Ingeniería del Software 2
Grado en Ingeniería Informática en Sistemas de Información
Enseñanzas Prácticas y de Desarrollo

EPD 6: Acceso a MySQL mediante JDBC

Objetivos

- Manejar el gestor de MySQL: crear tablas, relaciones entre éstas, gestionar usuarios y privilegios.
- Conocer algunas de las herramientas gráficas que permiten la manipulación de MySQL.
- Crear bases de datos con las diferentes herramientas presentadas ya sea haciendo restore o desde cero y realizar copias de la base datos haciendo backup.
- Establecer conexiones entre Netbeans y el gestor MySQL.
- Dominar el cliente JDBC.
- Realizar una conexión a MySQL desde Java mediante JDBC y manipular los datos almacenados en alguna base de datos.

Conceptos

1 Instalación y Configuración del SGBD MySQL

Para la instalación y configuración de MySQL se ha de descargar el instalador para Windows que se puede encontrar en la página web de MySQL. Entre las distintas versiones que se ofrecen allí la versión Community es la que posee licencia GPL y es la que vamos a utilizar.

Al localizar en la web la zona de descargas de MySQL Community podemos encontrar que se nos ofrece la descarga de múltiples herramientas: MySQL Server, MySQL Connectors, MySQL Workbench, MySQL Notifier,… Para nuestro caso existe un instalador que incluye todos los componentes y que se denomina MySQL on Windows que se puede descargar desde: https://dev.mysql.com/get/Downloads/MySQLInstaller/mysql-installer-community-8.0.15.0.msi

El proceso de instalación es muy sencillo, tan sólo deberá elegir la opción para desarrolladores (“Developer Default”) que incluye las herramientas más interesantes desde el punto de vista de un desarrollador: MySQL Server, MySQL Shell, MySQL Router, MySQL Workbench, MySQL for Excel, MySQL for Visual Studio, MySQL Connectors, ejemplos, tutoriales y documentación.

En el proceso de configuración seleccionar las siguientes opciones:

1. Standalone MySQL Server
2. Type and networking
   a) Development Computer
   b) TCP/IP,
   c) Puerto 3306
   d) X Protocol Port 33060
   e) Open Windows Firewall ports
3. Authentication method
   a) Use strong Password Encription
4. Account and roles
   a) MySQL root password – escoja una clave de administrador, por ejemplo, root
5. Windows Service
   a) Configure MySQL Server as Windows Service
   b) Windows Service Name (valor por defecto MySQL80)
   c) Start the MySQL Server at system startup
   d) Run Windows service as… Standard System Account
6. Apply configuration. En este paso se aplican las configuraciones seleccionadas.
7. MySQL Router Configuration – Valores por defecto
8. Samples and Examples
   a) Introducir clave usuario root (la creada en el paso 4.a) y pulsar check
9. Apply Configuration – Pulsar Execute para que se instalen las bases de datos de ejemplo
10. Finaliza la instalación

Llegado este momento, su ordenador está preparado para manejar y manipular bases de dato de MySQL. Una de las formas de hacerlo sería mediante línea de comandos. Debido a que esto puede ser una tarea ardua en esta práctica se presentarán las herramientas gráficas que hemos obtenido en el proceso de instalación.

En versiones anteriores existían varias herramientas, con funciones específicas cada una de ellas, que permitían realizar la gestión del servidor MySQL. Actualmente todas esas herramientas se encuentran integradas en una sola: MySQL Workbench.

Entre todas las herramientas y opciones que disponemos en MySQL Workbench hay que destacar:

1. **Manage Server Connections:** Gestor de conexiones al servidor. Permite definir conexiones al servidor local o a servidores remotos, utilizando distintos usuarios y modos de conexión.
2. **MySQL Model (Ctrl + N):** Permite diseñar un modelo de base de datos gráficamente y obtener posteriormente la implementación del modelo como una base de datos de MySQL.
3. **Reverse Engineer (Ctrl + R):** Herramienta de ingeniería inversa que construye y documenta un diagrama E/R a partir de una base de datos existente.
4. **Migration Wizard:** Asistente para la migración de una base de datos desde otro SGBD (Oracle, PostgreSQL, …) a MySQL
5. **Connect to database (Ctrl + U):** Permite seleccionar una conexión guardada para utilizarla. Una vez abierta la conexión dispondremos de distintas opciones:
   - **MANAGEMENT**
     i. **Server status:** Monitor de estado general del servidor.
     ii. **Client connection:** Monitor de conexiones cliente.
     iii. **Users and privilegies:** Usuarios y privilegios.
     iv. **Status and system variables:** Permite consultar el valor de las variables de estado y de sistema de MySQL.
     v. **Data Export:** Herramienta para exportación de datos. Muy útil para migrar datos a otra instancia de MySQL.
     vi. **Data Import/Restore:** Herramienta de importación de datos. La utilizaremos para restaurar las exportaciones realizadas con la herramienta Data Export.
   - **INSTANCE**
     i. **Startup / Shutdown:** Arranque y parada del servidor.
     ii. **Server logs:** Acceso a los logs generados durante la ejecución del servidor.
     iii. **Options file:** Acceso al fichero de configuración del servidor.
   - **PERFOMANCE**
     i. **Dashboard:** Cuadro de mandos con información sobre el rendimiento del servidor.
     ii. **Perfomance reports:** Acceso a informes predefinidos sobre distintos aspectos del rendimiento del servidor.
     iii. **Perfomance Schema Setup:** Completa herramienta que permite recopilar información sobre el rendimiento del servidor a lo largo del tiempo para obtener una información precisa de lo que ocurre en el servidor. Sirve para optimizar el diseño de bases de datos, consultas y transacciones.
   - **SCHEMAS -** Acceso a cada uno de las bases de datos o esquemas existentes en la instancia del SGBD a la que estamos conectados.
   - **QUERY TAB –** Cada una de las pestañas que se abren (Ctrl + T) con objeto de ejecutar sentencias SQL en la base de datos que se seleccione.
   - **NEW SCHEMA –** Pestaña que se abre al seleccionar la opción “New Schema” en la barra de herramientas. Permite crear una nueva base de datos (esquema) sin datos proporcionando simplemente su nombre.
   - **CREATE A NEW TABLE –** Pestaña que se abre al selecciona la opción “Create a new table in the active schema”. Permite diseñar una tabla nueva en la base de datos seleccionada. Hay que definir campo a campo con su tipo de datos y los parámetros que se necesiten: valor por defecto, permitir o no valores nulos, clave primaria, etc

También existen herramientas web para administrar y gestionar MySQL. Dentro de éstas, la herramienta más usada es PhpMyAdmin, la cual requiere que la máquina tenga instalado PHP y un servidor web. Es sencillo instalar todo el software necesario instalando el paquete XAMPP.

