

# ***DOCUMENTACIÓN: API DE CONSULTA PARA DB SAKILA***

# *Índice:*

<b>1. Proyecto base.....</b>	<b>3</b>
<b>2. Diagrama de trabajo y equipos.....</b>	<b>6</b>
<b>3. Diagrama de relacion entre tablas de Sakila.....</b>	<b>8</b>
<b>4. Recordatorio de GIT.....</b>	<b>9</b>
<b>5. Resumen manual de Boostrap.....</b>	<b>11</b>
<b>6. Consultas verificadas en el código base.....</b>	<b>24</b>
<b>7. Corrección de problemas en la instalación.....</b>	<b>47</b>
<b>8. Consulta individual.....</b>	<b>49</b>
<b>9. Node y Express (resumen manual).....</b>	<b>54</b>
<b>10. Guía final de instalación.....</b>	<b>58</b>

## 1. Proyecto base:

Documentación Guía de ChatGPT:

1 - Creación de las carpetas \_( Windows & Linux )\_

2 - Parte Frontend:

- index.html \_( con main.js )\_
- main.ts \_( front listo y esperando datos de sakila )\_
- configuración básica ts \_( tsconfig.json )\_
- package.json \_( front )\_

3 - Parte backend:

- server.ts \_(Servidor escuchando )\_
- app.ts \_( conexion con la base )\_
- configuración básica ts \_( tsconfig.json )\_
- package.json \_( back )\_

4 - Compilar y levantar los dos proyectos automáticamente:

- Windows o Linux \_( Resultado esperado ( Comprobar cuales ) )\_

5 - Back - Modelo,Controlar,Ruta

- Film.ts \_( interface de Film )\_
- filmController.ts \_( Query de consulta? )\_
- filmRoutes.ts

Modificar el app.ts y agregar una lineas más

6 - Front - Consumir Api y Mostrar Tabla - main.ts \_( actualizado )\_

7 - Mejorar back - filmController.ts

8 - Mejorar front - main.ts

9 - Mejorar front de nuevo - main.ts \_( mejorar )\_

10 - Mejora de nuevo del front - main.ts \_( ordenamiento )\_

11 - Mostrar total de resultados:

- Mejorar back - filmcontroller.ts
- Adaptar front - main.ts \_( cambiarlo )\_
- loadFilm.ts \_( cambiarlo )\_

- renderFilms.ts \_( agregar una linea )\_

12 - Mostrar filtro por año:

- Back - Soportar filtro año \_( filmController.ts - actualización )\_
- Front - Agregar select de años
- main.ts \_( añadir una cosa )\_
- renderFilms.ts \_( Modificar ?? )\_

13 - Cerrar modal (main.ts)

14 - Btn limpiar filtros \_( renderFilms )\_

15 - Paginación numérica - Front - \_( Actualizar na cosa en renderFilms )\_

16 - Mostrar lenguaje modal

- Back:
- Modificar el filmController.ts - Actualizar el Film.ts
- Front: - Añadir una linea Nueva \_( html )\_

17 - Front and Back juntos \_( similar deploy ):\_

- Back: - Actualizar server.ts
- Comando similar deploy

18

- Creación del README.md
- Configurar base de datos
- Instalar dependencias
- Compilar proyectos
- Correr en modo producción
- Licencia

19 - Publicar sakila front + back opción A o B

20 - Dockerfile \_( Comandos )\_

21 - DockerCompose \_( Pequeña modificaicón app.ts )\_

22 - Publicar Sakila app Railway

- Publicar Back + front + mariadb
- Seguir pasos del 1 al 5

23 - últimos archivos de calidad ( backend [.env.example, .dockerignore])

24 - Script deploy\_local.sh(Linux/Mac) && Script deplo\_local.bat (Windows)

25 - Mejorado el readme

26 - Alternativas deploy Sakila

- Render
- Fly.io
- Google Cloud Run
- ...

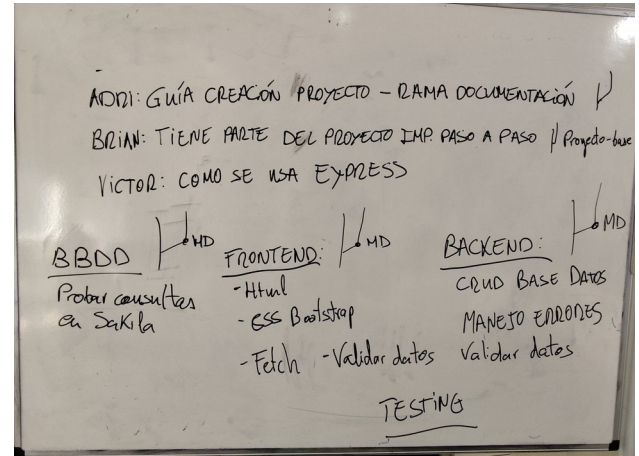
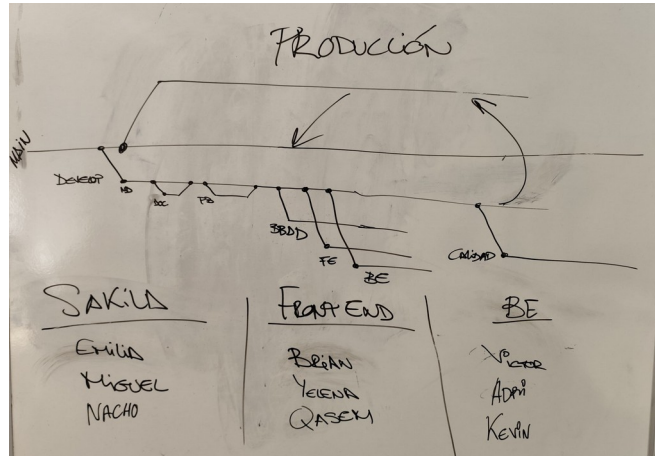
27 - Front en Vercel conectado back en railway

- Back ( Habilitar cors)
- Front ( Apuntar URL to Back)

28 - Desplegar app en Render \_( Seguir puntos )\_

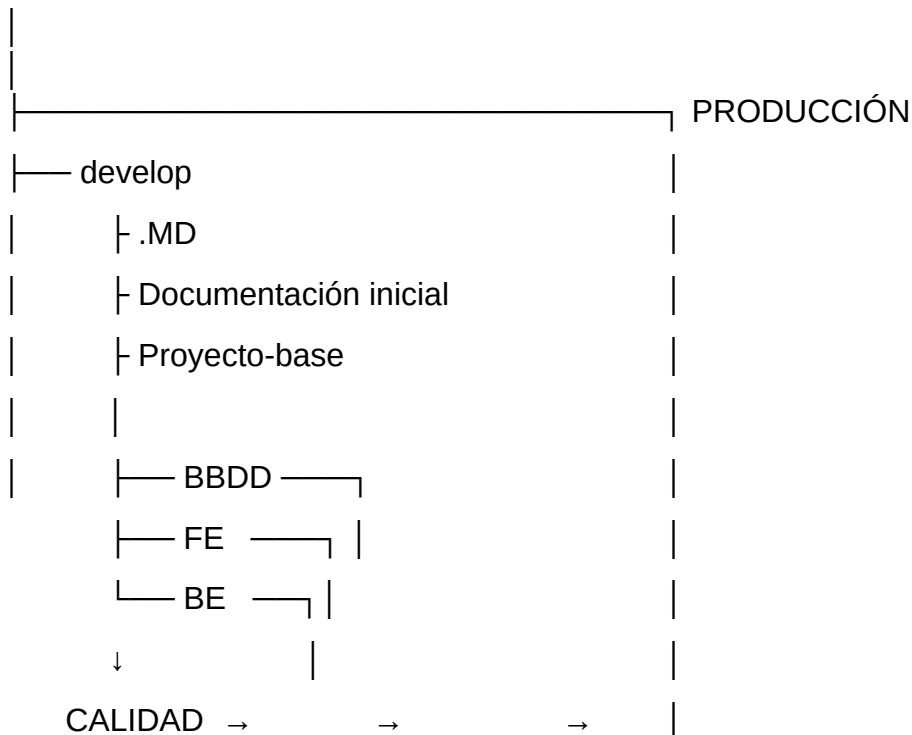
29 - Ultra optimización.

## 2. Diagrama de trabajo y equipos:



## Flujo de Ramas (Git)

### MAIN



## Equipos:

### BD

- Emilia
- Miguel
- Nacho

### FRONTEND

- Brian
- Yelena
- Qasem

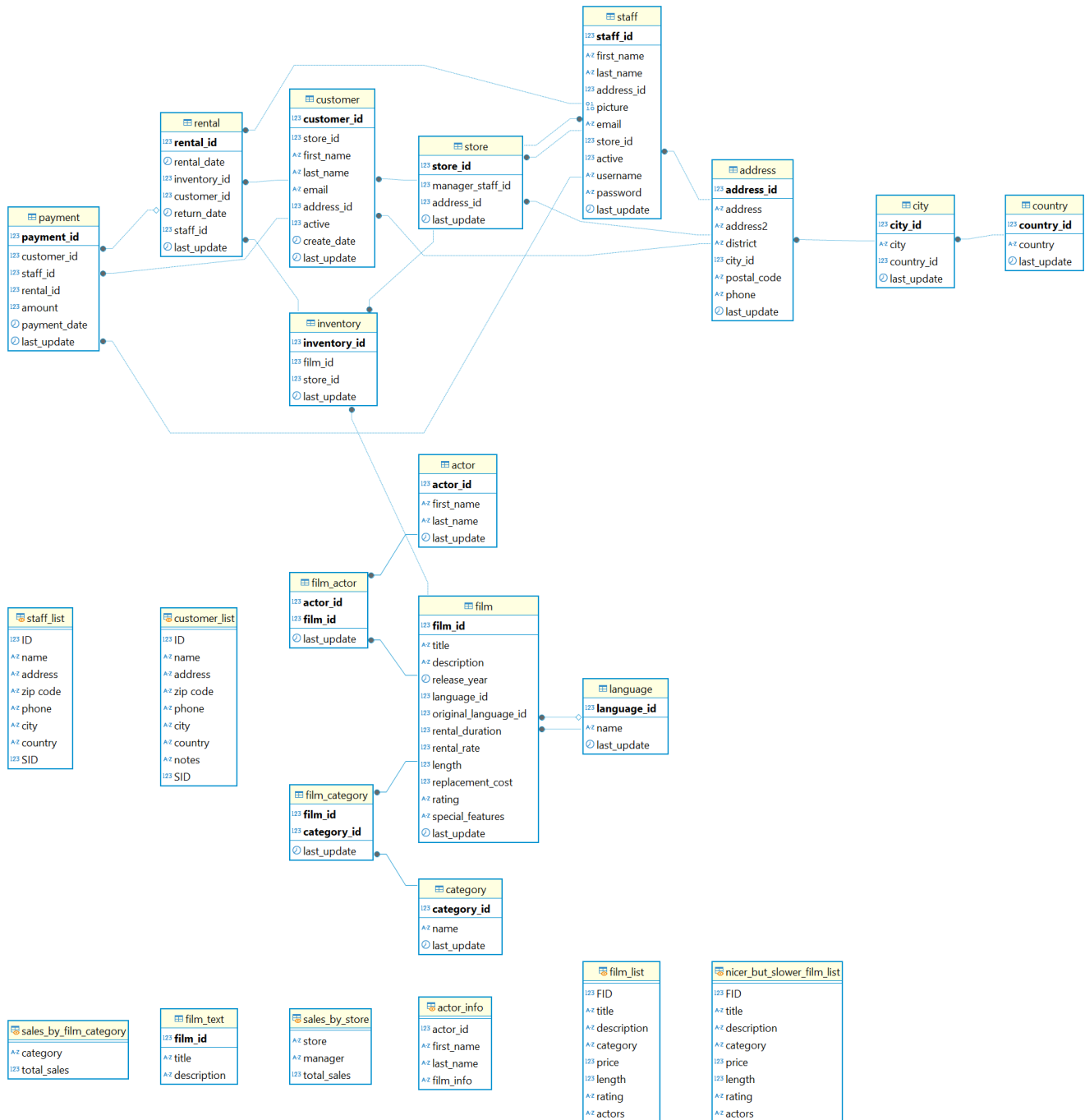
### BE (Backend)

- Víctor
- Adri
- Kevin

## Interpretación del flujo:

1. **MAIN** es la rama principal.
2. **DEVELOP** es una rama de desarrollo.
3. A partir de DEVELOP se crean ramas paralelas:
  - BBDD (Base de datos)
  - FE (Frontend)
  - BE (Backend)
4. Estas se integran a develop y posteriormente allí se pasan por **CALIDAD** (testing o QA).
5. Finalmente, el código aprobado se mueve a **PRODUCCIÓN**.

