
TEMA 5: ACCESO PROGRAMÁTICO

1. INTRODUCCIÓN

- **Sistema Gestor de Bases de Datos (SGBD):** su funcionamiento clásico es funcionar como un servidor al que se conectan clientes (aunque cliente y servidor podrían estar en la misma máquina, ejecutando la conexión a través de la máquina virtual). Algunos ejemplos podrían ser *MySQL*, *Oracle*, *SQL Server*, *BD2*, entre otros. También pueden funcionar como un sistema empotrado/embebido como, por ejemplo, *Apache Derby*.
- **Base de Datos (BDs):** es donde se guarda la información. Un mismo SGBD puede tener varias BDs diferentes.
- **Clientes:** máquina o software que se conecta al servidor creado por el SGBD. Pueden ser aplicaciones web (*Amazon*, *Ebay*, ...), aplicaciones de escritorio (*Gestión Biblioteca*, ...) o aplicaciones/clientes de administración que permiten controlar todas las BBDD (*MySql Workbench*, *MySQL Command Line Tool*).
- **MySQL Workbench:** herramienta con múltiples funcionalidades entre las que destacan el acceso, la gestión, la administración y el diseño de las BBDD.
- **Usuarios:** puede haber dos tipos diferentes:
 - Acceso web: usuario con privilegios sobre la BDs **SOLO** para leer.
 - Administrador: usuario con privilegios sobre la misma BDs con la opción de modificar.
- **Conexión:** todo cliente que desee conectarse a un SGBD debe especificar la BDs a la que quiere acceder y el usuario y la contraseña (credenciales de usuario).

2. CONECTORES

Una vez tenemos un SGBD operativo con un esquema de BDs funcionando (tablas y datos) tenemos varias opciones para interactuar con él:

- **Acceso mediante Workbench (o similar):** es un entorno clásico para DBAs (Administradores de Bases de Datos) en el que se deben definir las operaciones a realizar (DDL y/o DML). Es un entorno orientado a usuarios expertos en BBDD ya que proporciona acceso directo a ellas.
- **Acceso mediante aplicaciones ya desarrolladas:** son las usadas por los no expertos ya que se despreocupa de todo (ni siquiera tiene que saber qué es una BDs ni lo que hay detrás). Por este motivo es una interfaz muy amigable en la que se pueden insertar, consultar o borrar datos sin necesidad de saber SQL o la estructura lógica de un esquema de BDs.
- **Acceso programático:** es necesario el conocimiento de SQL y del lenguaje a desarrollar. Según el lenguaje y el SGBD usaremos un driver u otro. Además, se debe controlar tanto el propio acceso a la BDs como el manejo de los datos, las sentencias a ejecutar (para evitar el SQL Injection), etc. Por todo ello, es un entorno orientado a programadores.

Además, hay varios tipos de conectores. Los más importantes son:

- **ODBC (Conectividad Abierta de Bases de Datos):** permite acceder a cualquier dato desde cualquier aplicación sin importar el SGBD que use. Esto se logra gracias a una capa intermedia, llamada Nivel de Interfaz de Cliente SQL, entre la aplicación y el SGBD. El principal objetivo es traducir las consultas de la aplicación en comandos que el SGBD entienda. Su requisito fundamental es que tanto la aplicación como el SGBD sean compatibles con ODBC.
- **JDBC (Conector a la Base de Datos en Java):** es una API que permite ejecutar operaciones sobre BBDD en Java. Como cualquier programa Java es independiente del Sistema Operativo o de la BDs a la que se accede, ya que utiliza SQL como lenguaje de acceso y manipulación. La conexión a un SGBD en particular se realiza junto con el driver/biblioteca de conexión apropiado. Los componentes básicos del JDBC son los siguientes:
 - **API** → proporciona el acceso programático a través de determinadas clases y métodos. Puede ejecutar sentencias SQL, obtener resultados y propagar cambios. Los paquetes que se usan para ello son: `java.sql` (principalmente) y `javax.sql` (estando este último es desuso).
 - **Clase DriverManager** → sirve para cargar un driver.
 - **Clase Connection** → sirve para establecer conexiones con las BBDD.
 - **Clase Statement/PreparedStatement** → sirve para ejecutar sentencias SQL y enviarlas a la BDs.
 - **Clase ResultSet** → sirve para almacenar el resultado de la consulta.

Es importante resaltar que en este caso no se usa una capa intermedia ya que se conecta directamente a la BDs.

TIPO SQL	TIPO JAVA	MÉTODOS DE RESULTSET
INTEGER o INT	int	<code>getInt</code>
SMALLINT	short	<code>getShort</code>
FLOAT	float	<code>getFloat</code>
DOUBLE	double	<code>getDouble</code>
CHAR(n)	String	<code>getString</code>
VARCHAR(n)	String	<code>getString</code>
BOOLEAN o BOOL	boolean	<code>getBoolean</code>
DATE	<code>java.sql.Date</code>	<code>getDate</code>
TIME	<code>java.sql.Time</code>	<code>getTime</code>
TIMESTAMP	<code>java.sql.Timestamp</code>	<code>getTimestamp</code>
BLOB	<code>java.sql.Blob</code>	

Mapeo entre tipo de datos en SQL y Java y los métodos que se usan con ResultSet

3. FASES DE COMUNICACIÓN CON JDBC

a. **Cargar el Driver:**

```
String drv = "com.mysql.jdbc.Driver";
Class.forName(drv);
```

b. **Establecer conexión con la BDs:**

```
String serverAddress = "localhost:3306";
String bd = "sakila";
String user = "bdg";
String pass = "bdg";
String url = "jdbc:mysql://" + serverAddress + "/" + db;
Connection conn = DriverManager.getConnection(url, user, pass);
```

c. **Crear y ejecutar una sentencia SQL:**

(Clases *Statement* y *PreparedStatement*, métodos *executeQuery* y *executeUpdate*)

```
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery("SELECT * FROM actor");
```

Statement VS. PreparedStatement	
Utilizamos la clase <i>Statement</i> cuando las consultas que vamos a realizar no tienen parámetros (<i>P. Ej.</i> “ <i>SELECT * FROM actor;</i> ”) o cuando tienen parámetro fijo (<i>P. Ej.</i> “ <i>SELECT * FROM actor where first_name=“Arnold”;</i> ” y siempre es Arnold)	Utilizamos la clase <i>PreparedStatement</i> cuando las consultas que vamos a realizar tienen parámetros (<i>P. Ej.</i> “ <i>SELECT * FROM actor where first_name=?;</i> ” donde ? puede ser Arnold, Pepe, Luis, etc.) para evitar el <u>SQL Injection</u> o cuando se hace una misma consulta varias veces a través del uso de un bucle con diferentes parámetros (<i>P. Ej.</i> se quieren realizar varios <i>inserts</i> de diferentes actores en la BDs).
Se declara de la siguiente forma: <pre>Statement st = conn.createStatement(); st.executeQuery(query);</pre>	Se declara de la siguiente forma: <pre>PreparedStatement pst = conn.prepareStatement(query); pst.setString(1,"Arnold"); → ahora lo vemos ResultSet rs = pst.executeQuery();</pre>
executeQuery VS. executeUpdate	
Se usa para consultas que <u>NO</u> vayan a modificar la BDs, es decir, con instrucciones DML, como <i>INSERT</i> , <i>UPDATE</i> o <i>DELETE</i> .	Se usa para consultas que vayan a modificar la BDs, es decir, con instrucciones DML, como <i>INSERT</i> , <i>UPDATE</i> o <i>DELETE</i> .

