```

USUARIO (1r × 3c)			
#	Cod_Partida	Nombre_Partida	Estado
1	1	Batalla Estelar Alfa	en curso

7.6 Consulta 4

```
1  /*Muestra el Total de las partidas que se han jugado*/
2
3  SELECT COUNT(*) AS Total_Partidas
4  FROM partida;
5
6  /*• Explicación:
7  1. SELECT COUNT(*): Utiliza la función de agregación COUNT(*) para
8     contar todas las filas de la tabla especificada.
9  2. AS Total_Partidas: Asigna un alias descriptivo ("Total_Partidas")
10     a la columna del resultado del conteo.
11  3. FROM PARTIDA: Indica que el conteo se realiza sobre la tabla PARTIDA.
12  */
```

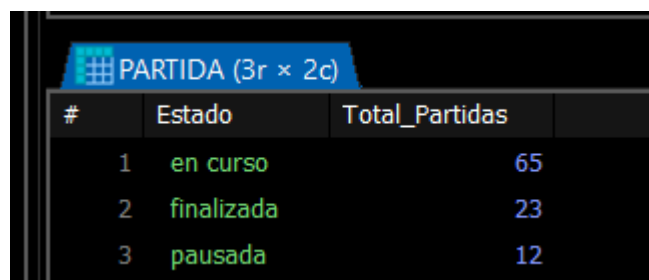


partida (1r x 1c)

#	Total_Partidas
1	100

7.7 Consulta 5

```
1  /*Muestra el Total de las partidas segun su estado*/
2
3  SELECT Estado, COUNT(*) AS Total_Partidas
4  FROM PARTIDA
5  GROUP BY Estado;
6
7  /*• Explicación:
8  1. SELECT Estado, COUNT(*): Selecciona el estado y cuenta las filas
9     para cada grupo de estado.
10  2. FROM PARTIDA: Especifica la tabla.
11  3. GROUP BY Estado: Agrupa las filas que tienen el mismo valor en la
12     columna Estado antes de aplicar la función COUNT(*). Esto permite
13     contar cuántas partidas hay por cada estado distinto.
14  */
```



PARTIDA (3r x 2c)

#	Estado	Total_Partidas
1	en curso	65
2	finalizada	23
3	pausada	12

7.8 Consulta 6

```

1  /*Muestra los Nicks y Juegos de los usuarios en partidas en curso*/
2
3  SELECT DISTINCT u.Nick, j.Nombre AS Nombre_Juego
4  FROM USUARIO u
5  INNER JOIN AVATAR a ON u.Login = a.Login
6  INNER JOIN PARTICIPACION pa ON a.Login = pa.Login AND a.Nick_Avatar = pa.Nick_Avatar
7  INNER JOIN PARTIDA p ON pa.Cod_Partida = p.Cod_Partida
8  INNER JOIN JUEGO j ON p.Cod_Juego = j.Cod_Juego
9  WHERE p.Estado = 'en curso';
10
11  /*• Explicación:
12  1. Se realizan múltiples INNER JOIN para conectar las tablas:
13     USUARIO -> AVATAR -> PARTICIPACION -> PARTIDA -> JUEGO, usando las claves foráneas
14     correctas en cada paso.
15  2. WHERE p.Estado = 'en curso': Filtra para considerar solo las participaciones en
16     partidas activas.
17  3. SELECT DISTINCT u.Nick, j.Nombre AS Nombre_Juego: Muestra el Nick del usuario y
18     el Nombre del juego correspondiente. DISTINCT evita mostrar la misma combinación
19     usuario-juego varias veces si un usuario participa en múltiples partidas en curso
20     del mismo juego.
21  */

```

USUARIO (75r × 2c)		
#	Nick	Nombre_Juego
1	Rhommel_Nick	Ajedrez Cosmico
2	AlexT	Ajedrez Cosmico
3	Bea_S	Ajedrez Cosmico
4	CharlieR	Ajedrez Cosmico
5	ZoeMol	Rey de la Lucha Libre
6	AaronP	Rey de la Lucha Libre
7	BerM	Rey de la Lucha Libre
8	CesarP	Rey de la Lucha Libre
9	DaniS	Rey de la Lucha Libre
10	JaviR	Chef Interplanetario

7.9 Consulta 7

```
1  /*Muestra los usuarios con Avatar de Nivel > 10*/
2
3  SELECT u.Nick, a.Nivel
4  FROM USUARIO u
5  INNER JOIN AVATAR a ON u.Login = a.Login
6  WHERE a.Nivel >10;
7
8  /*• Explicación:
9  1. INNER JOIN AVATAR a ON u.Login = a.Login: Conecta las tablas
10     USUARIO y AVATAR usando la clave correcta (Login). Se usa
11     INNER JOIN porque solo nos interesan los usuarios que sí
12     tienen avatares que cumplen la condición.
13  2. WHERE a.Nivel > 10: Filtra los resultados para mostrar solo
14     aquellos avatares cuyo nivel supera 10.
15  3. SELECT u.Nick, a.Nivel: Muestra el Nick del usuario propietario
16     y el nivel del avatar que cumple la condición.
17  */
```

USUARIO (27r x 2c)			
#	Nick	Nivel	
1	Rhommel_Nick	15	
2	Rhommel_Nick	20	
3	Fati_A	12	
4	LuciF	14	
5	MarcP	11	
6	UliCam	15	
7	YagSan	13	
8	CesarP	11	
9	HilPar	12	
10	MarBel	14	

7.10 Consulta 8

```

1  /*Muestra los Juegos y sus partidas finalizadas (limite 10)*/
2
3  SELECT j.Nombre AS Nombre_Juego, p.Nombre_Partida, p.Estado
4  FROM JUEGO j
5  LEFT JOIN PARTIDA p ON j.Cod_Juego = p.Cod_Juego AND p.Estado = 'finalizada'
6  LIMIT 10;
7
8  /* Explicación:
9  1. FROM JUEGO j LEFT JOIN PARTIDA p ...: Se usa un LEFT JOIN desde JUEGO hacia PARTIDA.
10 Esto asegura que se listen todos los juegos, incluso si no tienen ninguna partida
11 finalizada asociada.
12 2. ON j.Cod_Juego = p.Cod_Juego AND p.Estado = 'finalizada': La condición de unión
13 incluye no solo la clave (Cod_Juego), sino también el filtro por Estado. Esto hace
14 que solo se unan las partidas finalizadas. Si un juego no tiene partidas finalizadas,
15 las columnas de PARTIDA (p.Nombre_Partida, p.Estado) aparecerán como NULL para ese juego.
16 3. LIMIT 10: Restringe la salida a las primeras 10 filas del resultado general.
17 */

```

JUEGO (10r x 3c)				
#	Nombre_Juego	Nombre_Partida	Estado	
1	Academia de Magia Elemental	(NULL)	(NULL)	
2	Academia de Pilotos X	MagiaElemental_01	finalizada	
3	Agente Secreto 00X	(NULL)	(NULL)	
4	Ajedrez Cosmico	Duelo de Titanes	finalizada	
5	Ajedrez Cosmico	SesionAlfa_01	finalizada	
6	Ajedrez Cosmico	DefensaTotal_01	finalizada	
7	Ajedrez Cosmico	ViajeInaugural_01	finalizada	
8	Alquimista Prohibido	(NULL)	(NULL)	
9	Arena de Gladiadores Cyber	(NULL)	(NULL)	
10	Arquitecto de Rascacielos	ArenaSangrienta_01	finalizada	

7.11 Consulta 9

