



IES VALLE DEL JERTE PLASENCIA

SEGUNDA PRÁCTICA UNIDAD 3 1ºDAM

IMPORTAR/EXPORTAR LIBRERÍAS EN JAVA. MYSQL

Abel López Ruiz

Editado por



28 de diciembre de 2024

ÍNDICE GENERAL

| | |
|--|-----------|
| 1. Introducción | 2 |
| 1.1. Librerías en Java | 2 |
| 1.2. Librerías que utilizaremos en la práctica | 3 |
| 2. Importar/Exportar librerías en Java | 4 |
| 2.1. Importar librería | 4 |
| 2.2. Exportar Librería | 8 |
| 3. MySQL con los IDE's | 13 |
| 3.1. Introducción bases de datos y Mysql | 13 |
| 3.2. Instalando Mysql y Workbench | 14 |
| 4. Trabajando con las bases de datos | 18 |
| 4.1. sakila y algunas consultas | 18 |
| 4.2. Algunas consultas | 19 |
| 5. Mysql con Eclipse | 28 |
| 5.1. conectar la base de datos | 28 |
| 5.2. Prueba con la base de datos sakila | 32 |
| 6. Trabajando con MYSQL y SWING simultáneamente | 35 |
| 7. Entrega | 39 |
| 8. Bibliografía | 40 |

1. Introducción

1.1. Librerías en Java

En cualquier proyecto de Java el desarrollador normalmente utiliza librerías o bibliotecas externas para crear su proyecto. Estas librerías o bibliotecas también se conocen como dependencias, ya que tu proyecto dependerá de los métodos implementados que has usado de estas librerías externas. Estas librerías deberás importarlas a tu proyecto y además añadir el `import` o los `imports` necesarios en la cabecera de tu clase.

En las librerías que añadamos del exterior que no sean la de los API's de Java, podremos usar sus métodos y sus atributos de todas las clases que tengan implementadas, las podremos usar en nuestro proyecto según la necesidad que tengamos.

Esto es muy ventajoso porque nos da la oportunidad de reutilizar código, de hecho, código que hemos creado en otros proyectos tenemos la posibilidad de reutilizarlos en otros proyectos nuevos.

Para importar librerías, además de añadirlas en el proyecto (que veremos como se hace desde el IDE) hay que utilizar la cláusula `import` con la ruta de la clase según en el paquete o paquetes donde se encuentre. Un ejemplo de ello se puede observar a continuación:

```
1 package paquete.mipaquete;
2
3 import paquete_dam.otropaquete_dam.clase_dam;
4
5 class Ejemplo5{}
```

En el ejemplo anterior solo se importa la `clase_dam`, pero si queremos importar todas las clases que hay en `otropaquete_dam`, se usa la notación del asterisco como se mostrará a continuación. Cabe comentar que no tiene sentido importar clases que no vamos a usar ya que ralentiza la ejecución del programa.

```
6 import paquete_dam.otropaquete_dam.*;
7
8 class Ejemplo35{}
```

Con todo esto surge la pregunta, ¿qué formas hay de agregar una librería a mi proyecto?. Una de ellas es agregar el paquete entero o la clase con una opción de importar y otra que es muy típica con archivos `.jar` que son archivos comprimidos, realmente son `.rar` o `.zip`. En esta práctica se aprenderá a utilizar los archivos `.jar`.

1.2. Librerías que utilizaremos en la práctica

En esta práctica se va a exportar una librería externa en extensión .jar, se realizará en Eclipse, pero en el IDE Apache Netbeans es similar y se dejará como tarea para el alumno. Una vez importada además deberemos agregar el import correspondiente. Siempre que se utilice una librería deberíamos leer la documentación de todos los métodos y atributos que tienen y saber así que import añadir.

Luego una vez agregada, probaremos un método sencillo para ver si funciona y no hay ningún error. Después de esto se le propone al alumno crear un proyecto sencillo con una clase de un solo método y crear un archivo .jar para luego importarlo en otro proyecto vacío y probarlo. Es importante que el alumno aprenda a crear los .jar de sus proyectos para usarlos más adelante en otros proyectos o compartirlos con otros compañeros.

En nuestro caso vamos a probar unas librerías que están hechas en Java Swing ya usadas en la práctica anterior y la vamos a importar y utilizar para ver que funciona. El alumno puede probar otras librerías que se pueda descargar e incluirlas en su proyecto.

Una cosa que es importante en esta práctica que ya se vieron en anteriores, es que el alumno investigue sobre la variable de entorno CLASSPATH, saber que se guarda en esa variable y como podemos ver lo que hay dentro de esa variable.

También comentar que Java tiene unas API's o librerías propias que podemos usar, hay muchas, la mayoría más utilizadas son las de java.lang.*, está la librería Math de matemáticas, las de manejo de ficheros (métodos para leer y escribir de ficheros, crearlos, etc), librerías para String, etc.

Cuando se va a crear una librería es muy importante la documentación porque si la va a usar otra persona distinta al creador, debe saber cómo se utiliza, qué hace cada método, etc. Sino tiene documentación no le va a ser fácil utilizarla, pero como documentar una librería o clase se verá en secciones siguientes.

Otra librería que se va a emplear en esta práctica es la del conector de mysql, es una librería que tiene métodos para conectar con una base de datos y además hacer consultas. Es muy importante este tipo de librería pues casi en cualquier proyecto es normal usar bases de datos.

Se recomienda a los alumnos crear una máquina virtual en Virtualbox con Ubuntu o con windows 10, instalar los IDE's Netbeans y Eclipse, porque puede ser que se necesite instalar alguna otra herramienta.

2. Importar/Exportar librerías en Java

2.1. Importar librería

En esta práctica vamos a seguir usando Swing, la librería de interfaces gráficas de Java o constructor de ventanas. Vamos a descargarnos una librería conocida como JCalendar creada en Swing, y vamos a añadir el archivo .jar y probarla. Esta librería se puede descargar del siguiente lugar aunque la facilitaremos en la plataforma moodle (el alumno puede usar cualquier otra si lo desea, la práctica se trata de aprender a añadir una librería a nuestro proyecto):

Versión en paquete:

<https://toedter.com/jcalendar/>

Versión en jar:

<https://sourceforge.net/projects/agendaj/>

Crearemos primero una clase de una ventana sencilla de esta forma:

```
9 package importar;
10 /**
11  *
12  * @author Abel
13  */
14 import javax.swing.*;
15
16 public class Importar {
17
18     public static void main(String[] args) {
19
20         JFrame ventanilla= new JFrame();
21         ventanilla.setBounds(100,100,500,500);
22         ventanilla.setResizable(false);
23         ventanilla.setVisible(true);
24         ventanilla.setDefaultCloseOperation(JFrame.
25             EXIT_ON_CLOSE);
26     }
27 }
```

La salida que resulta es:

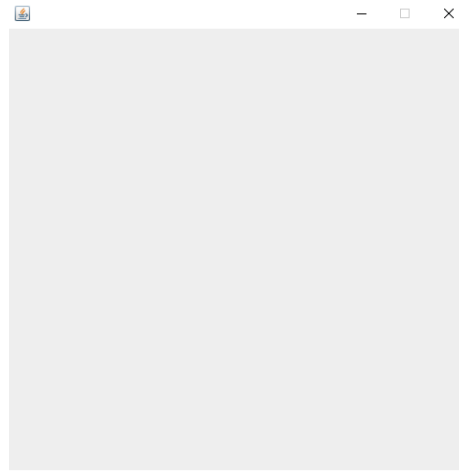


Figura 2.1: Ventana

Veamos como se agrega el JCalenda.jar a nuestro proyecto que es una librería hecha en Swing (le damos a la carpeta que pone Librerías del proyecto, click derecho y add Jar, todo esto en Netbeans, en Eclipse será diferente):

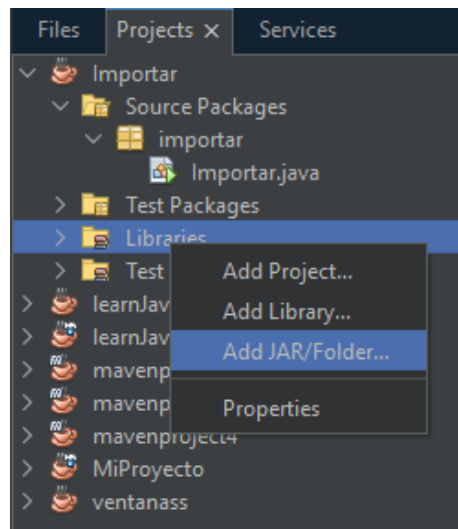


Figura 2.2: Añadir JAR

Como tiene tres puntos ..., nos aparecerá el siguiente diálogo en el que es un navegador de archivos y buscaremos el jar de JCalendar donde lo tengáis guardado en vuestra ruta:

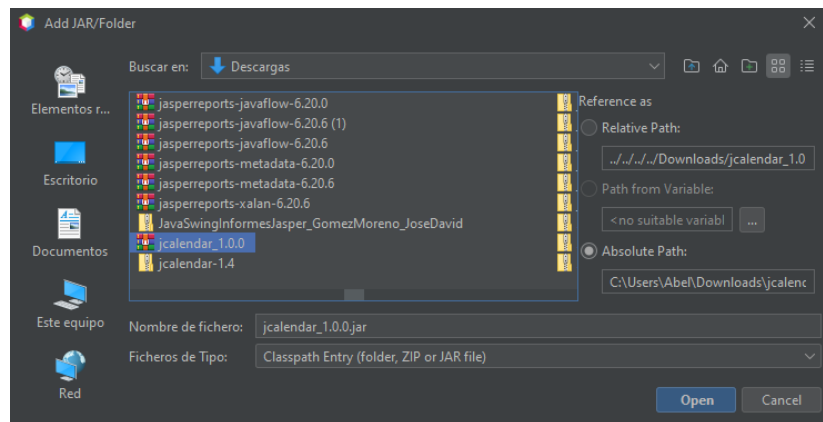


Figura 2.3: Añadir JAR

Le damos a Open y ya estaría agregada, para ver si se ha agregado, entonces, vamos a la carpeta de librerías y nos debería salir junto a las de JDK de Java:

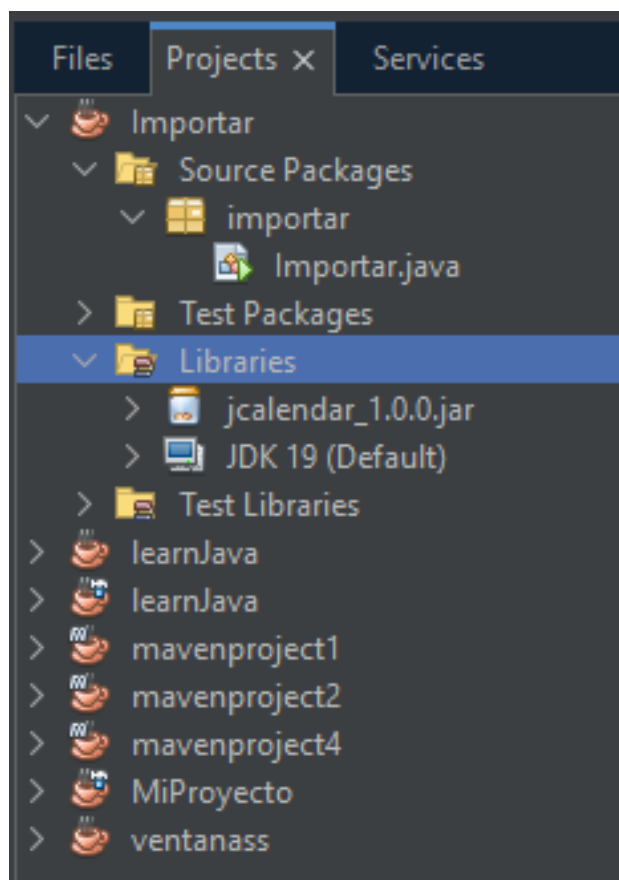


Figura 2.4: Añadir JAR

Después de esto, nos queda agregar el import, esto lo tendríamos que ver en la documentación. Pero no importa, se puede crear una instancia de esta forma: JCalendar cal=new JCalendar(), luego se le da a la bombilla y te sale en las opciones de reparar el error, una de ellas es agregar el import y le damos.