d. **Gestionar el resultado**

(Clase ResultSet)

```

while (rs.next()) {
    Date lastUpdate = rs.getDate(4);
    int id = rs.getInt("actor_id");
    String firstName = rs.getString("first_name");
    String lastName = rs.getString(3); → cogemos los datos de la 3ª columna.
    OJO que empieza en 1 y no en 0 !!!!  

}

```

e. **Liberar recursos:**

```

rs.close(); → OJO que no se nos olvide realizar en CADA MÉTODO el cierre
st.close(); de cada recurso abierto!!
conn.close(); → ¿Se debe cerrar la conexión tras cada una de las consultas que realicemos?
                ¿Por qué? NO!! SOLO SE HACE UNA VEZ AL FINALIZAR TODAS  

LAS CONSULTAS QUE QUERAMOS EJECUTAR!!

```

ERRORES MÁS COMUNES:

- Driver no cargado (no se encuentra en el CLASSPATH)
- Fallo de conexión
- No existe la BDs
- Error de sintaxis en consulta SQL
- Errores de permiso
- Error de violación de reglas de integridad referencial (FK)
- Error de violación de integridad de la tabla (PK duplicada)

Cualquiera de estos errores producirá una excepción, por lo que en función del tipo de excepción o de los datos que nos dé la misma (p. ej., códigos de error) podremos interpretar el error concreto.

4. INSERT, UPDATE Y DELETE

a. **INSERT:**

```

String nombres[] = {"Clara", "Dani", "Edward", "Denzel"};
String apellidos[] = {"Lago", "Rovira", "Norton", "Washington"};
String query = "INSERT INTO actor (first_name, last_name, last_update) VALUES (?, ?, ?);";
PreparedStatement pst = conn.prepareStatement(query);
for (int i = 0; i < nombres.length; i++) {
    pst.setString(1, nombres[i]);
    pst.setString(2, apellidos[i]);
    pst.setDate(3, new Date(new java.util.Date().getTime()));
    pst.executeUpdate();
}
pst.close();

```

b. ***UPDATE***:

```
Statement st = conn.createStatement();
int result=st.executeUpdate("UPDATE actor SET first_name='Daniel' WHERE
first_name='Dani';");
st.close();
```

Si la instrucción fuera DDL nos debería devolver 0, ya que no modifica la BDs. Como en este caso es una sentencia DML, puede retornar un valor diferente a 0. En concreto, nos devolverá el número de filas afectadas.

c. ***DELETE***:

```
Statement st = conn.createStatement();
int result = st.executeUpdate("DELETE FROM actor WHERE actor_id=5;");
st.close();
```

En este caso saltará un error ya que hay una tabla (“*film_actor*”) que apunta a ésta. Al estar creada con la opción *RESTRICT*, no nos permite eliminar esta fila mientras haya alguna película relacionada con este actor en la otra tabla, es decir, alguna fila o tupla que contenga al actor cuyo “*actor_id*” es 5.

5. INFORMACIÓN DE LOS DATOS

- a. ***Clase DatabaseMetaData***: permite obtener cualquier información posible sobre una BDs, es decir, los esquemas, tablas, tipos de tablas, columnas de las tablas, etc.
- b. ***Clase ResultSetMetaData***: otra forma de obtener información sobre la estructura de las tablas es con esta clase. La principal diferencia es que mientras que DatabaseMetaData actúa sobre toda una BDs, esta actúa sobre una tabla (o conjunto de tablas) en concreto a través del ResultSet obtenido tras ejecutar una consulta.

6. FUNCIONES DE TIEMPO EN JAVA - SQL

- a. Usando la fecha actual con las clases Date, Time y Timestamp del paquete java.sql

```
long msActuales = System.currentTimeMillis();
java.sql.Date fecha = new java.sql.Date(msActuales);
java.sql.Time hora = new java.sql.Time(msActuales);
java.sql.Timestamp timestamp = new java.sql.Timestamp(msActuales);

java.sql.Date fecha2 = new java.sql.Date(new java.util.Date().getTime());
```

- b. Usando una fecha específica con las clases Date (en desuso) y Calendar

```
java.util.Date fechaUtil = new java.util.Date("2013/01/15");
java.sql.Date fechal = new java.sql.Date(fechaUtil.getTime());
```

```
Calendar cal = Calendar.getInstance();
cal.set(Calendar.YEAR, 2013);
cal.set(Calendar.MONTH, Calendar.MAY);
cal.set(Calendar.DAY_OF_MONTH, 15);
java.sql.Date fecha2 = new java.sql.Date(cal.getTime().getTime());
```

- c. Comparación de fechas usando compareTo (como en cualquier clase en la que se define un orden)

Dadas las fechas de b) (fechal y fecha2), pasamos a compararlas como en cualquier otro caso:

```
int compare = fechal.compareTo(fecha2);
if (compare==0) {System.out.println("Fechas iguales");}
elseif (compare>0) {System.out.println("La fechal es posterior de la fecha2");}
else {System.out.println("La fechal es anterior de la fecha2");}
```

7. FICHEROS BINARIOS

Existen diversos tipos en SQL para definir ficheros binarios (dependiendo de su tamaño máximo). El tamaño definido es el máximo, pero son dinámicos, es decir, sólo ocupan el tamaño que cada fichero requiere. Sin embargo, el *tipo Binary* tiene un tamaño predefinido ocupando ese tamaño siempre.