```
1  /*Muestra los Usuarios con Avatares de Nivel >10 (Subconsulta)*/
2
3  SELECT Nick, Nombre
4  FROM USUARIO
5  WHERE Login IN ( -- Se filtra por Login, que es la FK en AVATAR
6      SELECT Login
7      FROM AVATAR
8      WHERE Nivel > 10
9  );
10
11 /*• Explicación:
12 1. Subconsulta: (SELECT Login FROM AVATAR WHERE Nivel > 10) se ejecuta
13 primero. Obtiene una lista de todos los Login de usuarios que tienen
14 al menos un avatar con nivel mayor a 10.
15 2. Consulta Principal: SELECT Nick, Nombre FROM USUARIO WHERE Login IN (...)
16 selecciona el Nick y Nombre de la tabla USUARIO, pero solo para aquellas
17 filas cuyo Login esté presente en la lista de Login devuelta por la subconsulta.
18 */
```

USUARIO (26r x 2c)		
#	Nick	Nombre
1	Rhommel_Nick	Rhommel
2	Fati_A	Fatima Alonso
3	LuciF	Lucia Fuentes
4	MarcP	Marcos Pena
5	UliCam	Ulises Campos
6	YagSan	Yago Santos
7	CesarP	Cesar Parra
8	HilPar	Hilda Pardo
9	MarBel	Marta Beltran
10	RenaM	Renata Macias

7.12 Consulta 10

```

1  /*Muestra los Juegos con partidas en Agosto 2024*/
2
3  SELECT j.Nombre AS Nombre_Juego, (
4      SELECT COUNT(*)
5      FROM PARTIDA p
6      WHERE p.Cod_Juego = j.Cod_Juego
7            AND p.Fecha_Creacion BETWEEN '2024-03-01' AND '2024-03-31' -- Usamos Fecha_Creacion
8  ) AS Partidas_Jugadas_Marzo24
9  FROM JUEGO j
10 HAVING Partidas_Jugadas_Marzo24 > 0;
11
12 /*• Explicación:
13 1. Subconsulta Correlacionada: (SELECT COUNT(*) ... WHERE p.Cod_juego = j.Cod_juego ...)
14 se ejecuta para cada fila de la tabla JUEGO (alias j). Cuenta cuántas partidas (p)
15 existen para ese juego específico (p.Cod_juego = j.Cod_juego) cuya Fecha_Creacion
16 caiga dentro del rango especificado (BETWEEN ...). El resultado de esta cuenta se
17 muestra como Partidas_Jugadas_Marzo24.
18 2. FROM JUEGO j: La consulta principal itera sobre todos los juegos.
19 3. HAVING Partidas_Jugadas_Marzo24 > 0: Después de calcular el conteo para cada juego,
20 HAVING filtra los resultados para mostrar solo aquellos juegos donde el conteo
21 (Partidas_Jugadas_Marzo24) sea mayor que cero.
22 */

```

PARTIDA (58r × 2c)		
#	Nombre_Juego	Partidas_Jugadas_Marzo24
1	Academia de Magia Elemental	3
2	Academia de Pilotos X	3
3	Ajedrez Cosmico	7
4	Arena de Gladiadores Cyber	1
5	Arquitecto de Rascacielos	2
6	Ascenso al Monte Olimpo	2
7	Aventura Pirata: Islas del Tesoro	1
8	Barista Express: Cafe de Autor	1
9	Biologo de Campo: Safari Africano	2
10	Campeon de Artes Marciales Mixtas	1

7.13. Reflexión Práctica: JOINS vs. Subconsultas en sitiojuegosonline (Consultas 7 y 9)

Como parte de la validación funcional, retomamos el análisis comparativo entre JOINS y subconsultas, evaluando ambos enfoques sobre nuestra base de datos poblada. Utilizamos como caso de estudio las consultas adaptadas nº 7 y nº 9, ambas diseñadas para "obtener usuarios con avatares de nivel superior a 10", pero implementadas con técnicas diferentes.

Consulta 7 (JOIN)

```
1 /*Muestra los usuarios con Avatar de Nivel > 10*/
2
3 SELECT u.Nick, a.Nivel
4 FROM USUARIO u
5 INNER JOIN AVATAR a ON u.Login = a.Login
6 WHERE a.Nivel >10;
```

Consulta 9 (Subconsulta)

```
1 /*Muestra los Usuarios con Avatares de Nivel >10 (Subconsulta)*/
2
3 SELECT Nick, Nombre
4 FROM USUARIO
5 WHERE Login IN ( -- Se filtra por Login, que es la FK en AVATAR
6     SELECT Login
7     FROM AVATAR
8     WHERE Nivel > 10
9 );
```

La ejecución y análisis de estas dos consultas nos permitió observar en la práctica las siguientes ventajas y desventajas:

7.13.1. Uso del JOIN (Consulta 7)

- **Ventajas Observadas:**

- **Flexibilidad Superior en Selección:** La ventaja más evidente fue la capacidad de seleccionar columnas de *ambas* tablas (USUARIO y AVATAR) en el resultado final. Pudimos obtener tanto el Nick del usuario como el Nivel específico del avatar que cumplía la condición, algo imposible con la subconsulta IN simple. Esto es crucial si se necesita mostrar información combinada.
- **Potencial de Rendimiento:** Aunque con nuestro volumen actual de datos la diferencia no fue perceptible, sabemos que los motores SQL están altamente optimizados para procesar JOINS eficientemente, especialmente utilizando los índices existentes en las claves (Login en este caso). Se considera generalmente más escalable para grandes cantidades de datos.

- **Desventajas Observadas:**

- **Sintaxis (Curva de Aprendizaje):** La sintaxis del INNER JOIN, aunque estándar, puede resultar inicialmente menos intuitiva que la estructura lógica de la subconsulta para alguien que empieza con SQL.
- **(Potencial) Duplicación:** Si no hubiéramos querido ver el nivel específico y un usuario tuviera *varios* avatares con nivel > 10, el JOIN habría devuelto una fila por cada avatar, duplicando la información del usuario. En ese caso, habríamos necesitado añadir DISTINCT al SELECT u.Nick para obtener una lista única de usuarios, añadiendo un paso extra.

7.13.2. Uso de la Subconsulta (Consulta 9)

- **Ventajas Observadas:**

- **Claridad y Simplicidad para Filtros:** La consulta es muy legible y refleja directamente la pregunta: "Selecciona usuarios SI SU Login está EN la lista de Logins de avatares de nivel alto". Para este tipo de filtro de existencia, la lógica es muy directa.
- **Enfoque Directo:** Se centra en obtener datos de la tabla principal (USUARIO) aplicando una condición basada en otra tabla, sin la necesidad explícita de "combinar" tablas en la cláusula FROM.

- **Desventajas Observadas:**

- **Selección Limitada:** Como se mencionó, la principal limitación fue no poder incluir fácilmente columnas de la tabla AVATAR (como Nivel) en el resultado final. Se requeriría modificar la consulta de forma más compleja para lograrlo.
- **(Potencial) Rendimiento:** Si la subconsulta devolviera una lista muy grande de Logins, o si la tabla USUARIO fuera enorme, la operación IN podría, en algunos escenarios o versiones antiguas de SGBD, ser menos eficiente que un JOIN optimizado.

7.13.3. Conclusión de la Comparativa

Nuestra experiencia práctica con las consultas 7 y 9 en la base de datos sitiojuegosonline confirma las consideraciones teóricas:

- Las **subconsultas IN** son una excelente opción por su **simplicidad y claridad** cuando el objetivo principal es **filtrar** una tabla basándose en si un valor existe en otra, y **no necesitamos datos** de esa segunda tabla en el resultado.
- Los **JOINS** son la herramienta **más flexible y potente** cuando necesitamos **combinar información** de varias tablas en el resultado final. Generalmente, ofrecen un **mejor rendimiento y escalabilidad**, siendo la opción preferente para consultas que requieren datos interrelacionados.

En nuestro proyecto, si bien ambas consultas son válidas para identificar a los usuarios, el **JOIN se perfila como la solución más completa** si quisiéramos mostrar más detalles (como el nivel del avatar), mientras que la **subconsulta destaca por su concisión** para la pregunta específica formulada. La elección dependerá del detalle de información requerido en cada caso de uso.

8 Índices

Para la creación de índices seguimos la siguiente estructura

CREATE INDEX...

Después comprobamos su uso con

EXPLAIN...

Y vemos que en la columna “Key” aparece el nombre del índice creado anteriormente, lo que confirma que se está usando.

Por último, proporcionamos una consulta de ejemplo para usar el índice correspondiente.