El código quedaría al final de esta forma, con la función add se añade el objeto calendario a la ventana para que salga en la ventana el calendario.

```
28     import com.toedter.calendar.JCalendar;
29     import javax.swing.*;
30
31     public class Importar {
32
33         /**
34         * @param args the command line arguments
35         */
36         public static void main(String[] args) {
37             // TODO code application logic here
38
39             JCalendar cal=new JCalendar();
40
41             JFrame ventanilla= new JFrame();
42             ventanilla.setBounds(100,100,500,500);
43             ventanilla.setResizable(false);
44             ventanilla.add(cal);
45             ventanilla.setVisible(true);
46             ventanilla.setDefaultCloseOperation(JFrame.
47                 EXIT_ON_CLOSE);
48
49         }
50
51     }
```

El resultado final por pantalla se puede ver que es:

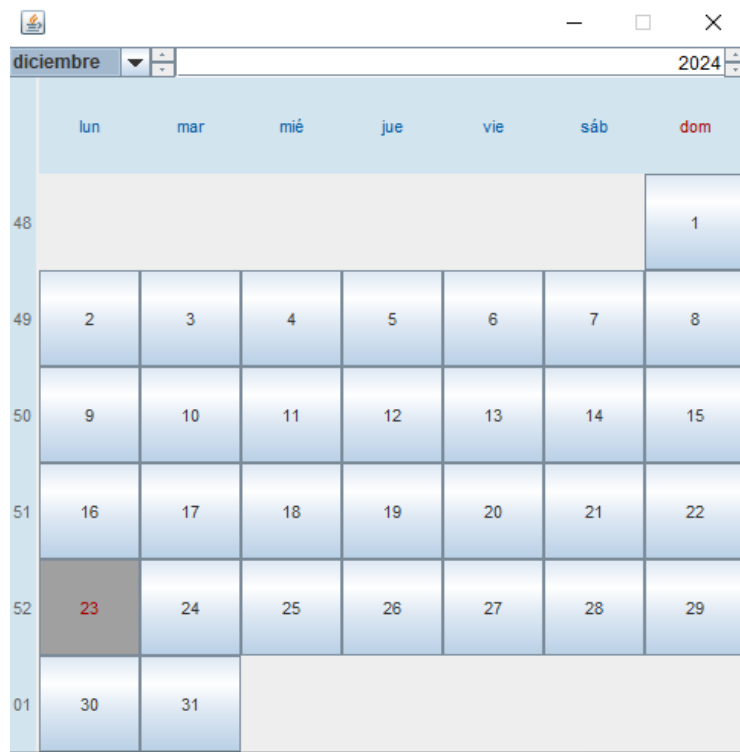


Figura 2.5: Ventana calendario

Se deja al alumno como tarea pendiente realizar todo lo anterior pero con el IDE Eclipse. Deberá aportar capturas de pantallas comentadas brevemente de cómo lo ha hecho.

2.2. Exportar Librería

En esta sección se va a crear esta vez el archivo .jar para más adelante poder usarlo en otros proyectos. Para ello crearemos un programa en Swing simple en Netbeans de una ventana con 4 botones como el que aparece en la imagen:

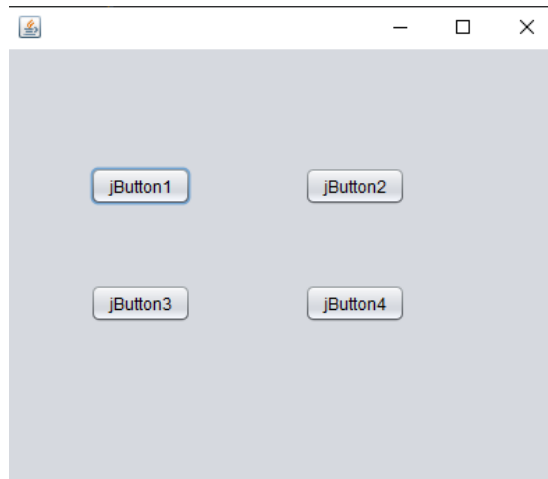


Figura 2.6: Proyecto a exportar

Una forma muy fácil es darle en el proyecto click derecho y le damos a clean and build y nos va a crear el archivo .jar que estará dentro de una carpeta dist que se genera, lo podemos ver en la imagen 2.8 en la pestaña de files.

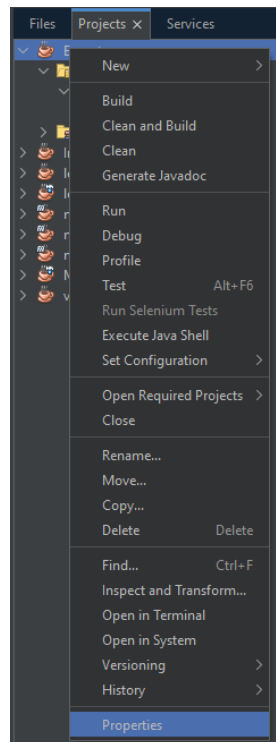


Figura 2.7: Proyecto a exportar

Una vez pulsado en la pestaña file vemos lo siguiente:

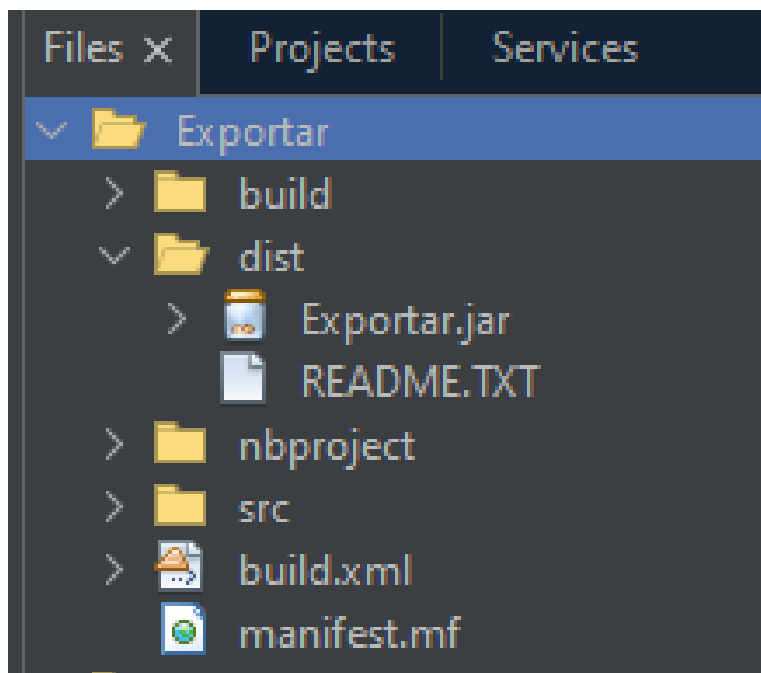


Figura 2.8: Proyecto a exportar

En la figura anterior se puede ver la carpeta dist (distribución) donde está nuestro archivo .jar y además con un README con información sobre el JAR.

Otra forma de crear el JAR es darle a clic derecho al proyecto, a propiedades y en packaging, pero solo se creará si ya está creado la carpeta dist:

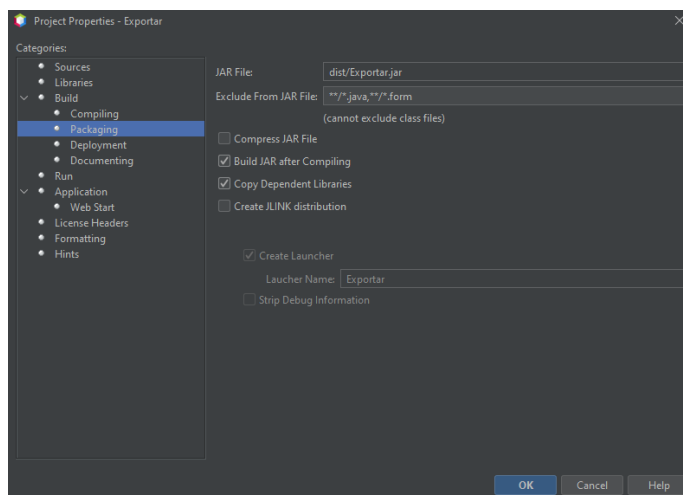


Figura 2.9: Proyecto a exportar

Otra forma también de exportar proyectos, mi carpeta proyecto en formato .zip y además importar también se puede es con la siguiente opción que nos aparece en la imagen:

