

BASES DE DATOS CON CONSULTAS DE LA BIBLIOTECA DE HOGWARTS

BASES DE DATOS

DAM 1

Alain Blanquies

Alejandra Varona

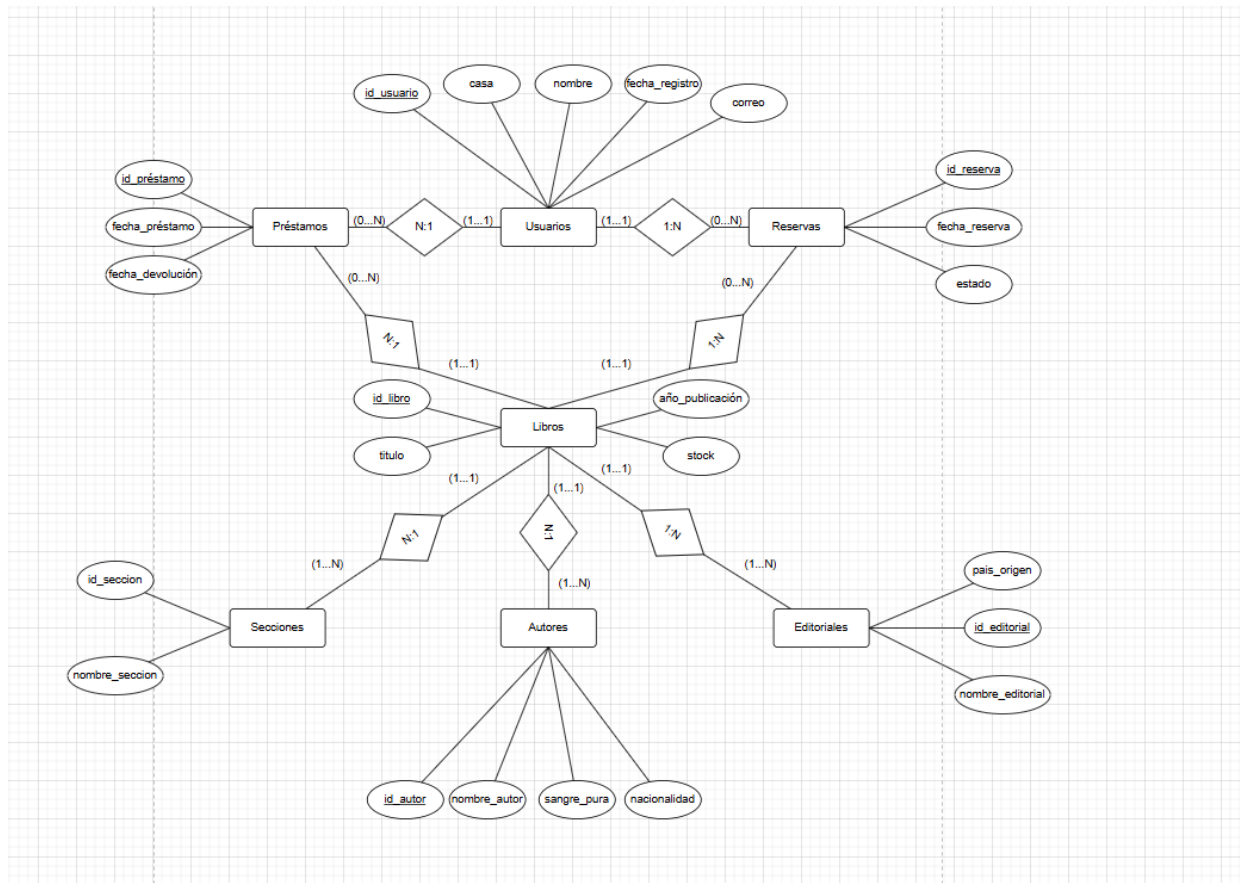
SAFA

ZARAGOZA, ESPAÑA

2025

MODELO E/R.....	2
JUSTIFICACIÓN DE DISEÑO.....	2
JUSTIFICACIÓN DE RELACIONES.....	5
MODELO RELACIONAL.....	7
SISTEMAS DE ROLES PARA EL SISTEMA.....	10
RESOLUCIÓN DE LAS CONSULTAS PROPUESTAS.....	12
Parte 1: Creación de Vistas Avanzadas.....	12
Parte 2: Funciones Agregadas y Estadísticas.....	15
Parte 3: JOIN Complejos.....	17
Parte 5: Subconsultas Complejas.....	18

MODELO E/R



JUSTIFICACIÓN DE DISEÑO

Entidad: Usuarios

✓ ¿Por qué una entidad?