```
--INDICE 1:
-- Índice para optimizar búsquedas por nombre de juego (consultas frecuentes en la tabla JUEGO)
CREATE INDEX idx_Juego_Nombre ON JUEGO(Nombre);

mysql> EXPLAIN SELECT *
-> FROM juego
-> WHERE Nombre = 'Conquista Galactica RTS';

+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | juego | NULL | ref | idx_Juego_Nombre | idx_Juego_Nombre | 402 |
+----+-----+-----+-----+-----+-----+-----+-----+

1 row in set, 1 warning (0.00 sec)

--consulta ejemplo indice 1:
SELECT * FROM juego WHERE Nombre = 'Ajedrez Cosmico';
```

```
--INDICE 2:
-- Índice compuesto para búsquedas que usan tanto el nick del avatar como el código de juego
CREATE INDEX idx_Avatar_Nick_CodJuego ON AVATAR(Nick_Avatar, Cod_Juego);

mysql> EXPLAIN SELECT * FROM AVATAR WHERE Nick_Avatar = 'PuzzleMaster' AND Cod_Juego = 3;

+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | AVATAR | NULL | ref | idx_Nick_Avatar,fk_Avatar_Juego | idx_Avatar_Nick_CodJuego | idx_Nick_Avatar |
+----+-----+-----+-----+-----+-----+-----+-----+

1 row in set, 1 warning (0.00 sec)

--consulta ejemplo indice 2:
SELECT * FROM avatar WHERE Nick_Avatar = 'Strategist001';
```

El optimizador de MySQL eligió idx_Nick_Avatar porque probablemente determinó que filtrar por Nick_Avatar era más eficiente, es decir, tenía un costo menor de consulta que usar el índice compuesto idx_Avatar_Nick_CodJuego.

```
--INDICE 3:
--Para búsquedas por nombre (si es un campo frecuentemente consultado)
CREATE INDEX idx_usuario_nombre ON usuario(Nombre);

mysql> EXPLAIN SELECT * FROM usuario WHERE Nombre LIKE 'Ana%';

+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | usuario | NULL | range | idx_usuario_nombre | idx_usuario_nombre | |
+----+-----+-----+-----+-----+-----+-----+-----+

1 row in set, 1 warning (0.00 sec)

--consulta ejemplo indice 3:
SELECT Login, Nombre, Email, Nick
FROM usuario
WHERE Nombre LIKE 'A%'
ORDER BY Nombre;
```

```
--INDICE 4:
--Para búsquedas combinadas de email y nick
CREATE INDEX idx_usuario_email_nick ON usuario(Email, Nick);

mysql> EXPLAIN SELECT * FROM usuario WHERE Email = 'ana.m@email.com' AND Nick = 'AnaPro';
+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | usuario | NULL | const | uq_Email,uq_Nick,idx_usuario_email_nick | uq_Email |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

--consulta ejemplo indice 4:
SELECT Login, Nombre, Email, Nick
FROM usuario
WHERE Email LIKE '%@mail.test'
      AND Nick LIKE 'A%'
ORDER BY Email, Nick;
```

En este caso ocurre lo mismo que en el índice 2.

```
--INDICE 5:
-- Para consultas que filtren por nivel
CREATE INDEX idx_avatar_nivel ON avatar(Nivel);

mysql> EXPLAIN SELECT * FROM avatar WHERE Nivel > 10;
+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | avatar | NULL | range | idx_avatar_nivel | idx_avatar_nivel |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

--consulta ejemplo indice 5:
SELECT Login, Nick_Avatar, Aspecto, Nivel, Cod_Juego
FROM avatar
WHERE Nivel >= 10
ORDER BY Nivel DESC;
```

```
--INDICE 6:
-- Para consultas que busquen avatares por aspecto
CREATE INDEX idx_avatar_aspecto ON avatar(Aspecto);

mysql> EXPLAIN SELECT * FROM avatar WHERE Aspecto = 'Peon metalico';
+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | avatar | NULL | ref | idx_avatar_aspecto | idx_avatar_aspecto |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

--consulta ejemplo indice 6:
SELECT Login, Nick_Avatar, Aspecto, Nivel, Cod_Juego
FROM avatar
WHERE Aspecto = 'Peon metalico'
ORDER BY nivel desc;
```

```
--INDICE 7:
-- Para búsquedas por estado y fecha
CREATE INDEX idx_partida_estado_fecha ON partida(Estado, Fecha_Creacion);

mysql> EXPLAIN SELECT * FROM partida WHERE Estado = 'finalizada' AND Fecha_Creacion >= '2024-01-01';
+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key |
+----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | partida | NULL | range | idx_partida_estado_fecha | idx_partida_estado_fecha |
+----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

--consulta ejemplo indice 7:
SELECT Cod_Partida, Cod_Juego, Nombre_Partida, Fecha_Creacion, Estado
FROM partida
WHERE Estado = 'en curso'
AND Fecha_Creacion >= '2024-03-01';
```

```
--INDICE 8:
-- Para búsquedas por nombre de partida
CREATE INDEX idx_partida_nombre ON partida(Nombre_Partida);

mysql> EXPLAIN SELECT * FROM partida WHERE Nombre_Partida LIKE 'Batalla%';
+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key |
+----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | partida | NULL | range | idx_partida_nombre | idx_partida_nombre |
+----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

--consulta ejemplo indice 8:
SELECT Cod_Partida, Nombre_Partida, Cod_Juego, Estado
FROM partida
WHERE Nombre_Partida LIKE 'Batalla%';
```

```
--INDICE 9:
-- Para consultas que busquen enfrentamientos por resultado
CREATE INDEX idx_enfrentamiento_resultado ON enfrentamiento(Resultado);

mysql> EXPLAIN SELECT * FROM enfrentamiento WHERE Resultado = 'Victoria A1';
+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key |
+----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | enfrentamiento | NULL | ref | idx_enfrentamiento_resultado | idx_enfrentamiento_resultado |
+----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

--consulta ejemplo indice 9:
SELECT *
FROM enfrentamiento
WHERE Resultado = 'Victoria A1';
```