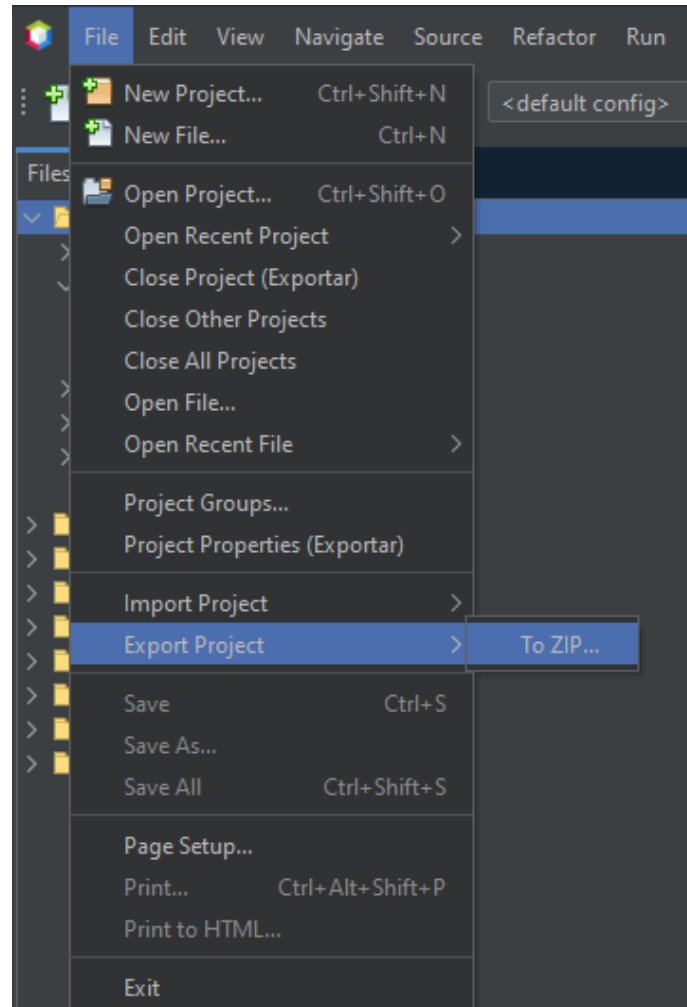


Figura 2.10: Proyecto a exportar

Después de creado nuestro .jar vamos a usarlo en un nuevo proyecto que creamos o en uno ya existente. Agregamos el jar a nuestro proyecto como se hizo anteriormente.

```

[ ] /**
[ ]  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt
[ ]  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit th
[ ]  */
[ ] package prueba_exp;
[ ]
[ ] import exportar.Exportado;
[ ]
[ ] /**
[ ]  *
[ ]  * @author Abel
[ ]  */
[ ] public class Prueba_exp {
[ ]
[ ]     /**
[ ]      * @param args the command line arguments
[ ]      */
[ ]     public static void main(String[] args) {
[ ]         // TODO code application logic here
[ ]
[ ]         Exportado exp=new Exportado();
[ ]
[ ]     }
[ ] }

```

Figura 2.11: Proyecto a exportar

Ejecutando este proyecto nos queda entonces (aunque no saldrá nada porque no lo hemos puesto visible con la siguiente sentencia `exp.setVisible(true)`):

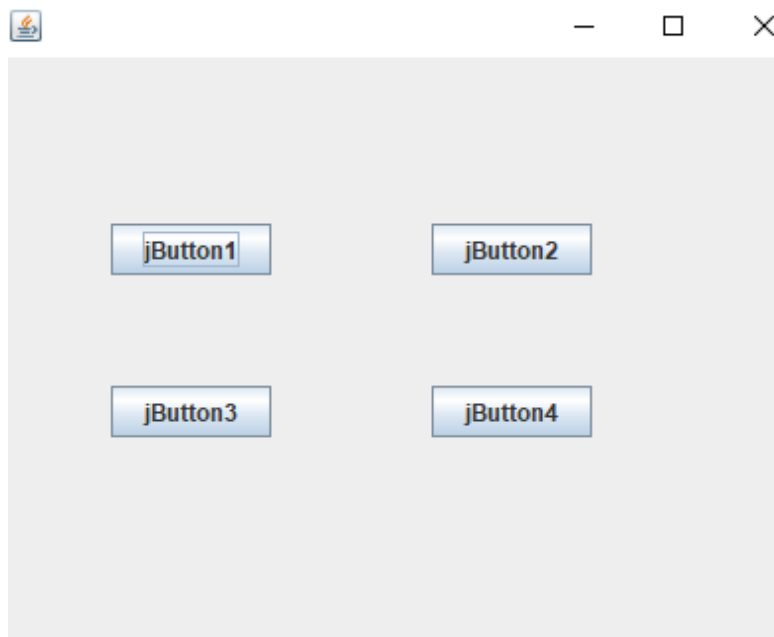


Figura 2.12: Proyecto a exportar

Se deja al alumno como tarea pendiente que haga todo lo anterior pero con el IDE Eclipse.

3. MySQL con los IDE's

3.1. Introducción bases de datos y Mysql

En esta sección se va a hablar sobre las bases de datos relacionales, ya que puede haber relaciones entre tablas, también existen bases de datos como las de MongoDB que no son relacionales como la MySQL. Una base de datos está compuestas por tablas y cada una de las tablas por campos. Los campos pueden ser de muchos tipos o clases, puede ser de tipo numérico, de tipo String, de tipo DATE (fechas), etc.

Por lo tanto, una base de datos (DB) está compuesta por tablas y a la vez la tabla puede tener varios campos, cada campo es de un tipo, hay infinidad de tipo, numérico, de fechas, de texto, etc. Un ejemplo es la tabla cliente, tendrá campos de nombre cliente, apellido primero, edad, etc. Si esta tabla tiene 2000 entradas, quiere decir que tendrá 2000 clientes.

Un campo además puede ser de muchos tipos, por ejemplo autoincremento, que es que se incrementa automáticamente cada entrada o registro que se añada a la tabla. También están los registros claves o key, una imagen que muestra varias tablas de ejemplo y alguna de sus relaciones:

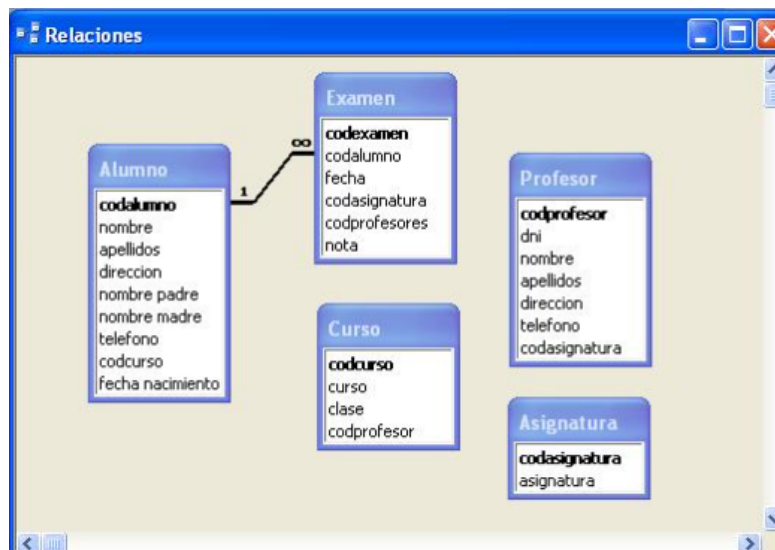


Figura 3.1: Tablas de ejemplo

Para hacer consulta a las bases de datos se usarán las Query, por ejemplo si queremos consultar los clientes mayores de 50 años, se obtendrá los registros que cumplan

dicha condición.

3.2. Instalando Mysql y Workbench

Primeramente hay que instalar mysql, se puede hacer escribiendo mysql.com en google y aparecerá la primera, pero de todas formas se facilita el enlace para instalar mysql:

<https://www.mysql.com/downloads/>

En esta instalación hay que marcar las bases de datos de sakila y otra que venga de ejemplo ya que vamos a trabajar con la base de datos sakila. Una vez instalado, que es un servicio que se instala por el puerto 3306 por defecto, puede ejecutarlo por la línea de comandos de windows por ejemplo escribiendo el comando mysql. Aunque nosotros instalaremos un gestor de bases de datos para no trabajar desde la línea de comandos conocido como mysql workbench, donde el enlaces de la instalación la indicamos se puede buscar en la página oficial de mysql:

<https://dev.mysql.com/downloads/workbench/>

Una vez instalado aparecerá algo como esto:

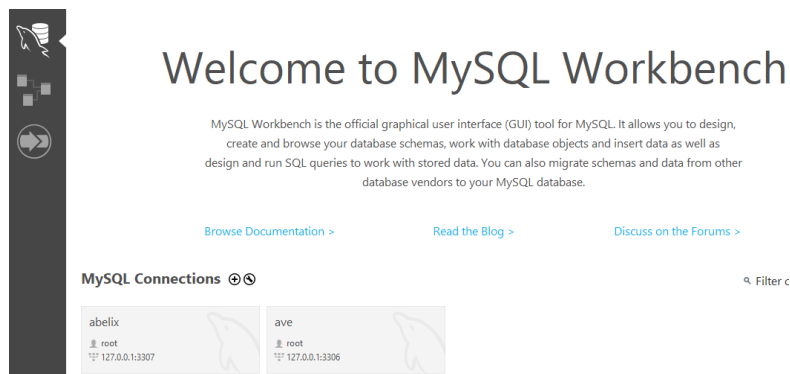


Figura 3.2: Workbench

Entramos en una de las conexiones con la clave que hayamos puesto y entonces aparecerá algo similar a la siguiente pantalla que es el gestor de base de datos, a la izquierda se puede ver como está la base de datos sakila que vamos a usar de pruebas. En ella se podrán ver las tablas y en la sección siguiente se explicará un poco con más detalle.

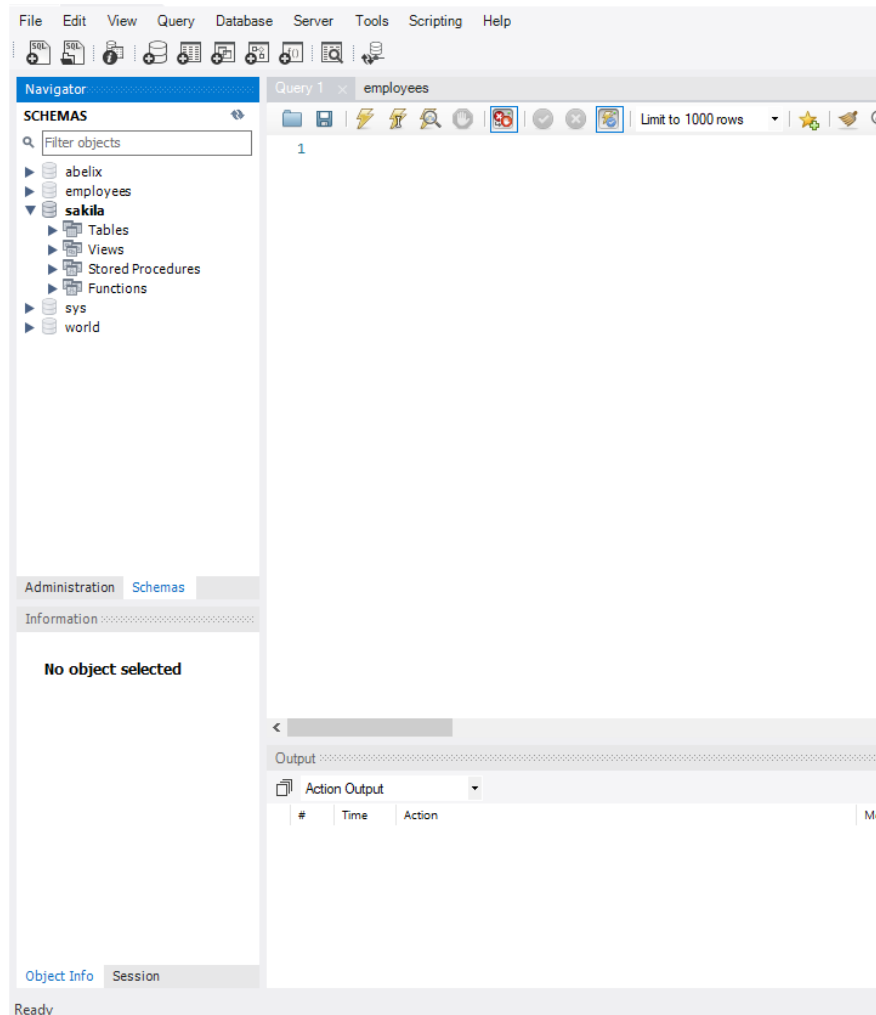


Figura 3.2: Workbench

Antes de nada comentar que en la parte izquierda en esquema, aparece todas mis bases de datos, incluida la de sakila, otra de empleados, otra que es la sys de sistema que es una base de datos del sistema, y otra world que son otra de las que vienen de ejemplo al instalar mysql. Si pinchamos en una de las bases de datos se despliega y se puede ver las tablas, las vistas, las funciones, es decir, que una base de datos está compuesta de algo más que las tablas, si pinchamos en las tablas podremos ver como sigue las tablas de sakila en la imagen 3.3.