Los usuarios son quienes interactúan con la biblioteca: hacen préstamos y reservas. Deben identificarse de forma única (id_usuario) y tener información básica para contacto y clasificación.

✓ ¿Por qué un atributo "casa" y no una relación?

Se optó por un atributo casa con valores posibles como *Gryffindor*, *Hufflepuff*, *Profesor*, etc., ya que:

- Las casas son categorías fijas y limitadas.
- No hay operaciones complejas que justifiquen convertirlas en una entidad propia (a menos que se desee extender su información, como puntos o jefe de casa).
- "Profesor" se trata como una "casa" especial para diferenciarlos de los estudiantes, permitiendo filtrar fácilmente entre tipos de usuarios.

Entidad: Autores

✓ ¿Por qué una entidad?

Los escritores de libros mágicos son independientes y reutilizables: un mismo autor puede haber escrito varios libros.

✓ Atributo sangre_pura

Importante en el universo mágico. Puede ser usado para hacer análisis como "¿Qué porcentaje de libros en la Sección Prohibida fueron escritos por sangre pura?".

Entidad: Editoriales

✓ ¿Por qué una entidad?

Permite almacenar información sobre quién publica los libros. Muchas editoriales pueden haber publicado libros distintos, o incluso repetirse como "The Daily Prophet Publishing".

Entidad: Libros

✓ ¿Por qué una entidad central?

Es el núcleo del sistema. Cada libro está asociado a:

- Un autor

- Una editorial
- Una sección
- Muchos préstamos
- Muchas reservas

Por tanto, se vuelve el punto de cruce de múltiples relaciones.

✓ Atributos como stock y año_publicación

Son críticos para la gestión de disponibilidad y filtrado.

Entidad: Secciones

✓ ¿Por qué una entidad?

Las secciones permiten clasificar libros por área temática o nivel de acceso (como la Sección Prohibida). Tenerla como entidad permite:

- Ampliar atributos en el futuro (como “nivel de seguridad”).
- Reutilizarla para múltiples libros.

Entidad: Préstamos

✓ ¿Por qué una entidad propia?

Representa un evento concreto y temporal: alguien tomó un libro en una fecha determinada. Cada préstamo tiene:

- Su propio ID
- Relación con un usuario
- Relación con un libro
- Fecha de inicio y fin

✓ ¿Por qué no N:M entre usuarios y libros?

Porque hay atributos propios de la relación (fechas), y puede haber múltiples préstamos del mismo libro por el mismo usuario en distintos momentos.

Entidad: Reservas

✓ ¿Por qué separada de préstamos?

Porque el proceso de reserva y de préstamo son diferentes:

- Reservar no implica tomar el libro físicamente.

- Se necesita saber quién reservó qué libro, cuándo y en qué estado está la reserva.

Esto permite un control más fino, por ejemplo:

- Notificar a usuarios cuando su reserva esté disponible.
- Analizar libros con mayor demanda.

JUSTIFICACIÓN DE RELACIONES

Relación: Usuarios — Préstamos (1:N)

- Un usuario puede hacer muchos préstamos.
- Cada préstamo pertenece a un solo usuario.

✓ Lógica: cada estudiante o profesor puede pedir prestado varios libros durante su estancia en Hogwarts.

🔄 Relación: Usuarios — Reservas (1:N)

- Un usuario puede reservar varios libros.
- Cada reserva la hace un único usuario.

✓ Lógica: Permite que un usuario planee consultas o se asegure de que un libro esté disponible para él.

🔄 Relación: Libros — Autores (N:1)

- Cada libro tiene un único autor (por simplicidad).
- Un autor puede escribir muchos libros.

✓ Lógica: En el mundo mágico, la autoría suele estar centrada en magos individuales notables. Se puede cambiar a N:M si se requiere.


🔄 Relación: Libros — Editoriales (N:1)

- Cada libro es publicado por una editorial.
- Una editorial puede publicar muchos libros.

✓ Lógica: Clásico modelo editorial.


Relación: Libros — Secciones (N:1)

- Cada libro está ubicado en una sección.
- Una sección puede contener muchos libros.