```
--INDICE 10:
-- índice compuesto para búsquedas frecuentes
CREATE INDEX idx_participacion_login_partida ON participacion(Login, Cod_Partida);

mysql> EXPLAIN SELECT * FROM participacion WHERE Login = 'RhommelLogin' AND Cod_Partida = 1;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key |
+----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | participacion | NULL | ref | PRIMARY, fk_Participacion_Partida, idx_participacion_login_partida | fk_Participacion_Partida |
+----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

--consulta ejemplo indice 10:
SELECT p.Login, p.Nick_Avatar, p.Cod_Partida, a.Cod_Juego
FROM participacion p
JOIN avatar a ON p.Login = a.Login AND p.Nick_Avatar = a.Nick_Avatar
WHERE p.Login = 'RhommelLogin'
AND p.Cod_Partida = 1;
```

Incluimos los script SQL en el **ANEXO 5**

9 Vistas

9.1. Definición y Aplicación en el Proyecto

Como se introdujo en la sección teórica, las vistas (Views) son tablas virtuales definidas por consultas SQL almacenadas, que no guardan datos físicamente pero ofrecen una representación dinámica de los mismos. En el proyecto `sitiojuegosonline`, hemos implementado varias vistas con el objetivo principal de **simplificar el acceso a datos combinados o filtrados frecuentemente**, mejorando la legibilidad de las consultas y encapsulando lógica de negocio o de presentación.

Se crearon un total de seis vistas, cada una con un propósito específico. La definición SQL completa de cada vista, junto con una explicación detallada de su funcionamiento y ejemplos de uso, se encuentra documentada en el **ANEXO 6** de este documento. A continuación, se resumen las vistas implementadas:

9.2. Vistas Creadas

Vista 1 avatares con nivel alto (mayor a 10):

```

1  /*Muestra los avatares que superan el nivel 10*/
2
3  CREATE OR REPLACE VIEW V_Avatares_NivelAlto AS
4  SELECT
5      Login,          -- Login del usuario propietario
6      Nick_Avatar,    -- Nick específico del Avatar
7      Aspecto,
8      Nivel,
9      Cod_Juego       -- Código del juego al que pertenece
10 FROM AVATAR
11 WHERE Nivel > 10;   -- Condición de filtrado por nivel

```

```
1 SELECT * FROM v_avatares_nivelalto
```

v_avatares_nivelalto (27r x 5c)							
#	Login	Nick_Avatar	Aspecto	Nivel	Cod_Juego		
1	RhommelLogin	Avatar_Rhommel02	Luchador	15		1	
2	RhommelLogin	Avatar_Rhommel1	Luchador	20		1	
3	user006	Solver006	Esfera logica	12		3	
4	user012	Solver012	Mente cristalina	14		3	
5	user013	Strategist013	Reina de plasma	11		1	
6	user021	Solver021	(NULL)	15		3	
7	user025	Strategist025	(NULL)	13		1	
8	user029	Racer029	(NULL)	11		2	
9	user034	Strategist034	(NULL)	12		1	
10	user038	Racer038	(NULL)	14		2	

Vista 2 de usuarios con avatares creados:

```

1  /*Muestra los usuarios con avatares creados*/
2
3  CREATE OR REPLACE VIEW V_Usuarios_Con_Avatares AS
4  SELECT
5      u.Login,
6      u.Nick AS Usuario_Nick,      -- Alias para el Nick del Usuario
7      a.Nick_Avatar AS Avatar_Nick, -- Alias para el Nick del Avatar
8      a.Aspecto,
9      a.Nivel,
10     a.Cod_Juego                  -- Código del juego del avatar
11 FROM USUARIO u
12 INNER JOIN AVATAR a ON u.Login = a.Login; -- Unión correcta por Login

```

```
1 SELECT * FROM v_usuarios_con_avatares
```

#	Login	Usuario_Nick	Avatar_Nick	Aspecto	Nivel	Cod_Juego
1	amartinez	AnaPro	PuzzleMaster	(NULL)	10	3
2	amartinez	AnaPro	ReinaCosmica	Figura majestuosa con cetro estelar	7	1
3	jgarcia	LoboJuan	CorredorEstelar	Nave ágil y roja	3	2
4	jgarcia	LoboJuan	LoboAlfa	Lobo gris con armadura ligera	5	1
5	RhommelLogin	Rhommel_Nick	Avatar_Rhommel02	Luchador	15	1
6	RhommelLogin	Rhommel_Nick	Avatar_Rhommel1	Luchador	20	1
7	user001	AlexT	Strategist001	Peon metalico	5	1
8	user002	Bea_S	Racer002	Nave agil azul	3	2
9	user003	CharlieR	Solver003	Cubo de datos	8	3
10	user004	DiNav	Strategist004	Torre laser	10	1

Vista 3 detallada de los datos de la partida:

```

1  /*Vista detallada de los datos de la partida*/
2
3  CREATE OR REPLACE VIEW V_Partidas_Detalladas AS
4  SELECT
5      p.Cod_Partida,
6      p.Nombre_Partida,
7      p.Password_Partida,
8      p.Fecha_Creacion,
9      p.Hora_Creacion,
10     p.Estado,
11     p.Cod_Juego,          -- FK al juego
12     j.Nombre AS Nombre_Juego -- Nombre del juego asociado
13 FROM PARTIDA p
14 INNER JOIN JUEGO j ON p.Cod_Juego = j.Cod_Juego;
15
16 /*Explicación:
17
18 FROM PARTIDA p INNER JOIN JUEGO j ON p.Cod_Juego = j.Cod_Juego: Une las tablas
19 PARTIDA y JUEGO usando la clave foránea Cod_Juego.
20
21 SELECT p.*, j.Nombre AS Nombre_Juego: Selecciona todas las columnas de la tabla
22 PARTIDA (usando p.* como atajo, aunque es mejor listar explícitamente como arriba)
23 y añade la columna Nombre de la tabla JUEGO, dándole el alias Nombre_Juego para claridad.*

```

```
1 SELECT * FROM v_partidas_detalladas
```

#	Login	Usuario_Nick	Avatar_Nick	Aspecto	Nivel	Cod_Juego
1	amartinez	AnaPro	PuzzleMaster	(NULL)	10	3
2	amartinez	AnaPro	ReinaCosmica	Figura majestuosa con cetro estelar	7	1
3	jgarcia	LoboJuan	CorredorEstelar	Nave ágil y roja	3	2
4	jgarcia	LoboJuan	LoboAlfa	Lobo gris con armadura ligera	5	1
5	RhommelLogin	Rhommel_Nick	Avatar_Rhommel02	Luchador	15	1
6	RhommelLogin	Rhommel_Nick	Avatar_Rhommel1	Luchador	20	1
7	user001	AlexT	Strategist001	Peon metalico	5	1
8	user002	Bea_S	Racer002	Nave agil azul	3	2
9	user003	CharlieR	Solver003	Cubo de datos	8	3
10	user004	DiNav	Strategist004	Torre laser	10	1
11	user005	FctoM	Racer005	(NULL)	2	2

Vista 4 de los participantes de cada partida:

```

1  /*Mostrar una lista clara de qué usuarios (con su Nick) y qué avatares específicos
2  (con su Nick_Avatar) están participando en cada partida (Cod_Partida).*/
3
4  CREATE OR REPLACE VIEW V_Participantes_Por_Partida AS
5  SELECT
6      pa.Cod_Partida,
7      u.Login,
8      u.Nick AS Usuario_Nick,
9      a.Nick_Avatar
10 FROM PARTICIPACION pa
11 INNER JOIN AVATAR a ON pa.Login = a.Login AND pa.Nick_Avatar = a.Nick_Avatar
12 INNER JOIN USUARIO u ON a.Login = u.Login;
13
14 /*Explicación:
15
16 FROM PARTICIPACION pa ...: Empieza desde la tabla de participaciones.
17
18 INNER JOIN AVATAR a ON ...: Une con AVATAR usando la clave compuesta (Login, Nick_Avatar) para
19 obtener los detalles del avatar.
20
21 INNER JOIN USUARIO u ON a.Login = u.Login: Une con USUARIO usando el Login (obtenido de AVATAR o
22 PARTICIPACION) para obtener el Nick del usuario.
23
24 SELECT pa.Cod_Partida, u.Login, u.Nick..., a.Nick_Avatar: Selecciona la información relevante
25 de las tres tablas unidas.*/

```

```
1 SELECT * FROM v_participantes_por_partida
```

v_participantes_por_partida (102r x 4c)					
#	Cod_Partida	Login	Usuario_Nick	Nick_Avatar	
1	1	RhommelLogin	Rhommel_Nick	Avatar_Rhommel02	
2	1	RhommelLogin	Rhommel_Nick	Avatar_Rhommel1	
3	1	user001	AlexT	Strategist001	
4	1	user002	Bea_S	Racer002	
5	1	user003	CharlieR	Solver003	
6	4	user075	MarGam	Solver075	
7	4	user076	NereHid	Strategist076	
8	4	user077	OliExp	Racer077	
9	4	user078	PiliJur	Solver078	
10	4	user079	RubLuj	Strategist079	

Vista 5 historial de enfrentamientos:

```

1  /*Vista de historial legible de Los enfrentamientos,*/
2
3  CREATE OR REPLACE VIEW V_Historial_Enfrentamientos AS
4  SELECT
5      e.id_Enfrentamiento,
6      e.Cod_Partida,
7      p.Nombre_Partida, -- Obtenido directamente de PARTIDA
8      j.Nombre AS Nombre_Juego, -- Obtenido directamente de JUEGO
9      u1.Nick AS Nick_Usuario1,
10     e.Nick_Avatar1,
11     u2.Nick AS Nick_Usuario2,
12     e.Nick_Avatar2,
13     e.Resultado
14 FROM ENFRENTAMIENTO e
15 -- Obtener info (Nick Usuario) del Participante 1
16 INNER JOIN AVATAR a1 ON e.Login_Avatar1 = a1.Login AND e.Nick_Avatar1 = a1.Nick_Avatar
17 INNER JOIN USUARIO u1 ON a1.Login = u1.Login
18 -- Obtener info (Nick Usuario) del Participante 2
19 INNER JOIN AVATAR a2 ON e.Login_Avatar2 = a2.Login AND e.Nick_Avatar2 = a2.Nick_Avatar
20 INNER JOIN USUARIO u2 ON a2.Login = u2.Login
21 -- Obtener info de la Partida y Juego
22 INNER JOIN PARTIDA p ON e.Cod_Partida = p.Cod_Partida
23 INNER JOIN JUEGO j ON p.Cod_Juego = j.Cod_Juego;
24
25 /*Explicación:
26
27 Esta vista es más compleja porque necesita unir ENFRENTAMIENTO con PARTICIPACION, AVATAR y USUARIO dos veces (una para obtener
28 los datos del Avatar1/Usuario1 y otra para el Avatar2/Usuario2).
29
30 Los JOINS se hacen usando las claves correspondientes (Login_AvatarX, Nick_AvatarX, Cod_Partida).*/

```

```
1 SELECT * FROM v_historial_enfrentamientos
```

#	id_Enfrentamiento	Cod_Partida	Nombre_Partida	Nombre_Juego	Nick_Usuario1	Nick_Avatar1	Nick_Usuario2	Nick_Avatar2	Resultado
1	1	11	TurboBoost_02	Chef Interplanetario	AlexT	Strategist001	Bea_S	Racer002	Victoria A1
2	2	74	TorneoGolf_02	Guerras de Mechas Prime	AlexT	Strategist001	CharlieR	Solver003	Empate
3	3	5	CarreraRapida_01	Rey de la Lucha Libre	Bea_S	Racer002	DiNav	Strategist004	Victoria A2
4	4	74	TorneoGolf_02	Guerras de Mechas Prime	CharlieR	Solver003	EsteM	Racer005	Victoria A1
5	5	35	CarreraFinal_03	Guerras de Mechas Prime	Fati_A	Solver006	GaboH	Strategist007	Victoria A2
6	6	38	SimuladorCombate_01	Maestro de Cartas Astrales	Fati_A	Solver006	ElenaV	Racer008	Empate
7	7	26	CazaMamut_01	Supervivencia Crio-Genesis	GaboH	Strategist007	NachoR	Solver009	Victoria A1
8	8	5	CarreraRapida_01	Rey de la Lucha Libre	ElenaV	Racer008	JimeP	Strategist010	Victoria A2
9	9	54	ViajeInaugural_01	Ajedrez Cosmico	KevB	Racer011	LuciF	Solver012	Victoria A1
10	10	54	ViajeInaugural_01	Ajedrez Cosmico	KevB	Racer011	MarcP	Strategist013	Victoria A1

Vista 6 clasificación de los juegos mas populares:

```

1  /*Mostrar los juegos ordenados por el número de partidas que se han creado para ellos, de más popular a menos popular.*/
2
3  CREATE OR REPLACE VIEW V_Juegos_Populares AS
4  SELECT
5      j.Cod_Juego,
6      j.Nombre,
7      COUNT(p.Cod_Partida) AS Numero_Partidas
8  FROM JUEGO j
9  LEFT JOIN PARTIDA p ON j.Cod_Juego = p.Cod_Juego
10 GROUP BY j.Cod_Juego, j.Nombre -- Agrupar por juego
11 ORDER BY Numero_Partidas DESC; -- Ordenar por número de partidas descendente
12
13 /*Explicación:
14
15 FROM JUEGO j LEFT JOIN PARTIDA p ...: Se usa LEFT JOIN para incluir todos los juegos, incluso aquellos que aún no tengan
16 ninguna partida creada (en cuyo caso Numero_Partidas será 0).
17
18 COUNT(p.Cod_Partida): Cuenta el número de partidas asociadas a cada juego. COUNT ignora los NULLs, por lo que para
19 juegos sin partidas dará 0.
20
21 GROUP BY j.Cod_Juego, j.Nombre: Agrupa las filas por juego antes de contar las partidas.
22
23 ORDER BY Numero_Partidas DESC: Ordena el resultado final para mostrar los juegos con más partidas primero.*/

```

```
3  SELECT * FROM v_juegos_populares
```

v_juegos_populares (93r x 3c)			
#	Cod_Juego	Nombre	Numero_Partidas
1	1	Ajedrez Cosmico	10
2	2	Carreras Nebulares	4
3	48	Campeonato Mundial de Surf	4
4	78	Arquitecto de Rascacielos	3
5	39	Academia de Magia Elemental	3
6	3	Puzzle de Agujeros Negros	3
7	32	Maestro Zen: Jardines Rocosos	3
8	8	Academia de Pilotos X	3
9	31	Orquesta Sinfonica VR	2
10	19	Supervivencia Crio-Genesis	2

Adjuntamos el **ANEXO 6** con los scripts SQL para la creación de estas vistas

10 Triggers

TRIGGER 1:

Este trigger se ejecuta antes de insertar un nuevo registro en la tabla ENFRENTAMIENTO. Su propósito es imponer dos reglas de integridad:

- Evitar que un avatar se enfrente a sí mismo y garantizar un orden canónico entre los participantes.