También comentar que en el cuadro más grande donde pone Query es donde escribo las consultas, que se verán algunos ejemplos, y cuando le doy al rayo se ejecutan y el resultado sale en la ventana de salida output de abajo como se puede apreciar en la Figura 3.2.

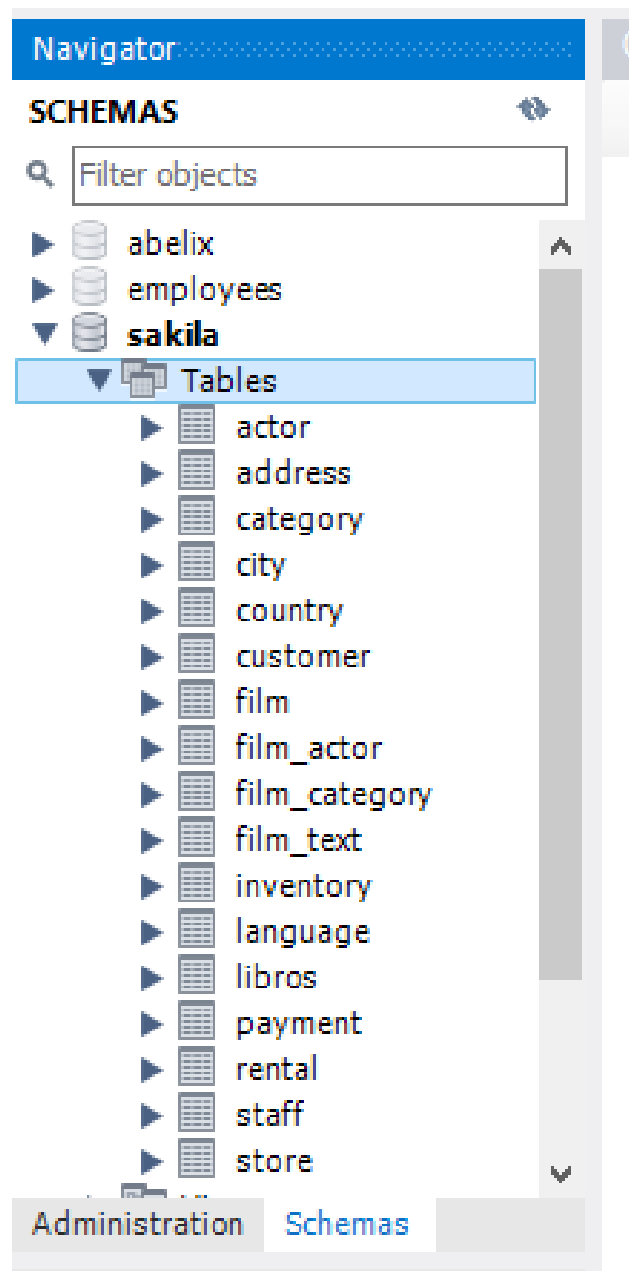


Figura 3.3: Tablas

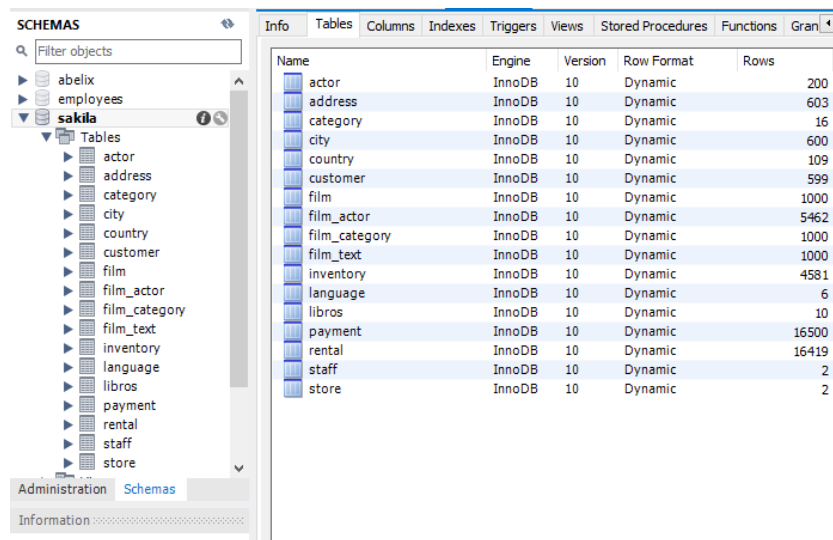
Antes de nada comentar que sakila es una base de datos de un videoclub, cuando existían las cintas VHS de películas y la gente las alquilaba, luego se adaptaron a los CD's y DVD's, apenas quedan ya abiertos negocios de videoclubs. Se podría comentar que pueden ser alguna de las tablas que vemos en la imagen.

En la tabla de actores, habrá los actores de cine de cada película, en address direcciones postales de cada videoclub, category será la categoría de la película (terror, suspense, etc). La tabla film, es la tabla de la lista de películas, language la tabla de idiomas de películas, y así podemos ir comentador qué puede ser cada una, aunque podremos hacer consultas sobre ellas y ver que hay.

4. Trabajando con las bases de datos

4.1. sakila y algunas consultas

En esta sección vamos a ver un poco la base de datos sakila y probar algunas cosas. Se puede ver un resumen de todo y sus columnas (campos) y tablas pulsando la i de información que aparece al lado:



| Name | Engine | Version | Row Format | Rows |
|---------------|--------|---------|------------|-------|
| actor | InnoDB | 10 | Dynamic | 200 |
| address | InnoDB | 10 | Dynamic | 603 |
| category | InnoDB | 10 | Dynamic | 16 |
| city | InnoDB | 10 | Dynamic | 600 |
| country | InnoDB | 10 | Dynamic | 109 |
| customer | InnoDB | 10 | Dynamic | 599 |
| film | InnoDB | 10 | Dynamic | 1000 |
| film_actor | InnoDB | 10 | Dynamic | 5462 |
| film_category | InnoDB | 10 | Dynamic | 1000 |
| film_text | InnoDB | 10 | Dynamic | 1000 |
| inventory | InnoDB | 10 | Dynamic | 4581 |
| language | InnoDB | 10 | Dynamic | 6 |
| libros | InnoDB | 10 | Dynamic | 10 |
| payment | InnoDB | 10 | Dynamic | 16500 |
| rental | InnoDB | 10 | Dynamic | 16419 |
| staff | InnoDB | 10 | Dynamic | 2 |
| store | InnoDB | 10 | Dynamic | 2 |

Figura 3.4: Sakila

También podemos ver un esquema o diagrama de las tablas y sus relaciones en la pestaña database y en reverse y entonces se creará un diagrama. Por ejemplo de la base sakila nos la muestra así:

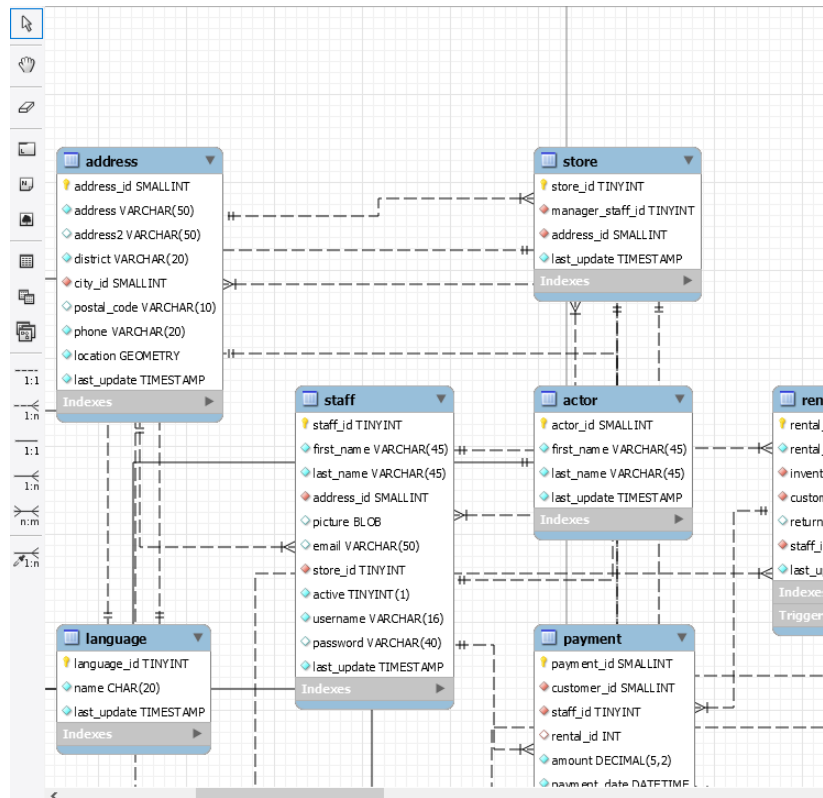


Figura 3.5: Diagrama relaciones Sakila

4.2. Algunas consultas

Ahora vamos a realizar algunas consultas o query sobre algunas de las tablas de la base de datos de sakila. Utilizaremos mayormente la instrucción SELECT. Las consultas las escribiremos en el cuadro grande y luego le daremos al rayo.

Se nos olvidó comentarlo, pero si pinchamos en la tabla y luego en columnas podemos ver los campos o columnas, por ejemplo en el caso de la tabla actor que mostramos en el ejemplo. Se puede ver que tiene 4 campos, la id del actor, el nombre (first_name), y el apellido y la fecha de última actualización.