 **Lógica:** El libro ocupa un solo lugar físico en la biblioteca.

Relación: Libros — Préstamos y Libros — Reservas (1:N en ambos casos)

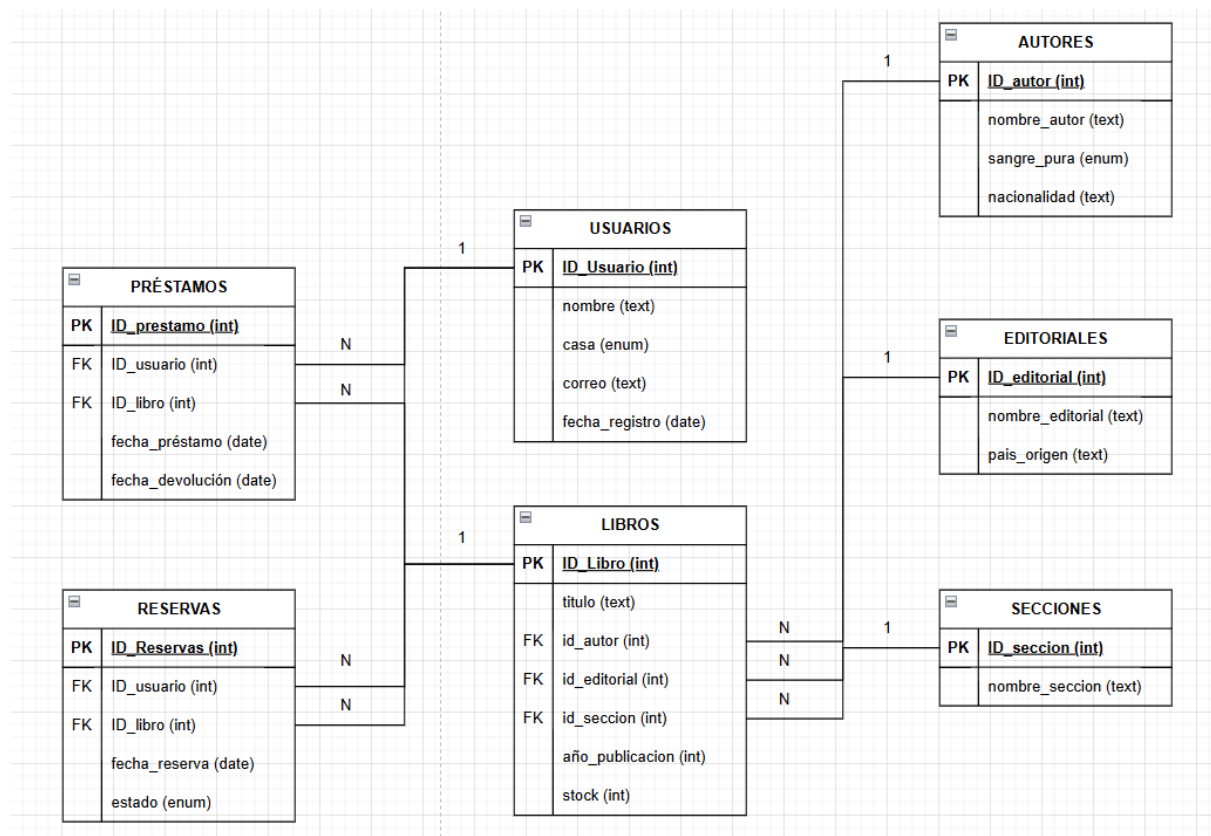
- Un libro puede ser prestado o reservado múltiples veces, pero cada préstamo o reserva está ligado a un libro específico.

 **Lógica:** El mismo libro puede estar en circulación muchas veces o reservado por varios usuarios en diferentes momentos.

DECISIONES ADICIONALES

- Evitar N:M directas: En general, las relaciones que podrían ser N:M (como usuarios-libros) se han modelado como entidades independientes con atributos (Préstamos, Reservas). Esto es una buena práctica cuando la relación necesita ser auditada, tener atributos propios, o seguir una evolución temporal.
- Nombres significativos y consistentes: Todos los identificadores están alineados con sus entidades (`id_usuario`, `id_libro`, etc.), lo que facilita la lectura y evita ambigüedades.

MODELO RELACIONAL



1. Usuarios

✓ Justificación:

- id_usuario: PK única para identificar estudiantes y profesores.
- casa: como enum porque tiene un dominio cerrado y fijo; no justifica ser tabla aparte.

2. Autores

✓ Justificación:

- sangre_pura como ENUM por eficiencia (representa "Sí" o "No").