Nótese que las herramientas presentadas no son las únicas existentes que pueden ser utilizadas con tal fin. Por ejemplo, podríamos usar el mismo entorno de desarrollo NetBeans para conectarnos a una base de datos almacenada en MySQL y poder ejecutar consultas SQL en la misma. Para ello es necesario seguir los siguientes pasos:

1. Descargar el driver de conexión de https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-8.0.15.zip
2. Descomprimir el archivo y copiar el archivo mysql-connector-java-8.0.15.jar a la ruta de librerías externas de java. Esta ruta es distinta según la versión del JDK que se esté utilizando. Sirva a modo de ejemplo la siguiente: C:\Program Files\Java\jdk1.8.0_161\jre\lib\ext
3. Definir una nueva conexión a MySQL en NetBeans

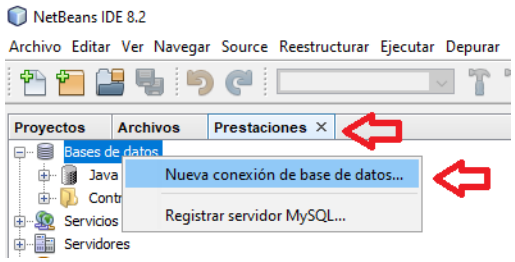

Figura 1. Creación de la conexión

4. Especificar la ruta del driver MySQL (controlador connector/J)

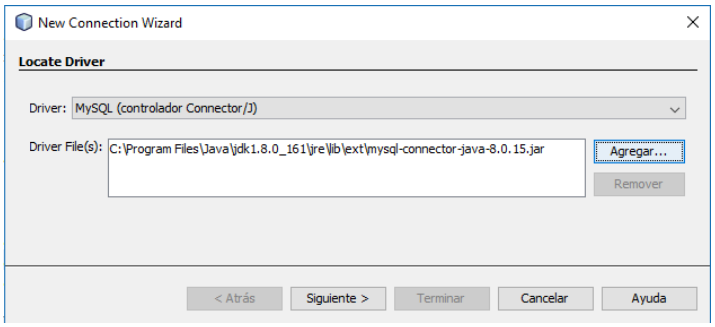

Figura 2. Ruta del driver

5. Definir los parámetros de la conexión, en particular el usuario y contraseña con el que acceder y la URL de JDBC. Respecto a esta última comentar que en algunas ocasiones hay inconvenientes con respecto a la zona horaria del servidor y hay que modificar el valor que aparece por defecto por este otro valor: jdbc:mysql://localhost:3306/mysql?zeroDateTimeBehavior=convertToNull&serverTimezone=UTC

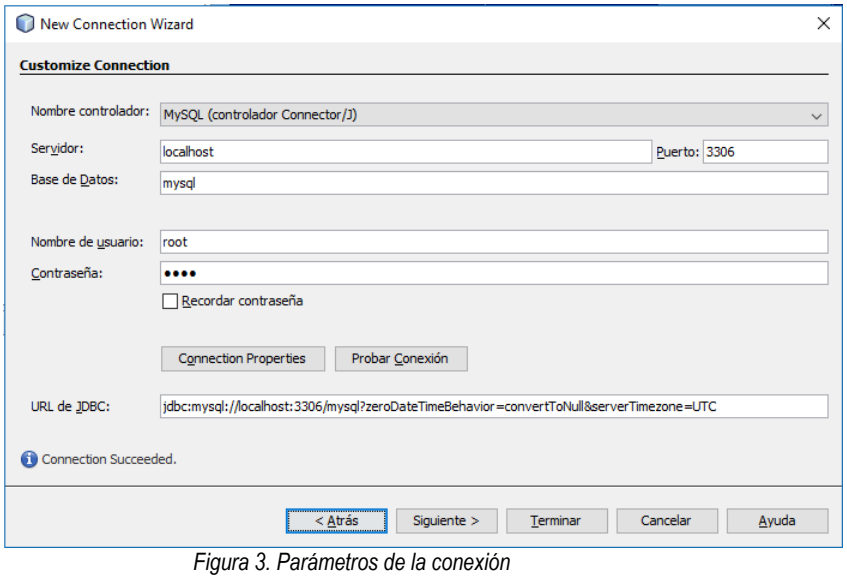

Figura 3. Parámetros de la conexión

6. En el resto de pasos del asistente es suficiente con dejar los valores por defecto. Al finalizar el asistente quedará almacenada en NetBeans la nueva conexión, que al desplegarla permite examinar el contenido de la base de datos y trabajar con ella.

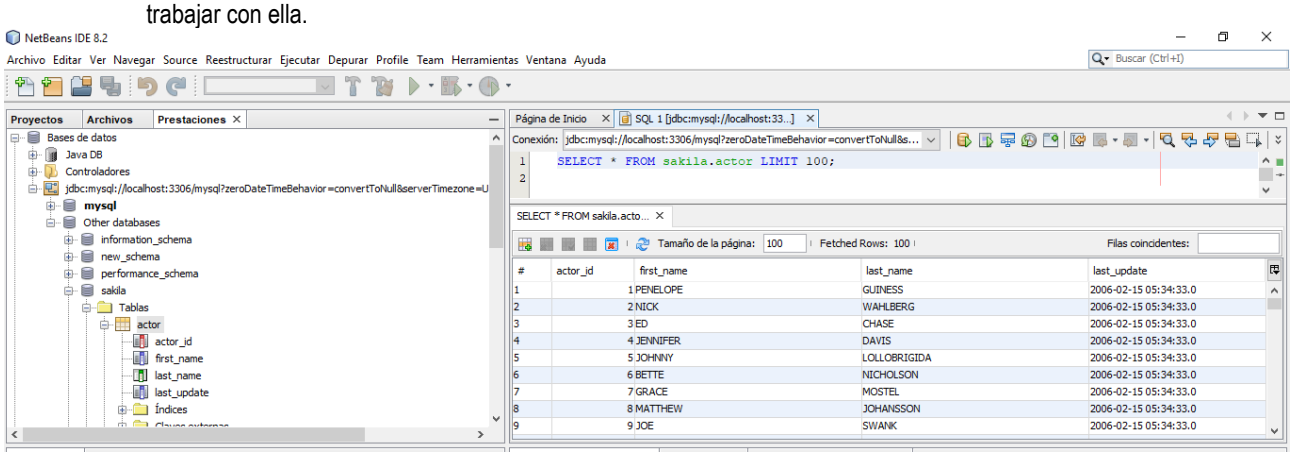

Figura 4. Acceso a MySQL desde NetBeans

Por otro lado, también podríamos vincular Access con MySQL para poder insertar registros a las distintas tablas de forma cómoda. Para ello, previamente, se debería preparar el ordenador para que soporte conexiones ODBC.