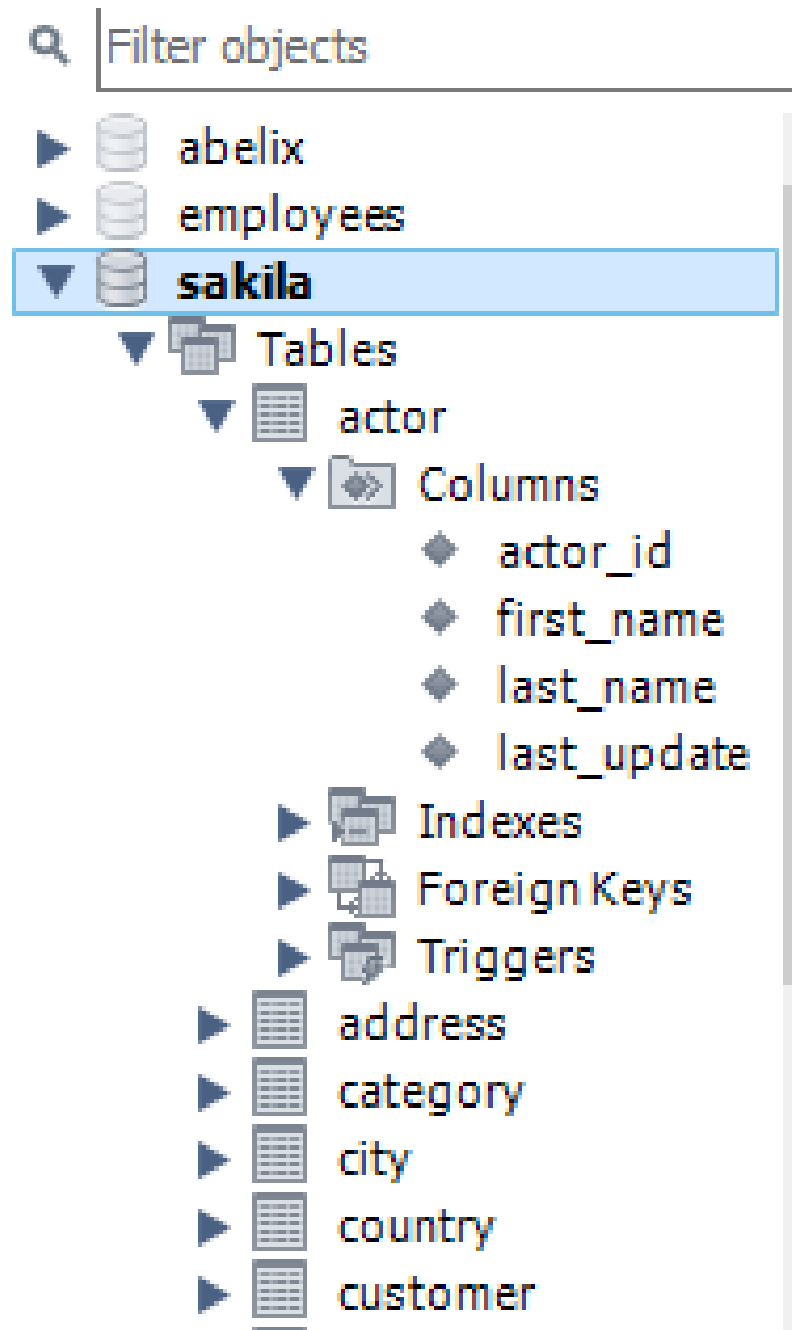


Figura 3.6: Campos de las tablas.

Para hacer una consulta para por ejemplo seleccionar el campo del primer nombre se hace de la siguiente forma:

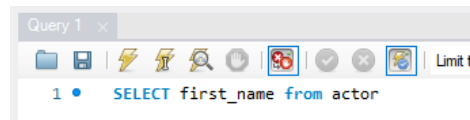


Figura 3.7: Query1.

La salida es una vez pulsado el rayo:

A screenshot of the 'Result Grid' showing the output of the query. It displays a single column with the following names: ANNE, NATALIE, GARY, CARMEN, MENA, PENELOPE, FAY, DAN, JUDE, CHRISTIAN, DUSTIN, and HENRY.

| first_name |
|------------|
| ANNE |
| NATALIE |
| GARY |
| CARMEN |
| MENA |
| PENELOPE |
| FAY |
| DAN |
| JUDE |
| CHRISTIAN |
| DUSTIN |
| HENRY |

Figura 3.8: Resultado1.

Veamos una serie de consultas más para practicar:

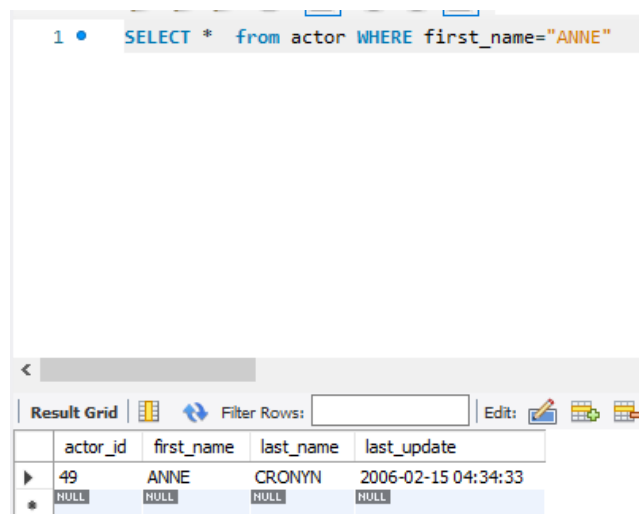


Figura 3.9: Consulta 2.

Query 1 x

Limit to 1000 rows

```
1 • SELECT * from actor WHERE first_name="PENELOPE";
```

2

3

Result Grid

| | actor_id | first_name | last_name | last_update |
|---|----------|------------|-----------|---------------------|
| ▶ | 1 | PENELOPE | GUINESS | 2006-02-15 04:34:33 |
| | 54 | PENELOPE | PINKETT | 2006-02-15 04:34:33 |
| | 104 | PENELOPE | CRONYN | 2006-02-15 04:34:33 |
| | 120 | PENELOPE | MONROE | 2006-02-15 04:34:33 |
| * | NULL | NULL | NULL | NULL |

Figura 3.10: Consulta 3.

Query 1 x

Limit to 1000 rows

```
1 • SELECT * from actor WHERE last_name="MONROE";
```

2

3

Result Grid

| | actor_id | first_name | last_name | last_update |
|---|----------|------------|-----------|---------------------|
| ▶ | 120 | PENELOPE | MONROE | 2006-02-15 04:34:33 |
| | 178 | LISA | MONROE | 2006-02-15 04:34:33 |
| * | NULL | NULL | NULL | NULL |

Figura 3.11: Consulta 4.

Query 1

Limit to 1000 rows

```
1 • SELECT * from payment WHERE amount>10;
```

2

3

Result Grid

| payment_id | customer_id | staff_id | rental_id | amount | payment_date |
|------------|-------------|----------|-----------|--------|---------------------|
| 44 | 2 | 2 | 9236 | 10.99 | 2005-07-30 13:47:43 |
| 69 | 3 | 2 | 7503 | 10.99 | 2005-07-27 20:23:12 |
| 324 | 12 | 2 | 10392 | 10.99 | 2005-08-01 06:50:26 |
| 342 | 13 | 2 | 8831 | 11.99 | 2005-07-29 22:37:41 |
| 551 | 21 | 1 | 3212 | 10.99 | 2005-06-21 01:04:35 |
| 793 | 29 | 2 | 5442 | 10.99 | 2005-07-09 21:55:19 |
| 908 | 33 | 1 | 1301 | 10.99 | 2005-06-15 09:46:33 |
| 1254 | 45 | 1 | 15812 | 10.99 | 2005-08-23 14:47:26 |
| 1347 | 49 | 2 | 8553 | 10.99 | 2005-07-29 11:15:36 |
| 1379 | 50 | 2 | 9691 | 10.99 | 2005-07-31 07:09:55 |
| 1511 | 54 | 1 | 15226 | 10.99 | 2005-08-22 17:20:17 |
| 2084 | 76 | 1 | 15566 | 10.99 | 2005-08-23 05:17:23 |

Figura 3.12: Consulta 5.

Query 1

Limit to 1000 rows

```
1 • SELECT * from payment WHERE amount>5 AND amount<6;
```

2

3

Result Grid

| payment_id | customer_id | staff_id | rental_id | amount | payment_date |
|------------|-------------|----------|-----------|--------|---------------------|
| 3 | 1 | 1 | 1185 | 5.99 | 2005-06-15 00:54:12 |
| 10 | 1 | 2 | 4526 | 5.99 | 2005-07-08 03:17:05 |
| 11 | 1 | 1 | 4611 | 5.99 | 2005-07-08 07:33:56 |
| 32 | 1 | 1 | 15315 | 5.99 | 2005-08-22 20:03:46 |
| 38 | 2 | 1 | 7376 | 5.99 | 2005-07-27 15:23:02 |
| 39 | 2 | 2 | 7459 | 5.99 | 2005-07-27 18:40:20 |
| 40 | 2 | 2 | 8230 | 5.99 | 2005-07-29 00:12:59 |
| 42 | 2 | 2 | 8705 | 5.99 | 2005-07-29 17:14:29 |
| 51 | 2 | 1 | 11087 | 5.99 | 2005-08-02 07:41:41 |
| 57 | 2 | 2 | 14743 | 5.99 | 2005-08-21 22:41:56 |
| 68 | 3 | 1 | 7096 | 5.99 | 2005-07-27 04:54:42 |
| 70 | 3 | 2 | 10597 | 5.99 | 2005-08-01 14:19:48 |

Figura 3.13: Consulta 6.

Query 1

Limit to 1000 rows

```
1 • SELECT * from film ORDER BY title ASC;
```

2

3

Result Grid

| | film_id | title | description |
|---|---------|------------------|--|
| ▶ | 1 | ACADEMY DINOSAUR | A Epic Drama of a Feminist And a Mad Scientist ... |
| | 2 | ACE GOLDFINGER | A Astounding Epistle of a Database Administrat... |
| | 3 | ADAPTATION HOLES | A Astounding Reflection of a Lumberjack And a ... |
| | 4 | AFFAIR PREJUDICE | A Fanciful Documentary of a Frisbee And a Lum... |
| | 5 | AFRICAN EGG | A Fast-Paced Documentary of a Pastry Chef An... |
| | 6 | AGENT TRUMAN | A Intrepid Panorama of a Robot And a Boy who... |
| | 7 | AIRPLANE SIERRA | A Touching Saga of a Hunter And a Butler who ... |
| | 8 | AIRPORT POLLOCK | A Epic Tale of a Moose And a Girl who must Con... |
| | 9 | ALABAMA DEVIL | A Thoughtful Panorama of a Database Administ... |
| | 10 | ALADDIN CALENDAR | A Action-Packed Tale of a Man And a Lumberjac... |
| | 11 | ALAMO VIDEOTAPE | A Roving Epistle of a Butler And a Cat who must |

Figura 3.14: Consulta 7.