### 3. Diagrama de relacion entre tablas de Sakila:





## 4. Recordatorio de GIT:

 **Tabla 1: Comandos básicos, ramas y cambios**

Comando	Descripción
<code>git config --global user.name "Nome Apellido"</code>	Cambia el nombre del usuario.
<code>git config --global user.email "correo@ejemplo.com"</code>	Cambia el email del usuario.
<code>git init</code>	Inicializa el repositorio.
<code>git add &lt;archivo&gt;</code>	Añade archivo(s) al staging.
<code>git status</code>	Muestra el estado de los archivos.
<code>git commit</code>	Crea un commit desde staging.
<code>git commit -m "mensaje"</code>	Commit con mensaje.
<code>git commit -a -m "mensaje"</code>	Commit sin usar add.
<code>git log --oneline --graph</code>	Historial del repositorio.
<code>git diff &lt;id&gt; / &lt;archivo&gt;</code>	Muestra diferencias.
<code>git restore &lt;archivo&gt; / checkout &lt;archivo&gt;</code>	Deshace cambios.
<code>git tag -a &lt;etiqueta&gt; -m "mensaje"</code>	Etiqueta un commit.

### **Ramas**

Comando	Descripción
<code>git branch &lt;rama&gt;</code>	Crea rama desde actual.
<code>git branch &lt;rama&gt; &lt;base&gt;</code>	Crea rama desde otra.
<code>git checkout &lt;rama&gt; / switch &lt;rama&gt;</code>	Cambia de rama.
<code>git checkout -b &lt;rama&gt; / switch -c &lt;rama&gt;</code>	Crea y cambia.
<code>git merge &lt;rama&gt;</code>	Fusiona ramas.
<code>git branch -d &lt;rama&gt;</code>	Elimina rama.
<code>git log --all --oneline --graph</code>	Historial en todas las ramas como grafo.

 **Tabla 2: Reversión de cambios, stash y remoto**

Comando	Descripción
<code>git reset --hard &lt;id&gt;</code>	Vuelve a commit anterior, elimina posteriores.
<code>git revert &lt;id&gt;</code>	Revierte un commit conservando el resto.
<code>git checkout &lt;id&gt;</code>	Revisión temporal de un commit.
<code>git checkout &lt;archivo&gt;</code>	Restaura archivo desde último commit.
<code>git reset &lt;archivo&gt;</code>	Saca archivo del staging.

## Stash

Comando	Descripción
<code>git stash</code>	Guarda cambios sin commit.
<code>git stash list</code>	Lista los stashes.
<code>git stash apply &lt;id&gt;</code>	Aplica un stash.
<code>git stash drop &lt;id&gt;</code>	Elimina un stash.

## Remoto

Comando	Descripción
<code>git remote add origin &lt;url&gt;</code>	Añade repositorio remoto.
<code>git remote remove &lt;nombre&gt;</code>	Elimina repositorio remoto.
<code>git push origin master</code>	Sube rama master.
<code>git push -u origin master</code>	Define upstream remoto.
<code>git pull origin master</code>	Trae y fusiona cambios.
<code>git fetch</code>	Verifica cambios sin descarga.
<code>git clone &lt;url&gt;</code>	Clona repositorio.

## 5. Resumen manual de Bootstrap:

### Estructura de Archivos Recomendada

Para personalizar Bootstrap sin modificar sus archivos fuente, se sugiere la siguiente estructura:

```
vbnet
CopiarEditar
tu-proyecto/
├── scss/
│   └── custom.scss
├── node_modules/
│   └── bootstrap/
│       ├── js/
│       └── scss/
└── index.html
```

En custom.scss, importarás los archivos Sass de Bootstrap y realizarás tus personalizaciones.

### Importación de Bootstrap en tu Archivo Sass

En tu archivo custom.scss, puedes importar Bootstrap de la siguiente manera:

```
scss
CopiarEditar
// Importa las funciones de Bootstrap
@import "node_modules/bootstrap/scss/functions";

// Aquí puedes sobrescribir variables predeterminadas de Bootstrap
$primary: #ff5733;

// Importa el resto de los archivos de Bootstrap
@import "node_modules/bootstrap/scss/bootstrap";
```

Este enfoque te permite sobrescribir variables antes de que se utilicen en los estilos de Bootstrap.

### Compilación de Sass

Para compilar tu archivo Sass en CSS, puedes utilizar herramientas como:

- **Dart Sass:** el compilador oficial de Sass.
- **Webpack, Parcel o Vite:** empaquetadores de módulos que pueden manejar la compilación de Sass.

Por ejemplo, usando Dart Sass desde la terminal:

```
bash
CopiarEditar
sass scss/custom.scss css/custom.css
```

Esto generará un archivo CSS que puedes incluir en tu HTML.

## Personalización Avanzada con Mapas y Mixins

Bootstrap utiliza **mapas** para definir conjuntos de variables, como colores o tamaños. Puedes modificar estos mapas para personalizar componentes:

```
scss
CopiarEditar
// Agrega un nuevo color al mapa de colores
$theme-colors: map-merge($theme-colors, (
  "nuevo-color": #123456
));
```

Además, Bootstrap proporciona **mixins** para aplicar estilos de manera consistente:

```
scss
CopiarEditar
@include media-breakpoint-up(md) {
  // Estilos para pantallas medianas en adelante
}
```

## Opciones Globales

Bootstrap ofrece variables Sass para habilitar o deshabilitar características globales, como:

- \$enable-rounded: controla los bordes redondeados.
- \$enable-shadows: activa o desactiva las sombras.
- \$enable-gradients: permite o no los degradados.

Estas variables se pueden ajustar en tu archivo custom.scss antes de importar los archivos principales de Bootstrap.

## Modos de Color

Bootstrap 5.3 introduce soporte para modos de color, como claro y oscuro, utilizando la variable data-bs-theme. Puedes definir tus propios modos de color personalizados y ajustar variables Sass y CSS según sea necesario.

## LAYOUT:



### 1. Contenedores (.container)

Los contenedores son elementos fundamentales en Bootstrap que sirven para alinear y espaciar el contenido dentro de un dispositivo o ventana de visualización. Existen tres tipos principales:

- **.container**: Establece un ancho máximo en cada punto de interrupción (breakpoint) y centra el contenido horizontalmente.
- **.container-{breakpoint}**: Tiene un ancho del 100% hasta el punto de interrupción especificado, donde se fija un ancho máximo.
- **.container-fluid**: Siempre tiene un ancho del 100%, independientemente del tamaño de la ventana.



### 2. Sistema de Rejilla (Grid System)

Bootstrap utiliza un sistema de rejilla basado en Flexbox que permite crear diseños responsivos y adaptables. Este sistema se basa en una estructura de 12 columnas y se compone de:

- **Contenedores (.container)**: Envoltorio general del contenido.
- **Filas (.row)**: Agrupan columnas y aseguran un alineamiento adecuado.
- **Columnas (.col)**: Contienen el contenido real y se ajustan según el tamaño de la pantalla.

Las clases de columna se pueden combinar con los puntos de interrupción para definir cómo se comportan en diferentes tamaños de pantalla, por ejemplo: `.col-6`, `.col-md-4`, `.col-lg-3`.



### 3. Puntos de Interrupción (Breakpoints)

Los puntos de interrupción son anchos de pantalla predefinidos que permiten adaptar el diseño a diferentes dispositivos. Bootstrap define los siguientes:

- **Extra pequeño (xs)**: <576px (sin clase infix)
- **Pequeño (sm)**: ≥576px
- **Mediano (md)**: ≥768px
- **Grande (lg)**: ≥992px

- **Extra grande (xl):**  $\geq 1200\text{px}$
- **Extra extra grande (xxl):**  $\geq 1400\text{px}$

Estos puntos de interrupción se utilizan en clases como `.col-md-6` o `.d-lg-none` para aplicar estilos específicos según el tamaño de la pantalla.

## 4. Columnas Avanzadas

Además de las clases básicas de columnas, Bootstrap ofrece opciones avanzadas para:

- **Alineación vertical y horizontal:** Utilizando clases como `.align-items-center` o `.justify-content-end`.
- **Ordenamiento:** Cambiar el orden de las columnas con clases como `.order-1` o `.order-md-2`.
- **Desplazamiento (Offset):** Mover columnas hacia la derecha con clases como `.offset-md-3`.

Estas opciones proporcionan un control más preciso sobre la disposición de las columnas en diferentes tamaños de pantalla.

## 5. CSS Grid (Experimental)

Bootstrap 5.1 introdujo un sistema de rejilla alternativo basado en CSS Grid, que es experimental y opcional. Para habilitarlo:

1. Desactiva las clases de la rejilla predeterminada estableciendo `$enable-grid-classes: false`.
2. Activa CSS Grid estableciendo `$enable-cssgrid: true`.
3. Recompila tu Sass.

Este sistema utiliza clases como `.grid` y `.g-col-4` y se basa en propiedades de CSS Grid como `grid-template-columns` y `gap`.

## 6. Utilidades de Diseño

Bootstrap incluye numerosas clases utilitarias para facilitar el diseño y la disposición de elementos:

- **Visualización (display):** Controla cómo se muestra un elemento, por ejemplo, `.d-none` para ocultar o `.d-flex` para aplicar Flexbox.

- **Flexbox:** Clases como `.justify-content-center` o `.align-items-start` para alinear elementos.
- **Espaciado (margin y padding):** Clases como `.m-3` o `.p-2` para ajustar márgenes y rellenos.
- **Visibilidad:** Clases como `.visible` o `.invisible` para mostrar u ocultar elementos sin afectar el diseño.

Estas utilidades permiten aplicar estilos de manera rápida y coherente sin necesidad de escribir CSS personalizado.

## CONTENT:



### 1. Reboot: Normalización de Estilos

**Reboot** es el conjunto de estilos base de Bootstrap que proporciona una base coherente y moderna para los navegadores. Incluye:

- **Box-sizing:** Se establece en `border-box` para todos los elementos, lo que facilita el cálculo de anchos y alturas.
- **Estilos tipográficos básicos:** Define estilos predeterminados para encabezados, párrafos, listas, enlaces, entre otros.

**Correcciones de inconsistencias:** Ajustes para asegurar una apariencia uniforme en diferentes navegadores.



### 2. Tipografía

Bootstrap ofrece una tipografía limpia y accesible mediante:

- **Sistema de fuentes:** Utiliza una pila de fuentes que prioriza las fuentes del sistema operativo para mejorar el rendimiento y la coherencia visual.
- **Escala tipográfica:** Basada en `rem`, lo que facilita la adaptabilidad y la accesibilidad.

**Clases utilitarias:** Para ajustar rápidamente estilos como tamaño (`.fs-1` a `.fs-6`), peso (`.fw-bold`, `.fw-light`), alineación (`.text-center`, `.text-end`), entre otros.

### 3. Imágenes

Bootstrap facilita el trabajo con imágenes mediante:

- **Clases responsivas:** `.img-fluid` para que las imágenes se escalen adecuadamente dentro de su contenedor.

**Estilos adicionales:** `.rounded`, `.rounded-circle`, `.img-thumbnail` para aplicar bordes redondeados, formas circulares o estilos de miniatura.

### 4. Tablas

Las tablas en Bootstrap se pueden estilizar y hacer responsivas con:

- **Clases básicas:** `.table` para aplicar estilos predeterminados.
- **Variantes:** `.table-striped`, `.table-bordered`, `.table-hover`, `.table-sm` para diferentes estilos visuales.
- **Responsividad:** `.table-responsive` para envolver la tabla y permitir el desplazamiento horizontal en dispositivos pequeños.

### 5. Figuras

Para asociar imágenes con leyendas, Bootstrap ofrece:

- **Contenedor** `.figure`: Agrupa la imagen y su leyenda.
- **Imagen** `.figure-img`: Aplica estilos específicos a la imagen dentro de una figura.
- **Leyenda** `.figure-caption`: Estiliza el texto descriptivo asociado a la imagen.

### 6. Utilidades de Contenido

Bootstrap proporciona diversas clases utilitarias para mejorar la presentación del contenido:

- **Listas:** `.list-unstyled`, `.list-inline` para listas sin estilos predeterminados o en línea.
- **Bloques de código:** `.pre-scrollable` para permitir el desplazamiento en bloques de código largos.
- **Textos:** `.text-muted`, `.text-primary`, `.text-success`, etc., para aplicar colores contextuales al texto.



## FORMS & CONTENT:



### Formularios (Forms)

Bootstrap 5.3 ofrece una amplia gama de herramientas para crear formularios accesibles y personalizables. A continuación, se destacan los principales aspectos:

#### 1. Controles de Formulario

Bootstrap proporciona estilos personalizados para elementos como `<input>`, `<select>`, `<textarea>`, y más. Al aplicar clases como `.form-control` o `.form-select`, se mejora la apariencia y funcionalidad de estos elementos, asegurando una experiencia de usuario coherente y moderna. [Bootstrap](#)

#### 2. Grupos de Entrada (Input Groups)

Los grupos de entrada permiten combinar campos de formulario con elementos adicionales, como botones o textos, utilizando la clase `.input-group`. Esto es útil para crear componentes como campos de búsqueda con botones integrados.

#### 3. Etiquetas Flotantes (Floating Labels)

Con las etiquetas flotantes, puedes colocar las etiquetas dentro de los campos de entrada, las cuales se desplazan hacia arriba cuando el usuario comienza a escribir. Esto se logra utilizando la clase `.form-floating`.

#### 4. Diseño de Formularios

Bootstrap facilita la creación de diferentes diseños de formularios: [Bootstrap](#)

- **Formularios en línea:** Utilizando la clase `.row` y clases de columna para alinear elementos horizontalmente.
- **Formularios horizontales:** Aplicando clases como `.row` y `.col-form-label` para alinear etiquetas y campos en una misma línea.
- **Formularios con cuadrícula personalizada:** Combinando el sistema de cuadrícula de Bootstrap con clases de formulario para diseños más complejos.