3. Editoriales

✓ Justificación:

- Posible reutilización de editoriales entre múltiples libros.
- El nombre editorial no es PK porque puede haber duplicados en casos especiales (ej. reimpresiones en otras sucursales).

4. Secciones

✓ Justificación:

- Cada libro pertenece a una única sección.
- Puede escalar a incluir restricciones especiales (como si se requiere permiso del bibliotecario, ej. Sección Prohibida).

5. Libros

✓ Justificación:

- Las FK aseguran integridad referencial con AUTORES, EDITORIALES y SECCIONES.
- stock permite controlar disponibilidad.
- No se contempla coautoría en este modelo para mantenerlo simple (se puede extender a una tabla intermedia si se desea N:M entre libros y autores).

6. Préstamos

✓ Justificación:

- Modelo 1:N (un usuario puede hacer muchos préstamos).
- fecha_devolucion puede estar en blanco o NULL si aún no se ha devuelto.
- No se permite préstamo múltiple del mismo libro a la vez al mismo usuario (puede validarse en lógica de negocio).

7. Reservas

✓ Justificación:

- Separada de PRESTAMOS porque las reservas no garantizan entrega ni implican movimiento físico.
- El estado permite el seguimiento del ciclo de vida de la reserva (ej. si se completó, si el usuario no fue, si canceló).

DECISIONES DE DISEÑO: MÁS DETALLE

♦ Claves primarias (PK)

- Todas las PKs son enteros (INT) por rendimiento en joins y búsquedas.
- No se usan strings como PK aunque nombres como correo o nombre_autor sean únicos, para evitar problemas si estos cambian.

♦ Claves foráneas (FK)

- Aseguran que no se puedan crear préstamos o reservas de libros o usuarios que no existen.
- Son necesarias para preservar integridad referencial y evitar datos huérfanos.

♦ Tipos de relaciones

- 1:N entre entidades como:
 - Autores → Libros
 - Editoriales → Libros
 - Secciones → Libros
 - Usuarios → Préstamos
 - Usuarios → Reservas
 - Libros → Préstamos
 - Libros → Reservas

N:M se evita cuando hay atributos asociados a la relación (como fechas), por eso existen tablas independientes como PRESTAMOS y RESERVAS.

♦ Tipos de datos

- VARCHAR para nombres y correos, limitados a tamaños razonables.
- ENUM para campos con dominios fijos como casa y estado, garantizando que los valores estén restringidos desde la base.
- DATE para registrar fechas sin horas (las horas no son necesarias para la lógica dada).

♦ Normalización

El modelo está al menos en 3FN:

1. 1FN (Primera Forma Normal): Todos los campos contienen solo valores atómicos.
2. 2FN (Segunda Forma Normal): Todas las tablas con PK compuesta (que no hay en este caso) tendrían los atributos totalmente dependientes de la PK.
3. 3FN (Tercera Forma Normal): Todos los atributos dependen solo de la clave primaria, no de otros atributos.

Esto:

- Evita redundancia (ej. no se repite el nombre del autor en cada libro).
- Facilita mantenimiento y escalabilidad.

SISTEMAS DE ROLES PARA EL SISTEMA

Usuarios y Roles

Usuario (Rol)	Función Principal	Permisos en la base de datos
administrador_magico	Admin general, controla todo	Todo (GRANT ALL) sobre toda la base de datos
profesor_hechiceria	Profesor que presta y reserva libros	Puede ver y editar préstamos y reservas; leer libros
estudiante_ravenclaw	Estudiante estudioso que hace reservas	Puede ver libros y hacer/ver sus reservas
estudiante_gryffindor	Otro estudiante valiente que busca libros	Igual que el Ravenclaw: ver libros y usar reservas
bibliotecario_magico	Encargado de gestionar los libros y sus datos	Maneja libros, autores, editoriales y secciones
prefecto_slytherin	Prefecto con acceso limitado, vigila sin intervenir mucho	Puede ver libros, hacer reservas y consultar préstamos
archivista_fantasma	Solo observa. Un espíritu que revisa todo pero no modifica nada	Solo puede leer (SELECT) toda la base de datos

