**Prueba Objetiva practica final // UF1845\_3/**

**Acceso a datos en Aplicaciones Web del entorno Servidor**

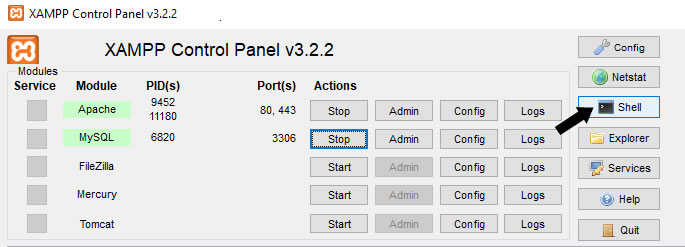
**Rune Brito N.**

**RESPUESTAS:**

**1.0 Importación de Base de Datos**

**1.1 Importación de Base de Datos**

Se instaló y accedió al programa XAMPP, en esta plataforma se activó el Mysql, y a su vez, se ejecutó “El terminal” o consola para dar inicio al ejercicio.



Dentro del terminal o consola, se introdujo la secuencia que indica que queremos conectarnos a MySQL con el usuario “root” por defecto y con contraseña genérica (intro) .

**# mysql -u root –p**

**MariaDB [(none)]>**

Del archivo que nos hemos descargado de la base de datos Sakila (RAR), vemos que hay dos archivos, “sakila-schema.sql” corresponde a la estructura de la base de datos y “sakila-data.sql” que corresponden a los datos. Estos se descomprimieron en el disco C:/ del ordenador. Luego a través de los siguientes comandos se importó la base de datos Shakila.sql y posteriormente se importaron los datos que contiene.

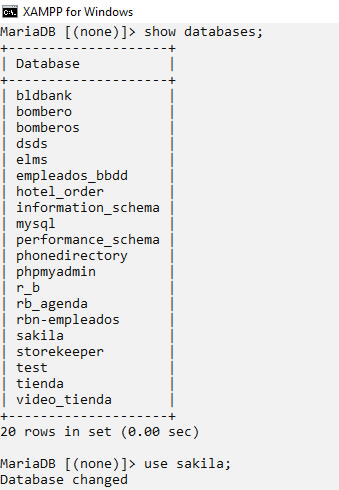
Lo primero fue importar la estructura, para lo cual teniendo el archivo “sakila-schema.sql” descargado en C:/

**MariaDB [(none)]>SOURCE C:/sakila-schema.sql;**

Posteriormente se utilizó, el mismo procedimiento para los datos con el nombre de fichero en la ubicado también en C:/

**MariaDB [(none)]>SOURCE D:/sakila-data.sql;**

Una vez importado los dos archivos, se ha creado la base de datos shakila con toda su estructura y datos en su interior



**1.2 y creación de tablas, registros y campos**

Se creó una tabla con sus campos y contenidos en dentro de la base de datos Sakila que lleva por nombre “Artistas\_CD” y que lleva como campos para ser rellenados a futuro: nombre, apellidos, agrupación, email, teléfono y país.

CREATE TABLE IF NO EXIST `artista\_cd` (`nombre` varchar(16) NOT NULL,

`Apellidos` varchar(30) NOT NULL,

`agrupacion` varchar(30) NOT NULL,

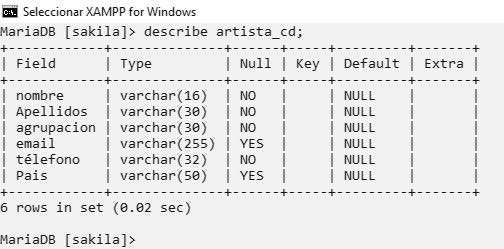
`email` varchar(255) DEFAULT NULL,

`télefono` varchar(32) NOT NULL,

`Pais` varchar(50) DEFAULT NULL )

ENGINE=InnoDB DEFAULT CHARSET=UTT-8

Luego dentro de la línea de comando se escribió la sentencia, la tabla y su contenido

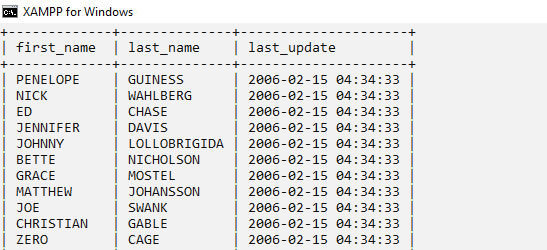


**2. CONSULTAS EN LA BASE DE DATOS SAKILA**

**2.1 Consulta Básicas**

Se realizó una consulta básica en la tabla “actor” por “nombre”, “apellidos” y “fecha”

SELECT first\_name,last\_name, last\_update FROM actor;

****

Con esta sentencia se generó una consulta que muestra los clientes que están en la tabla rental.