1. Descargar la versión instalable del driver ODBC de 32 bits desde la página de MySQL en la URL: https://dev.mysql.com/get/Downloads/Connector-ODBC/8.0/mysql-connector-odbc-8.0.15-win32.msi
2. Proceder a la instalación del driver con el fichero descargado.
3. Ejecutar la herramienta administrativa de Windows “Orígenes de datos ODBC (32 bits)”
4. En la pestaña “DNS de usuario” se debe agregar un nuevo origen de datos.
5. Seleccionar el driver “MySQL ODBC 8.0 Unicode Driver”

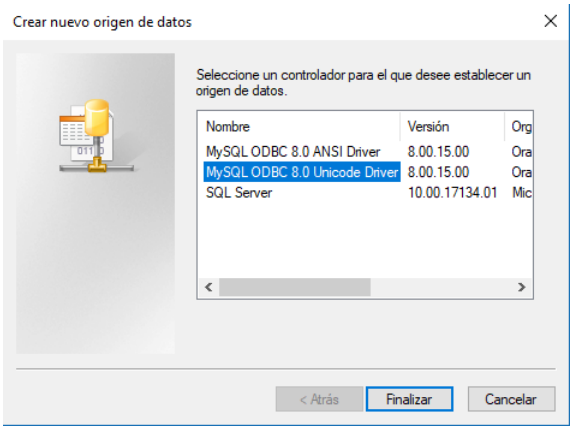

Figura 5. Selección de driver ODBC

6. Definir los parámetros de la conexión

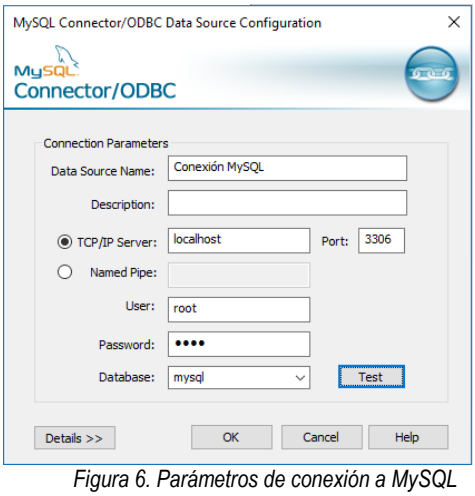

Figura 6. Parámetros de conexión a MySQL

7. Desde Access se debe agregar el nuevo origen de datos a cualquier base de datos existente o una nueva.

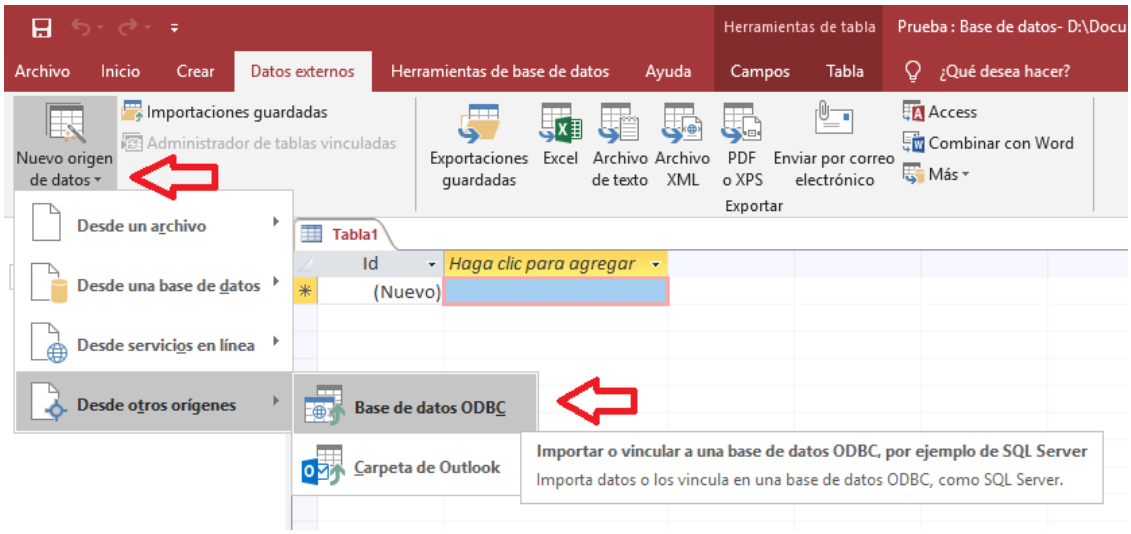

Figura 7. Agregar origen de datos

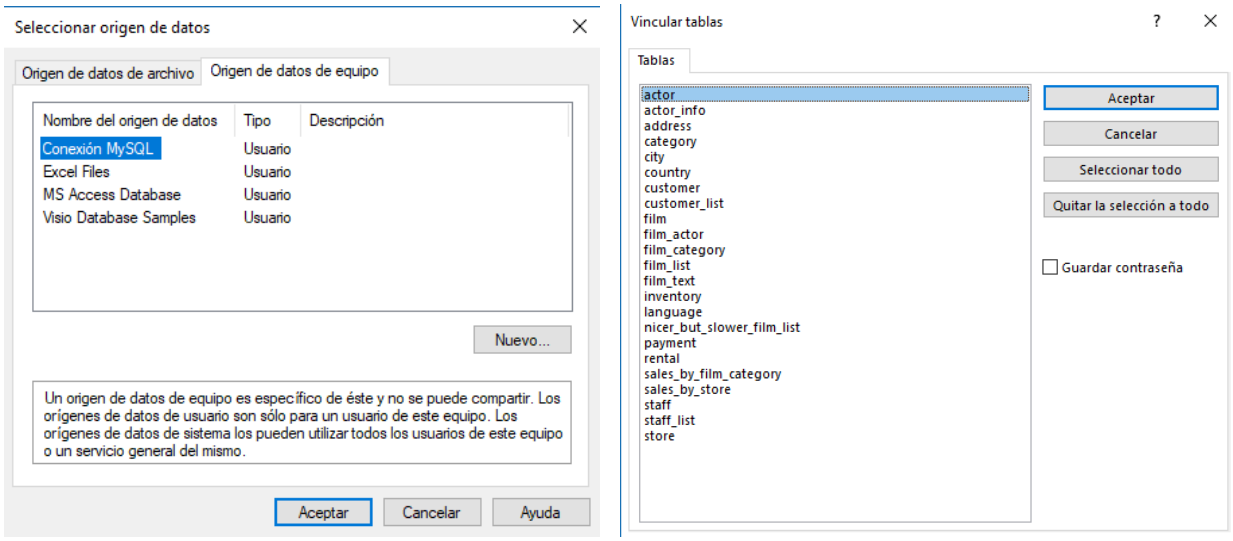

Figura 8. Selección de conexión y tabla a vincular

8. Vincular al origen de datos creando una tabla vinculada por cada tabla a la que necesitemos acceder.

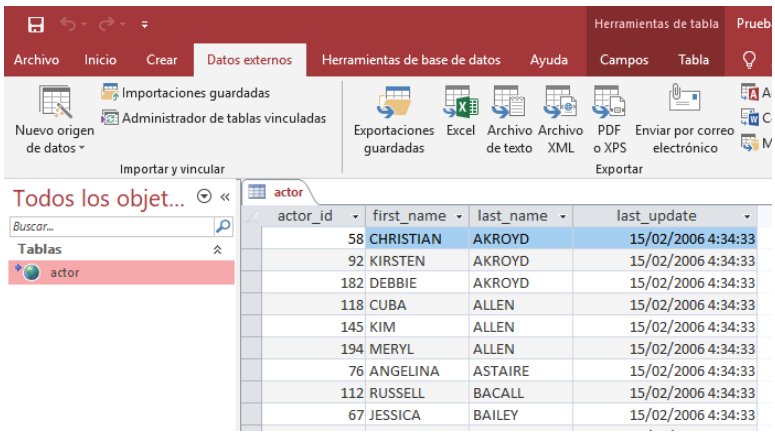

Figura 9. Tabla vinculada mediante ODBC en Access

2 Gestor de Base de Datos MySQL

MySQL, el sistema de gestión de bases de datos SQL Open Source más popular, lo desarrolla, distribuye y soporta Oracle Corporation. Actualmente se distribuye con licencia dual: Licencia Pública General (GPL) en el caso de la versión Cummunity Edition / Licencia Comercial en el caso de la versión Enterprise Edition.

El sitio web MySQL (proporciona la última información sobre MySQL, de la que se destaca:

- ***MySQL es un sistema de gestión de bases de datos***

  Una base de datos es una colección estructurada de datos. Puede ser cualquier cosa, desde una simple lista de compra a una galería de pintura o las más vastas cantidades de información en una red corporativa. Para añadir, acceder, y procesar los datos almacenados en una base de datos, necesita un sistema de gestión de base de datos como MySQL Server. Al ser los computadores muy buenos en tratar grandes cantidades de datos, los sistemas de gestión de bases de datos juegan un papel central en computación, como aplicaciones autónomas o como parte de otras aplicaciones.

- ***MySQL es un sistema de gestión de bases de datos relacionales***

  Una base de datos relacional almacena datos en tablas separadas en lugar de poner todos los datos en un gran almacén. Esto añade velocidad y flexibilidad. La parte SQL de "MySQL" se refiere a "Structured Query Language". SQL es el lenguaje estandarizado más común para acceder a bases de datos y está definido por el estándar ANSI/ISO SQL. El estándar SQL ha evolucionado desde 1986 y existen varias versiones.