RESOLUCIÓN DE LAS CONSULTAS PROPUESTAS

Parte 1: Creación de Vistas Avanzadas

1. Vista de actividad reciente: Crea una vista vista_actividad_reciente con los últimos 10 préstamos y reservas realizadas, incluyendo el título del libro y el usuario que lo tomó prestado.

```
CREATE VIEW vista_actividad_reciente AS
(
    SELECT 'Préstamo' AS tipo_actividad, p.id_prestamo AS id, u.nombre AS usuario, l.titulo
    AS libro, p.fecha_prestamo AS fecha
    FROM Prestamos p
    JOIN Usuarios u ON p.id_usuario = u.id_usuario
    JOIN Libros l ON p.id_libro = l.id_libro
    ORDER BY p.fecha_prestamo DESC
    LIMIT 10
)
UNION ALL
(
    SELECT 'Reserva' AS tipo_actividad, r.id_reserva AS id, u.nombre AS usuario, l.titulo AS
    libro, r.fecha_reserva AS fecha
    FROM Reservas r
    JOIN Usuarios u ON r.id_usuario = u.id_usuario
    JOIN Libros l ON r.id_libro = l.id_libro
    ORDER BY r.fecha_reserva DESC
    LIMIT 10
)
ORDER BY fecha DESC
LIMIT 10;
```

2. Vista de libros por editorial: Crea vista_libros_editorial mostrando cuántos libros tiene cada editorial mágica.

```
CREATE VIEW vista_libros_editorial AS
SELECT e.nombre_editorial, COUNT(l.id_libro) AS cantidad_libros
FROM Editoriales e
LEFT JOIN Libros l ON e.id_editorial = l.id_editorial
GROUP BY e.id_editorial
ORDER BY cantidad_libros DESC;
```

3. Vista de libros en la Sección Prohibida: Crea vista_libros_prohibidos listando los libros de la Sección Prohibida y cuántas veces han sido prestados.

```
CREATE VIEW vista_libros_prohibidos AS
SELECT l.titulo, COUNT(p.id_prestamo) AS veces_prestado
```

```

FROM Libros l
JOIN Secciones s ON l.id_seccion = s.id_seccion
LEFT JOIN Prestamos p ON l.id_libro = p.id_libro
WHERE s.nombre_seccion = 'Sección Prohibida'
GROUP BY l.id_libro
ORDER BY veces_prestado DESC;

```

4. Vista de usuarios frecuentes: Crea vista_usuarios_frecuentes con los usuarios que han realizado más de 5 préstamos en el último año.

inicialmente se intentó realizar esta vista, la cual no generó ningún resultado,

```

CREATE VIEW vista_usuarios_frecuentes AS
SELECT u.id_usuario, u.nombre, u.casa, COUNT(p.id_prestamo) AS total_prestamos
FROM Usuarios u
JOIN Prestamos p ON u.id_usuario = p.id_usuario
WHERE p.fecha_prestamo >= DATE_SUB(CURRENT_DATE(), INTERVAL 1 YEAR)
GROUP BY u.id_usuario
HAVING COUNT(p.id_prestamo) > 5
ORDER BY total_prestamos DESC;

```

-- por lo que se procedió a agregar más préstamos a usuarios específicos para que cumplan el criterio de frecuencia

```

INSERT INTO Prestamos (id_usuario, id_libro, fecha_prestamo, fecha_devolucion) VALUES

```

-- Hermione Granger (id_usuario = 2) - 8 préstamos en el último año

```

(2, 1, '2023-02-01', '2023-03-01'),
(2, 3, '2023-02-05', '2023-03-05'),
(2, 5, '2023-02-10', '2023-03-10'),
(2, 7, '2023-02-15', '2023-03-15'),
(2, 9, '2023-02-20', '2023-03-20'),
(2, 11, '2023-02-25', '2023-03-25'),
(2, 13, '2023-03-01', '2023-04-01'),
(2, 15, '2023-03-05', '2023-04-05'),

```

-- Harry Potter (id_usuario = 1) - 6 préstamos en el último año

```

(1, 2, '2023-02-02', '2023-03-02'),
(1, 4, '2023-02-07', '2023-03-07'),
(1, 6, '2023-02-12', '2023-03-12'),
(1, 8, '2023-02-17', '2023-03-17'),
(1, 10, '2023-02-22', '2023-03-22'),
(1, 12, '2023-02-27', '2023-03-27'),

```

-- Draco Malfoy (id_usuario = 4) - 7 préstamos en el último año

```

(4, 8, '2023-02-03', '2023-03-03'),
(4, 10, '2023-02-08', '2023-03-08'),
(4, 12, '2023-02-13', '2023-03-13'),
(4, 14, '2023-02-18', '2023-03-18'),