TIPO SQL	TAMAÑO MÁXIMO
TINYBLOB	255bytes
BLOB	64 kB
MEDIUMBLOB	16 MB
LONGBLOB	4 GB

- a. **Enviar:** para insertar una imagen u otro fichero binario desde Java se pueden usar las clases *File* y *FileInputStream*.

```
String query = "INSERT INTO actor (firs_name, last_name,picture) VALUES (?,?,?);";
PreparedStatement pst = conn.prepareStatement(query);
pst.setString(1, "Megan");
pst.setString(2, "Fox");
File file = new File("pics/MeganFox.jpg");
FileInputStream fis = new FileInputStream(file);
pst.setBinaryStream(3, fis, (int) file.length());
pst.executeUpdate();
```

- b. **Obtener:** también podemos requerir la imagen o fichero binario desde la BDs y guardarla como *FileOutputStream*.

```

Statement st = conn.createStatement();
ResultSet rs = st.executeQuery("SELECT picture FROM actor WHERE first_name= "Megan" and
last_name= "Fox"; ");
byte data[] = null;
Blob myBlob = null;
while (rs.next()) {
    myBlob = rs.getBlob("picture");
    data = myBlob.getBytes(1, (int) myBlob.length());
}
FileOutputStream fos = new FileOutputStream("pics/meganfox_downloaded.jpg");
fos.write(data);
fos.close();

```

8. PROCEDIMIENTOS Y TRIGGERS

- a. **Procedimientos almacenados:** conjunto de sentencias SQL que se pueden almacenar en el servidor. Estos procedimientos pueden ser creados directamente desde MySQL Workbench o vía código. En otras palabras, es un pequeño programa que tenemos en la BDs y se puede llamar para realizar una serie de operaciones.

Las diferencias principales entre los procedimientos almacenados y las funciones son:

PROCEDIMIENTOS ALMACENADOS	FUNCIONES
Puede devolver 0, 1 o varios valores	Debe devolver solo un valor
Puede usar transacciones	NO puede usar transacciones
Puede tener parámetros tanto de entrada como de salida	Solo puede tener parámetros de entrada

```

CREATE PROCEDURE 'ADD.OSCAR' (IN act_id INT, IN film_id INT, IN typePrize
VARCHAR(45))
BEGIN
START TRANSACTION
CREATE TABLE IF NOT EXISTS 'sakila'.'oscar_winners' (
    'actor_id' INT NOT NULL,
    'film_id' INT NOT NULL,
    'type_oscar' VARCHAR(45) NOT NULL,
    PRIMARY KEY ('actor_id', 'film_id', 'type_oscar'));
SELECT @cnt:=COUNT(*) FROM film_actor WHERE actor_id=2 AND film_id=3;
IF @cnt>=1 THEN
    INSERT INTO oscars_winners (actor_id, film_id, type_oscar)
    VALUES (act_id, film_id, typePrize);
END IF;
COMMIT
END

```

Una forma de solucionar el acceso a determinados Metadatos requeridos al intentar ejecutarlos podría ser poner al final de la URL: **?noAccessToProcedureBodies=true**

Si lo que queremos es poder lanzar varias consultas de una sola vez, es decir, en una sola query, entonces debemos añadir tras nuestra URL: **?allowMultiQueries=true**

- b. **Triggers (disparadores)**: elementos que están asociados con tablas y se almacenan en la BDs. Son elementos asociados a eventos que se ejecutan cuando se produce un evento/acción sobre la tabla a la que están asociados (operaciones de manifestación de datos de la tabla como INSERT, UPDATE, DELETE). Pueden estar asociados a la ejecución ANTES (BEFORE) o DESPUÉS (AFTER) de la modificación de la tabla. Se puede definir dentro del cuerpo del Trigger que la acción que vaya especificada es para cada fila que se vea afectada por la operación con el comando FOR EACH ROW. El operador NEW recibe después del punto el nombre de la variable a la que se hace referencia (NEW.var).

Por ejemplo: un trigger que se activa antes de hacer la inserción de una persona comprueba si el DNI tiene letra; y en caso contrario llama a la función '*'calcularLetra'*' que añade esta letra de tal forma que el DNI se inserte con la letra siempre.

9. TRANSACCIONES

Son conjuntos de acciones a ejecutar de forma indivisible o atómica. Un SGBD se dice que es transaccional (ACID complaint) si tiene las funcionalidades necesarias: ATOMICIDAD, CONSISTENCIA, AISLAMIENTO y PERSISTENCIA. Las transacciones pueden ser satisfactorias (ejecución de las operaciones de modificación) o insatisfactorias (por fallo, ejecución manual de ROLLBACK) entendiendo como transacción todo lo que está dentro del bloque de las operaciones que la delimita. Estas operaciones son:

- a. **COMMIT**: se encarga de que los datos se fijen en la BDs y por lo tanto estén disponibles las modificaciones que se hayan hecho. Define un final y comienzo de transacción.
- b. **ROLLBACK**: deshace los cambios desde el comienzo de la última transacción.
- c. **AUTOCOMMIT (AC)**: si es TRUE(1) cada operación individual es una transacción, si es FALSE(0) hay que definir con las operaciones necesarias cuando empieza o acaba una transacción y qué conjunto de operaciones forma parte del mismo.
- d. **START TRANSACTION (ST)**: delimita el comienzo de una transacción nueva o bloque de operaciones que forma una transacción. Esta operación tiene sentido fundamentalmente cuando AC=1 (aunque también lo podemos hacer con AC=0) ya que permite que un grupo de operaciones sea una transacción (en lugar de que cada operación sea individual). En otras palabras, es como si AC se deshabilitara hasta un nuevo COMMIT o ROLLBACK.

Por defecto AC=1. Esto puede cambiarse con el método setAutoCommit() de la clase Connection (lo que implicaría hacer un COMMIT). La clase Connection también tiene los métodos commit() y rollback(), los cuales pueden generar excepciones que deben ser tratadas de forma adecuada.

Aunque se puede realizar un ST es preferible poner AC=0 para comenzar transacciones, restaurandolo a TRUE(1) cuando acabemos.

NOTA 1: Si se ejecuta un ST, se lanzan operaciones y se ejecuta otro ST antes de un COMMIT o ROLLBACK, el SGBD hará un COMMIT e introducirá los datos.

NOTA 2: Con AC=1, salvo que se inicie una transacción con ST un ROLLBACK no tiene ningún efecto.

10. OTROS EJEMPLOS CON JDBC

a. *Actualizar la BDs a través de ResultSet*

i. INSERT ROW

```
Statement st = conn.createStatement	ResultSet.TYPE_SCROLL_SENSITIVE,
	ResultSet.CONCUR_UPDATABLE);
ResultSet rs = st.executeQuery("SELECT * FROM actor WHERE actir_id>=200;");
rs.moveToInsertRow(); → nos movemos a la posición para insertar
rs.updateString("first_name", "Drew");
rs.updateString("last_name", "Barrymore");
rs.updateDate("last_update", new java.sql.Date(System.currentTimeMillis()));
rs.insertRow(); → insertamos el registro (la fila)
rs.beforeFirst(); → movemos el cursor al principio, justo antes del primer elemento
```

ii. UPDATE ROW

```
rs.absolute(2) → nos movemos a la 2a fila
rs.updateString(2, "Morgan"); → actualizamos la columna 2 ("first_name")
de este registro (fila)
rs.updateRow(); → actualizamos el registro
rs.beforeFirst();
```

iii. DELETE ROW

```
rs.absolute(4);
rs.deleteRow(); → borramos el registro (fila)
rs.beforeFirst();
```

b. ArrayList para guardar datos

MÉTODO	DESCRIPCIÓN
<code>ArrayList<Actor> nombre = new ArrayList<Actor>();</code>	→ Declaramos el objeto de la clase Actor (p. ej.)
<code>nombre.add("Elemento1");</code>	→ Añade el elemento al final
<code>nombre.add(3, "Elemento2");</code>	→ Añade el elemento en la posición 3
<code>nombre.size();</code>	→ Devuelve el tamaño
<code>nombre.get(2);</code>	→ Devuelve el elemento en la posición 2
<code>nombre.contains("Elemento1");</code>	→ Comprueba si existe el elemento que se pasa como parámetro
<code>nombre.indexOf("Elemento");</code>	→ Devuelve la posición de la primera ocurrencia
<code>nombre.remove(5);</code>	→ Borra el elemento de la 5 ^a posición
<code>nombre.remove("Elemento");</code>	→ Borra la primera ocurrencia de "Elemento"
<code>nombre.clear();</code>	→ Borra todos los elementos
<code>nombre.clone();</code>	→ Copia el Array