#### 5. Validación de Formularios

Bootstrap integra estilos para la validación de formularios, permitiendo mostrar mensajes de error o éxito. Puedes utilizar la validación nativa de HTML5 o implementar validaciones personalizadas con JavaScript. [Bootstrap](#)

## Componentes (Components)

Bootstrap 5.3 incluye una variedad de componentes reutilizables que facilitan la construcción de interfaces de usuario consistentes y responsivas:

### 1. Botones (Buttons)

Los botones se estilizan con la clase `.btn` y se pueden personalizar con variantes como `.btn-primary`, `.btn-secondary`, entre otras. También es posible ajustar su tamaño y estado (activo, deshabilitado, etc.).

### 2. Tarjetas (Cards)

Las tarjetas son contenedores flexibles que pueden incluir títulos, texto, imágenes, enlaces y más. Se construyen utilizando la clase `.card` y sus elementos internos, como `.card-body`, `.card-title`, y `.card-text`. [Bootstrap](#)

### 3. Listas de Grupo (List Groups)

Las listas de grupo permiten mostrar una serie de contenidos relacionados en una lista con estilo. Se crean con la clase `.list-group` y sus elementos `.list-group-item`. [Bootstrap](#)

### 4. Alertas (Alerts)

Las alertas proporcionan mensajes de retroalimentación para acciones del usuario. Se implementan con la clase `.alert` y variantes como `.alert-success`, `.alert-danger`, etc.

### 5. Modales (Modals)

Los modales son ventanas emergentes que se superponen al contenido principal. Se construyen con la clase `.modal` y se controlan mediante JavaScript para mostrar u ocultar según sea necesario.

### 6. Menús Desplegables (Dropdowns)

Los menús desplegables permiten mostrar una lista de opciones al hacer clic en un elemento. Se crean con la clase `.dropdown` y se personalizan con variantes y opciones de alineación.

### 7. Sistema de Clases Base y Modificadores

Bootstrap utiliza una nomenclatura de clases base y modificadoras para sus componentes. Por ejemplo, `.btn` es la clase base para botones, mientras que `.btn-primary` es una variante modificadora. Esto permite una personalización y extensión más sencilla de los estilos.

## Principales Componentes

A continuación, se describen algunos de los componentes más utilizados en Bootstrap 5.3:

- **Botones (.btn):** Permiten crear botones con diferentes estilos y tamaños. Se pueden combinar con clases como `.btn-primary`, `.btn-lg`, `.btn-block`, entre otras.
- **Tarjetas (.card):** Contenedores flexibles que pueden incluir encabezados, pies de página, imágenes y contenido variado. Son ideales para mostrar información agrupada de manera atractiva.
- **Listas de grupo (.list-group):** Permiten mostrar una serie de elementos en una lista con estilos consistentes. Se pueden utilizar para menús, listas de tareas, entre otros.
- **Alertas (.alert):** Muestran mensajes de retroalimentación al usuario, como confirmaciones o advertencias. Se pueden personalizar con clases como `.alert-success`, `.alert-danger`, etc.
- **Modales (.modal):** Ventanas emergentes que se superponen al contenido principal, útiles para mostrar información adicional o formularios sin abandonar la página actual.
- **Menús desplegables (.dropdown):** Permiten mostrar una lista de opciones al hacer clic en un elemento, como un botón o enlace.
- **Acordeones (.accordion):** Componentes que permiten expandir y contraer secciones de contenido, útiles para mostrar información de manera compacta.
- **Carruseles (.carousel):** Permiten mostrar una serie de imágenes o contenido en un formato de presentación deslizante.
- **Barras de navegación (.navbar):** Componentes de navegación que se adaptan a diferentes tamaños de pantalla y pueden incluir enlaces, formularios y otros elementos interactivos.
- **Paginación (.pagination):** Proporcionan controles para navegar entre páginas de contenido, como listas de artículos o resultados de búsqueda.

## Personalización con Sass

Bootstrap 5.3 está construido con Sass, lo que permite una personalización avanzada de los componentes: [bootstrap.p2hp.com+2GitHub+2Bootstrap+2](#)

- **Variables Sass:** Se pueden modificar variables predefinidas para cambiar aspectos como colores, tamaños y espacios.
- **Mixins y funciones:** Facilitan la creación de estilos personalizados reutilizables.

**Mapas de Sass:** Permiten generar variantes de componentes de manera eficiente mediante bucles @each.

## HELPERS:

### Helpers en Bootstrap 5.3

Los *helpers* en Bootstrap 5.3 son clases utilitarias diseñadas para facilitar tareas comunes de diseño y comportamiento en tus proyectos web. Estas clases permiten aplicar estilos específicos de manera rápida y eficiente, sin necesidad de escribir CSS personalizado.

### 1. Stacks (.vstack y .hstack)

Introducidos en Bootstrap 5.1, los *stacks* son atajos que simplifican la creación de diseños utilizando Flexbox: [Bootstrap](#)

- .vstack: Apila elementos verticalmente. [Bootstrap+6Bootstrap+6Bootstrap+6](#)
- .hstack: Coloca elementos horizontalmente. [Bootstrap Blog+5Bootstrap+5Bootstrap+5](#)

Ambas clases pueden combinarse con las utilidades de espaciado, como .gap-\*, para controlar el espacio entre los elementos. [Bootstrap](#)

### 2. Color y Fondo

Estas clases combinan las utilidades de texto (.text-\*) y fondo (.bg-\*) para establecer combinaciones de colores con contraste adecuado. Utilizan la función color-contrast() de Sass para garantizar la legibilidad del texto sobre diferentes fondos. [Bootstrap+1Bootstrap+1](#)

### 3. Posicionamiento

Bootstrap ofrece clases para posicionar elementos de manera fija o adhesiva:

- .fixed-top y .fixed-bottom: Fijan un elemento en la parte superior o inferior de la ventana.

- `.sticky-top` y `.sticky-bottom`: Hacen que un elemento se adhiera al borde correspondiente al hacer scroll.

También existen variantes responsivas como `.sticky-sm-top` para aplicar estos comportamientos en diferentes tamaños de pantalla. [Bootstrap](#)

#### 4. Ratios

Las clases `.ratio` permiten mantener una relación de aspecto específica para elementos como videos o imágenes. Las relaciones predefinidas incluyen:

- `.ratio-1x1`
- `.ratio-4x3`
- `.ratio-16x9`
- `.ratio-21x9`

También es posible definir relaciones personalizadas utilizando variables CSS. [Bootstrap+3Bootstrap+3Bootstrap+3](#)

#### 5. API de Utilidades Personalizadas

Bootstrap 5.3 introduce una API que permite crear utilidades personalizadas mediante Sass. Algunas características incluyen:

- **Variables CSS:** Genera variables locales en lugar de reglas tradicionales de propiedad-valor. [Bootstrap+1Bootstrap+1](#)
- **Clases Personalizadas:** Permite cambiar el prefijo de las clases generadas.
- **Estados:** Genera variantes para pseudo-clases como `:hover` o `:focus`. [Bootstrap](#)

Esta flexibilidad facilita la creación de estilos reutilizables y adaptados a las necesidades específicas de tu proyecto.

## UTILITIES:

### Utilidades en Bootstrap 5.3

Bootstrap 5.3 ofrece una amplia gama de clases utilitarias que permiten aplicar estilos específicos de manera rápida y eficiente, sin necesidad de escribir CSS personalizado. Estas utilidades están diseñadas para ser responsivas y facilitar el desarrollo de interfaces modernas y adaptables.

### Espaciado: Márgenes y Relleno

Las clases de espaciado permiten controlar los márgenes (m-) y el relleno (p-) de los elementos. Se utilizan combinaciones como:

- .m-3: Aplica un margen de 1rem en todos los lados. [elchininet.github.io+1Bootstrap+1](https://elchininet.github.io/1Bootstrap+1)
- .pt-2: Aplica un relleno superior de 0.5rem.

Estas clases también admiten variantes responsivas, como .mb-md-4, que aplica un margen inferior de 1.5rem a partir del punto de interrupción md. [elchininet.github.io](https://elchininet.github.io)

### Posicionamiento y Visualización

Bootstrap proporciona clases para controlar la propiedad display y la visibilidad de los elementos: [Bootstrap+3getbootstrap.cn+3elchininet.github.io+3](https://Bootstrap+3getbootstrap.cn+3elchininet.github.io+3)

- .d-none: Oculta un elemento.
- .d-flex: Convierte un elemento en un contenedor flex. [elchininet.github.io+2getbootstrap.cn+2Bootstrap+2](https://elchininet.github.io+2getbootstrap.cn+2Bootstrap+2)
- .visible / .invisible: Controlan la visibilidad sin afectar el flujo del documento. [Bootstrap](https://Bootstrap)

Estas clases también tienen variantes responsivas, como .d-sm-block, que muestra el elemento como bloque a partir del punto de interrupción sm.

### Colores y Fondos

Las utilidades de color permiten aplicar estilos de texto y fondo:

- .text-primary: Aplica el color primario al texto. [AlmaBetter+2Bootstrap+2Bootstrap+2](https://AlmaBetter+2Bootstrap+2Bootstrap+2)
- .bg-success: Aplica un fondo de color verde indicando éxito.

Estas clases ayudan a mantener una paleta de colores coherente en toda la aplicación.

## Tipografía

Bootstrap incluye clases para modificar la apariencia del texto: [getbootstrap.cn](https://getbootstrap.cn)

- `.text-center`: Centra el texto. [Bootstrap+6elchininet.github.io+6Bastaki Software Solutions L.L.C-FZ+6](https://getbootstrap.cn/docs/4.0/css/text/#text-align)
- `.fw-bold`: Aplica negrita al texto. [Bootstrap+9Bootstrap+9getbootstrap.cn+9](https://getbootstrap.cn/docs/4.0/css/text/#font-weight)
- `.fst-italic`: Aplica cursiva al texto.
- `.text-uppercase`: Convierte el texto a mayúsculas.

Estas utilidades facilitan la personalización de la tipografía sin necesidad de escribir CSS adicional.

## Flexbox

Bootstrap aprovecha Flexbox para crear diseños flexibles y responsivos. Algunas clases útiles incluyen:

- `.d-flex`: Establece un contenedor flex.
- `.justify-content-center`: Centra los elementos horizontalmente.
- `.align-items-start`: Alinea los elementos al inicio verticalmente.
- `.flex-column`: Organiza los elementos en una columna.

Estas clases permiten construir diseños complejos de manera sencilla.

## API de Utilidades Personalizadas

Bootstrap 5.3 introduce una API basada en Sass que permite crear utilidades personalizadas:

- **Variables CSS**: Se pueden generar variables locales en lugar de reglas tradicionales de propiedad-valor.
- **Clases Personalizadas**: Es posible cambiar el prefijo de las clases generadas.

**Estados**: Se pueden generar variantes para pseudo-clases como `:hover` o `:focus`.

## 6. Consultas verificadas en el código base:

### Consulta 1:

```
const [films] = await db.query<Film[]>(`
  SELECT film.film_id, film.title, film.description, film.release_year, film.language_id,
         film.rental_duration, film.rental_rate, film.length, film.replacement_cost,
         film.rating, language.name as language_name
  FROM film
  JOIN language ON film.language_id = language.language_id
  ${conditions.length ? 'WHERE ' + conditions.join(' AND ') : ''}
  LIMIT ? OFFSET ?
`, [...params, FILMS_PER_PAGE, offset]);
```

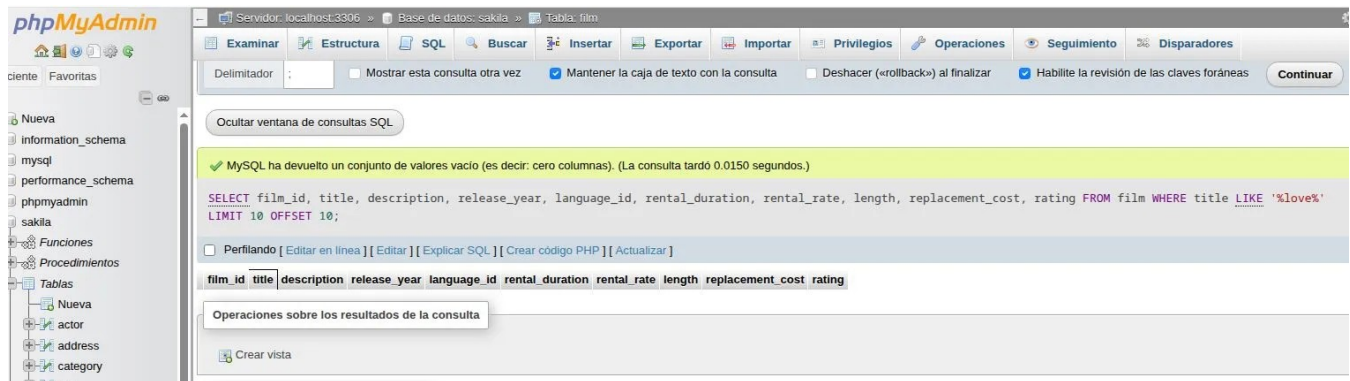
Este es un trozo típico de una plantilla SQL embebida en TypeScript, donde se construye dinámicamente la cláusula **WHERE** y se usan parámetros (?) para paginación.

```
const [films] = await db.query<Film[]>(
  SELECT film.film_id, film.title, film.description,
  film.release_year, film.language_id,
         film.rental_duration, film.rental_rate, film.length,
  film.replacement_cost,
         film.rating, language.name as language_name
  FROM film
  JOIN language ON film.language_id = language.language_id
  ${conditions.length ? 'WHERE ' + conditions.join(' AND ') : ''}
  LIMIT ? OFFSET ?
, [...params, FILMS_PER_PAGE, offset]);
```

Si asumimos que `${conditions}` tiene condiciones como `film.rating = 'PG'` y `film.length > 90`, y que los valores de paginación son, por ejemplo, `LIMIT 10 OFFSET 0`, la consulta estaría lista para probar en lenguaje SQL.