SELECT rental.rental\_id,customer.first\_name,

customer.last\_name,rental.rental\_date,rental.return\_date

FROM rental

INNER JOIN customer

ON rental.customer\_id=customer.customer\_id

Con esta sentencia, se muestran las películas alquiladas con mayor frecuencia en orden descendente.

SELECT COUNT(r.rental\_id), f.title as "Film"

FROM film f

INNER JOIN inventory i ON f.film\_id = i.film\_id

INNER JOIN rental r ON i.inventory\_id = r.inventory\_id

GROUP BY 2

ORDER BY 2 DESC;

Con esta sentencia, se consultan los apellidos de actores y la cantidad de actores que tienen dicho apellido, pero sólo para nombres compartidos por al menos dos.

SELECT last\_name, COUNT(\*)

FROM actor

GROUP BY 1

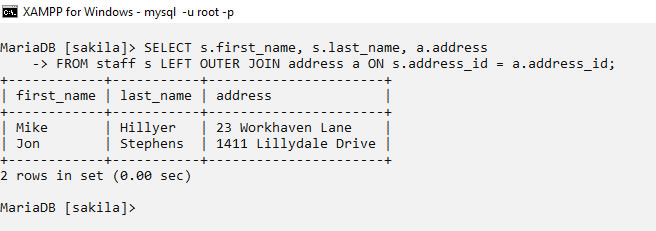
HAVING COUNT(\*) >= 2

**2.2 Consultas con leftJoint**

Se utilizó joint para la consulta de los nombres y apellidos, así como la dirección de cada miembro del personal y nombres del personal del staff.

SELECT s.first\_name, s.last\_name, a.address

FROM staff s LEFT OUTER JOIN address a ON s.address\_id = a.address\_id;

****

**2.3 Consultas elaboradas con Inner Joint**

Se utilizaron un par de consultas “Joint”, para ello se tomó como ejemplo de la vida real dentro de Sakila y por ende de una tienda de DVD, la cuales pueden producir una lista diaria de alquileres vencidos para que los clientes puedan ser contactados y pedirles que devuelvan sus DVD vencidos.

Para esta consulta se buscaron películas en la tabla de alquiler con una fecha de retorno NULL y donde la fecha de alquiler sea más antigua que la duración del alquiler especificada en la tabla de películas. Con ello, si la película está vencida, muestra el nombre de la película junto con el nombre del cliente y su número de teléfono.

SELECTCONCAT(customer.last\_name,', ',customer.first\_name)AScustomer,

address.phone,film.title

FROMrentalINNERJOINcustomer

ONrental.customer\_id=customer.customer\_id

INNERJOINaddressONcustomer.address\_id=address.address\_id

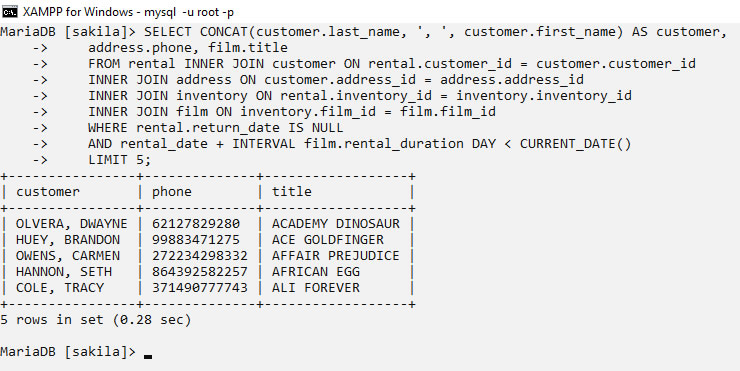
INNERJOINinventoryONrental.inventory\_id=inventory.inventory\_id

INNERJOIN film ONinventory.film\_id=film.film\_id

WHERErental.return\_dateISNULL

ANDrental\_date+INTERVALfilm.rental\_duration DAY <CURRENT\_DATE()