Ejemplo de uso:

```
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery("SELECT * FROM actor;");
ArrayList<Actor> actores = new ArrayList<Actor>();
while(rs.next()) {
    int id = rs.getInt("actor_id");
    String firstName = rs.getString(2);
    String lastName = rs.getString("last_name");
    Date lastUpdate = rs.getDate(4);
    actores.add(new Actor(id, firstName, lastName, lastUpdate));
}
```

c. Escritura en fichero como CSV

- i. Añadimos a la clase Actor el método toCSV

```
public String toCSV() {
    return id + "," + firstName + "," + lastName + "," + lastUpdate.toString();
}
```

- ii. Escribimos los resultados en un fichero

```
ArrayList<Actor> nombresActores = getNombresActores();
BufferedWriter bw = new BufferedWriter (new FileWriter ("actores.csv"));
for (int i = 0; i < nombresActores.size(); i++) {
    Actor act = nombresActores.get(i);
    bw.write(act.toCSV());
    bw.newLine();
}
bw.close();
```

EXÁMENES ACCESO PROGRAMÁTICO

JUNIO 2019 (EVALUACIÓN CONTINUA)

Instrucciones: Las preguntas tienen una única solución. Las respuestas correctas suman +1 punto. Las preguntas no contestadas ni suman ni restan. Las respuestas incorrectas restan -0.25 puntos. Contestar en el enunciado marcando la respuesta con un círculo.

1. La variable query contiene una sentencia DDL y conn es una variable de tipo Connection (la conexión está abierta y funciona sin problemas), ¿cuál de las siguientes llamadas a métodos de la clase Connection debe usarse para ejecutar la sentencia DDL?
 - a. conn.executeUpdate(query);
 - b. conn.executeQuery(query);
 - c. conn.prepareStatement(query);
 - d. conn.getConnection(query);

2. Señala la afirmación falsa:
 - a. El mismo driver para conectarnos a un DBMS MySQL será válido para conectarnos a un DBMS Oracle.
 - b. La API de JDBC es la misma para conectarse a un DBMS MySQL y a un DBMS Oracle.
 - c. Al importar el driver de JDBC siempre podremos usar la clase Connection.
 - d. Al importar el driver de JDBC siempre podremos usar la clase PreparedStatement.

3. ¿Cuántos bloques de transacción hay en el código de la derecha (AC=1)?
 - a. 3
 - b. 4
 - c. 5
 - d. 8

4. ¿Cuántas operaciones han modificado la base de datos en el código de la derecha (AC=1)?
 - a. 3
 - b. 4
 - c. 5
 - d. 8

<pre>INSERT DELETE START TRANSACTION INSERT INSERT DELETE START TRANSACTION DELETE INSERT INSERT ROLLBACK</pre>

Código para las preguntas 3 y 4.

5. En la práctica de JDBC, la llamada a DBclose() debe realizarse:
 - a. Despues de cada modificación o consulta de la base de datos.
 - b. En el main, tras la llamada a mainMenu().
 - c. Antes de acabar cada método de los pedidos.
 - d. No debe llamarse nunca a DBclose().

6. En la práctica de JDBC, para obtener las cervezas más populares, ¿qué tablas deben usarse?
 - a. "cerveza".
 - b. "gusta".
 - c. Tanto "cerveza" como "gusta".
 - d. Ni "cerveza" ni "gusta".

7. ¿En qué método de la práctica de JDBC es necesario desactivar el autocommit?
 - a. createTableEmpleado().
 - b. createTableGusta(). → no lo podemos saber sin haber hecho la práctica
 - c. loadEmpleados().
 - d. loadGustos().

8. En la práctica de JDBC, ¿dónde debe declararse el objeto de tipo Connector?
 - a. En cada método.
 - b. No es necesario un objeto de la clase Connector de forma explícita.
 - c. En las variables de la clase.
 - d. En el main.

9. En la práctica de JDBC, ¿cuál es el tipo de dato de la columna que se pide añadir en el método addFotoColumn()?
 - a. BINARY.
 - b. BLOB. → no lo podemos saber sin haber hecho la práctica
 - c. PICTURE.
 - d. Ninguno de los anteriores.

10. En la práctica JDBC, al insertar un empleado con foto, ¿qué clase de las siguientes no hay que usar?
 - a. File.
 - b. FileInputStream.
 - c. Statement.
 - d. PreparedStatement.

JUNIO 2019 (PRUEBA FINAL)

Se proporcionan los siguientes fragmentos de código que pertenecen a un programa que pretende conectarse a un SGBD MySQL en su puerto por defecto. Las credenciales de acceso proporcionadas son correctas y no dará ningún tipo de problema en este sentido.

También se considera que la definición de la clase y el método main, aunque no aparezca, es correcto. El main, llama al constructor de la propia clase Main(), que como se ve, llama a los métodos “connect()” y “executeQueries()”.

Dado por lo tanto esos fragmentos de código, hay un total de 5 errores (se considera error todo lo que afecte a una línea de código, pudiendo haber más de un error en una línea, pero a efectos de este enunciado, eso implicaría un solo error). Los errores a identificar pueden dar lugar a errores secundarios cuando estos no han sido solventados. Esos errores secundarios no serán tenidos en cuenta como posibles errores. Las soluciones se deben proporcionar en el espacio dado.