## Preparada para probar en php/myadmin:



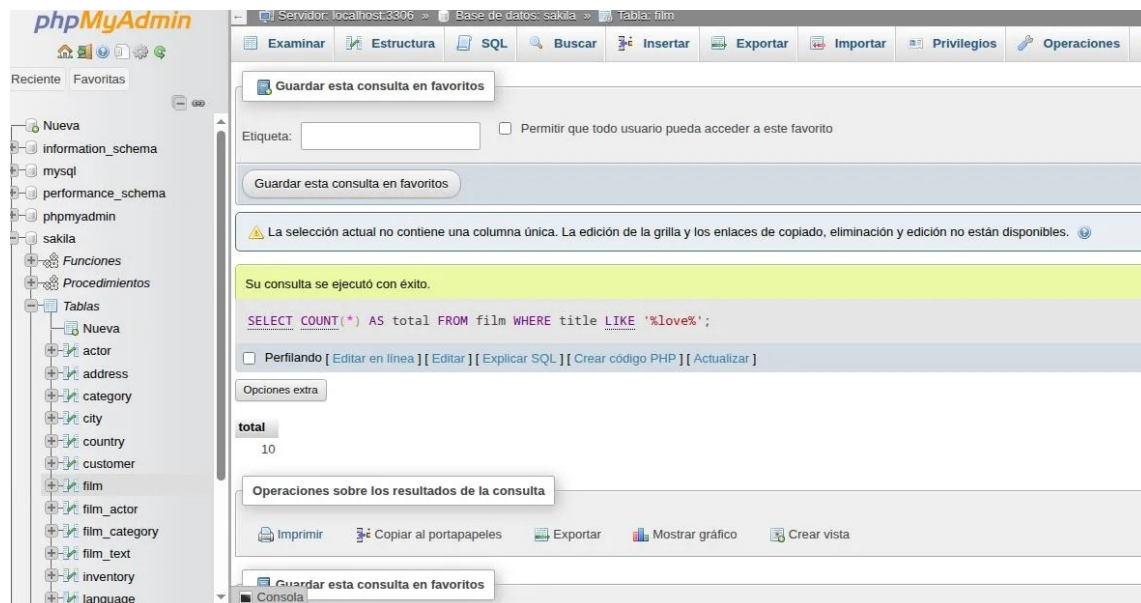
The screenshot shows the phpMyAdmin interface with the 'film' table selected in the 'sakila' database. The SQL query executed is: `SELECT film_id, title, description, release_year, language_id, rental_duration, rental_rate, length, replacement_cost, rating FROM film WHERE title LIKE '%love%' LIMIT 10 OFFSET 10;`. The result is a message: 'MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0.0150 segundos.)'. Below the message, there are links for 'Perfilando', 'Editar en línea', 'Editar', 'Explicar SQL', 'Crear código PHP', and 'Actualizar'. A table header is visible: 

film_id	title	description	release_year	language_id	rental_duration	rental_rate	length	replacement_cost	rating
---------	-------	-------------	--------------	-------------	-----------------	-------------	--------	------------------	--------

. Below the header, there is a section for 'Operaciones sobre los resultados de la consulta' with a 'Crear vista' button.

SELECT

film.film\_id,  
film.title,



The screenshot shows the phpMyAdmin interface with the 'film' table selected in the 'sakila' database. The SQL query executed is: `SELECT COUNT(*) AS total FROM film WHERE title LIKE '%love%';`. The result is a message: 'Su consulta se ejecutó con éxito.' Below the message, there are links for 'Perfilando', 'Editar en línea', 'Editar', 'Explicar SQL', 'Crear código PHP', and 'Actualizar'. A table header is visible: 

total
10

. Below the header, there is a section for 'Operaciones sobre los resultados de la consulta' with buttons for 'Imprimir', 'Copiar al portapapeles', 'Exportar', 'Mostrar gráfico', and 'Crear vista'. At the bottom, there is a 'Consola' section.

film.description,  
film.release\_year,  
film.language\_id,  
film.rental\_duration,  
film.rental\_rate,  
film.length,  
film.replacement\_cost,  
film.rating,  
language.name AS language\_name

FROM film

JOIN language ON film.language\_id = language.language\_id

```
WHERE film.rating = 'PG' AND film.length > 90  
LIMIT 10 OFFSET 0;
```

Qué hace el **JOIN** en las consultas:

El JOIN se utiliza para **combinar filas de dos o más tablas** en una consulta, basándose en una relación entre columnas que tienen en común (normalmente llaves primarias y foráneas).

Para qué se usa:

Se usa para obtener información que está **repartida entre varias tablas**. En bases de datos relacionales, es buena práctica dividir los datos en múltiples tablas normalizadas. Entonces, cuando quieres obtener una visión completa, necesitas unirlos.

Qué hace el **LIMIT** y **OFFSET**:

**LIMIT**: Se usa para **restringir el número de filas** que devuelve una consulta. Usado con OFFSET salta los x primeros resultados (por el OFFSET) y muestra los x siguientes primeros resultados (según los que le pidas con LIMIT).

¿Para qué se usa?

- Para **ver solo una parte de los resultados**, por ejemplo, los primeros 10 registros.
- Para **probar consultas** sin recuperar miles de filas.
- En paginación de resultados (como cuando ves los primeros 10, luego los siguientes 10, etc.).

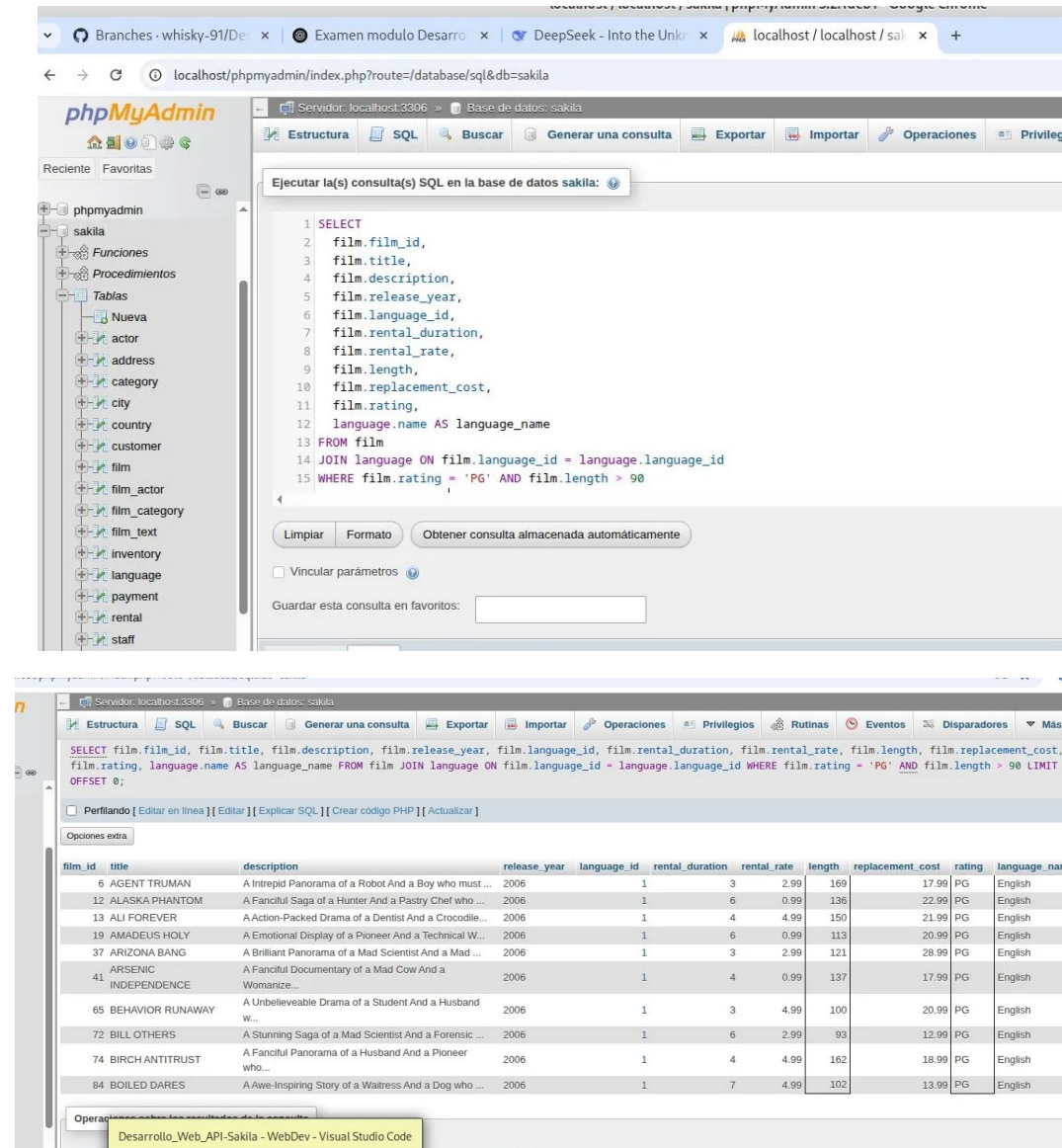
**OFFSET**: Se usa para **saltar un número específico de filas** antes de comenzar a devolver resultados. Se usa **junto con LIMIT** para paginar o controlar desde dónde empiezas a ver los datos.

¿Para qué sirve?

- Para **paginación de resultados** (por ejemplo, mostrar la página 2, 3, etc.).
- Para **omitir los primeros registros** de una consulta.
- Muy útil cuando tienes grandes volúmenes de datos y quieres mostrar fragmentos (como en una web con muchos productos o usuarios).

**WHERE**: Filtra las películas que cumplen: rating = 'PG' (clasificación PG) length > 90 (duración mayor a 90 minutos)

## Prueba en Sakila directamente mediante phpMyAdmin:



The screenshot displays the phpMyAdmin web interface for the 'sakila' database. The left sidebar shows the database structure with tables like actor, address, category, city, country, customer, film, film\_actor, film\_category, film\_text, inventory, language, payment, rental, and staff. The main area shows a SQL query being executed:

```
1 SELECT
2   film.film_id,
3   film.title,
4   film.description,
5   film.release_year,
6   film.language_id,
7   film.rental_duration,
8   film.rental_rate,
9   film.length,
10  film.replacement_cost,
11  film.rating,
12  language.name AS language_name
13 FROM film
14 JOIN language ON film.language_id = language.language_id
15 WHERE film.rating = 'PG' AND film.length > 90
```

Below the query, there are buttons for 'Limpiar', 'Formato', and 'Obtener consulta almacenada automáticamente'. A checkbox for 'Vincular parámetros' is also present. A text box for 'Guardar esta consulta en favoritos:' is visible.