Query 1

Limit to 1000 rows

```
1 • SELECT SUM(amount) from payment GROUP BY staff_id;
```

2

Result Grid

| | SUM(amount) |
|---|-------------|
| ▶ | 33482.50 |
| | 33924.06 |

Figura 3.15: Consulta 8.

Query 1 x

Limit to 1000 rows

```
1 • SELECT SUM(amount) from payment GROUP BY customer_id;
2
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content

| SUM(amount) |
|-------------|
| 118.68 |
| 128.73 |
| 135.74 |
| 81.78 |
| 144.62 |
| 93.72 |
| 151.67 |
| 92.76 |
| 89.77 |
| 99.75 |
| 106.76 |
| 103.72 |

Result 17 x Read Only

Result Grid, Form Editor, Field Types

Figura 3.16: Consulta 9.

Query 1 x

Limit to 1000 rows

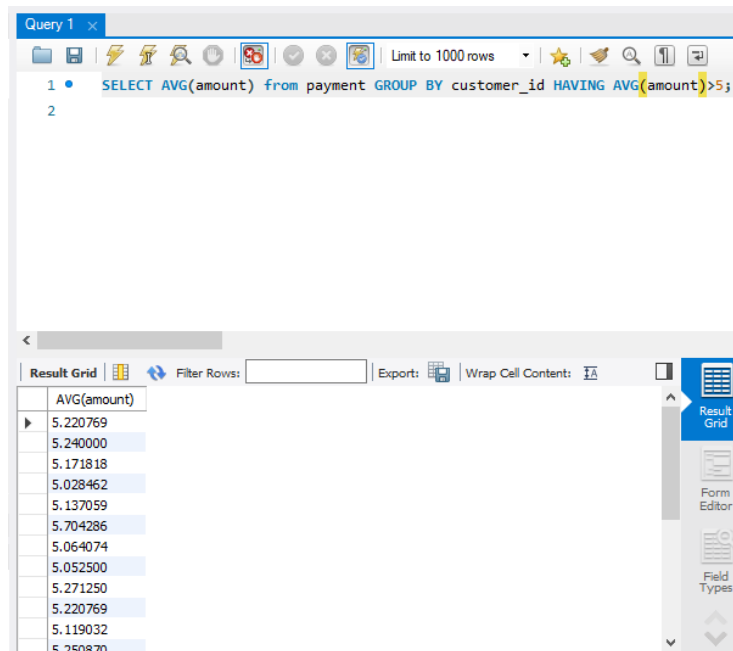
```
1 • SELECT AVG(amount) from payment GROUP BY customer_id;
2
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content

| AVG(amount) |
|-------------|
| 3.708750 |
| 4.767778 |
| 5.220769 |
| 3.717273 |
| 3.805789 |
| 3.347143 |
| 4.596061 |
| 3.865000 |
| 3.903043 |
| 3.990000 |
| 4.448333 |
| 3.704286 |

Result 18 x

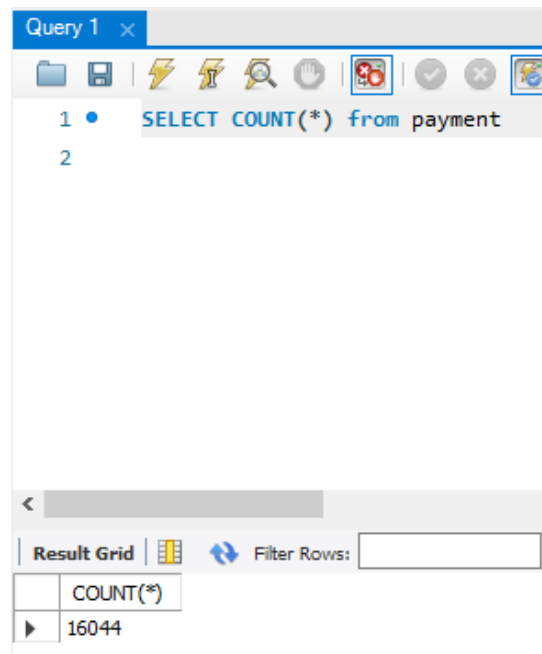
Figura 3.17: Consulta 10.



The screenshot shows a database query editor window titled "Query 1". The query text is: `SELECT AVG(amount) from payment GROUP BY customer_id HAVING AVG(amount)>5;`. The results are displayed in a "Result Grid" table with the following data:

| AVG(amount) |
|-------------|
| 5.220769 |
| 5.240000 |
| 5.171818 |
| 5.028462 |
| 5.137059 |
| 5.704286 |
| 5.064074 |
| 5.052500 |
| 5.271250 |
| 5.220769 |
| 5.119032 |
| 5.250870 |

Figura 3.18: Consulta 11.



The screenshot shows a database query editor window titled "Query 1". The query text is: `SELECT COUNT(*) from payment`. The results are displayed in a "Result Grid" table with the following data:

| COUNT(*) |
|----------|
| 16044 |

Figura 3.19: Consulta 12.

Query 1

Limit to 1000 rows

```
1 • SELECT * from payment LIMIT 10
```

2

Result Grid

Filter Rows:

Edit: Export/Import:

| | payment_id | customer_id | staff_id | rental_id | amount | payment_date | last_update |
|---|------------|-------------|----------|-----------|--------|---------------------|---------------------|
| ▶ | 1 | 1 | 1 | 76 | 2.99 | 2005-05-25 11:30:37 | 2006-02-15 22:11:04 |
| | 2 | 1 | 1 | 573 | 0.99 | 2005-05-28 10:35:23 | 2006-02-15 22:11:04 |
| | 3 | 1 | 1 | 1185 | 5.99 | 2005-06-15 00:54:12 | 2006-02-15 22:11:04 |
| | 4 | 1 | 2 | 1422 | 0.99 | 2005-06-15 18:02:53 | 2006-02-15 22:11:04 |
| | 5 | 1 | 2 | 1476 | 9.99 | 2005-06-15 21:08:46 | 2006-02-15 22:11:04 |
| | 6 | 1 | 1 | 1725 | 4.99 | 2005-06-16 15:18:57 | 2006-02-15 22:11:04 |
| | 7 | 1 | 1 | 2308 | 4.99 | 2005-06-18 08:41:48 | 2006-02-15 22:11:04 |
| | 8 | 1 | 2 | 2363 | 0.99 | 2005-06-18 13:33:59 | 2006-02-15 22:11:04 |
| | 9 | 1 | 1 | 3284 | 3.99 | 2005-06-21 06:24:45 | 2006-02-15 22:11:04 |
| | 10 | 1 | 2 | 4526 | 5.99 | 2005-07-08 03:17:05 | 2006-02-15 22:11:04 |
| • | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

Figura 3.20: Consulta 13.

5. Mysql con Eclipse

5.1. conectar la base de datos

Primeramente, se incluirá una librería .jar que hay para ello que es la mysql-conector-java.jar o un nombre similar, esta librería se facilitará en la plataforma pero se puede descargar. Esto se realizará pinchando en el proyecto con click derecho, a propiedades como se muestra en la imagen:

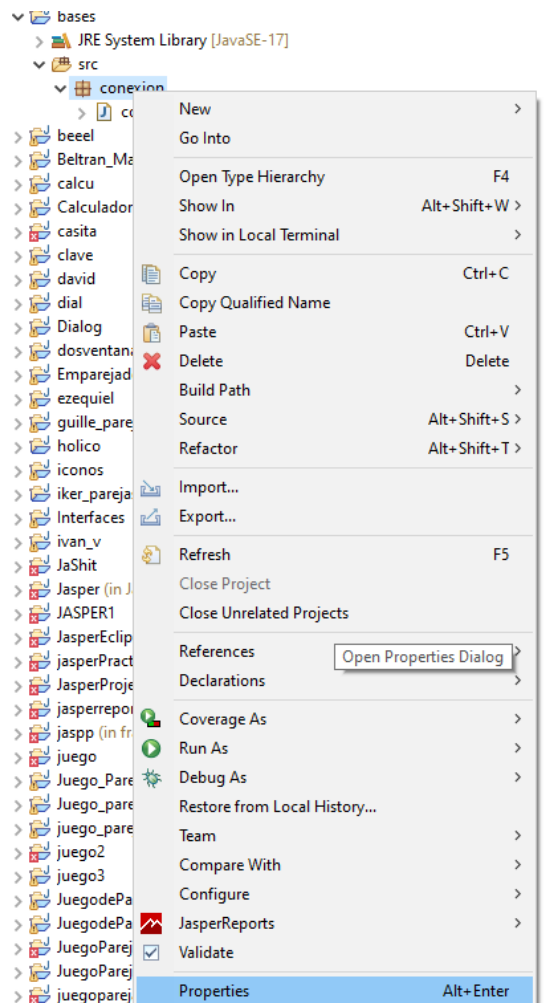


Figura 5.1: Agregar mysql-conector.

Una vez en el menú de propiedades, vamos a la opción build path y en la pestaña librería como se indica en la imagen:

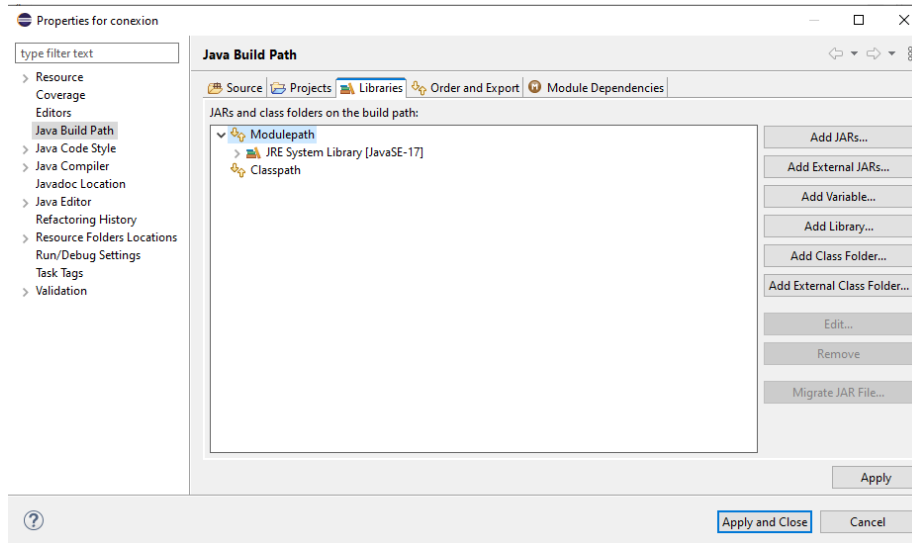


Figura 5.2: Agregar mysql-conector.