- ***MySQL software es Open Source.***

  Open Source significa que es posible para cualquiera usar y modificar el software. Cualquiera puede bajar el software MySQL desde internet y usarlo sin pagar nada. Si lo desea, puede estudiar el código fuente y cambiarlo para adaptarlo a sus necesidades.

- ***El servidor de base de datos MySQL es muy rápido, fiable y fácil de usar.***

  MySQL Server se desarrolló originalmente para tratar grandes bases de datos mucho más rápido que soluciones existentes y ha sido usado con éxito en entornos de producción de alto rendimiento durante varios años. MySQL Server ofrece hoy en día una gran cantidad de funciones. Su conectividad, velocidad, y seguridad hacen de MySQL Server altamente apropiado para acceder a bases de datos en Internet

- ***MySQL Server trabaja en entornos cliente/servidor o incrustados***

  El software de bases de datos MySQL es un sistema cliente/servidor que consiste en un servidor SQL multi-threaded que trabaja con diferentes backends, programas y bibliotecas cliente, herramientas administrativas y un amplio abanico de interfaces de programación para aplicaciones (APIs).
  También proporcionan el MySQL Server como biblioteca incrustada multi-threaded que se puede enlazar en la aplicación para obtener un producto más pequeño, rápido y fácil de administrar.

3 JDBC: el cliente java para acceder a MySql

3.1 Introducción

En el mundo Windows, JDBC es para Java lo que ODBC es para Windows. Windows en general no sabe nada acerca de las bases de datos, pero define el estándar ODBC consistente en un conjunto de primitivas que cualquier driver o fuente ODBC debe ser capaz de entender y manipular. Los programadores que a su vez deseen escribir programas para manejar bases de datos genéricas en Windows utilizan las llamadas ODBC.

Con JDBC ocurre exactamente lo mismo: JDBC es una especificación de un conjunto de clases y métodos de operación que permiten a cualquier programa Java acceder a sistemas de bases de datos de forma homogénea. Lógicamente, al igual que ODBC, la aplicación de Java debe tener acceso a un driver JDBC adecuado. Este driver es el que implementa la funcionalidad de todas las clases de acceso a datos y proporciona la comunicación entre el API JDBC y la base de datos real.

La necesidad de JDBC, a pesar de la existencia de ODBC, viene dada porque ODBC es un interfaz escrito en lenguaje C, que al no ser un lenguaje portable, haría que las aplicaciones Java también perdiesen la portabilidad. Además, ODBC tiene el inconveniente de que se ha de instalar manualmente en cada máquina; al contrario que los drivers JDBC, que al estar escritos en Java son automáticamente instalables, portables y seguros.