The results of the query are displayed in a table with the following columns: film\_id, title, description, release\_year, language\_id, rental\_duration, rental\_rate, length, replacement\_cost, rating, and language\_name. The table contains 10 rows of data.

film_id	title	description	release_year	language_id	rental_duration	rental_rate	length	replacement_cost	rating	language_name
6	AGENT TRUMAN	A Intrepid Panorama of a Robot And a Boy who must ...	2006	1	3	2.99	169	17.99	PG	English
12	ALASKA PHANTOM	A Fanciful Saga of a Hunter And a Pastry Chef who ...	2006	1	6	0.99	136	22.99	PG	English
13	ALI FOREVER	A Action-Packed Drama of a Dentist And a Crocodile...	2006	1	4	4.99	150	21.99	PG	English
19	AMADEUS HOLY	A Emotional Display of a Pioneer And a Technical W...	2006	1	6	0.99	113	20.99	PG	English
37	ARIZONA BANG	A Brilliant Panorama of a Mad Scientist And a Mad ...	2006	1	3	2.99	121	28.99	PG	English
41	ARSENIC INDEPENDENCE	A Fanciful Documentary of a Mad Cow And a Womanize...	2006	1	4	0.99	137	17.99	PG	English
65	BEHAVIOR RUNAWAY	A Unbelievable Drama of a Student And a Husband w...	2006	1	3	4.99	100	20.99	PG	English
72	BILL OTHERS	A Stunning Saga of a Mad Scientist And a Forensic ...	2006	1	6	2.99	93	12.99	PG	English
74	BIRCH ANTITRUST	A Fanciful Panorama of a Husband And a Pioneer who...	2006	1	4	4.99	162	18.99	PG	English
84	BOILED DARES	A Awe-Inspiring Story of a Waitress And a Dog who ...	2006	1	7	4.99	102	13.99	PG	English

## Consulta 2:

```
import express from 'express';
import { createPool } from 'mysql2/promise';
import cors from 'cors';

const app = express();

// Middleware
app.use(cors());
app.use(express.json());

// Conexión base de datos (Sakila)
export const db = createPool({
  host: 'localhost',
  user: 'root',
  password: '', // Cambiar según tu config
  database: 'sakila',
  waitForConnections: true,
  connectionLimit: 10,
  queueLimit: 0
});

// Ruta de prueba
app.get('/api/films', async (_req, res) => {
  try {
    const [rows] = await db.query('SELECT * FROM film LIMIT 10');
    res.json(rows);
  } catch (error) {
    res.status(500).json({ message: 'Error al consultar la base de datos', error });
  }
});

export default app;
```

esta es la consulta

### Preparada para probar en phpmyadmin:

La consulta embebida en este caso ya está preparada para probar en lenguaje SQL (ya que no hay ninguna parte que esté construida dinamicamente):

```
SELECT *
FROM film
LIMIT 10;
```

## Prueba en Sakila directamente mediante phpMyAdmin:

st/phpmyadmin/index.php?route=/database/sql&db=sakila

Examinar Estructura SQL Buscar Insertar Exportar Importar Privilegios Operaciones Seguimiento Disparar

Ocultar ventana de consultas SQL

✓ Mostrando filas 0 - 9 (total de 10, La consulta tardó 0.0021 segundos.)

```
SELECT * FROM film LIMIT 10;
```

Perfilando [ Editar en línea ] [ Editar ] [ Explicar SQL ] [ Crear código PHP ] [ Actualizar ]

Opciones extra

	film_id	title	description	release_year	language_id	original_language_id	rental_duration	rental_rate	length	replacem
<input type="checkbox"/> Editar Copiar Borrar	1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who...	2006	1	NULL	6	0.99	86	
<input type="checkbox"/> Editar Copiar Borrar	2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrator	2006	1	NULL	3	4.99	48	
<input type="checkbox"/> Editar Copiar Borrar	3	ADAPTATION HO			1	NULL	7	2.99	50	

consultas-bd-embedidas-en-el-codigo

Consola

### Consulta 3:

```
import { Request, Response } from 'express';
import { db } from '../app';
import { Film } from '../models/Film';

export const getFilms = async (_req: Request, res: Response) => {
  try {
    const [rows] = await db.query<Film[]>('SELECT film_id, title, description, release_year, language_id,
    rental_duration, rental_rate, length, replacement_cost, rating FROM film LIMIT 20');
    res.json(rows);
  } catch (error) {
    console.error('Error al obtener films:', error);
    res.status(500).json({ message: 'Error al obtener films', error });
  }
};
```

esta es la consulta

### Preparada para probar en phpmyadmin:

La consulta embebida en este caso ya está preparada para probar en lenguaje SQL (ya que no hay ninguna parte que esté construida dinámicamente):

```
SELECT film_id, title, description, release_year, language_id, rental_duration, rental_rate, length,
replacement_cost, rating
FROM film
LIMIT 20;
```

### Prueba en Sakila directamente mediante phpMyAdmin:

Ocultar ventana de consultas SQL

Mostrando filas 0 - 19 (total de 20, La consulta tardó 0.0003 segundos.)

```
SELECT film_id, title, description, release_year, language_id, rental_duration, rental_rate, length, replacement_cost, rating FROM film LIMIT 20;
```

☐ Perfilando [\[ Editar en línea \]](#) [\[ Editar \]](#) [\[ Explicar SQL \]](#) [\[ Crear código PHP \]](#) [\[ Actualizar \]](#)

Opciones extra

			film_id	title	description	release_year	language_id	rental_duration	rental_rate	length	replacement_cost	rating
<input type="checkbox"/>	<a href="#">Editar</a>	<a href="#">Copiar</a>	<a href="#">Borrar</a>	1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who...	2006	1	6	0.99	86	20.99 PG
<input type="checkbox"/>	<a href="#">Editar</a>	<a href="#">Copiar</a>	<a href="#">Borrar</a>	2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrator A...	2006	1	3	4.99	48	12.99 G
<input type="checkbox"/>	<a href="#">Editar</a>	<a href="#">Copiar</a>	<a href="#">Borrar</a>	3	ADAPTATION HOLES	A Astounding Reflection of a Lumberjack And a Car ...	2006	1	7	2.99	50	18.99 NC-17
<input type="checkbox"/>	<a href="#">Editar</a>	<a href="#">Copiar</a>	<a href="#">Borrar</a>	4	AFFAIR PREJUDICE	A Fanciful Documentary of a Frisbee And a Lumberja...	2006	1	5	2.99	117	26.99 G
<input type="checkbox"/>	<a href="#">Editar</a>	<a href="#">Copiar</a>	<a href="#">Borrar</a>	5	AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef And a De...	2006	1	6	2.99	130	22.99 G
<input type="checkbox"/>	<a href="#">Editar</a>	<a href="#">Copiar</a>	<a href="#">Borrar</a>	6	AGENT TRUMAN	A Intrepid Panorama of a Robot And a Boy who must ...	2006	1	3	2.99	169	17.99 PG
<input type="checkbox"/>	<a href="#">Editar</a>	<a href="#">Copiar</a>	<a href="#">Borrar</a>	7	AIRPLANE SIERRA	A Touching Saga of a Hunter And a Butler who must ...	2006	1	6	4.99	62	28.99 PG-13
<input type="checkbox"/>	<a href="#">Editar</a>	<a href="#">Copiar</a>	<a href="#">Borrar</a>	8	AIRPORT POLLOCK	A Epic Tale of a Moose And a Girl who must Confron...	2006	1	6	4.99	54	15.99 R

#### Consulta 4:

```
const [films] = await db.query<Film[]>(`
  SELECT film.film_id, film.title, film.description, film.release_year, film.language_id,
         film.rental_duration, film.rental_rate, film.length, film.replacement_cost,
         film.rating, language.name as language_name
  FROM film
  JOIN language ON film.language_id = language.language_id
  ${conditions.length ? 'WHERE ' + conditions.join(' AND ') : ''}
  LIMIT ? OFFSET ?
`, [...params, FILMS_PER_PAGE, offset]);
```

consulta

Ese código TypeScript está usando una consulta SQL parametrizada con interpolación condicional en una plantilla de cadena para construir dinámicamente una consulta.

Dado que `conditions`, `FILMS_PER_PAGE` y `offset` son variables, aquí tienes la plantilla SQL completa equivalente, suponiendo que se sustituyen por valores reales.

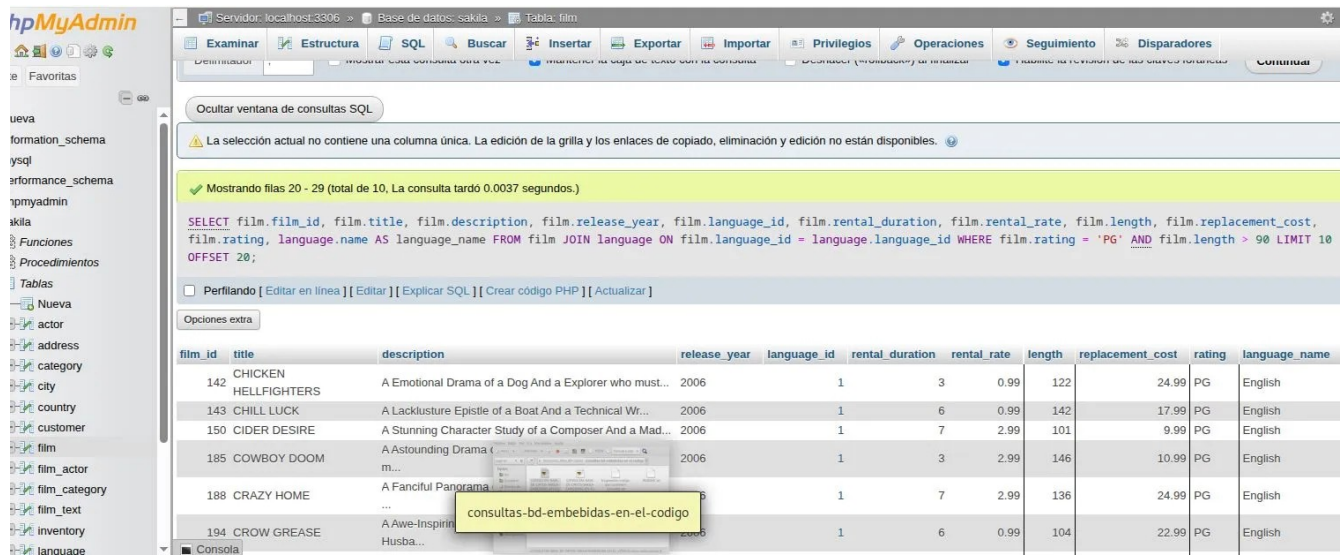
#### Preparada para probar en php/myadmin:

```
SELECT
film.film_id,
film.title,
film.description,
film.release_year,
film.language_id,
film.rental_duration,
film.rental_rate,
film.length,
film.replacement_cost,
film.rating,
language.name AS language_name
FROM film
JOIN language ON film.language_id = language.language_id
WHERE film.rating = 'PG' AND film.length > 90
LIMIT 10 OFFSET 20;
```

**LIMIT 10 OFFSET 20:** Paginación que: Omite las primeras 20 películas (OFFSET 20)  
Muestra solo 10 resultados (LIMIT 10).



## Prueba en Sakila directamente mediante phpMyAdmin:



The screenshot shows the phpMyAdmin interface with the 'film' table selected in the 'sakila' database. The SQL query is displayed in the query editor, and the results are shown in a table below. A yellow callout box points to the query, indicating that database queries are embedded in the code.

La selección actual no contiene una columna única. La edición de la grilla y los enlaces de copiado, eliminación y edición no están disponibles.

Mostrando filas 20 - 29 (total de 10, La consulta tardó 0.0037 segundos.)

```
SELECT film.film_id, film.title, film.description, film.release_year, film.language_id, film.rental_duration, film.rental_rate, film.length, film.replacement_cost, film.rating, language.name AS language_name FROM film JOIN language ON film.language_id = language.language_id WHERE film.rating = 'PG' AND film.length > 90 LIMIT 10 OFFSET 20;
```

Perfilando [ Editar en línea ] [ Editar ] [ Explicar SQL ] [ Crear código PHP ] [ Actualizar ]

Opciones extra

film_id	title	description	release_year	language_id	rental_duration	rental_rate	length	replacement_cost	rating	language_name
142	CHICKEN HELLFIGHTERS	A Emotional Drama of a Dog And a Explorer who must...	2006	1	3	0.99	122	24.99	PG	English
143	CHILL LUCK	A Lacklustre Epistle of a Boat And a Technical Wr...	2006	1	6	0.99	142	17.99	PG	English
150	CIDER DESIRE	A Stunning Character Study of a Composer And a Mad...	2006	1	7	2.99	101	9.99	PG	English
185	COWBOY DOOM	A Astounding Drama (m...	2006	1	3	2.99	146	10.99	PG	English
188	CRAZY HOME	A Fanciful Panorama...		1	7	2.99	136	24.99	PG	English
194	CROW GREASE	A Awe-Inspiring Husba...	2006	1	6	0.99	104	22.99	PG	English

consultas-bd-embebidas-en-el-codigo



## Consulta 5:

```
export const getFilms = async (req: Request, res: Response) => {
  try {
    const page = parseInt(req.query.page as string) || 1;
    const search = (req.query.search as string) || '';
    const year = parseInt(req.query.year as string) || null;

    const offset = (page - 1) * FILMS_PER_PAGE;

    let baseSql = 'FROM film';
    const conditions: string[] = [];
    const params: any[] = [];

    if (search) {
      conditions.push('title LIKE ?');
      params.push(`%${search}%`);
    }

    if (year) {
      conditions.push('release_year = ?');
      params.push(year);
    }

    if (conditions.length > 0) {
      baseSql += ' WHERE ' + conditions.join(' AND ');
    }

    const [films] = await db.query<Film[]>(`SELECT film_id, title, description, release_year, language_id,
    rental_duration, rental_rate, length, replacement_cost, rating ${baseSql} LIMIT ? OFFSET ?`, [...params,
    FILMS_PER_PAGE, offset]);

    const [countRows] = await db.query<any[]>(`SELECT COUNT(*) as total ${baseSql}`, params);

    res.json({ films, total: countRows[0].total });
  } catch (error) {
    console.log('voy a pasar un trozo de código (creo que es de un .ts) necesito que me lo pases a una consulta SQL');
    res.status(500).json({ message: 'Error al obtener films', error });
  }
};
```

OJO: DOS CONSULTAS

### Este controlador de TypeScript realiza dos consultas SQL:

1. Una para obtener una lista de películas con paginación y posibles filtros (search, year).
2. Otra para obtener el conteo total de registros que coinciden con esos filtros (sin paginación).