Puntuaciones:

- Cada error identificado y solventado vale dos puntos. La mera identificación del error es un punto.
- Identificar una línea de código correcta como error resta 0.5 puntos.

```

import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.Connection;

public class Main {
    private Connection conn;

    public Main() {
        try {
            connect();
            executeQueries();
        } catch (ClassNotFoundException e) {
            // e.printStackTrace();
            System.err.println("Clase no encontrada. ¿Se ha cargado el driver en el proyecto?");
            System.err.println("URL: http://dev.mysql.com/downloads/connector/j/");
        } catch (Exception e) {
            System.err.println("Error desconocido: " + e.getMessage());
            e.printStackTrace();
        }
    }

    private void connect() throws Exception {
        String drv = "com.mysql.jdbc.DriverManager";
        Class.forName(drv);

        String serverAddress = "localhost:3309";
        String db = "sakila";
        String user = "bd";
        String pass = "bdupm";
        String url = "jdbc:oracle://" + serverAddress + "/";
        conn = DriverManager.getConnection(url, user, pass);
        System.out.println("Conectado a la base de datos!");
    }
}

```

```
private void executeQueries() throws SQLException {  
  
    String nombres[] = { "Clara", "Dani", "Edward", "Denzel" };  
    String apellidos[] = { "Lago", "Rovira", "Norton", "Washington" };  
  
    String query = "INSERT INTO actor (first_name, last_name, last_update) VALUES (?,?,?);";  
  
    Statement pst = conn.createStatement();  
    for (int i = 0; i < nombres.length; i++) {  
        pst.setString(0, nombres[i]);  
        pst.setString(1, apellidos[i]);  
        pst.setDate(2, new java.sql.Date(new java.util.Date().getTime()));  
        boolean res = pst.executeUpdate();  
    }  
    System.out.println("Query ejecutada!");  
  
    pst.close();  
    conn.close();  
}
```

JUNIO 2018 (EVALUACIÓN CONTINUA)

1. ¿Cuál fue el organismo que planteó el sistema ODBC?
 - a. SAG (SQL Access Group).
 - b. DBG (Data Bases Group).
 - c. DAG (Data Access Group).
 - d. MySQL.

2. ¿Cuál es la excepción más correcta (la más cercana a la excepción real) que va a devolver un error de violación de clave foránea durante una inserción en la base de datos utilizando JDBC?
 - a. Exception.
 - b. ForeignKeyIntegrityException.
 - c. IntegrityConstraintViolationException.
 - d. SQLException.

3. En base a lo visto en la práctica, ¿cuál de las siguientes operaciones lleva implícito un commit independientemente del estado del autocommit?
 - a. CREATE TABLE
 - b. INSERT
 - c. DELETE
 - d. UPDATE

4. ¿Cuál es la clase que se encarga, llamando al método apropiado, de realizar realmente la conexión?
 - a. Connection
 - b. DriverManager
 - c. Driver
 - d. com.mysql.jdbc.Driver

5. Si queremos realizar una consulta que modifique la BD, ¿cuál de los siguientes métodos de la clase Statement debemos llamar?
 - a. executeUpdate
 - b. executeQuery
 - c. executeSelect
 - d. executeUpgrade

6. ¿Cuál es el método en un ResultSet que devuelve un booleano que me indica si hay más registros en él mismo?
 - a. hasNext().
 - b. next().
 - c. isNext().
 - d. Ninguna de las anteriores.

7. ¿A qué operación NO afecta un trigger?
 - a. SELECT
 - b. INSERT
 - c. DELETE
 - d. UPDATE

8. Ante la siguiente secuencia (con Autocommit = 0): 1. INSERT, 2. DELETE, 3. COMMIT, 4. INSERT, 5. UPDATE. ¿Cuántas operaciones podemos asegurar que han modificado la base de datos?
 - a. 1
 - b. 2
 - c. 3
 - d. 4
9. ¿Cuál es el parámetro que debe pasarse a la hora de crear un Statement al método de creación para que el ResultSet permita que se actualice la base de datos a través de operaciones realizadas en el mismo ResultSet?
 - a. Statement.UPDATABLE_CONCUR
 - b. ResultSet.UPDATABLE_CONCUR
 - c. ResultSet.CONCUR_UPDATABLE
 - d. Statement.CONCUR_UPDATABLE
10. ¿Cuál es el método que se debe usar, de la clase PreparedStatement, para pasar el valor de un tipo de dato binario y que se sustituya dicho valor por el contenido a la hora de hacer una operación de inserción en la base de datos?
 - a. setBinaryStream
 - b. setStream
 - c. setBlob
 - d. setStreamBinary

JUNIO 2017 (EVALUACIÓN CONTINUA)

1. Queremos hacer una consulta (SELECT) en una tabla. La condición de la cláusula WHERE se recibe en un método como parámetro. ¿Cuál es la mejor opción a usar en el código?
 - a. Usar la clase Statement.
 - b. Usar la clase PreparedStatement.
 - c. Cualquiera de las dos es válida.
 - d. Ninguna es válida.
2. ¿Cuál es el tipo de datos de la columna que se debía añadir en la tabla paciente para resolver el ejercicio 9 que pedía que se pudiera añadir una foto cuando se añadiera un paciente?
 - a. VARCHAR
 - b. STRING
 - c. LONGBLOG
 - d. Ninguna de las anteriores
3. ¿Cuál es la clase a usar para cargar el driver en JDBC?
 - a. com.mysql.jdbc.Driver
 - b. org.mysql.jdbc.Driver
 - c. com.jdbc.mysql.Driver
 - d. org.jdbc.mysql.Driver

4. ¿Cuál de las siguientes clases me permite obtener metadatos de la base de datos?
 - a. MetaDatabase
 - b. DatabaseMetaData
 - c. ResultSetMetadata
 - d. Ninguna de las anteriores

5. Si queremos realizar una consulta de tipo SELECT, ¿cuál de los siguientes métodos de la clase Statement debemos llamar?
 - a. executeUpdate
 - b. executeQuery
 - c. executeUpgrade
 - d. executeSelect

6. ¿Cuál de los siguientes elementos no se debe cerrar al acabar de ejecutar cada operación/ejercicio en la práctica?
 - a. ResultSet
 - b. Statement
 - c. PreparedStatement
 - d. Connection

7. ¿Cuál de las siguientes cláusulas de SQL era clave para la realización del ejercicio 7 que pedía obtener la distribución de pacientes que sufren una determinada enfermedad y cuántas veces?
 - a. Order by
 - b. Group by
 - c. Average (AVG)
 - d. Suma (SUM)

8. ¿Qué clase permitía obtener los metadatos de una tabla?
 - a. TableMetaData
 - b. ResultSetMetaData
 - c. EntityMetaData
 - d. Todas las anteriores

9. ¿Cuál es el puerto que utiliza por defecto MySQL para las conexiones y que se ha usado en la práctica?
 - a. 3307
 - b. 3063
 - c. 3306
 - d. 3606

10. ¿Cuál de las siguientes sentencias de SQL permite iniciar una transacción?
 - a. AUTOCOMMIT = 0
 - b. AUTOCOMMIT = 1
 - c. START TRANSACTION
 - d. BEGIN TRANSACTION

JUNIO 2017 (PRUEBA FINAL)