Toda la conectividad de bases de datos de Java se basa en sentencias SQL, por lo que se hace imprescindible un conocimiento adecuado de SQL para realizar cualquier clase de operación de bases de datos. Aunque, afortunadamente, casi todos los entornos de desarrollo Java ofrecen componentes visuales que proporcionan una funcionalidad suficientemente potente sin necesidad de que sea necesario utilizar SQL, para usar directamente el JDK es imprescindible. La especificación JDBC requiere que cualquier driver JDBC sea compatible con al menos el nivel «de entrada» de ANSI SQL 92 (ANSI SQL 92 Entry Level).

En esta asignatura, el driver JDBC que usaremos para manejar MySql se encuentra en: https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-8.0.15.zip o en el curso virtual de la asignatura (mysql-connector-java-5.1.5.zip);

3.2 El API de JDBC

JDBC define ocho interfaces para operaciones con bases de datos, de las que se derivan las clases correspondientes. La figura 1, en formato OMT, con nomenclatura UML, muestra la interrelación entre estas clases según el modelo de objetos de la especificación de JDBC.

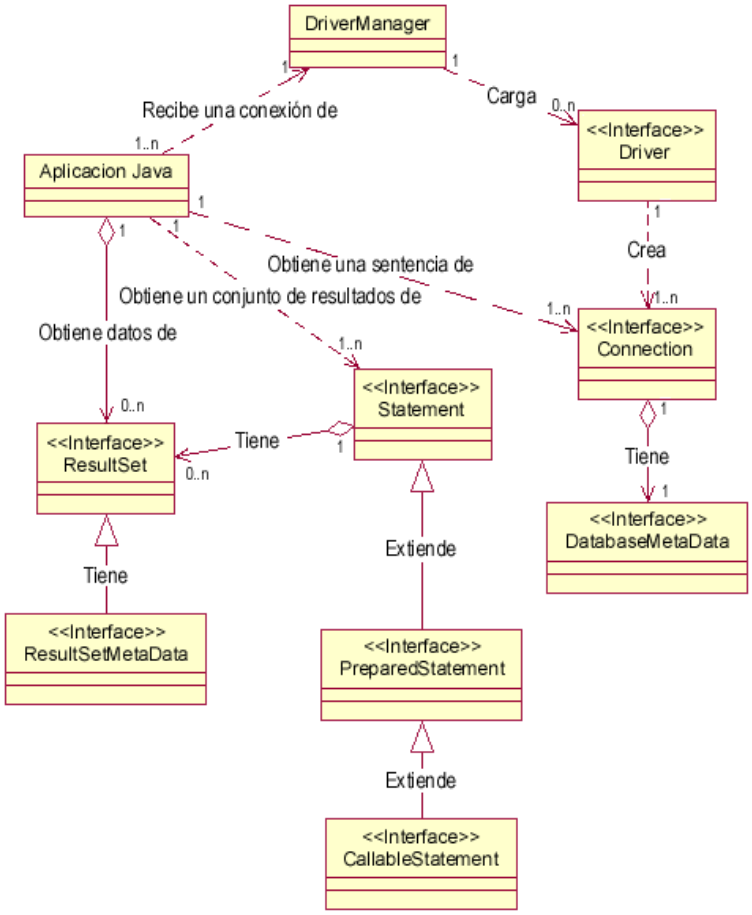

Figura 10. Modelo de objetos de la especificación de JDBC

La clase que se encarga de cargar inicialmente todos los drivers JDBC disponibles es DriverManager. Una aplicación puede utilizar DriverManager para obtener un objeto de tipo conexión, Connection, con una base de datos. La conexión se especifica siguiendo una sintaxis basada en la especificación más amplia de los URL, de la forma:

jdbc:subprotocolo//servidor:puerto/base de datos

Por ejemplo, si se utiliza MySql el nombre del subprotocolo será *mysql.* En algunas ocasiones es necesario identificar aún más el protocolo. Por ejemplo, si se usa el puente JDBC-ODBC no es suficiente con jdbc:odbc, ya que pueden existir múltiples drivers ODBC, y en este caso, hay que especificar aún más, mediante jdbc:odbc:fuente de datos.

Una vez que se tiene un objeto de tipo Connection, se pueden crear sentencias, statements o ejecutables. Cada una de estas sentencias puede devolver cero o más resultados, que se devuelven como objetos de tipo ResultSet.

En la figura 2 contiene el diagrama que relaciona las cuatro clases principales que va a usar cualquier programa Java con JDBC, y representa el esqueleto de cualquiera de los programas que se desarrollan para atacar a bases de datos.

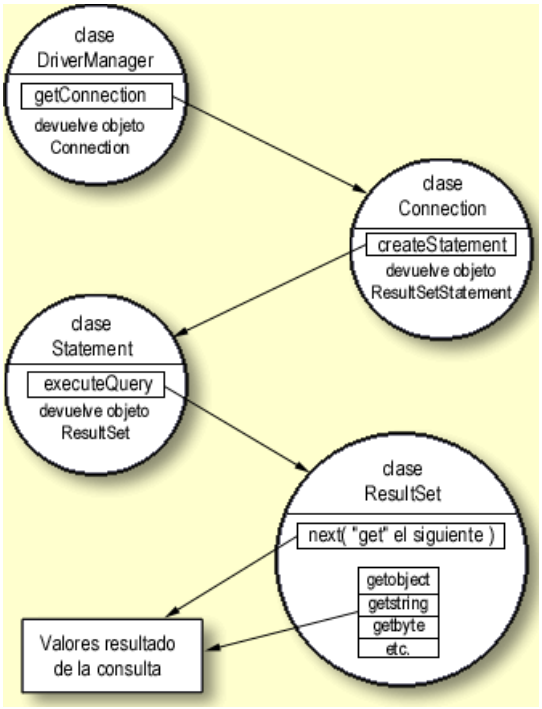

Figura 11. Diagrama del uso básico de JDBC en una aplicación Java

Seguidamente, pasaremos a describir cada uno de los pasos necesarios para poder manipular una base de datos MySql desde Java usando JDBC.

3.2.1 Cargar el controlador

Para trabajar con el API JDBC se tiene que importar el paquete **java.sql** (`import java.sql.*;`). En este paquete se definen los objetos que proporcionan toda la funcionalidad que se requiere para el acceso a bases de datos.

El siguiente paso después de importar el paquete java.sql consiste en cargar el controlador JDBC, es decir un objeto **Driver** específico para una base de datos que define cómo se ejecutan las instrucciones para esa base de datos en particular.