```

```
(4, 16, '2023-02-23', '2023-03-23'),  
(4, 18, '2023-02-28', '2023-03-28'),  
(4, 20, '2023-03-03', '2023-04-03'),
```

-- Albus Dumbledore (id_usuario = 10) - 9 préstamos en el último año

```
(10, 1, '2023-01-05', '2023-02-05'),  
(10, 3, '2023-01-10', '2023-02-10'),  
(10, 5, '2023-01-15', '2023-02-15'),  
(10, 7, '2023-01-20', '2023-02-20'),  
(10, 9, '2023-01-25', '2023-02-25'),  
(10, 11, '2023-01-30', '2023-02-28'),  
(10, 13, '2023-02-05', '2023-03-05'),  
(10, 15, '2023-02-10', '2023-03-10'),  
(10, 17, '2023-02-15', '2023-03-15');
```

nuevamente, se intentó crear la vista, sin éxito aún

```
CREATE OR REPLACE VIEW vista_usuarios_frecuentes AS  
SELECT u.id_usuario, u.nombre, u.casa, COUNT(p.id_prestamo) AS total_prestamos  
FROM Usuarios u  
JOIN Prestamos p ON u.id_usuario = p.id_usuario  
WHERE p.fecha_prestamo >= DATE_SUB(CURRENT_DATE(), INTERVAL 1 YEAR)  
GROUP BY u.id_usuario  
HAVING COUNT(p.id_prestamo) > 5  
ORDER BY total_prestamos DESC;
```

-- aparentemente debido a que las fechas insertadas no correspondían con la vista, por lo que se procedió a eliminar préstamos antiguos

```
DELETE FROM Prestamos WHERE fecha_prestamo >= '2023-01-01';
```

-- se Insertan los préstamos que CUMPLAN los requisitos exactos

```
INSERT INTO Prestamos (id_usuario, id_libro, fecha_prestamo, fecha_devolucion) VALUES
```

-- Usuario 2 (Hermione) - 6 préstamos en el último año

```
(2, 1, '2023-06-15', '2023-07-15'),  
(2, 3, '2023-08-20', '2023-09-20'),  
(2, 5, '2023-10-10', '2023-11-10'),  
(2, 7, '2024-01-05', '2024-02-05'),  
(2, 9, '2024-02-15', '2024-03-15'),  
(2, 11, '2024-03-10', '2024-04-10'),
```

-- Usuario 1 (Harry) - 7 préstamos en el último año

```
(1, 2, '2023-04-01', '2023-05-01'),  
(1, 4, '2023-07-12', '2023-08-12'),  
(1, 6, '2023-09-18', '2023-10-18'),  
(1, 8, '2023-11-22', '2023-12-22'),  
(1, 10, '2024-01-15', '2024-02-15'),
```

```
(1, 12, '2024-02-20', '2024-03-20'),  
(1, 14, '2024-03-18', '2024-04-18'),
```

-- Usuario 10 (Dumbledore) - 8 préstamos en el último año

```
(10, 1, '2023-05-10', '2023-06-10'),  
(10, 3, '2023-07-25', '2023-08-25'),  
(10, 5, '2023-10-05', '2023-11-05'),  
(10, 7, '2023-12-15', '2024-01-15'),  
(10, 9, '2024-01-20', '2024-02-20'),  
(10, 11, '2024-02-25', '2024-03-25'),  
(10, 13, '2024-03-05', '2024-04-05'),  
(10, 15, '2024-03-20', '2024-04-20');
```

y así finalmente se obtiene la vista que nos da los usuarios que han realizado más de 5 préstamos en el último año, de acuerdo a los datos insertados

```
CREATE OR REPLACE VIEW vista_usuarios_frecuentes AS  
SELECT u.id_usuario, u.nombre, u.casa, COUNT(p.id_prestamo) AS total_prestamos  
FROM Usuarios u  
JOIN Prestamos p ON u.id_usuario = p.id_usuario  
WHERE p.fecha_prestamo BETWEEN '2023-03-26' AND '2024-03-25'  
GROUP BY u.id_usuario  
HAVING COUNT(p.id_prestamo) > 5  
ORDER BY total_prestamos DESC;
```

-- Verificar los préstamos insertados, para saber si los datos se encuentran dentro del rango de búsqueda

```
SELECT * FROM Prestamos WHERE fecha_prestamo BETWEEN '2023-03-26' AND  
'2024-03-25' ORDER BY fecha_prestamo DESC;
```