Pregunta 1 (5 puntos): Tenemos una base de datos cuya tabla “Empleados” contiene registros de empleados de la empresa. Esta tabla tiene 4 columnas, todas de tipo varchar excepto el ID que es numérico y no auto incremental: *id_empleado*, *nombre*, *apellidos* y *telefono*. Tenemos un programa escrito en Java que utiliza la librería JDBC para realizar las operaciones. Asumiendo que el siguiente código se conecta de forma correcta y que el objeto “conn” es el objeto *Connection* que maneja la conexión:

```
119 private void addEmpleado() throws Exception {  
120     conn.setAutoCommit(false);  
121     System.out.println("Introduce ID del empleado");  
122     int id = readInt();  
123     String nombre = readString();  
124     String apellido = readString();  
125     String telefono = readString();  
126     String query = "INSERT INTO EMPLEADO(id_empleado, nombre,apellidos, telefono) VALUES (?,?,?,?)";  
127     PreparedStatement pst = conn.prepareStatement(query);  
128     pst.setInt(0, id);  
129     pst.setString(1, nombre);  
130     pst.setString(2, apellido);  
131     pst.setString(3, telefono);  
132     pst.executeQuery();  
133 }
```

Identificar los diferentes errores que hay en el código y proponer solución. Solo se considerará como correcto el par error-solución (si se proporciona error pero no solución o viceversa, no puntúa). Existen algunos errores que son de funcionamiento (el programa no va a funcionar bien) y otros referidos a que una vez arreglado todo, el programa podría dar problemas en determinadas situaciones. Identificar nuevamente cuáles son esas situaciones y como las solventarías. Ser **breve y conciso** tanto en la identificación de errores como en la solución. Se pueden usar los números de líneas como referencia.

Pregunta 2 (1 punto): ¿Qué clase debemos utilizar si queremos obtener metadatos de la base de datos? ¿Y qué clase debemos utilizar si queremos obtener cuales son los nombres de las columnas de una tabla?

Pregunta 3 (2 puntos): Explicar la diferencia entre Statement y PreparedStatement y cuándo y por qué se debe utilizar uno u otro. Ser breve y usar ejemplos a ser posible.

Pregunta 4 (2 puntos):

Estamos en una base de datos con autocommit = 1 y se va a ejecutar el flujo de operaciones que se describe al final.

- 1) Indicar el comando que permite empezar una transacción (en la BD, no en código) y que sustituiría por lo tanto en el flujo a la sentencia de comienzo de transacción (*0,5 puntos*).
 - 2) Dado el siguiente flujo de instrucciones y sabiendo que la sentencia número 7 va a fallar debido a un error de integridad referencial. ¿Qué operaciones se puede garantizar que se han ejecutado de tal forma que sus modificaciones van a persistir en la base de datos? (*1 punto*)
 - 3) ¿Cuál es la operación que hace que se deshagan los cambios en la base de datos en una transacción? (*0,5 puntos*)
- | | |
|---------------------------------------|---------------|
| 1. [SENTENCIA COMIENZO TRANSACCIÓN] | 6. [INSERT] |
| 2. [INSERT] | 7. [INSERT] |
| 3. [INSERT] | 8. [UPDATE] |
| 4. [COMMIT] | 9. [COMMIT] |
| 5. [SENTENCIA COMIENZO TRANSACCIÓN] | |
- 1)
 - 2)
 - 3)

JUNIO 2016 (PRUEBA FINAL)

1. Se pide (0.5 por respuesta correcta - Total 2.5 puntos):
 - a. ¿Qué clase debemos utilizar para construir sentencias SQL en las que se permita evitar SQL Injection?
 - b. ¿Qué clase debemos usar para construir sentencias SQL que se vayan a repetir múltiples veces?
 - c. ¿Qué método tenemos que ejecutar (y con qué parámetro) para que podamos trabajar contra la base de datos de tal manera que cada operación no sea gestionada como una transacción separada?
 - d. ¿En qué clase debemos llamar a ese método?
 - e. ¿Qué devuelve el método executeUpdate() (tipo de dato y qué significa)?

2. El siguiente código contiene errores. Se asume que las posibles tablas, esquema, usuario, contraseña, etc., existen y son correctos. No se debe juzgar como erróneo que la consulta pueda devolver múltiples registros y solo se intente obtener uno. Identificar los errores y proponer una solución (en código). Enumerar los errores (error 1, error 2, ...) y su solución. Se valora la brevedad en la respuesta (7.5 puntos)

```

public class Examen {
    String drv = "com.mysql.jdbc.Driver";
    String serverAddress = "localhost:3306";
    String db = "sakila";
    String user = "bd3";
    String pass = "bdupm";
    String url = "jdbc:mysql://" + serverAddress + "/" + db;

    private void metodo() throws Exception {
        Class.forName(drv);
        Connection conn = Connection.getConnection(url, user, pass);
        PreparedStatement pst = conn.createStatement("SELECT * FROM actor WHERE first_name = %");
        pst.setString("Alberto", 0);
        ResultSet rs = pst.executeQuery();
        int id = rs.getInt("actor_id");
        String firstName = rs.getString("first_name");
        String lastName = rs.getString(3);
        Date lastUpdate = rs.getDate("last_update");
        System.out.println("ID: " + id + " , Name: " + firstName + " , Last name: " + lastName);
    }
}
  
```

JULIO 2019

Ejercicio 1: Dado un conjunto de operaciones y diferentes estados del AC, responder a las preguntas planteadas. Cada pregunta correcta (completamente, no se admiten respuestas parciales) supone 0.75 puntos. Total: 4.5 puntos.

Asumimos una base de datos con las siguientes operaciones:

- | | |
|-------------------------------------|-----------------------|
| 1. INSERT | 10. UPDATE |
| 2. DELETE | 11. DELETE |
| 3. INSERT | 12. SELECT |
| 4. ROLLBACK (ejecutado manualmente) | 13. UPDATE |
| 5. INSERT | 14. COMMIT |
| 6. DELETE | 15. UPDATE |
| 7. UPDATE | 16. START TRANSACTION |
| 8. START TRANSACTION | 17. UPDATE |
| 9. DELETE | 18. COMMIT |

Pregunta: Asumiendo primero AC = 0 y luego AC = 1:

- A. ¿Cuántos bloques de transacción tenemos?
- B. ¿Cuántas operaciones podemos garantizar que han modificado la base de datos?
- C. ¿Cuáles son esas operaciones que podemos garantizar que han modificado la base de datos (número de operación)?

Indicar respuestas en las siguientes cajas:

AC = 0

A (Nº bloques)		B (Nº operaciones)		C (Operaciones)	
----------------	--	--------------------	--	-----------------	--

AC = 1

A (Nº bloques)		B (Nº operaciones)		C (Operaciones)	
----------------	--	--------------------	--	-----------------	--

Ejercicio 2: Identificar (brevemente, una línea por motivo) los motivos por los que se suele usar la clase PreparedStatement. 1 punto.