Hay varias formas de hacerlo, pero la más sencilla es utilizar el método **forName()** de la clase **Class:**

```
Class.forName("Controlador JDBC");
```

Para el caso particular del controlador para MySQL, Connector/J, se tiene lo siguiente:

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

Debe tenerse en cuenta que el método estático forName() definido por la clase Class genera un objeto de la clase especificada. Cualquier controlador JDBC tiene que incluir una parte de iniciación estática que se ejecuta cuando se carga la clase. En cuanto el cargador de clases carga dicha clase, se ejecuta la iniciación estática, que pasa a registrarse como un controlador JDBC en el **DriverManager**.

Es decir, el siguiente código:

```
Class.forName("Controlador JDBC");
```

Es equivalente a:

```
Class c = Class.forName("Controlador JDBC");
Driver driver = (Driver)c.newInstance();
DriverManager.registerDriver(driver);
```

Algunos controladores no crean automáticamente una instancia cuando se carga la clase. Si forName() no crea por sí solo una instancia del controlador, se tiene que hacer esto de manera explícita:

```
Class.forName("Controlador JDBC").newInstance();
```

De nuevo, para el Connector/J:

```
Class.forName("com.mysql.cj.jdbc.Driver").newInstance();
```

3.2.2 Establecer la conexión

Una vez registrado el controlador con el DriverManager, se debe especificar la fuente de datos a la que se desea acceder. En JDBC, una fuente de datos se especifica por medio de un URL con el prefijo de protocolo **jdbc:** , la sintaxis y la estructura del protocolo es la siguiente:

jdbc:{subprotocolo}:{subnombre}

El {subprotocolo} expresa el tipo de controlador, normalmente es el nombre del sistema de base de datos, como db2, oracle o mysql. El contenido y la sintaxis de {subnombre} dependen del {subprotocolo}, pero en general indican el nombre y la ubicación de la fuente de datos. Por ejemplo, para acceder a una base de datos denominada *productos* en un sistema *Oracle* local, el URL sería de la siguiente manera:

```
String url = "jdbc:oracle:productos";
```

Si la misma base de datos estuviera en un sistema *DB2* en un servidor llamado *dbserver.ibm.com*, el URL sería el siguiente:

```
String url = "jdbc:db2:dbserver.ibm.com/productos";
```

El formato general para conectarse a MySQL es:

jdbc:mysql://[servidor][:puerto]/[base_de_datos][?param1=valor1][param2=valor2]...

Si queremos acceder de manera local a una base de datos llamada “MiniAgenda”, el URL sería :

```
String url = "jdbc:mysql://localhost/MiniAgenda?serverTimezone=UTC";
```

Una vez que se ha determinado la URL, se puede establecer una conexión a una base de datos. El objeto **Connection** es el principal objeto utilizado para proporcionar un vínculo entre las bases de datos y una aplicación Java. Connection proporciona métodos para manejar el procesamiento de transacciones, para crear objetos y ejecutar instrucciones SQL y para crear objetos para la ejecución de procedimientos almacenados.

Se puede emplear tanto el objeto Driver como el objeto DriverManager para crear un objeto Connection. Se utiliza el método **connect()** para el objeto Driver, y el método **getConnection()** para el objeto DriverManager.

El objeto Connection proporciona una conexión estática a la base de datos. Esto significa que hasta que se llame en forma explícita a su método **close()** para cerrar la conexión o se destruya el objeto Connection, la conexión a la base de datos permanecerá activa.

La manera más usual de establecer una conexión a una base de datos es invocando el método getConnection() de la clase DriverManager. A menudo, las bases de datos están protegidas con nombres de usuario (login) y contraseñas (password) para restringir el acceso a las mismas. El método getConnection() permite que el nombre de usuario y la contraseña se pasen también como parámetros.

```
String login = "usuario";
String password = "clave";
Connection conn = DriverManager.getConnection(url,login,password);
```

3.2.3 Problemas en la conexión

En toda aplicación de bases de datos con MySQL es indispensable poder establecer la conexión al servidor para posteriormente enviarle las consultas. Los programas en Java no son la excepción. El siguiente código nos servirá para verificar que podemos establecer una conexión a nuestra base de datos MiniAgenda.

```
import java.sql.*;

public class TestConnection
{
   static String bd = "MiniAgenda";
   static String login = "usuario";


   static String password = "clave";
   static String url = "jdbc:mysql://localhost/"+bd+"?serverTimezone=UTC";


   public static void main(String[] args) throws Exception
   {
      Connection conn = null;

      try
      {
         Class.forName("com.mysql.cj.jdbc.Driver").newInstance();

         conn = DriverManager.getConnection(url,login,password);

         if (conn != null)
         {
            System.out.println("Conexión a base de datos "+url+" ... Ok");
            conn.close();
         }
      }
      catch(SQLException ex)
      {
         System.out.println(ex);
      }
      catch(ClassNotFoundException ex)
      {
         System.out.println(ex);
      }

   }
}
```

A continuación, se listan algunas de las salidas que se pueden obtener al ejecutar el programa anterior:

```
Conexión a base de datos jdbc:mysql://localhost/MiniAgenda ... Ok
-- Todo funciona bien =)

java.lang.ClassNotFoundException: com.mysql.cj.jdbc.Driver
-- Verificar que se ha agregado el driver al proyecto

java.sql.SQLException: Invalid authorization specification: Access denied for
user: 'bingo@localhost' (Using password: YES)
-- El login o el password proporcionados no nos permiten el acceso al servidor.

java.sql.SQLException: No suitable driver
-- Probablemente se ha escrito de forma incorrecta el URL para la base de datos.

java.sql.SQLException: General error: Access denied for user: 'bingo@localhost'
to database 'agendota'
-- Probablemente se ha escrito de manera incorrecta el nombre de la base de datos.
```

3.2.4 Creación de sentencias

Como se mencionó en el apartado anterior, el objeto Connection permite establecer una conexión a una base de datos. Para ejecutar instrucciones SQL y procesar los resultados de las mismas, debemos hacer uso de un objeto **Statement**. Los objetos Statement envían comandos SQL a la base de datos, que pueden ser de cualquiera de los tipos siguientes:

- Un comando de definición de datos como CREATE TABLE o CREATE INDEX.
- Un comando de manipulación de datos como INSERT, DELETE o UPDATE.
- Un sentencia SELECT para consulta de datos.