Luego en modulepath, el damos a agregar a external jars y buscamos el lugar donde está el mysql connector como se muestra en la siguiente figura:

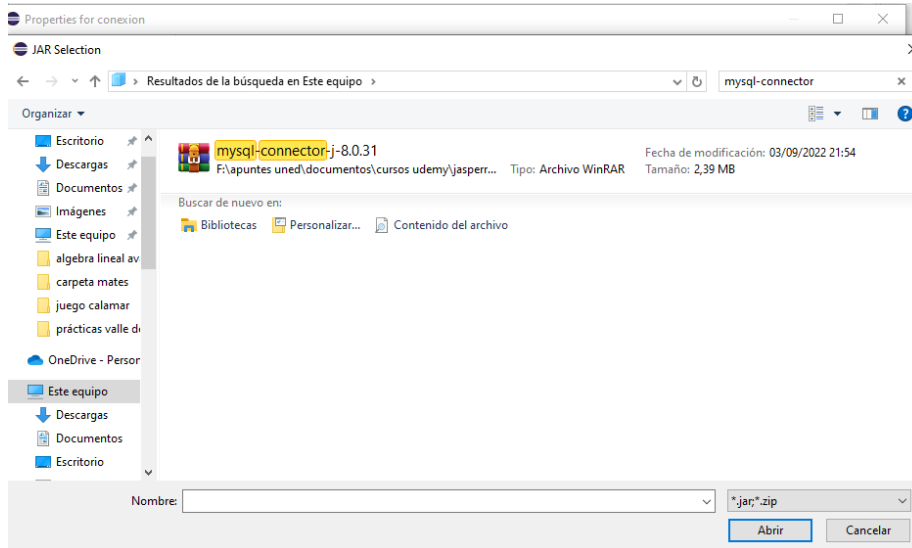


Figura 5.3: Agregar mysql-conector.

Luego quedará agregado como sigue y se le da a apply.

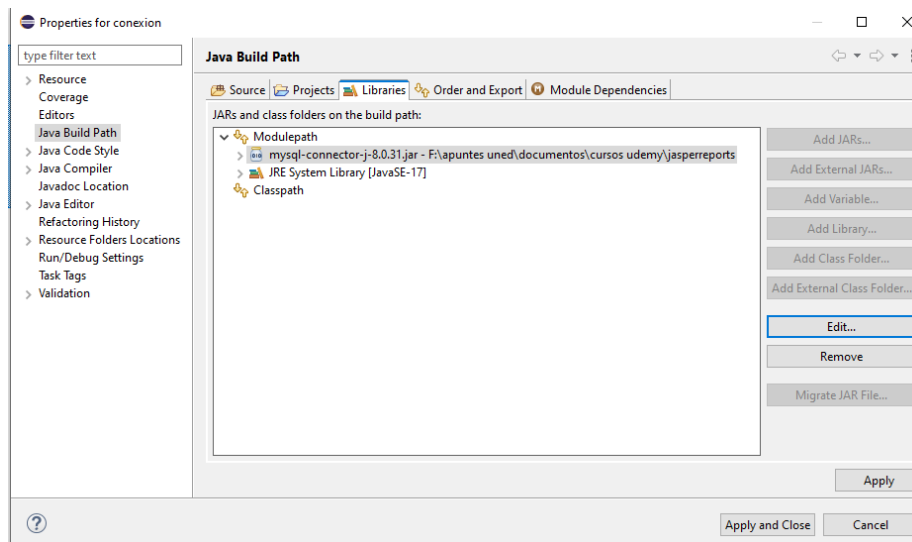


Figura 5.4: Agregar mysql-conector.

Una vez pulsado apply and close, en el proyecto nos aparecerá la librería de esta forma:

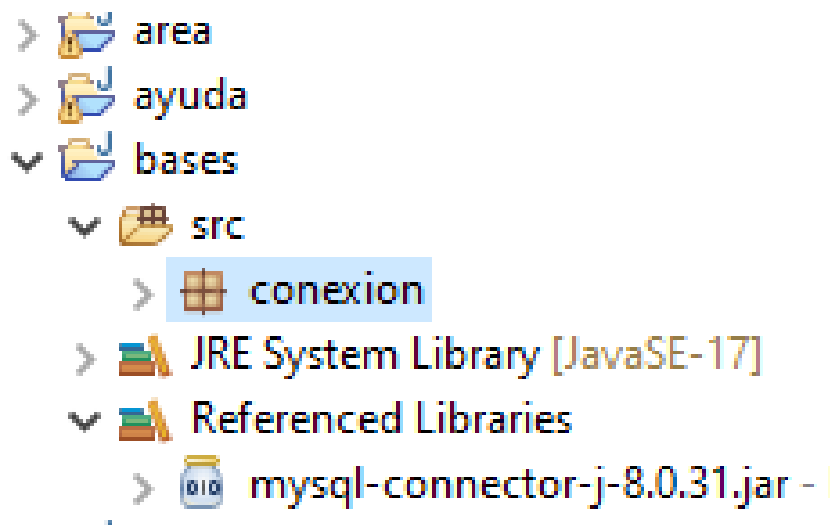


Figura 5.5: Agregar mysql-conector.

Bueno ahora se va a poner un poco de código para ver como conectarse a la base de datos sakila, los imports se van agregando dándole a la bombilla cuando salga y add imports.:

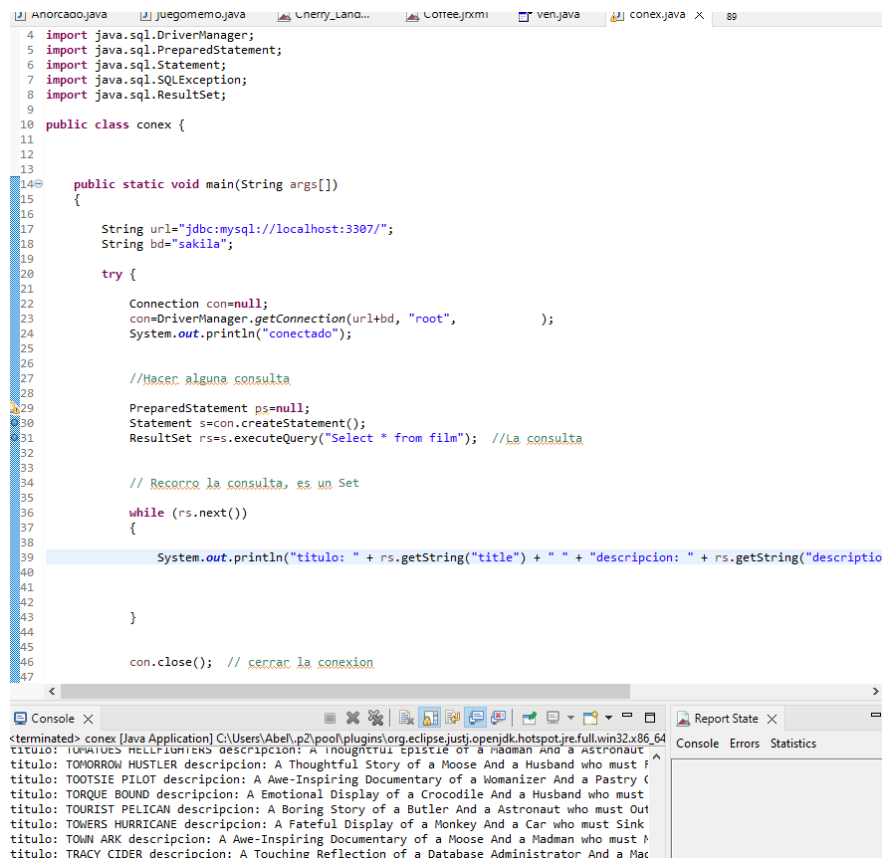


Figura 5.6: Conexión.

Primeramente vamos a conectar y lo vamos a hacer con un bloque try/catch por si salta alguna excepción, si se conecta entonces es que está funcionando, en la variable bd se pone el nombre de la base de datos, importante poner el puerto si es distinto al 3306 para que funcione.

5.2. Prueba con la base de datos sakila

Una vez lograda la conexión procederemos a hacer alguna consulta a la base de datos como se hacía en el gestor workbench pero con java. Para ello se usará el siguiente código:



```

4 import java.sql.DriverManager;
5 import java.sql.PreparedStatement;
6 import java.sql.Statement;
7 import java.sql.SQLException;
8 import java.sql.ResultSet;
9
10 public class conex {
11
12
13
14 public static void main(String args[])
15 {
16
17     String url="jdbc:mysql://localhost:3307/";
18     String bd="sakila";
19
20     try {
21
22         Connection con=null;
23         con=DriverManager.getConnection(url+bd, "root", "");
24         System.out.println("conectado");
25
26
27         //Hacer alguna consulta
28
29         PreparedStatement ps=null;
30         Statement s=con.createStatement();
31         ResultSet rs=s.executeQuery("Select * from film"); //La consulta
32
33
34         // Recorro la consulta, es un Set
35
36         while (rs.next())
37         {
38
39             System.out.println("titulo: " + rs.getString("title") + " " + "descripcion: " + rs.getString("descriptio
40
41
42         }
43
44
45         con.close(); // cerrar la conexion
46
47

```

Console Output:

```

conectado
titulo: TOOTSIE PILOT descripcion: A Awe-Inspiring Documentary of a Womanizer And a Pastry C
titulo: TORQUE BOUND descripcion: A Emotional Display of a Crocodile And a Husband who must f
titulo: TOURIST PELICAN descripcion: A Boring Story of a Butler And a Astronaut who must Out
titulo: TOWNERS HURRICANE descripcion: A Fateful Display of a Monkey And a Car who must Sink
titulo: TOWN ARK descripcion: A Awe-Inspiring Documentary of a Moose And a Madman who must f
titulo: TRACY CIDER descripcion: A Touchine Reflection of a Database Administrator And a Mac

```

Figura 5.7: Consultas.