```

1 DELIMITER $$
2
3 CREATE TRIGGER `trg_enfrentamiento_before_insert`
4 BEFORE INSERT ON `ENFRENTAMIENTO`
5 FOR EACH ROW
6 BEGIN
7     -- 1. Verificar que un avatar no se enfrente a sí mismo
8     IF (NEW.Login_Avatar1 = NEW.Login_Avatar2 AND NEW.Nick_Avatar1 = NEW.Nick_Avatar2) THEN
9         -- Si son iguales, lanzar un error y cancelar la inserción
10        SIGNAL SQLSTATE '45000' -- Estado de error genérico definido por el usuario
11        SET MESSAGE_TEXT = 'Error de Integridad (trg_enfrentamiento_before_insert): Un avatar no puede enfrentarse a sí mismo.';
12    END IF;
13
14    -- 2. Verificar el orden canónico (A1 debe ser "menor o igual" que A2)
15    -- Compara primero por Login, luego por Nick Avatar
16    IF (NEW.Login_Avatar1 > NEW.Login_Avatar2 OR (NEW.Login_Avatar1 = NEW.Login_Avatar2 AND NEW.Nick_Avatar1 > NEW.Nick_Avatar2)) THEN
17        -- Si el orden es incorrecto, lanzar un error y cancelar la inserción
18        SIGNAL SQLSTATE '45000'
19        SET MESSAGE_TEXT = 'Error de Integridad (trg_enfrentamiento_before_insert):
20        El Participante 1 debe ser alfabéticamente menor o igual que el Participante 2.
21        Por favor, invierta los participantes antes de insertar.';
22    END IF;
23
24 END
25 $$

```

TRIGGER 2:

Se ejecuta antes de actualizar un registro en la tabla enfrentamiento, es decir, se ejecuta antes de que un registro existente en ENFRENTAMIENTO sea MODIFICADO.

Su propósito es garantizar que cualquier actualización de un registro cumpla con dos reglas de integridad:

- Evitar que un avatar se enfrente a sí mismo y garantizar un orden canónico entre los participantes.

```

1 DELIMITER $$
2
3 CREATE TRIGGER `trg_enfrentamiento_before_update`
4 BEFORE UPDATE ON `ENFRENTAMIENTO`
5 FOR EACH ROW
6 BEGIN
7     -- 1. Verificar que un avatar no se enfrente a sí mismo (con los nuevos valores)
8     IF (NEW.Login_Avatar1 = NEW.Login_Avatar2 AND NEW.Nick_Avatar1 = NEW.Nick_Avatar2) THEN
9         SIGNAL SQLSTATE '45000'
10        SET MESSAGE_TEXT = 'Error de Integridad (trg_enfrentamiento_before_update): Un avatar no puede enfrentarse a sí mismo.';
11    END IF;
12
13    -- 2. Verificar el orden canónico (con los nuevos valores)
14    IF (NEW.Login_Avatar1 > NEW.Login_Avatar2 OR (NEW.Login_Avatar1 = NEW.Login_Avatar2 AND NEW.Nick_Avatar1 > NEW.Nick_Avatar2)) THEN
15        SIGNAL SQLSTATE '45000'
16        SET MESSAGE_TEXT = 'Error de Integridad (trg_enfrentamiento_before_update):
17        El Participante 1 debe ser alfabéticamente menor o igual que el Participante 2.
18        Por favor, invierta los participantes antes de actualizar.';
19    END IF;
20
21 END
22 $$

```

TRIGGER 3:

Este trigger se activa antes de insertar un nuevo registro en la tabla avatar. Verificará que el nivel del avatar sea un valor positivo (mayor o igual a 1), ya que un nivel negativo o cero no tendría sentido en el contexto de un juego. Si el nivel es inválido, el trigger lanzará un error con mensaje y cancelará la inserción. Este trigger no afectará los registros existentes, pero sí protegerá futuras inserciones

```
1 DELIMITER $$
2
3 CREATE TRIGGER `trg_avatar_nivel_positivo`
4 BEFORE INSERT ON `avatar`
5 FOR EACH ROW
6 BEGIN
7     IF NEW.Nivel < 1 THEN
8         SIGNAL SQLSTATE '45000'
9         SET MESSAGE_TEXT = 'Error de Integridad (trg_avatar_nivel_positivo):
10             El Nivel del avatar debe ser mayor o igual a 1.';
11     END IF;
12 END
13 $$
```

TRIGGER 4:

Creamos un trigger que se active después de insertar un registro en la tabla partida y registre la acción en historico_partidas.

```
1 DELIMITER $$
2
3 CREATE TRIGGER `trg_historico_partidas_insert`
4 AFTER INSERT ON `partida`
5 FOR EACH ROW
6 BEGIN
7     INSERT INTO `historico_partidas` (
8         Cod_Partida,
9         Nombre_Partida,
10        Cod_Juego,
11        Fecha_Creacion,
12        Hora_Creacion,
13        Fecha_Registro,
14        Accion
15    )
16    VALUES (
17        NEW.Cod_Partida,
18        NEW.Nombre_Partida,
19        NEW.Cod_Juego,
20        NEW.Fecha_Creacion,
21        NEW.Hora_Creacion,
22        NOW(),
23        'CREACION'
24    );
25 END
26 $$
27
```


TRIGGER 5:

El trigger se activa antes de insertar un nuevo registro en la tabla participacion.

Verificará que la partida (Cod_Partida) a la que un avatar intenta unirse no tenga el estado 'finalizada' en la tabla partida.

Si la partida ya está finalizada, el trigger lanzará un error y cancelará la inserción.

```
1 DELIMITER $$
2
3 CREATE TRIGGER `trg_participacion_no_efectuada`
4 BEFORE INSERT ON `participacion`
5 FOR EACH ROW
6 BEGIN
7     DECLARE partida_estado VARCHAR(20);
8
9     -- Obtener el estado de la partida
10    SELECT Estado INTO partida_estado
11    FROM partida
12    WHERE Cod_Partida = NEW.Cod_Partida;
13
14    -- Verificar si la partida está finalizada
15    IF partida_estado = 'finalizada' THEN
16        SIGNAL SQLSTATE '45000'
17        SET MESSAGE_TEXT = 'Error de Integridad (trg_participacion_no_efectuada):
18                               No se puede unir a una partida finalizada.';
19    END IF;
20 END
21 $$
```

TRIGGER 6:

El trigger se activa antes de insertar un nuevo registro en la tabla enfrentamiento.

Verificará que los dos avatares involucrados en el enfrentamiento (Login_Avatar1 y Login_Avatar2) no sean del mismo usuario, ya que un usuario no debería enfrentarse a sí mismo en una partida.