Un comando de manipulación de datos devuelve un contador con el número de filas (registros) afectados, o modificados, mientras una instrucción SELECT devuelve un conjunto de registros denominado conjunto de resultados (*result set*). La interfaz Statement no tiene un constructor, sin embargo, podemos obtener un objeto Statement al invocar el método **createStatement()** de un objeto Connection.

```
conn = DriverManager.getConnection(url,login,pasword);
Statement stmt = conn.createStatement();
```

Una vez creado el objeto Statement, se puede emplear para enviar consultas a la base de datos usando los métodos execute(), executeUpdate() o executeQuery(). La elección del método depende del tipo de consulta que se va a enviar al servidor de bases de datos:

Método	Descripción
execute( )	Se usa principalmente cuando una sentencia SQL regresa varios conjuntos de resultados. Esto ocurre principalmente cuando se está haciendo uso de procedimientos almacenados.
executeUpdate( )	Este método se utiliza con instrucciones SQL de manipulación de datos tales como INSERT, DELETE o UPDATE.
executeQuery( )	Se usa en las instrucciones del tipo SELECT.

Es recomendable que se cierren los objetos Connection y Statement que se hayan creado cuando ya no se necesiten. Lo que sucede es que cuando en una aplicación en Java se están usando recursos externos, como es el caso del acceso a bases de datos con el API JDBC, el recolector de basura de Java (*garbage collector*) no tiene manera de conocer cuál es el estado de esos recursos, y por lo tanto, no es capaz de liberarlos en el caso de que ya no sean útiles. Lo que sucede en estos casos es que pueden quedar almacenados en memoria grandes cantidades de recursos relacionados con la aplicación de bases de datos que se está ejecutando. Es por esto que se recomienda que se cierren de manera explícita los objetos Connection y Statement.

De manera similar a Connection, la interfaz Statement tiene un método **close()** que permite cerrar de manera explícita un objeto Statement. Al cerrar un objeto Statement se liberan los recursos que están en uso tanto en la aplicación Java como en el servidor de bases de datos.

```
Statement stmt = conn.createStatement();
....
stmt.close();
```

3.2.5 Ejecución de consultas

Cuando se ejecutan sentencias SELECT usando el método executeQuery(), se obtiene como respuesta un conjunto de resultados, que en Java es representado por un objeto ResultSet.

```
Statement stmt = conn.createStatement();
ResultSet res  = stmt.executeQuery("SELECT * FROM contactos");
```

Se puede pensar en un conjunto de resultados como una tabla (filas y columnas) en la que están los datos obtenidos por una sentencia SELECT:

Id_contacto	nombre	telefono	email
1	Pepe Pecas	8282-7272	pepe@hotmail.net
2	Laura Zarco	2737-9212	lauris@micorreo.com
3	Juan Penas	7262-8292	juan@correo.com.cx

La información del conjunto de resultados se puede obtener usando el método next() y los diversos métodos getXXX() del objeto ResultSet. El método next() permite moverse fila por fila a través del ResultSet, mientras que los diversos métodos getXXX() permiten acceder a los datos de una fila en particular.

Los métodos getXXX() toman como argumento el índice o nombre de una columna, y regresan un valor con el tipo de datos especificado en el método. Así por ejemplo, getString() devolverá una cadena, getBoolean() un booleano y getInt() un entero. Cabe mencionar que estos métodos deben tener una correspondencia con los tipos de datos que se tienen en el ResultSet, y que son a las vez los tipos de datos provenientes de la consulta SELECT en la base de datos, sin embargo, si únicamente se desean mostrar los datos se puede usar getString() sin importar el tipo de dato de la columna. Seguidamente, se muestra la relación entre tipos Java y Sql estándar

Tipo Java	Tipo SQL
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
java.math.BigDecimal	NUMERIC
String	VARCHAR o LONGVARCHAR
byte[]	VARBINARY o LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP

Por otra parte, si en estos métodos se utiliza la versión que toma el índice de la columna, se debe considerar que los índices empiezan a partir de 1, y no en 0 (cero) como en los arrays, los vectores, y algunas otras estructuras de datos de Java.

Existe un objeto ResultSetMetaData que proporciona varios métodos para obtener información sobre los datos que están dentro de un objeto ResultSet. Estos métodos permiten entre otras cosas obtener de manera dinámica el número de columnas en el conjunto de resultados, así como el nombre y el tipo de cada columna.

```
ResultSet res  = stmt.executeQuery("SELECT * FROM contactos");
ResultSetMetaData metadata = res.getMetaData();
```

3.3 Transacciones en JDBC

Las transacciones permiten ejecutar bloques de código con las propiedades ACID (Atomicity-Consistency-Isolation-Durability).

Por defecto, cuando se crea una conexión en JDBC está en modo auto-commit: cada Statement y PreparedStatment que se ejecute sobre la base de datos abierta, irá en su propia transacción. Si se desea ejecutar varias queries en una misma transacción es preciso:

- Deshabilitar el modo auto-commit de la conexión (`connection.setAutocomit(false)`)
- Lanzar las queries
- Terminar con connection.commit() si todo va bien, o connection.rollback() en otro caso

En principio, las transacciones garantizan que no hay problemas si dos o más transacciones modifican datos comunes concurrentemente (propiedad I en ACID). Aunque, en realidad, dependiendo de la estrategia de bloqueo que se use, pueden darse los siguientes problemas:

- ***Dirty-reads:*** una transacción lee datos actualizados por otra transacción pero todavía no comprometidos (figura 3)
- ***Non-repeatable reads:*** una transacción relee un dato que ha cambiado "mágicamente" desde la primera lectura (figura 4)
- ***Phantom reads:*** una transacción relanza una query de consulta y obtiene "mágicamente" más filas la segunda vez (figura 5)

Transacción 1	Transacción 2
begin	
SELECT X /* X = 0 */	
X = X + 10 /* X = 10 */	
UPDATE X /* X = 10 en BD pero no comprometido */	**begin**
	SELECT X /* X = 10 */
rollback /* X = 0 en BD */	**X = X + 10** /* X = 20 */
	UPDATE X /* X = 20 en BD pero no comprometido */
	commit /* X = 20 en BD y comprometido */

Figura 3

Transacción 1	Transacción 2
begin	
SELECT X /* X = 0 */	**begin**
	X = 20
	UPDATE X /* X = 20 en BD pero no comprometido */
	commit /* X = 20 en BD y comprometido */
SELECT X /* X = 20 */	
...	
commit	

Figura 4

Transacción 1	Transacción 2
begin	
SELECT WHERE condition	**begin**
	INSERT /* Inserta nuevas filas en BD (no comprometidas), algunas cumpliendo "condition" */
	commit /* Inserciones comprometidas */
SELECT WHERE condition /* Ahora devuelve más filas */	
...	
commit	