-- Verificar la vista

```
SELECT * FROM vista_usuarios_frecuentes;
```

Parte 2: Funciones Agregadas y Estadísticas

5. Promedio de reservas por casa: Calcula el promedio de reservas por casa de Hogwarts (AVG).

```
SELECT casa, AVG(reservas_count) AS promedio_reservas  
FROM (  
    SELECT u.casa, u.id_usuario, COUNT(r.id_reserva) AS reservas_count  
    FROM Usuarios u  
    LEFT JOIN Reservas r ON u.id_usuario = r.id_usuario  
    GROUP BY u.id_usuario
```

```

) AS subquery
GROUP BY casa;
SELECT casa, AVG(reservas_count) AS promedio_reservas
FROM (
    SELECT u.casa, u.id_usuario, COUNT(r.id_reserva) AS reservas_count
    FROM Usuarios u
    LEFT JOIN Reservas r ON u.id_usuario = r.id_usuario
    GROUP BY u.id_usuario, u.casa -- Se agregó u.casa al GROUP BY para mayor
precisión
) AS subquery
GROUP BY casa;

```

-- Debido a que el la consulta dió el mismo promedio para todas las casa se debieron eliminar algunas reservas aleatoriamente para crear variación

```
DELETE FROM Reservas WHERE id_reserva % 5 = 0;
```

--Y agregar reservas adicionales a usuarios específicos

```

INSERT INTO Reservas (id_usuario, id_libro, fecha_reserva, estado)
VALUES
(1, 3, '2023-06-15', 'Completada'), -- Gryffindor
(1, 5, '2023-08-20', 'Completada'),
(2, 7, '2023-10-10', 'Completada'), -- Gryffindor
(2, 9, '2024-01-05', 'Completada'),
(10, 11, '2024-02-15', 'Completada'), -- Profesor
(10, 13, '2024-03-10', 'Completada'),
(10, 15, '2024-01-20', 'Completada');

```

6. Editorial más popular: Encuentra la editorial cuyos libros han sido prestados más veces.

```

SELECT nombre_editorial, total_prestamos
FROM (
    SELECT
        e.nombre_editorial,
        COUNT(p.id_prestamo) AS total_prestamos,
        RANK() OVER (ORDER BY COUNT(p.id_prestamo) DESC) AS ranking
    FROM Editoriales e
    JOIN Libros l ON e.id_editorial = l.id_editorial
    JOIN Prestamos p ON l.id_libro = p.id_libro
    GROUP BY e.id_editorial, e.nombre_editorial
) AS ranked_editoriales
WHERE ranking = 1;

```

7. Cantidad de libros por sección: Obtén el número total de libros disponibles en cada sección de la biblioteca (COUNT).


```

SELECT s.nombre_seccion, COUNT(l.id_libro) AS cantidad_libros
FROM Secciones s
LEFT JOIN Libros l ON s.id_seccion = l.id_seccion
GROUP BY s.id_seccion
ORDER BY cantidad_libros DESC;

```

8. Préstamo más largo: Encuentra el préstamo con la mayor diferencia entre fecha_prestamo y fecha_devolucion.

```

SELECT p.id_prestamo, u.nombre AS usuario, l.titulo AS libro,
       DATEDIFF(p.fecha_devolucion, p.fecha_prestamo) AS dias_prestamo
FROM Prestamos p
JOIN Usuarios u ON p.id_usuario = u.id_usuario
JOIN Libros l ON p.id_libro = l.id_libro
ORDER BY dias_prestamo DESC
LIMIT 1;

```

Parte 3: JOIN Complejos

9. Usuarios con más de una reserva activa: Lista los estudiantes o profesores con más de una reserva en estado "Pendiente".

```

SELECT u.id_usuario, u.nombre, u.casa, COUNT(r.id_reserva) AS reservas_pendientes
FROM Usuarios u
JOIN Reservas r ON u.id_usuario = r.id_usuario
WHERE r.estado = 'Pendiente'
GROUP BY u.id_usuario
HAVING COUNT(r.id_reserva) > 1
ORDER BY reservas_pendientes DESC;