```
1 DELIMITER $$
2
3 CREATE TRIGGER `trg_enfrentamiento_distintos_usuarios`
4 BEFORE INSERT ON `enfrentamiento`
5 FOR EACH ROW
6 BEGIN
7     IF NEW.Login_Avatar1 = NEW.Login_Avatar2 THEN
8         SIGNAL SQLSTATE '45000'
9         SET MESSAGE_TEXT = 'Error de Integridad (trg_enfrentamiento_distintos_usuarios):
10                               Un usuario no puede enfrentarse a sí mismo.';
11     END IF;
12 END
13 $$
```

11 Procedimientos y funciones

En este punto decidimos crear una serie de procedimientos y funciones acordes con la esencia del proyecto, un sitio de juegos online para ello creamos procedimientos para el registro, creación de avatares, y los diversos escenarios de partida, además creamos funciones para verificar el Nick del usuario en la BBDD, información relativa a los avatares y partidas jugadas por el usuario.

En el **ANEXO 8** se incluye los script SQL de los procedimientos y las funciones.

Procedimiento 1: **Registrar Nuevo Usuario**

Valida la unicidad del login y nick antes de insertar.

```
1 DELIMITER $$
2 CREATE PROCEDURE pr_RegistrarUsuario(
3     IN p_Login VARCHAR(50),
4     IN p_Nombre VARCHAR(100),
5     IN p_Correo VARCHAR(100),
6     IN p_Nick VARCHAR(50),
7     IN p_Password VARCHAR(255)
8 )
9 BEGIN
10     -- Verificar si el login o nick ya existen
11     IF EXISTS (SELECT 1 FROM USUARIO WHERE Login = p_Login OR Nick = p_Nick) THEN
12         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Login o Nick ya registrado';
13     ELSE
14         INSERT INTO USUARIO (Login, Nombre, Correo, Nick, Password)
15         VALUES (p_Login, p_Nombre, p_Correo, p_Nick, p_Password);
16     END IF;
17 END
18 $$
```

Procedimiento 2: **Crear Avatar para Usuario**

Asocia un avatar a un usuario y juego específico.

```
1 DELIMITER $$
2 CREATE PROCEDURE pr_CrearAvatar(
3     IN p_Login VARCHAR(50),
4     IN p_Nick_Avatar VARCHAR(50),
5     IN p_Aspecto TEXT,
6     IN p_Cod_Juego INT
7 )
8 BEGIN
9     IF NOT EXISTS (SELECT 1 FROM USUARIO WHERE Login = p_Login) THEN
10         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Usuario no existe';
11     ELSE
12         INSERT INTO AVATAR (Login, Nick_Avatar, Aspecto, Nivel, Cod_Juego)
13         VALUES (p_Login, p_Nick_Avatar, p_Aspecto, 1, p_Cod_Juego); -- Nivel inicial: 1
14     END IF;
15 END
16 $$
```

Procedimiento 3: Iniciar Nueva Partida

Crea una partida sin registrar al creador (cumple requisito de anonimato).

```
1 DELIMITER $$
2 CREATE PROCEDURE pr_IniciarPartida(
3     IN p_Nombre_Partida VARCHAR(100),
4     IN p_Password_Partida VARCHAR(50),
5     IN p_Cod_Juego INT
6 )
7 BEGIN
8     INSERT INTO PARTIDA (Nombre_Partida, Password_Partida, Fecha_Creacion, Hora_Creacion, Estado, Cod_Juego)
9     VALUES (p_Nombre_Partida, p_Password_Partida, CURDATE(), CURTIME(), 'en curso', p_Cod_Juego);
10 END
11 $$
```

Procedimiento 4: Unirse a Partida

Valida la contraseña y el estado de la partida.

```
1 DELIMITER $$
2 CREATE PROCEDURE pr_UnirseAPartida(
3     IN p_Login VARCHAR(50),
4     IN p_Nick_Avatar VARCHAR(50),
5     IN p_Cod_Partida INT,
6     IN p_Password_Partida VARCHAR(50)
7 )
8 BEGIN
9     DECLARE v_Estado VARCHAR(20);
10    DECLARE v_Password_Correcto VARCHAR(50);
11
12    SELECT Estado, Password_Partida INTO v_Estado, v_Password_Correcto
13    FROM PARTIDA WHERE Cod_Partida = p_Cod_Partida;
14
15    IF v_Estado = 'finalizada' THEN
16        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Partida finalizada';
17    ELSEIF v_Password_Correcto IS NOT NULL AND v_Password_Correcto != p_Password_Partida THEN
18        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Contraseña incorrecta';
19    ELSE
20        INSERT INTO PARTICIPACION (Login, Nick_Avatar, Cod_Partida)
21        VALUES (p_Login, p_Nick_Avatar, p_Cod_Partida);
22    END IF;
23 END
24 $$
```

Procedimiento 5: Finalizar Partida

Cambia el estado de una partida a "finalizada".

```
1 DELIMITER $$
2 CREATE PROCEDURE pr_FinalizarPartida(IN p_Cod_Partida INT)
3 BEGIN
4     UPDATE PARTIDA
5     SET Estado = 'finalizada'
6     WHERE Cod_Partida = p_Cod_Partida;
7 END
8 $$
```

Procedimiento 6: Generar Reporte de Usuario

Lista avatares y partidas de un usuario.

```
1 DELIMITER $$
2 CREATE PROCEDURE pr_ReporteUsuario(IN p_Login VARCHAR(50))
3 BEGIN
4     SELECT
5         u.Nick AS Usuario,
6         a.Nick_Avatar AS Avatar,
7         a.Nivel,
8         p.Nombre_Partida,
9         p.Estado
10    FROM USUARIO u
11   LEFT JOIN AVATAR a ON u.Login = a.Login
12   LEFT JOIN PARTICIPACION pa ON a.Login = pa.Login AND a.Nick_Avatar = pa.Nick_Avatar
13   LEFT JOIN PARTIDA p ON pa.Cod_Partida = p.Cod_Partida
14  WHERE u.Login = p_Login;
15 END
16 $$
```

Función 1: Verificar Disponibilidad de Nick

```
1 DELIMITER $$
2 CREATE FUNCTION fn_NickDisponible(p_Nick VARCHAR(50))
3 RETURNS BOOLEAN
4 DETERMINISTIC
5 BEGIN
6     RETURN (SELECT COUNT(*) FROM USUARIO WHERE Nick = p_Nick) = 0;
7 END
8 $$
```

Función 2: Calcular Nivel Promedio de Avatares

```
1 DELIMITER $$
2 CREATE FUNCTION fn_NivelPromedio(p_Login VARCHAR(50))
3 RETURNS DECIMAL(5,2)
4 DETERMINISTIC
5 BEGIN
6     DECLARE v_Promedio DECIMAL(5,2);
7     SELECT AVG(Nivel) INTO v_Promedio FROM AVATAR WHERE Login = p_Login;
8     RETURN IFNULL(v_Promedio, 0);
9 END
10 $$
```

Función 3: Contar Partidas Activas por Juego

```
1 DELIMITER $$
2 CREATE FUNCTION fn_PartidasActivasJuego(p_Cod_Juego INT)
3 RETURNS INT
4 DETERMINISTIC
5 BEGIN
6     DECLARE v_Total INT;
7     SELECT COUNT(*) INTO v_Total FROM PARTIDA
8     WHERE Cod_Juego = p_Cod_Juego AND Estado = 'en curso';
9     RETURN v_Total;
10 END
11 $$
```

Función 4: Validar Acceso a Partida