### Ejemplo simulando que ya hemos recibido valores en la petición:

El usuario ha hecho una petición con estos parámetros en la URL:

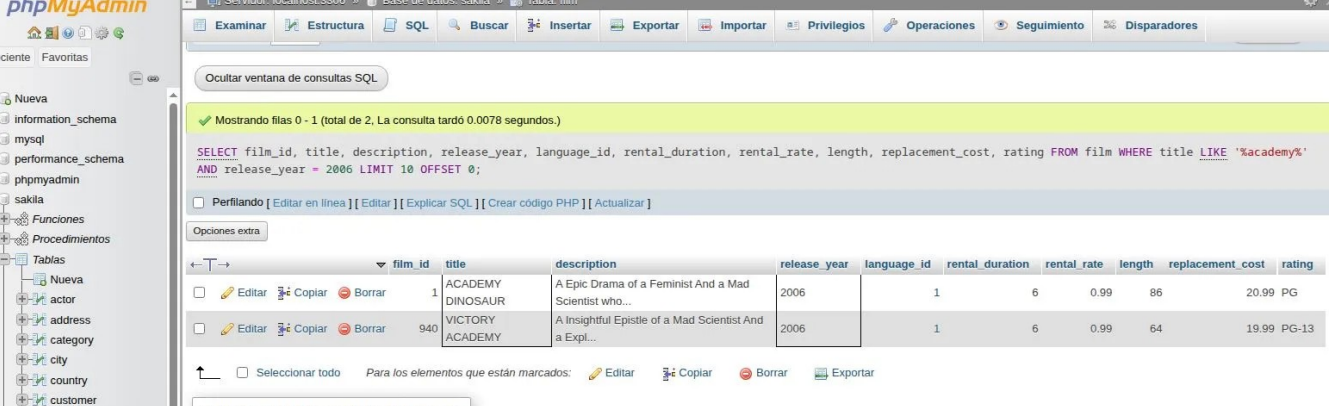
`?page=1&search=academy&year= 2006`

### Simulación de valores recibidos:

```
search = "academy"
year = 2006
FILMS_PER_PAGE = 10
page = 1 → offset = (1 - 1) * 10 = 0
```

### Consulta SQL resultante de la primera parte:

```
SELECT
film_id,
title,
description,
release_year,
language_id,
rental_duration,
rental_rate,
length,
replacement_cost,
rating
FROM film
WHERE title LIKE '%academy%' AND release_year = 2006
LIMIT 10 OFFSET 0;
```



Mostrando filas 0 - 1 (total de 2, La consulta tardó 0.0078 segundos.)

```
SELECT film_id, title, description, release_year, language_id, rental_duration, rental_rate, length, replacement_cost, rating FROM film WHERE title LIKE '%academy%' AND release_year = 2006 LIMIT 10 OFFSET 0;
```

Perfilando [ Editar en línea ] [ Editar ] [ Explicar SQL ] [ Crear código PHP ] [ Actualizar ]

Opciones extra

	film_id	title	description	release_year	language_id	rental_duration	rental_rate	length	replacement_cost	rating
<input type="checkbox"/>	1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who...	2006	1	6	0.99	86	20.99	PG
<input type="checkbox"/>	940	VICTORY ACADEMY	A Insightful Epistle of a Mad Scientist And a Expl...	2006	1	6	0.99	64	19.99	PG-13

Seleccionar todo Para los elementos que están marcados: [ Editar ] [ Copiar ] [ Borrar ] [ Exportar ]

En la segunda habría que sustituir ‘%busqueda%’ por el título de una película que esté en la base de datos.

Aquí tenemos un ejemplo con **busqueda** ya sustituido por ‘dorp’ que es la primera palabra del título de la película: DROP WATERFRONT. (que está en la base de datos).

### Preparada para probar en php/myadmin:

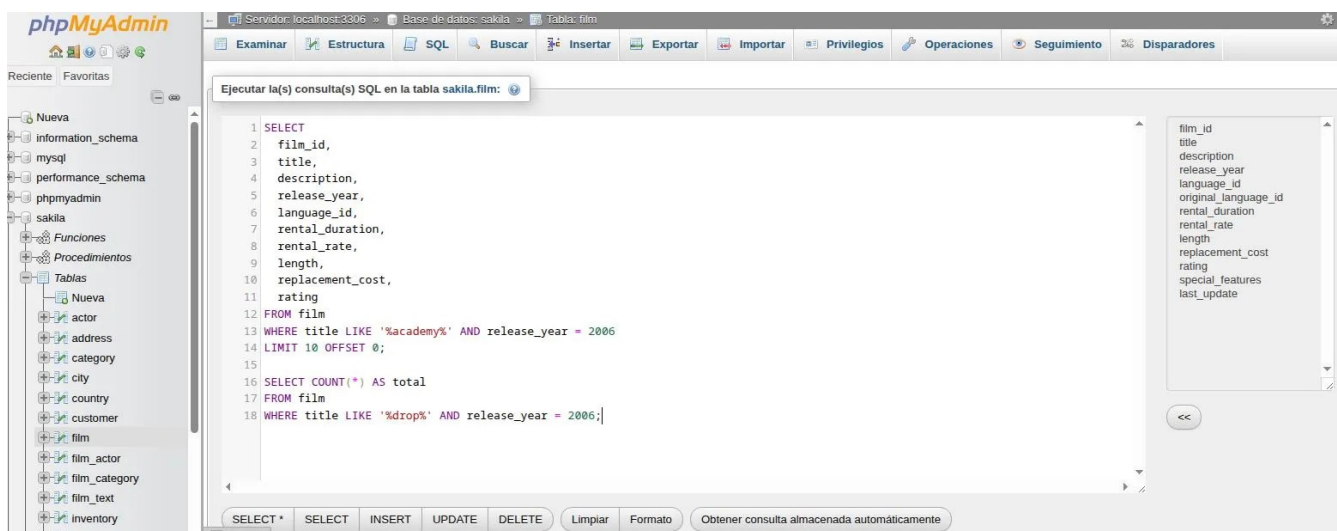
```
SELECT COUNT(*) AS total
FROM film
WHERE title LIKE '%drop%' AND release_year = 2006;
```

### Ambas consultas realizadas a la vez quedarían:

```
SELECT
film_id,
title,
description,
release_year,
language_id,
rental_duration,
rental_rate,
length,
replacement_cost,
rating
FROM film
WHERE title LIKE '%academy%' AND release_year = 2006
LIMIT 10 OFFSET 0;
```

```
SELECT COUNT(*) AS total
FROM film
WHERE title LIKE '%drop%' AND release_year = 2006;
```

### Prueba en Sakila directamente mediante phpMyAdmin:



## Resultado de la primera consulta:

Ocultar ventana de consultas SQL

✓ Mostrando filas 0 - 1 (total de 2, La consulta tardó 0.0138 segundos.)

```
SELECT film_id, title, description, release_year, language_id, rental_duration, rental_rate, length, replacement_cost, rating FROM film WHERE title LIKE '%academy%' AND release_year = 2006 LIMIT 10 OFFSET 0;
```

☐ Perfilando [ Editar en línea ] [ Editar ] [ Explicar SQL ] [ Crear código PHP ] [ Actualizar ]

Opciones extra

	film_id	title	description	release_year	language_id	rental_duration	rental_rate	length	replacement_cost	rating
<input type="checkbox"/> [ Editar ] [ Copiar ] [ Borrar ]	1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who...	2006	1	6	0.99	86	20.99	PG
<input type="checkbox"/> [ Editar ] [ Copiar ] [ Borrar ]	940	VICTORY ACADEMY	A Insightful Epistle of a Mad Scientist And a Expl...	2006	1	6	0.99	64	19.99	PG-13

☐ Seleccionar todo Para los elementos que están marcados: [ Editar ] [ Copiar ] [ Borrar ] [ Exportar ]

Operaciones sobre los resultados de la consulta

[ Imprimir ] [ Copiar al portapapeles ] [ Exportar ] [ Crear vista ]

## Resultado de la segunda consulta:

La selección actual no contiene una columna única. La edición de la grilla y los enlaces de copiado, eliminación y edición no están disponibles.

Su consulta se ejecutó con éxito.

```
SELECT COUNT(*) AS total FROM film WHERE title LIKE '%drop%' AND release_year = 2006;
```

☐ Perfilando [ Editar en línea ] [ Editar ] [ Explicar SQL ] [ Crear código PHP ] [ Actualizar ]

Opciones extra

**total**

3

Operaciones sobre los resultados de la consulta

[ Imprimir ] [ Copiar al portapapeles ] [ Exportar ] [ Mostrar gráfico ] [ Crear vista ]

## Consulta 6:

```
import { Request, Response } from 'express';
import { db } from '../app';
import { Film } from '../models/Film';

const FILMS_PER_PAGE = 10;

export const getFilms = async (req: Request, res: Response) => {
  try {
    const page = parseInt(req.query.page as string) || 1;
    const search = (req.query.search as string) || '';

    const offset = (page - 1) * FILMS_PER_PAGE;

    let baseSql = `FROM film`;
    const params: any[] = [];

    if (search) {
      baseSql += ` WHERE title LIKE ?`;
      params.push(`%${search}%`);
    }

    const [films] = await db.query<Film[]>(`SELECT film_id, title, description, release_year, language_id,
    rental_duration, rental_rate, length, replacement_cost, rating ${baseSql} LIMIT ? OFFSET ?`, [...params,
    FILMS_PER_PAGE, offset]);

    const [countRows] = await db.query<any[]>(`SELECT COUNT(*) as total ${baseSql}`, params);

    res.json({ films, total: countRows[0].total });
  } catch (error) {
    console.error('Error al obtener films:', error);
    res.status(500).json({ message: 'Error al obtener films', error });
  }
};
```

Este endpoint en TypeScript/Express hace dos consultas SQL dinámicas basadas en los parámetros `page` y `search`.

## Consultas preparadas para probar en php/myadmin:

### Primera consulta:

Vamos a suponer que:

```
page = 2
search = 'love'
FILMS_PER_PAGE = 10
offset = (2 - 1) * 10 = 10
```

```

SELECT
    film_id,
    title,
    description,
    release_year,
    language_id,
    rental_duration,
    rental_rate,
    length,
    replacement_cost,
    rating
FROM film
WHERE title LIKE '%love%'
LIMIT 10 OFFSET 10;

```

**NOTA:** Como hay en total 10 resultados en esta búsqueda simulada, si descartamos con OFFSET los 10 primeros el resultado será un conjunto vacío.

### Segunda consulta:

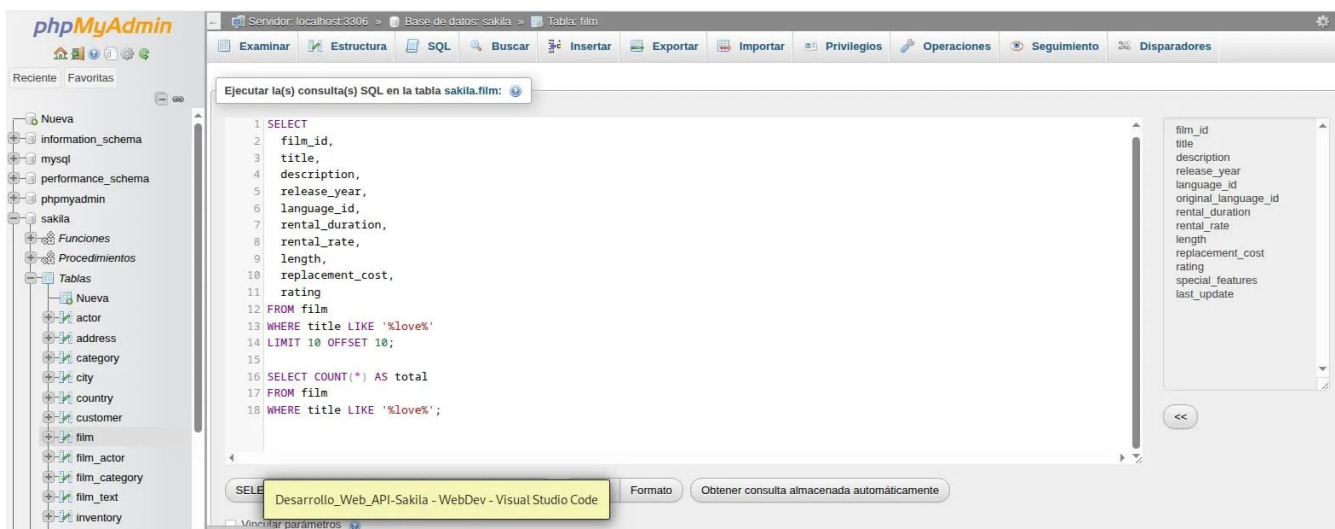
Siguiendo con los datos que hemos simulado para la primera:

```

SELECT COUNT(*) AS total
FROM film
WHERE title LIKE '%love%';

```

### Prueba en Sakila directamente mediante phpMyAdmin:



## Resultado de la primera consulta:

The screenshot shows the phpMyAdmin interface with the 'film' table selected. The SQL query is: `SELECT film_id, title, description, release_year, language_id, rental_duration, rental_rate, length, replacement_cost, rating FROM film WHERE title LIKE '%love%' LIMIT 10 OFFSET 10;`. The result is a message: 'MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0.0150 segundos.)'. Below the message, there are links for 'Perfilando', 'Editar en línea', 'Editar', 'Explicar SQL', 'Crear código PHP', and 'Actualizar'. A table header is visible with columns: film\_id, title, description, release\_year, language\_id, rental\_duration, rental\_rate, length, replacement\_cost, rating. Below the table header, there is a section for 'Operaciones sobre los resultados de la consulta' with a 'Crear vista' button.