LIMIT5;

****

**3.0 Creación de Vistas**

**3.1 Creación de Vista #1**

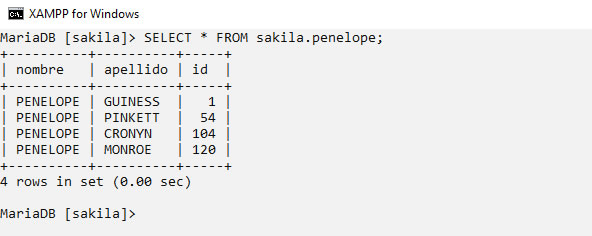
Con este código o sentencia se obtuvo una vista con los datos de películas cuyos nombres de actrices empiezan por un nombre” en este caso Penelope”.

CREATE VIEW penelope AS

SELECT first\_name as nombre,last\_name as apellido,actor\_id as id

FROM actor

WHERE first\_name= 'PENELOPE';



**3.2 Creación de Vista #2**

Con este código o sentencia se obtuvo una obteniendo una vista para ver los cinco primeros géneros, que generan mayores ingresos a la tienda de dvd.

CREATE VIEW top\_5 as

SELECT c.name as 'Film', sum(p.amount) as 'GrossRevenue'

fromcategory as c

joinfilm\_category as fconfc.category\_id = c.category\_id

joininventory as i oni.film\_id = fc.film\_id

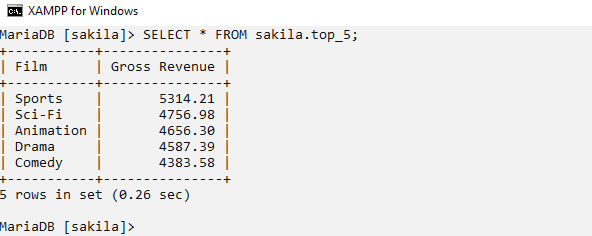
joinrental as r onr.inventory\_id = i.inventory\_id

joinpayment as p onp.rental\_id = r.rental\_id

groupby c.name

orderby sum(p.amount) desc

limit 5;



**4.0 DESCENCADENADORES ( TRIGGERS)**

**4.1 Trigger para generar una actualización de pago por el alquiler de DVD**

CREATE DEFINER=`root`@`localhost` TRIGGER `sakila`.`payment\_BEFORE\_UPDATE` BEFORE UPDATE ON `payment` FOR EACH ROW

BEGIN

if NEW.amount > OLD.amount then

set NEW.amount=OLD.amount;

end if;

END

**4.2 Trigger para insertar una oferta de 7.50$ cuando tenga un precio de 10$**

CREATE DEFINER=`root`@`localhost` TRIGGER `sakila`.`payment\_BEFORE\_INSERT` BEFORE INSERT ON `payment` FOR EACH ROW

BEGIN

ifNEW.amount>=10 then

set NEW.amount=7.50;

endif;

END

**5.0 Funciones**

La siguiente sentencia es una función que devuelve el “customer\_id” del cliente que actualmente alquila el artículo, o NULL si el artículo está en stock. donde “p\_inventory\_id”: es el ID o numero entre () del artículo de inventario que se debe verificar.

SELECT inventory\_held\_by\_customer(9);

**6.0 Procedimientos**

Con este procedimiento se produce una tabla de números ID de inventario para las copias de la película en stock y devuelve (en el parámetro p\_film\_count) un recuento que indica el número de filas en esa tabla.

CALL film\_in\_stock(1,1,@count);

Con este procedimiento se aumenta en una semana la fecha de entrega de la película alquilada (rental) y que el precio del alquiler aumente en 0,25$.

UPDATE sakila.rental

SET rental.return\_date = DATE\_ADD(rental.return\_date, INTERVAL 7 DAY);

UPDATE sakila.payment

SET payment.amount = payment.amount + 0.5;

Con este procedimiento la función inventory\_held\_by\_customer devuelve el customer\_id del cliente que ha alquilado el artículo de un inventario especifico.

BEGIN

DECLARE v\_customer\_id INT;

DECLARE EXIT HANDLER FOR NOT FOUND RETURN NULL;

SELECT customer\_id INTO v\_customer\_id

FROM rental

WHERE return\_date IS NULL

AND inventory\_id = p\_inventory\_id;

RETURN v\_customer\_id;

END