Para la consulta se usa `executeQuery`, en una variable `ResultSet` se guarda la consulta, es de tipo Set (conjunto), por lo que para recorrer un conjunto no se hace con un iterador, se hace con `next` como se ve en la imagen, para tomar un campo de String se hace con `getString()`, para un entero `getInt()`, para un Float, `getFloat()`, que se puede apreciar en la imagen. Hemos trabajado con la tabla `film` de las películas, se ha sacado por pantalla el título de la película, la descripción y el precio, se va a poner el código para que quede más claro a continuación (no olvidar cerrar la conexión con `con.close()`):

```

52 package conexion;
53 import java.sql.Connection;
54 import java.sql.DriverManager;

```

```
55 import java.sql.PreparedStatement;
56 import java.sql.Statement;
57 import java.sql.SQLException;
58 import java.sql.ResultSet;
59
60 public class conex {
61
62     public static void main(String args[])
63     {
64
65         String url="jdbc:mysql://localhost:3307/";
66         String bd="sakila";
67
68         try {
69
70             Connection con=null;
71             con=DriverManager.getConnection(url+bd, "root", "*****");
72             System.out.println("conectado");
73
74
75             //Hacer alguna consulta
76
77             PreparedStatement ps=null;
78             Statement s=con.createStatement();
79             ResultSet rs=s.executeQuery("Select * from film"); //La
                consulta
80
81             // Recorro la consulta, es un Set
82             while (rs.next())
83             {
84
85                 System.out.println("titulo: " + rs.getString("title") + " " + "
                    descripcion: " + rs.getString("description") + " precio : "
                    + rs.getFloat("rental_rate") );
86
87             }
88
89             con.close(); // cerrar la conexion
90
91         } catch (SQLException e) {
92
93             System.out.println("Fallo en la conexion");
94
95         }
96     }
97
98 }
99
100 }
```

Se pueden hacer todas las consultas que se hicieron con workbench aquí, cambiando lo que ha dentro del executeQuery, se propone de ejercicio las siguientes:

1. **De la tabla film, seleccionar title y rating, siempre que sea PG-13**
2. **Contar cuántas películas hay del Rating PG-13, PG, G, R, NC-17**
3. **Muestre los 10 primeras consultas de título, descripción y rating de una película cuando rental_rate sea menor que 3**
4. **De la tabla clientes customer, escriba los nombres e emails de solo 15 clientes**
5. **En la tabla film, sumar las películas de cada rating, mostrar el rating y su suma.**
6. **En la tabla de actores, ¿de qué actor hay más películas?. Para contar usar COUNT, u ORDER BY ASC, ¿hay más de un actor, quiénes son?**
7. **Con la función MAX en la tabla de film, mostrar el máximo de rental_duration**

6. Trabajando con MYSQL y SWING simultáneamente

Después de haber trabajado con Mysql en los IDE's, vamos a trabajar en Eclipse por ejemplo conjuntamente con Swing y MySQL, en lugar de mostrar las entradas de una consulta de MYSQL en la salida, la vamos a mostrar en una ventana o Frame, más concretamente en una tabla o JTable.

Para ello vamos a crear una ventana con una tabla de 10 entradas, le daremos a la tabla unas dimensiones que se pueda ver completamente, y 3 botones, uno de los botones hará una consulta de 10 entradas de los nombres de los actores, otra 10 títulos de películas y otro botón para consulta del nombre de 10 clientes (Es decir 3 JButton). Por lo tanto habrá que implementar los evento de mouseclick de estos botones. La ventana quedará algo como se muestra en la imagen:

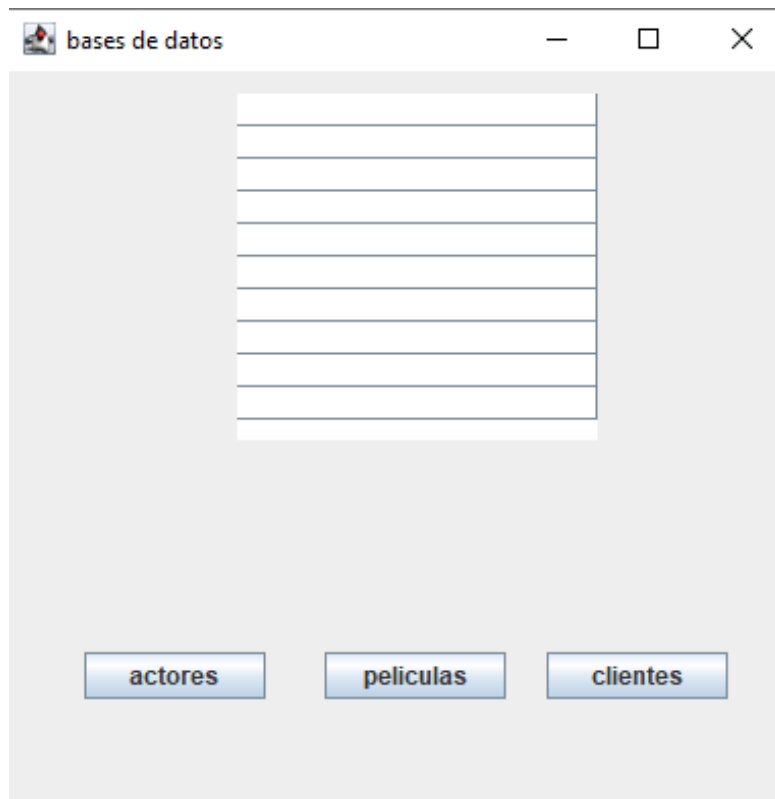


Figura 6.1: Swing y MySQL.

Hay un método que mostraremos a continuación para añadir los elementos de la entrada a la tabla, se llama el método `setValueAt`(elemento a agregar, fila, columna). Las consultas que haremos serán 10 para que no nos salga una tabla muy grande, serán estas consultas:

```
101 ResultSet rs=s.executeQuery("Select * from actor limit 10");
    //La consulta
102 ResultSet rs=s.executeQuery("Select * from film limit 10"); //
    La consulta
103 ResultSet rs=s.executeQuery("Select * from customer limit 10");
    //La consulta
```

Para agregar a la tabla se usará en cada uno de los 3 casos para cada uno de los botones, lo siguiente:

```
105 table.setValueAt(rs.getString("first_name"), i, 0); //para
    cliente y el actor
106 table.setValueAt(rs.getString("title"), i, 0); // para film
```

Vamos a poner el ejemplo del evento del botón película, en los demás casos serán similares, solo cambiando 2 líneas (el código que va dentro del evento):

```
108 btnNewButton_1.addMouseListener(new MouseAdapter() {
109     @Override
110     public void mouseClicked(MouseEvent e) {
111         try {
112
113             String url="jdbc:mysql://localhost:3307/";
114             String bd="sakila";
115
116             Connection con=null;
117             con=DriverManager.getConnection(url+bd, "root", "
                *****");
118             System.out.println("conectado");
119
120
121             //Hacer alguna consulta
122
123             PreparedStatement ps=null;
124             Statement s=con.createStatement();
125             ResultSet rs=s.executeQuery("Select * from film
                limit 10"); //La consulta
126
127
128             // Recorro la consulta, es un Set
129             int i=0;
130             while (rs.next())
131             {
132
133                 table.setValueAt(rs.getString("title"), i, 0);
```

```
134         i++;
135
136     }
137
138
139     con.close(); // cerrar la conexion
140
141
142     }catch(SQLException e3)
143     {
144
145         System.out.println("Fallo en la conexion");
146
147     }
148
149
150
151     }
152 }
153 };
```

Para los otros dos botones es similar solo cambiando la línea de la consulta 125 y la 133 para introducir en la tabla según el campo que se quiera meter en la tabla.

Si se pulsa en películas aparecerá la siguiente imagen:

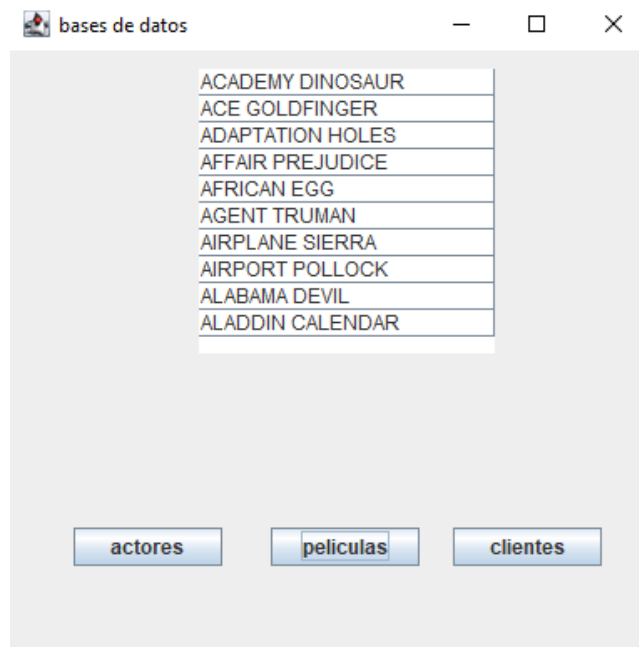


Figura 6.2: Botón películas.

Se deja como tarea pendiente que el alumno ponga 2 columnas más en el caso de película al lado, una para el rating y otra para el precio o rental_rate. Además si puede poner una cabecera o título que ponga título, rating y precio arriba de cada columna.

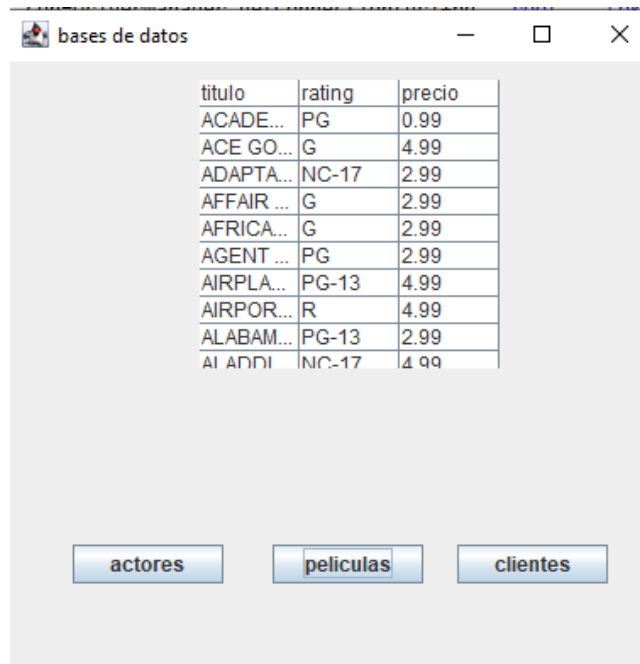


Figura 6.3: Botón películas.

7. Entrega

La entrega de la práctica se enviará por la plataforma moodle con el formato siguiente: nombrealumno_apellido1_apellido2.pdf, deberá incluir todas las capturas comentadas brevemente lo que se ha hecho en ellas con una o dos líneas. El plazo queda fijado un día antes de las evaluaciones.

Además se tendrá en cuenta la presentación del documento de la práctica, y aportaciones extras que se hagan.

8. Bibliografía

- [1] Gestor de una biblioteca Autor: Jose Alberto Benítez Andrade Gestor de una base de datos de una biblioteca de libros <https://www.jabenitez.com/personal/UNI/docu/documentacionBiblioteca.pdf>
- [2] Tutorial de incorporaci3nn del driver JDBC a Eclipse 20 de septiembre de 2016, UCA (Universidad de Cádiz) autores: Angel Manuel Gamaza Domínguez, José Miguel Otte Sainz-Aguirre https://rodin.uca.es/bitstream/handle/10498/18602/incorporacion-driver-JDBC-eclipse_Gamaza_Otte.pdf?sequence=1&isAllowed=y
- [3] Driver JDBC <https://cs.uns.edu.ar/~drg/bd/downloads/Clases%20Practicas/Java%20y%20MySQL/Clase-Java-MySQL.pdf>
- [4] Vídeo Octavio Sánchez Youtube autor: Octavio Sánchez <https://www.youtube.com/watch?v=oX7EyR6tAe0>