Ejercicio 3: Dado el siguiente fragmento de código, donde se asume que el resto de código que no se muestra (definición de clase, método main, etc.) es correcto, las credenciales y la BD existen y son correctas, y donde también se asume que el código llamará al método init(). Identificar cada error (indicar número de línea y error concreto) y solventarlo correctamente implica 1.5 puntos. Identificar erróneamente un error resta 0.5 puntos. Total: 4.5 puntos.

```
1  private void init() throws Exception {
2      String drv = "com.mysql.jdbc.DriverManager";
3      Class.forName(drv);
4
5      String serverAddress = "localhost:3306";
6      String db = "sakila";
7      String user = "bd";
8      String pass = "bdupm";
9      String url = "jdbc:mysql://" + serverAddress + "/" + db;
10     Connection conn = DriverManager.getConnector(pass, user, url);
11     System.out.println("Conectado a la base de datos!");
12
13     Statement st = conn.createStatement("DELETE FROM rental where rental_id = '1'");
14     int result = st.executeUpdate();
15
16     System.out.println("Número de filas afectadas: " + result);
17     System.out.println("Query ejecutada!");
18
19     st.close();
20     conn.close();
21 }
```

JULIO 2018

Ejercicio 1 (6 puntos):

Dados los siguientes fragmentos de código, llenar lo que corresponda en cada uno. Nótese que en el fragmento 2, la operación que se quiere ejecutar es una inserción. La respuesta debe entregarse en la tabla anexa, simplemente incluyendo la solución propuesta. Sólo se considerará válida la respuesta si es 100% correcta (**se diferencia mayúsculas y minúsculas y una sola letra omitida es motivo para no dar la respuesta por válida**), no habiendo interpretaciones o valoraciones parciales. Es también responsabilidad del alumno asegurarse de que pone la respuesta correcta asociada a cada hueco, no siendo susceptible de revisión problemas asociados a “equivocarse” al indicar el número y cosas similares. Cada respuesta correcta suma 0.3 puntos. Las respuestas vacías o incorrectas ni suman ni restan.

Comentarios:

- Para el hueco 6, se solicita la excepción más general que existe en Java.
- Para el hueco 7, la excepción más correcta y específica que aplique a todo el método.
- El hueco 13 pretende que se ejecute y se inserten los datos en la base de datos.
- El hueco 17 pretende que se cierre el PreparedStatement.

Fragmento 1:

```

private void conectar() {
    try {
        String drv = "com.mysql. 1 ";
        Class.forName(drv);
        String serverAddress = "localhost: 2 ";
        String url = "jdbc: 3 ://" + serverAddress + "/";
        conn = 4 .getConnection( 5 , USER, PASS);
    } catch ( 6 e) {
        System.err.println("Error al conectar: " + e.getMessage());
    }
}

```

Fragmento 2:

```

private void insertarDatos(int v1, String v2, Date v3) throws 7 {
    PreparedStatement pstl = conn. 18 Statement(" 20 19 tabla(edad,nombre,f_nac) 8 ( 9 );");
    pstl.set 14 (10,v1);
    pstl.set 15 (11,v2);
    pstl.set 16 (12),v3);
    pstl. 13 ();
    pstl. 17
}

```

Hueco	Respuesta	Hueco	Respuesta	Hueco	Respuesta	Hueco	Respuesta
1.		6.		11.		16.	
2.		7.		12.		17.	
3.		8.		13.		18.	
4.		9.		14.		19.	
5.		10.		15.		20.	

Ejercicio/pregunta 2 (1,5 puntos): Explicar que es un Trigger, sobre que operaciones aplica y cuales son las opciones a la hora de crearlo. Máximo 5 líneas.

Ejercicio/pregunta 3 (2,5 puntos): Explicar en que consiste una transacción. Explicar que es el AUTOCOMMIT en MySQL, que valores puede tener y que implica cada uno (incluyendo en ambos modos cuando empieza y acaba una transacción). Máximo 10 líneas.

JULIO 2018 (DOBLE GRADO ADE + ING. INFORMÁTICA)

Ejercicio/pregunta 1 (4.5 puntos): Responder a las siguientes preguntas (usar solamente el espacio proporcionado entre pregunta y pregunta, no más, y con una letra clara y legible).

1. Explicar las diferencias entre las clases PreparedStatement y Statement. Que diferencias existen a nivel de código (cuando se usan: poner un ejemplo para cada una) y en qué casos concretos se usa cada una. (2.5 puntos)
 2. Explicar, dentro de esos métodos, cuando se utiliza el método executeUpdate() y cuando se utiliza el método executeQuery(). (1 punto)
 3. Explicar que es un ResultSet, para que se usa, y con que tipo de operación en SQL se usa dicho elemento. Explicar los principales métodos que tiene. (1 punto)

Ejercicio/pregunta 2 (1 punto): ¿Qué es un SGBD (definición, características, operaciones que permite, ...) ?, ¿Qué es una base de datos? ¿en qué se diferencian ambos? (Usar como máximo el espacio proporcionado con letra clara y legible)

Ejercicio/pregunta 3 (5.5 puntos): Dado el flujo de operaciones que se proporciona, y partiendo de la base de que consideramos:

1. Operación: cada una de las acciones individuales que tienen un impacto (modifican) la base de datos.
2. Transacción: el conjunto de operaciones que se ejecutan de forma atómica. Distinguimos entre: a) fallidas/sin impacto en la BD (no producen cambios en la BD) y b) correctas (aquellas que producen un cambio en la BD).

1	INSERT
2	DELETE
3	UPDATE
4	COMMIT
5	INSERT
6	INSERT
7	UPDATE
8	ROLLBACK
9	INSERT
10	DELETE
11	START TRANSACTION
12	INSERT
13	DELETE
14	COMMIT
15	INSERT
16	DELETE

Se debe contestar en las tablas adjuntas. Cada pregunta tiene asignado un código de pregunta. Asegurarse de que se responde la respuesta adecuada en la celda adecuada ya que no se darán por válidos errores de “equivocarse de celda”.

Se solicita:

Con Autocommit = 0:

- AC0.1: ¿Cuántas operaciones podemos asegurar que van a modificar realmente la base de datos? (0.25 puntos)
- AC0.2: ¿Qué operaciones son (indicar número de operación)? (0.5 puntos)

De las transacciones que hay,

- AC0.3: ¿Cuántas se consideran fallidas/sin impacto en la BD (no producen cambios en la BD)? (0.25 puntos)
- AC0.4: Definir que operaciones conforman las transacciones fallidas/sin impacto en la BD (no producen cambios en la BD) (0.75 puntos)
- AC0.5: ¿Cuántas se consideran correctas (producen un cambio en la BD)? (0.25 puntos)
- AC0.6: Definir que operaciones conforman las transacciones correctas (0.75 puntos)

Con Autocommit = 1:

- AC1.1: ¿Cuántas operaciones podemos asegurar que van a modificar realmente la base de datos? (0.25 puntos)
- AC1.2: ¿Qué operaciones son (indicar número de operación)? (0.5 puntos)