Figura 5

La clase Connection, proporciona el método setTransactionIsolation, que permite especificar el nivel de aislamiento deseado:

- TRANSACTION_NONE: transacciones no soportadas
- TRANSACTION_READ_UNCOMMITED: pueden ocurrir "dirty reads", "non-repeatable reads" y "phantom reads"
- TRANSACTION_READ_COMMITED: pueden ocurrir "non-repeatable reads" y "phantom reads"
- TRANSACTION_REPEATABLE_READ: pueden ocurrir "phantom reads"
- TRANSACTION_SERIALIZABLE: elimina todos los problemas de concurrencia

Por defecto, dependiendo del driver, el nivel de aislamiento suele ser TRANSACTION_READ_COMMITED

3.4 Ejemplo Completo

```
/*
Esta es una pequeña aplicación que muestra cómo obtener los
datos de una tabla usando una consulta SELECT. Se ejecuta
desde la línea de comandos del sistema operativo y los datos
obtenidos son mostrados en la consola.
*/

import java.sql.*;

public class MostrarMiniAgenda
{
   static String login = "root";
   static String password = "root";
   static String url = "jdbc:mysql://localhost/MiniAgenda?serverTimezone=UTC";

   public static void main(String[] args) throws Exception
   {
      Connection conn = null;
      try
      {
         Class.forName("com.mysql.cj.jdbc.Driver").newInstance();
         conn = DriverManager.getConnection(url,login,password);

         if (conn != null)
         {
            Statement stmt = conn.createStatement();
            ResultSet res  = stmt.executeQuery("SELECT * FROM contactos");

            System.out.println("\nNOMBRE \t\t EMAIL \t\t\t TELEFONO \n");


            while(res.next())
            {
               String nombre  = res.getString("nombre");
               String email   = res.getString("email");
               String telefono= res.getString("telefono");

               System.out.println(nombre +" \t "+email+" \t "+telefono);
            }

            res.close();
            stmt.close();
            conn.close();
         }
       }
       catch(SQLException ex)
       {
          System.out.println(ex);
       }
       catch(ClassNotFoundException ex)
       {
          System.out.println(ex);
       }
   }
}
```

Bibliografía

Básicas:

- Documentación de Mysql: http://dev.mysql.com/doc/
    - MySql JDBC: https://dev.mysql.com/doc/connector-j/8.0/en/
    - MySql Workbench: https://dev.mysql.com/doc/workbench/en/

Experimentos

Con estos experimentos se pretende realizar todos los pasos necesarios para desarrollar una aplicación que acceda a MySQL. Concretamente, llevaremos a cabo la aplicación explicada a lo largo de la práctica. Para ello realice los siguientes experimentos en orden:

E1. (15 min) Cargue la base de datos MiniAgenda (miniagenda.sql) en el gestor de MySQL que se encuentra en su máquina (localhost) y cree un usuario con nombre "usuario" y clave "clave", asignándole todos los permisos para la base de datos creada. Haga uso de la herramienta MySQL Workbench.

E2. (10 min) Cree un proyecto en netbeans e inserte el driver de conexión de Java a MySQL (mysql-connector-java-8.0.15.jar)

E3. (15 min) Cree una clase con el código descrito en el apartado 3.4 y compruebe que todo funciona correctamente.

Ejercicios

EJ1. (10 min) Cree en la base de datos MiniAgenda una tabla denominada "usuarios". La tabla contendrá dos campos; login y password. Un posible script de creación sería:

```
CREATE TABLE usuarios ("
            + "login VARCHAR(10) NOT NULL,"
            + "password VARCHAR(10) NOT NULL,"
            + "PRIMARY KEY (login)"
            + ");

INSERT INTO usuarios VALUES ('adrian', 'adrian');
INSERT INTO usuarios VALUES ('ernesto', 'ern123');
INSERT INTO usuarios VALUES ('juan', 'jua123');
INSERT INTO usuarios VALUES ('pedro', 'ped123');
```

EJ2. (15 min) Cree una clase, denominada ConexionSelect, que imprima por pantalla todos los valores de la tabla. La salida debería ser:

```
Conexion a BD establecida
Login Password
adrian adrian
ernesto ern123
juan jua123
pedro ped123
```

EJ3. (25 min) Cree una clase ConexionInsert que reciba como parámetros de ejecución el nombre y clave de un nuevo registro de la tabla y lo inserte. La llamada java ConexionInsert julio jul123 provocaría la siguiente salida:

```
Conexion a BD establecida
Se agrego el registro de manera exitosa
```

No olvide de gestionar las posibles excepciones.

EJ4. (40 min) Cree las clases correspondientes para realizar borrados y actualizaciones de los registros existentes.

Problema

Suponga que la biblioteca de su barrio necesita llevar un control básico de préstamos de sus libros. Para ello, se requiere una base de datos que almacene información acerca de los libros (título, autor y signatura) y los usuarios (dni, nombre, teléfono, población), manteniendo un registro activo de todos y cada uno de los préstamos (libro, usuario, fecha de préstamo y fecha de devolución).

Se pide:

- Diseñar y generar dicha base de datos en MySql
- Implementar en java las siguientes consultas:
    - Libros de un autor determinado (título y signatura).
    - Libros que se encuentran prestados (título del libro y dni y nombre del usuario que lo tiene prestado).
    - Libros que llevan prestados más de un mes (título del libro y dni y nombre del usuario que lo tiene prestado).
- Además, se requiere que el sistema pueda introducir nuevos préstamos y renovaciones de préstamos existentes

Datos de la Práctica

Autor del documento: Norberto Díaz Díaz (Marzo 2012).

Revisiones:

- Norberto Díaz Díaz (Abril 2013): Revisión de formato, estimación temporal y modificación de ejercicios.
- Norberto Díaz Díaz (Marzo 2015): Actualización del orden de la práctica.
- Carlos Alberto Rodríguez Parrales (Marzo 2019): Actualización versión MySQL 8.0
- Adrián Gil Gamboa (Marzo 2024): Actualización y mejora en la redacción del ejercicio 1.

Estimación temporal:

- Parte presencial: 120 minutos.
    - Experimentos: 35 minutos.
    - Ejercicios: 85 minutos.
- Parte no presencial: 210 minutos.
    - Lectura y estudio del guión y Bibliografía: 90 minutos
    - Problema: 120 minutos