## Resultado de la segunda consulta:

The screenshot shows the phpMyAdmin interface with the 'film' table selected. The SQL query is: `SELECT COUNT(*) AS total FROM film WHERE title LIKE '%love%';`. The result is a message: 'Su consulta se ejecutó con éxito.' followed by a table with one row: 'total' with the value '10'. Below the table, there is a section for 'Operaciones sobre los resultados de la consulta' with buttons for 'Imprimir', 'Copiar al portapapeles', 'Exportar', 'Mostrar gráfico', and 'Crear vista'. At the bottom, there is a 'Consola' tab.

## Consula 7:

```
import { Request, Response } from 'express';
import { db } from '../app';
import { Film } from '../models/Film';

const FILMS_PER_PAGE = 10;

export const getFilms = async (req: Request, res: Response) => {
  try {
    const page = parseInt(req.query.page as string) || 1;
    const search = (req.query.search as string) || '';

    const offset = (page - 1) * FILMS_PER_PAGE;

    let sql = `SELECT film_id, title, description, release_year, language_id, rental_duration, rental_rate,
length, replacement cost, rating FROM film`;
    const params: any[] = [];

    if (search) {
      sql += ` WHERE title LIKE ?`;
      params.push(`%${search}%`);
    }

    sql += ` LIMIT ? OFFSET ?`;
    params.push(FILMS_PER_PAGE, offset);

    const [rows] = await db.query<Film[]>(sql, params);

    res.json(rows);
  } catch (error) {
    console.error('Error al obtener films:', error);
    res.status(500).json({ message: 'Error al obtener films', error });
  }
};
```

Consulta

Este controlador TypeScript de Express construye dinámicamente una consulta SQL para obtener películas de la tabla **film**, aplicando búsqueda y paginación si es necesario.

### Consulta preparada para probar en php/myadmin:

Para convertir esto a SQL puro, vamos a simular estos valores:

```
page = 1
search = "alien"
FILMS_PER_PAGE = 10
offset = (1 - 1) * 10 = 0
```



```
SELECT
film_id,
title,
description,
release_year,
language_id,
rental_duration,
rental_rate,
length,
replacement_cost,
rating
FROM film
WHERE title LIKE '%alien%'
LIMIT 10 OFFSET 0;
```

**NOTA:** Si el parámetro `search` no se proporciona (es decir, está vacío), la consulta resultante sería simplemente:

```
SELECT
film_id,
title,
description,
release_year,
language_id,
rental_duration,
rental_rate,
length,
replacement_cost,
rating
FROM film
LIMIT 10 OFFSET 0;
```

## Prueba en Sakila directamente mediante phpMyAdmin:

phpMyAdmin interface showing the SQL query editor for the 'film' table in the 'sakila' database. The query is:

```
1 SELECT
2   film_id,
3   title,
4   description,
5   release_year,
6   language_id,
7   rental_duration,
8   rental_rate,
9   length,
10  replacement_cost,
11  rating
12 FROM film
13 WHERE title LIKE '%alien%'
14 LIMIT 10 OFFSET 0;
```

The right sidebar shows the table structure for 'film' with columns: film\_id, title, description, release\_year, language\_id, original\_language\_id, rental\_duration, rental\_rate, length, replacement\_cost, rating, special\_features, last\_update.

phpMyAdmin interface showing the results of the SQL query. The results table has 11 columns: film\_id, title, description, release\_year, language\_id, rental\_duration, rental\_rate, length, replacement\_cost, rating. The results show 3 rows (total of 3, La consulta tardó 0.0077 segundos.):

film_id	title	description	release_year	language_id	rental_duration	rental_rate	length	replacement_cost	rating
15	ALIEN CENTER	A Brilliant Drama of a Cat And a Mad Scientist who...	2006	1	5	2.99	46	10.99	NC-17
223	DESIRE ALIEN	A Fast-Paced Tale of a Dog And a Forensic Psycholo...	2006	1	7	2.99	76	24.99	NC-17
418	HOBBIT ALIEN	A Emotional Drama of a Husband And a Girl who must...	2006	1	5	0.99	157	27.99	PG-13

Operations sobre los resultados de la consulta: Seleccionar todo, Para los elementos que están marcados: Editar, Copiar, Borrar, Exportar.

## Consulta 8:

```
import { Request, Response } from 'express';
import { db } from '../app';
import { Film } from '../models/Film';

const FILMS_PER_PAGE = 10;

export const getFilms = async (req: Request, res: Response) => {
  try {
    const page = parseInt(req.query.page as string) || 1;
    const search = (req.query.search as string) || '';

    const offset = (page - 1) * FILMS_PER_PAGE;

    let sql = `SELECT film_id, title, description, release_year, language_id, rental_duration, rental_rate,
length, replacement_cost, rating FROM film`;
    const params: any[] = [];

    if (search) {
      sql += ` WHERE title LIKE ?`;
      params.push(`%${search}%`);
    }

    sql += ` LIMIT ? OFFSET ?`;
    params.push(FILMS_PER_PAGE, offset);

    const [rows] = await db.query<Film[]>(sql, params);

    res.json(rows);
  } catch (error) {
    console.error('Error al obtener films:', error);
    res.status(500).json({ message: 'Error al obtener films', error });
  }
};
```

Consulta

En este código, los valores que se reciben dinámicamente vienen de los parámetros de la URL (query string) del objeto `req`. Específicamente:

### Parámetros recibidos del cliente:

```
const page = parseInt(req.query.page as string) || 1;
const search = (req.query.search as string) || '';
```

Estos se extraen desde la URL de una petición, por ejemplo:

GET /api/films?page=2&search=alien

Entonces:

`req.query.page = "2" → page = 2`

`req.query.search = "alien" → search = 'alien'`

Derivados:

$FILMS\_PER\_PAGE = 10$  (constante del servidor)

$offset = (page - 1) * FILMS\_PER\_PAGE = (2 - 1) * 10 = 10$

$params = ['\%alien\%', 10, 10]$  (para la consulta SQL)

**Consulta preparada para probar en php/myadmin:**

SELECT

film\_id,  
title,  
description,  
release\_year,  
language\_id,  
rental\_duration,  
rental\_rate,  
length,  
replacement\_cost,  
rating

FROM film

WHERE title LIKE '%alien%'

LIMIT 10 OFFSET 10;

**Prueba en Sakila directamente mediante phpMyAdmin:**

The screenshot shows the phpMyAdmin interface for the 'sakila' database. The left sidebar lists the database structure, including tables like 'actor', 'address', 'category', 'city', 'country', 'customer', and 'film'. The main panel displays a SQL query: `SELECT film_id, title, description, release_year, language_id, rental_duration, rental_rate, length, replacement_cost, rating FROM film WHERE title LIKE '%alien%' LIMIT 10 OFFSET 10;`. Below the query, there are buttons for 'Perfilando', 'Editar en línea', 'Editar', 'Explicar SQL', 'Crear código PHP', and 'Actualizar'. The results section shows a table with columns: film\_id, title, description, release\_year, language\_id, rental\_duration, rental\_rate, length, replacement\_cost, and rating. The table contains three rows of data:

film_id	title	description	release_year	language_id	rental_duration	rental_rate	length	replacement_cost	rating
15	ALIEN CENTER	A Brilliant Drama of a Cat And a Mad Scientist who...	2006	1	5	2.99	46	10.99	NC-17
223	DESIRE ALIEN	A Fast-Paced Tale of a Dog And a Forensic Psycholo...	2006	1	7	2.99	76	24.99	NC-17
418	HOBBIT ALIEN	A Emotional Drama of a Husband And a Girl who must...	2006	1	5	0.99	157	27.99	PG-13

## Consulta 9:

```
import { Request, Response } from 'express';
import { db } from '../app';
import { Film } from '../models/Film';

export const getFilms = async (_req: Request, res: Response) => {
  try {
    const [rows] = await db.query<Film[]>('SELECT film_id, title, description, release_year, language_id,
    rental_duration, rental_rate, length, replacement_cost, rating FROM film LIMIT 20');
    res.json(rows);
  } catch (error) {
    console.error('Error al obtener films:', error);
    res.status(500).json({ message: 'Error al obtener films', error });
  }
};
```

Consulta

Consulta preparada para probar en php/myadmin:

SELECT

film\_id,

title,

description,

release\_year,

language\_id,

rental\_duration,

rental\_rate,

length,

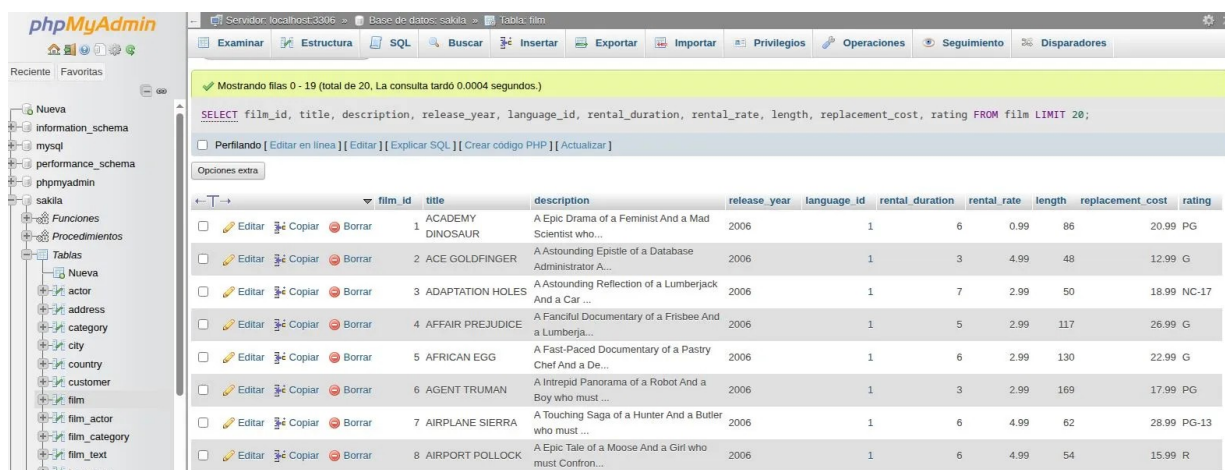
replacement\_cost,

rating

FROM film

LIMIT 20;

Prueba en Sakila directamente mediante phpMyAdmin:



Mostrando filas 0 - 19 (total de 20. La consulta tardó 0.0004 segundos.)

SELECT film\_id, title, description, release\_year, language\_id, rental\_duration, rental\_rate, length, replacement\_cost, rating FROM film LIMIT 20;

Perfilando [ Editar en línea ] [ Editar ] [ Explicar SQL ] [ Crear código PHP ] [ Actualizar ]

Opciones extra

	film_id	title	description	release_year	language_id	rental_duration	rental_rate	length	replacement_cost	rating
<input type="checkbox"/>	1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who...	2006	1	6	0.99	86	20.99	PG
<input type="checkbox"/>	2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrator A...	2006	1	3	4.99	48	12.99	G
<input type="checkbox"/>	3	ADAPTATION HOLES	A Astounding Reflection of a Lumberjack And a Car ...	2006	1	7	2.99	50	18.99	NC-17
<input type="checkbox"/>	4	AFFAIR PREJUDICE	A Fanciful Documentary of a Frisbee And a Lumberja...	2006	1	5	2.99	117	26.99	G
<input type="checkbox"/>	5	AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef And a De...	2006	1	6	2.99	130	22.99	G
<input type="checkbox"/>	6	AGENT TRUMAN	A Intrepid Panorama of a Robot And a Boy who must ...	2006	1	3	2.99	169	17.99	PG
<input type="checkbox"/>	7	AIRPLANE SIERRA	A Touching Saga of a Hunter And a Butler who must ...	2006	1	6	4.99	62	28.99	PG-13
<input type="checkbox"/>	8	AIRPORT POLLOCK	A Epic Tale of a Moose And a Girl who must Confront...	2006	1	6	4.99	54	15.99	R

## Consulta 10:

```
import express from 'express';
import { createPool } from 'mysql2/promise';
import cors from 'cors';

const app = express();

// Middleware
app.use(cors());
app.use(express.json());

// Conexión base de datos (Sakila)
export const db = createPool({
  host: 'localhost',
  user: 'root',
  password: '', // Cambiar según tu config
  database: 'sakila',
  waitForConnections: true,
  connectionLimit: 10,
  queueLimit: 0
});

// Ruta de prueba
app.get('/api/films', async (_req, res) => {
  try {
    const [rows] = await db.query('SELECT * FROM film LIMIT 10');
    res.json(rows);
  } catch (error) {
    res.status(500).json({ message: 'Error al consultar la base de datos', error });
  }
});
```

Consulta

Consulta preparada para probar en php/myadmin:

`SELECT * FROM film LIMIT 10;`

**NOTA:** Idéntica a la consulta 2.

Prueba en Sakila directamente mediante phpMyAdmin:

Mostrando filas 0 - 9 (total de 10, La consulta tardó 0.0030 segundos.)