```
1 DELIMITER $$
2 CREATE FUNCTION fn_PuedeUnirseAPartida(
3     p_Cod_Partida INT,
4     p_Password_Ingresada VARCHAR(50)
5 )
6 RETURNS BOOLEAN
7 DETERMINISTIC
8 BEGIN
9     DECLARE v_Password_Correcta VARCHAR(50);
10    SELECT Password_Partida INTO v_Password_Correcta FROM PARTIDA
11    WHERE Cod_Partida = p_Cod_Partida;
12    RETURN (v_Password_Correcta IS NULL OR v_Password_Correcta = p_Password_Ingresada);
13 END
14 $$
```

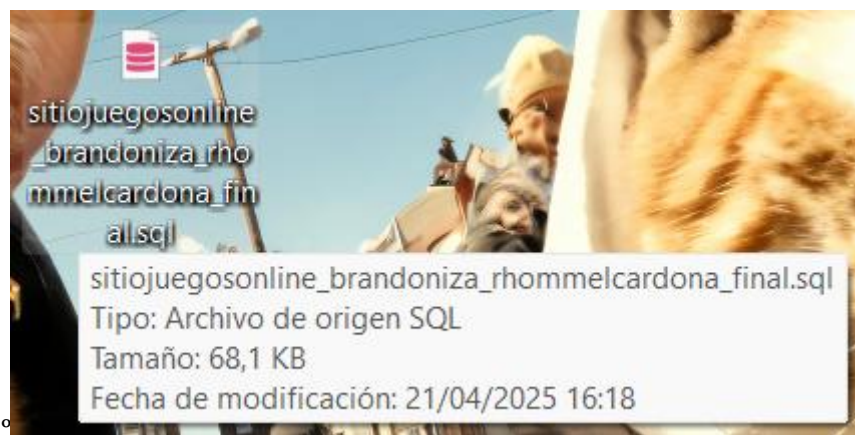
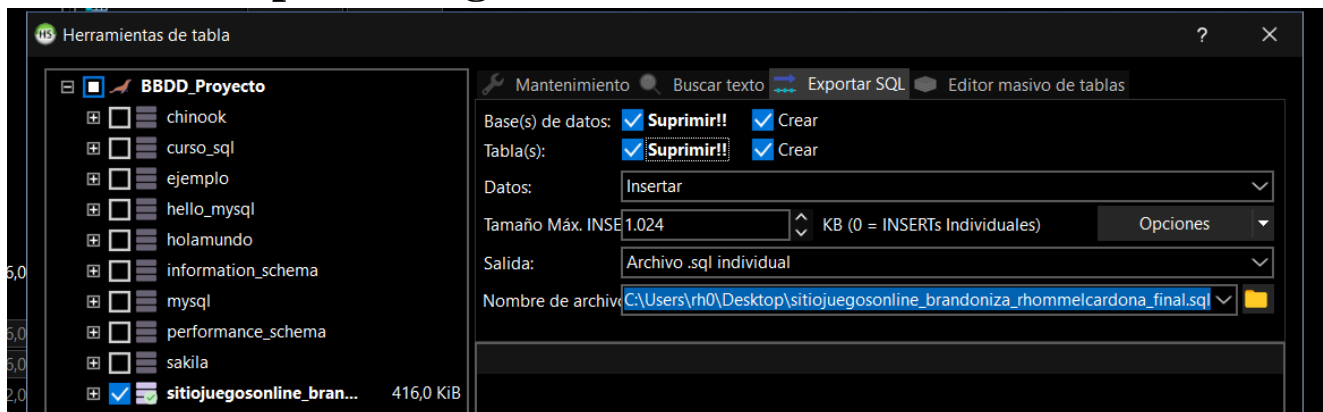
Función 5: Contar Enfrentamientos de un Avatar

```
1 DELIMITER $$
2 CREATE FUNCTION fn_EnfrentamientosAvatar(
3     p_Login VARCHAR(50),
4     p_Nick_Avatar VARCHAR(50)
5 )
6 RETURNS INT
7 DETERMINISTIC
8 BEGIN
9     DECLARE v_Total INT;
10    SELECT COUNT(*) INTO v_Total FROM ENFRENTAMIENTO
11    WHERE (Login_P1 = p_Login AND Nick_Avatar_P1 = p_Nick_Avatar)
12    OR (Login_P2 = p_Login AND Nick_Avatar_P2 = p_Nick_Avatar);
13    RETURN v_Total;
14 END
15 $$
```

Función 6: Obtener Última Partida de un Usuario

```
1 DELIMITER $$
2 CREATE FUNCTION fn_UltimaPartidaUsuario(p_Login VARCHAR(50))
3 RETURNS DATE
4 DETERMINISTIC
5 BEGIN
6     DECLARE v_Fecha DATE;
7     SELECT MAX(p.Fecha_Creacion) INTO v_Fecha
8     FROM PARTICIPACION pa
9     INNER JOIN PARTIDA p ON pa.Cod_Partida = p.Cod_Partida
10    WHERE pa.Login = p_Login;
11    RETURN v_Fecha;
12 END
13 $$
```

12 Copias de seguridad



```
C:\Users\rh0\Desktop>"C:\Program Files\MariaDB 11.5\bin\mysqldump.exe" -u root -p sitiojuegosOnline_brandoniza_rhommelcardona_final > backup_sitiojuegosOnline_brandoniza_rhommelcardona_final.sql
Enter password: ****

C:\Users\rh0\Desktop>z
```



En el ANEXO 9 incluimos el código de la copia de seguridad (backup) realizada al final del proyecto con todas las modificaciones a la base de datos.

12 Conclusión

El desarrollo de este proyecto ha sido una experiencia enriquecedora que nos ha permitido consolidar los conceptos clave del módulo de Bases de Datos y aplicarlos en un contexto práctico y desafiante. Desde el estudio inicial hasta la implementación final, cada fase ha reforzado nuestra capacidad para traducir requisitos técnicos en soluciones funcionales, equilibrando eficiencia y coherencia lógica. Entre los logros más significativos destacan:

1. **Diseño iterativo del MER:** La evolución desde un modelo inicial con relaciones ambiguas hasta un esquema normalizado y robusto, respetando restricciones como el anonimato del creador de partidas.
2. **Implementación técnica rigurosa:** La creación de tablas, claves foráneas y restricciones en MariaDB, asegurando la integridad de los datos incluso bajo cargas de prueba generadas mediante IA.
3. **Optimización y validación:** El uso estratégico de índices, vistas y consultas complejas para garantizar rendimiento, así como triggers y procedimientos almacenados para automatizar reglas de negocio.

Los desafíos enfrentados —como la resolución de relaciones N:M mediante tablas asociativas o la prevención de enfrentamientos inválidos entre avatares— han sido oportunidades para profundizar en el pensamiento crítico y la innovación técnica. Asimismo, la integración de herramientas como *HeidiSQL* y la generación de copias de seguridad automatizadas han ampliado nuestra competencia en entornos profesionales.

Como futuras mejoras, proponemos explorar:

- La integración de métricas de rendimiento en tiempo real para partidas.
- La expansión del modelo para incluir sistemas de logros y recompensas.
- La implementación de APIs que conecten la base de datos con interfaces de usuario dinámicas.

Este proyecto no solo cumple con los objetivos académicos, sino que sienta las bases para enfrentar proyectos de mayor envergadura, demostrando que el aprendizaje continuo y la adaptación son esenciales en el campo de la administración de sistemas y bases de datos.

Brandon Iza y Rhommel Cardona

Abril de 2025