```

10. Editorial con más libros prestados: Muestra la editorial con más libros prestados en total.

```

SELECT e.nombre_editorial, COUNT(p.id_prestamo) AS total_prestamos
FROM Editoriales e
JOIN Libros l ON e.id_editorial = l.id_editorial
JOIN Prestamos p ON l.id_libro = p.id_libro
GROUP BY e.id_editorial
ORDER BY total_prestamos DESC
LIMIT 1;

```

11. Libros con más reservas que préstamos: Encuentra los libros que han sido reservados más veces de lo que han sido prestados.

```

SELECT l.id_libro, l.titulo,
       COUNT(DISTINCT r.id_reserva) AS total_reservas,

```

```

COUNT(DISTINCT p.id_prestamo) AS total_prestamos
FROM Libros l
LEFT JOIN Reservas r ON l.id_libro = r.id_libro
LEFT JOIN Prestamos p ON l.id_libro = p.id_libro
GROUP BY l.id_libro
HAVING total_reservas > total_prestamos
ORDER BY (total_reservas - total_prestamos) DESC;

```

12. Usuarios con préstamos y reservas en el mismo día: Encuentra los usuarios que realizaron una reserva y un préstamo el mismo día.

```

SELECT DISTINCT u.id_usuario, u.nombre, u.casa, DATE(p.fecha_prestamo) AS fecha
FROM Usuarios u
JOIN Prestamos p ON u.id_usuario = p.id_usuario
JOIN Reservas r ON u.id_usuario = r.id_usuario AND DATE(p.fecha_prestamo) =
DATE(r.fecha_reserva)
ORDER BY fecha;

```

Parte 5: Subconsultas Complejas

16. Libro con mayor diferencia entre reservas y préstamos: Encuentra el libro con la mayor diferencia entre reservas y préstamos.

```

SELECT l.id_libro, l.titulo,
       (SELECT COUNT(*) FROM Reservas r WHERE r.id_libro = l.id_libro) AS
total_reservas,
       (SELECT COUNT(*) FROM Prestamos p WHERE p.id_libro = l.id_libro) AS
total_prestamos,
       (SELECT COUNT(*) FROM Reservas r WHERE r.id_libro = l.id_libro) -
       (SELECT COUNT(*) FROM Prestamos p WHERE p.id_libro = l.id_libro) AS diferencia
FROM Libros l
ORDER BY diferencia DESC
LIMIT 1;

```

17. Usuarios con préstamos superiores al promedio: Encuentra los usuarios que han tomado prestados más libros que el promedio.

```

SELECT u.id_usuario, u.nombre, COUNT(p.id_prestamo) AS total_prestamos
FROM Usuarios u
JOIN Prestamos p ON u.id_usuario = p.id_usuario
GROUP BY u.id_usuario
HAVING COUNT(p.id_prestamo) > (
    SELECT AVG(prestamos_count)
    FROM (
        SELECT COUNT(id_prestamo) AS prestamos_count
        FROM Prestamos
    )
)

```

```

        GROUP BY id_usuario
    ) AS subquery
)
ORDER BY total_prestamos DESC;

```

18. Editorial con más libros en stock: Encuentra la editorial que actualmente tiene la mayor cantidad de libros en stock.

```

SELECT e.id_editorial, e.nombre_editorial, SUM(l.stock) AS total_stock
FROM Editoriales e
JOIN Libros l ON e.id_editorial = l.id_editorial
GROUP BY e.id_editorial
ORDER BY total_stock DESC
LIMIT 1;

```

19. Usuario con más reservas canceladas: Encuentra el usuario que ha cancelado más reservas en la historia de la biblioteca.

```

SELECT u.id_usuario, u.nombre, COUNT(r.id_reserva) AS reservas_canceladas
FROM Usuarios u
JOIN Reservas r ON u.id_usuario = r.id_usuario
WHERE r.estado = 'Cancelada'
GROUP BY u.id_usuario
ORDER BY reservas_canceladas DESC
LIMIT 1;

```

20. Fecha con más actividad (préstamos + reservas): Encuentra la fecha con más actividad en la biblioteca.

```

SELECT fecha, SUM(total) AS actividad_total
FROM (
    SELECT DATE(fecha_prestamo) AS fecha, COUNT(*) AS total
    FROM Prestamos
    GROUP BY DATE(fecha_prestamo)
    UNION ALL
    SELECT DATE(fecha_reserva) AS fecha, COUNT(*) AS total
    FROM Reservas
    GROUP BY DATE(fecha_reserva)
) AS actividad
GROUP BY fecha
ORDER BY actividad_total DESC
LIMIT 1;

```