`SELECT * FROM film LIMIT 10;`

☐ Perfilando [\[ Editar en línea \]](#) [\[ Editar \]](#) [\[ Explicar SQL \]](#) [\[ Crear código PHP \]](#) [\[ Actualizar \]](#)

Opciones extra

	film_id	title	description	release_year	language_id	original_language_id	rental_duration	rental_rate	length	replacement_cost	rating
<input type="checkbox"/> <a href="#">Editar</a> <a href="#">Copiar</a> <a href="#">Borrar</a>	1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who...	2006	1	NULL	6	0.99	86	20.99	PG
<input type="checkbox"/> <a href="#">Editar</a> <a href="#">Copiar</a> <a href="#">Borrar</a>	2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrator A...	2006	1	NULL	3	4.99	48	12.99	G

## 7. Corrección de problemas en la instalación:

Antes de abrir la API es necesario abrir manualmente el servidor y la base de datos (salvo que arranque con el ordenador).

En caso de windows será necesario levantar previamente apache, xampp y MySQL.

Para ello, en un terminal de gitbash (windows) o de linux:

```
mysql -u root -p  
contraseña (en caso de windows dejar en blanco)  
use sakila;
```

**6.1 Instalación en linux:** Para instalar correctamente en linux es necesario también agregar una contraseña (por defecto mariadb) y retomamos:

```
use sakila;
```

Procedemos a irnos en VSCode a la raíz del proyecto, e instalamos nodemon. Para ello tenemos dos formas, la global y por proyecto:

```
global → npm install -g nodemon / sudo npm install -g nodemon  
local → npm install --save-dev nodemon
```

En backend/src/app.ts tenemos el apartado:

```
// Conexión base de datos (Sakila)  
export const db = createPool({  
  host: 'localhost',  
  user: 'root',  
  password: 'mariadb', // Cambiar según tu config  
  database: 'sakila',  
  waitForConnections: true,  
  connectionLimit: 10,  
  queueLimit: 0  
});
```

Aquí tenemos que colocar en “password” “mariadb” si estamos en linux con la contraseña por

defecto, dejarlo vacío en windows si tenemos todo por defecto o en su lugar colocar nuestra contraseña para root de SQL que tengamos asignada.

Una vez terminemos esta parte, guardamos el cambio y transpilamos, sirve una terminal situada en backend con el comando de typescript:

```
tsc app.ts
```

**Finaliza empaquetando (linux/windows):** Una vez seguimos estos pasos, en la consola situada en backend cuando terminemos de transpilar realizamos los comandos para empaquetar la API e iniciar la aplicación:

```
npm run build  
npm run start
```

Tomamos la dirección y el puerto de localhost que nos proporciona la terminal y la colocamos en el navegador para poder ver nuestra API.



## 8. Consulta individual:

### CONSULTAS SQL EN BASE DE DATOS SAKILA

---

#### 1. Listar los actores que han participado en películas con categoría Comedy (Yelena)

SQL:

```
SELECT DISTINCT a.actor_id, a.first_name, a.last_name
-> FROM actor a
-> JOIN film_actor fa ON a.actor_id = fa.actor_id
-> JOIN film f ON fa.film_id = f.film_id
-> JOIN film_category fc ON f.film_id = fc.film_id
-> JOIN category c ON fc.category_id = c.category_id
-> WHERE c.name = 'Comedy';
```

Explicación:

Esta consulta obtiene una lista única de actores que han participado en películas de la categoría "Comedy". Lo hace uniendo las tablas actor, film\_actor, film, film\_category y category, y filtrando por las películas cuya categoría es 'Comedy'. Usa DISTINCT para evitar repetir actores si han trabajado en más de una película de comedia.

#### 2. Clientes que han alquilado más de 20 películas (Miguel)

SQL:

```
SELECT c.customer_id, c.first_name, c.last_name, COUNT(r.rental_id) AS total_alquileres
FROM customer c
JOIN rental r ON c.customer_id = r.customer_id
GROUP BY c.customer_id, c.first_name, c.last_name
HAVING COUNT(r.rental_id) > 20
ORDER BY total_alquileres DESC;
```

Explicación:

- Cuenta cuántas veces ha alquilado cada cliente.
- Usa HAVING (en lugar de WHERE) para filtrar clientes con más de 20 alquileres.

### 3. Películas disponibles en inventario con su categoría (Emilia)

SQL:

```
SELECT f.title, c.name AS category, COUNT(i.inventory_id) AS disponibles
FROM film f
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category c ON fc.category_id = c.category_id
JOIN inventory i ON f.film_id = i.film_id
LEFT JOIN rental r ON i.inventory_id = r.inventory_id AND r.return_date IS NULL
WHERE r.rental_id IS NULL
GROUP BY f.film_id, f.title, c.name
ORDER BY disponibles DESC;
```

Explicación:

- Encuentra copias no alquiladas actualmente (disponibles).
- Agrupa por película y categoría y cuenta las disponibles.

### 4. Ingreso total por cada tienda (Qasem)

SQL:

```
SELECT s.store_id, SUM(p.amount) AS ingresos_totales
FROM store s
JOIN staff st ON s.store_id = st.store_id
JOIN payment p ON st.staff_id = p.staff_id
GROUP BY s.store_id;
```

Explicación:

- Relaciona tienda → empleados → pagos.
- Suma los pagos gestionados por los empleados de cada tienda.

### 5. Películas que nunca han sido alquiladas (Brian)

SQL:

```
SELECT f.film_id, f.title
FROM film f
JOIN inventory i ON f.film_id = i.film_id
LEFT JOIN rental r ON i.inventory_id = r.inventory_id
WHERE r.rental_id IS NULL
GROUP BY f.film_id, f.title;
```

Explicación:

- Busca películas cuyas copias nunca se hayan alquilado (rental\_id es NULL).

## 6. Clientes que viven en Canadá (Victor)

SQL:

```
SELECT c.first_name, c.last_name, c.email
FROM customer c
JOIN address a ON c.address_id = a.address_id
JOIN city ci ON a.city_id = ci.city_id
JOIN country co ON ci.country_id = co.country_id
WHERE co.country = 'Canada';
```

Explicación:

- Navega desde el cliente hasta el país a través de address y city.
- Filtra clientes que viven en ciudades de Canadá.

## 7. Duración media de películas por categoría (Adrián)

SQL:

```
SELECT c.name AS categoria, ROUND(AVG(f.length), 2) AS duracion_media
FROM film f
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category c ON fc.category_id = c.category_id
GROUP BY c.name;
```

Explicación:

- Calcula el promedio de duración de películas agrupadas por categoría.

## 8. Empleados y cuántos pagos han gestionado

SQL:

```
SELECT st.staff_id, st.first_name, st.last_name, COUNT(p.payment_id) AS pagos_realizados
FROM staff st
LEFT JOIN payment p ON st.staff_id = p.staff_id
GROUP BY st.staff_id, st.first_name, st.last_name
ORDER BY pagos_realizados DESC;
```

Explicación:

- Cuenta los pagos procesados por cada empleado.
- LEFT JOIN permite incluir empleados sin pagos.

## 9. Las 10 películas más alquiladas (Kevin)

SQL:

```
SELECT f.film_id, f.title, COUNT(r.rental_id) AS total_alquileres
FROM film f
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r ON i.inventory_id = r.inventory_id
GROUP BY f.film_id, f.title
ORDER BY total_alquileres DESC
LIMIT 10;
```

Explicación:

- Cuenta los alquileres por película considerando todas sus copias.
- Muestra las 10 más rentadas.

## 10. Actores que han trabajado en películas de la categoría 'Horror' (Nacho)

SQL:

```
SELECT DISTINCT a.actor_id, a.first_name, a.last_name
FROM actor a
JOIN film_actor fa ON a.actor_id = fa.actor_id
JOIN film f ON fa.film_id = f.film_id
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category c ON fc.category_id = c.category_id
WHERE c.name = 'Horror';
```

Explicación:

- Une actores con sus películas y categorías.
- Filtra por la categoría 'Horror'.
- DISTINCT evita actores duplicados si han hecho varias películas de esa categoría.

## **11.Actrices con nombre Penelope**

Versión realista en Sakila estándar (sin campo de género):

SQL:

```
SELECT actor_id, first_name, last_name  
FROM actor  
WHERE first_name = 'Penelope';
```

Explicación:

- Busca todos los actores cuyo primer nombre es 'Penelope'.
- La base de datos Sakila no tiene un campo de género, por lo tanto no distingue entre actor y actriz.

## 9. Node y Express:

### ¿Qué es Node.js y Express?

- Node.js es un entorno para ejecutar JavaScript fuera del navegador, por ejemplo, en un servidor.
- Express es un framework para Node.js que simplifica la creación de APIs y servidores web.

### Requisitos previos

#### Debes tener instalado:

- Node.js (<https://nodejs.org/>)
- MySQL (con la base de datos sakila ya importada)
- Un editor de código como VS Code

### 1 Crear el proyecto

#### Abre una terminal y crea una carpeta nueva:

```
sakila-api/  
├──  
├── index.js      # archivo principal  
├── db.js         # configuración de conexión a MySQL  
└── package.json
```

### 3 Conectarse a la base de datos (MySQL)

#### db.js:

```
const mysql = require('mysql2');  
  
const connection = mysql.createConnection({  
  host: 'localhost',  
  user: 'tu_usuario',  
  password: 'tu_contraseña',  
  database: 'sakila'  
});  
  
connection.connect((err) => {  
  if (err) throw err;  
  console.log('Conectado a la base de datos Sakila');  
});
```

```
module.exports = connection;
```

#### **Crear un servidor con Express**

**index.js:**

```
const express = require('express');
const app = express();
const port = 3000;

const db = require('./db');

// Ruta de prueba
app.get('/', (req, res) => {
  res.send('API de Sakila funcionando 🚀');
});

// Ruta para obtener todos los actores
app.get('/actores', (req, res) => {
  db.query('SELECT * FROM actor LIMIT 10', (err, results) => {
    if (err) {
      console.error('Error al obtener actores:', err);
      res.status(500).send('Error en el servidor');
    } else {
      res.json(results);
    }
  });
});

app.listen(port, () => {
  console.log(`Servidor corriendo en http://localhost:${port}`);
});
```

#### **Probar la API**

##### **1. Inicia el servidor:**

```
node index.js
```

##### **2. Entra a tu navegador y abre:**

<http://localhost:3000/> → Verás “API de Sakila funcionando”

http://localhost:3000/actores → Verás una lista de 10 actores

En Express, los **métodos básicos** se usan para manejar las distintas **acciones** que un cliente (como tu navegador o una app) puede hacer sobre un servidor. Estos métodos están basados en los verbos HTTP:

✚ Métodos HTTP básicos en Express

Método	Usado para...	Ejemplo en Express
GET	Leer datos	app.get('/peliculas', ...)
POST	Crear datos nuevos	app.post('/peliculas', ...)
PUT	Actualizar datos	app.put('/peliculas/:id', ...)
DELETE	Borrar datos	app.delete('/peliculas/:id', ...)

## 📦 1. GET

👉 ¿Para qué sirve?

Para obtener información del servidor, como una lista de películas o un actor.

📌 Ejemplo:

```
app.get('/actores', (req, res) => {  
  // Aquí podrías consultar la base de datos  
  res.send('Aquí iría la lista de actores');  
});
```

## 🖋️ 2. POST

👉 ¿Para qué sirve?

Para **crear un nuevo recurso**. Se usa, por ejemplo, para agregar un nuevo actor o cliente.

📌 Ejemplo:

```
app.post('/actores', (req, res) => {  
  // Aquí insertarías un nuevo actor en la BD  
  res.send('Actor creado');  
});
```

Para leer datos enviados por el cliente, necesitas usar `express.json()` como middleware.

```
app.use(express.json());
```



### 3. PUT

👉 ¿Para qué sirve?

Para **modificar un recurso existente** (por ejemplo, actualizar la información de una película).

📌 Ejemplo:

```
app.put('/peliculas/:id', (req, res) => {  
  const id = req.params.id;  
  // Aquí modificarías la película con ese ID  
  res.send(`Película ${id} actualizada`);  
});
```

### 4. DELETE

👉 ¿Para qué sirve?

Para **eliminar un recurso** del servidor, como borrar un cliente o actor.

📌 Ejemplo:

```
app.delete('/clientes/:id', (req, res) => {  
  const id = req.params.id;  
  // Aquí eliminarías el cliente con ese ID  
  res.send(`Cliente ${id} eliminado`);  
});
```

🧠 Resumen visual:

```
app.get('/recurso')    // Leer  
app.post('/recurso')   // Crear  
app.put('/recurso/:id') // Actualizar  
app.delete('/recurso/:id') // Eliminar
```

## 10. Guía final de instalación:

### INSTALACIÓN

-En la raíz del proyecto creamos archivo de configuración: `npx tsc --init`

#### -En la carpeta backend:

```
cd backend
```

```
npm init -y
```

```
npm install express mysql2 cors
```

```
npm install --save-dev typescript ts-node nodemon @types/express @types/node
```

-Y en "scripts" del package.json backend añade:

```
"scripts": {  
  "build": "tsc",  
  "start": "nodemon dist/server.js",  
  "dev": "nodemon src/server.ts"  
}
```

#### -En la carpeta frontend:

```
cd ../frontend
```

```
npm init -y
```

```
npm install --save-dev typescript
```

-Y en "scripts" de package.json frontend:

```
"scripts": {  
  "build": "tsc --project tsconfig.json"  
}
```

### PUESTA EN MARCHA

-En la raíz de frontend:

compilamos: `npm run build`

-En la raíz de backend:

compilamos: `npm run build && npm run start`

-Accedemos poniendo la url `http://localhost:3000/`