De las transacciones que hay,

- AC1.3: ¿Cuántas se consideran fallidas/sin impacto en la BD (no producen cambios en la BD)? (0.25 puntos)
- AC1.4: Definir que operaciones conforman las transacciones fallidas (0.75 puntos)
- AC1.5: ¿Cuántas se consideran correctas (producen un cambio en la BD)? (0.25 puntos)
- AC1.6: Definir que operaciones conforman las transacciones correctas (0.75 puntos)

Con Autocommit = 0

AC0.1 (0.25)	AC0.2 (0.5)	AC0.3 (0.25)	AC0.4 (0.75)	AC0.5 (0.25)	AC0.6 (0.75)

Con Autocommit = 1

AC0.1 (0.25)	AC0.2 (0.5)	AC0.3 (0.25)	AC0.4 (0.75)	AC0.5 (0.25)	AC0.6 (0.75)

JULIO 2016

Dado el siguiente código en el que se pretende que se trabaje en modo transaccional (varias operaciones puedan estar dentro de una sola transacción):

```
1 import java.sql.*;
2 public class Examen {
3
4     String drv = "com.mysql.jdbc.Driver";
5     String serverAddress = "localhost:3306";
6     String db = "sakila";
7     String user = "bd";
8     String pass = "bdupm";
9
10
11    public void borrarCuentas(String c, int i) throws Exception {
12        try {
13            1 conn = DriverManager.getConnection(url, user, pass);
14            PreparedStatement st = conn. 2 ("delete from movimiento where Cuenta_num_cuenta = ? and
15            importe total > ?");
16            st. 3 (1, c);
17            st.setInt(2, i);
18            int res = st.execute 4 ();
19            if (res > 0) {
20                PreparedStatement st2 = conn. 2 ("INSERT INTO LOG_DELETE(Cuenta_num_cuenta, importe_total,
21                num movs) VALUES (?, ?, ?);");
22                st2. 3 (1, c);
23                st2.setInt(2, i);
24                st2.setInt(3, res);
25                conn.commit();
26                conn.rollback();
27            }
28        } catch (SQLException e) {
29            System.err.println("Error: " + e.getMessage());
30        }
31    }
32}
```

1. Rellenar los huecos. Identificar el método o clase concreto. (4 puntos)
 2. El código requiere de líneas adicionales para su correcto funcionamiento. Identificar las líneas necesarias y situarlas en el lugar correcto. Se debe definir la línea donde se debe introducir () o escribir el código de antes y después y escribir la línea de código concreta. (4 puntos)
 3. El código contiene una incoherencia con respecto a cómo se pretende que funcione. Identificar la(s) línea(s) que causan la incoherencia y explicar el porqué es dicho código incoherente. (2 puntos)

JULIO 2017

1. Tenemos el código que se muestra en la imagen. Asumiendo que se han importado todas las clases correctas y que este código tiene un main que permite que se ejecute/llame al método init() de forma correcta:
 - a. El método conectar actualmente está lanzando hacia arriba la excepción genérica ‘Exception’, pero el código en realidad puede lanzar dos tipos de excepción diferentes. ¿Cuáles son y qué sentencias específicamente son las que las pueden lanzar? (0.5 puntos)
 - b. En la query de inserción del método ejecutar() no se pasa un ID. Asumiendo que funciona, ¿a qué se debe? ¿Cuál es el ID que le asignará (conceptualmente, no se pregunta por un valor numérico)? ¿Qué sentencia SQL y operación tendrías que ejecutar para obtener el ID que se le asignaría? (0.5 puntos)
 - c. ¿Qué significa el número (entero) que devuelve el método executeUpdate? Partiendo de que este código se ejecute bien y sin errores, ¿qué valor tendrá esa variable ‘result’? (0.5 puntos)
 - d. Rellena los huecos con el código que corresponda para que el código funcione correctamente (3.5 puntos; 0.5 puntos por hueco)

```

11 public class Examen {
12     private Connection conn;
13     private void init() {
14         try {
15             conectar();
16             ejecutar();
17         } catch (Exception e) {
18         }
19     }
20     private void ejecutar() throws Exception {
21         conn.setAutoCommit(false);
22         String query = "INSERT INTO actor (first_name, last_name, last_update, foto) VALUES [a] :";
23         PreparedStatement pst = conn.[b](query);
24         pst.setString([c], "Luis");
25         pst.setString([d], "Tosar");
26         pst.setDate([e], new Date(new java.util.Date().getTime()));
27         File file = new File("pics/luistosar.jpg");
28         FileInputStream fis = new FileInputStream(file);
29         pst.set[f]([g], fis, (int) file.length());
30         int result = pst.[h]();
31         System.out.println("Result: " + result);
32         Statement st2 = conn.createStatement();
33         ResultSet rs2 = st2
34             .executeQuery("SELECT * from actor where last_name = 'Tosar'");
35         while (rs2.next()) {
36             int id = rs2.getInt("actor_id");
37             String firstName = rs2.getString("first_name");
38             String lastName = rs2.getString("last_name");
39             Date lastUpdate = rs2.getDate("last_update");
40             System.out.println("Actor:" + id + " Nombre: " + firstName
41                 + " Apellido: " + lastName + " Update: "
42                 + lastUpdate.toString());
43         }
44     }
45     private void conectar() throws Exception {
46         String drv = "com.mysql.jdbc.Driver";
47         Class.forName(drv);
48         String serverAddress = "localhost:3306";
49         String db = "sakila";
50         String user = "bd";
51         String pass = "bdupm";
52         String url = "[f]://" + serverAddress + "/" + db;
53         conn = DriverManager.[g](url, user, pass);
54     }
55 }
```

2. En JDBC usamos fundamentalmente 4 clases: 2 para la creación/manipulación de sentencias, una para el manejo de resultados y una para el manejo de la conexión y las operaciones a realizar tras realizar ésta. Escribir el nombre de las 4 clases (y a qué operación/elemento se asocia) y explicar las diferencias entre las 2 clases de creación/manipulación de sentencias muy brevemente. (1.25 puntos)

3. Describe brevemente qué es:
- a. MySQL Server (0.25 puntos) →
 - b. MySQL Workbench (0.25 puntos) →
 - c. JDBC (explica el acrónimo) (0.25 puntos) →
4. Un operador está trabajando contra una base de datos MySQL que tiene el AUTOCOMMIT a 1. El operador realiza las siguientes operaciones desde MySQL Workbench (la letra tras las operaciones de modificación de la BD representan el hipotético dato insertado/modificado en una simplificación de las instrucciones ejecutadas).

1. START TRANSACTION	8. START TRANSACTION	15. START TRANSACTION
2. INSERT A	9. INSERT E	16. INSERT I
3. INSERT B	10. SELECT	17. INSERT J
4. SELECT	11. ROLLBACK	18. SELECT
5. COMMIT	12. INSERT F	19. ROLLBACK
6. INSERT C	13. INSERT G	20. INSERT K
7. INSERT D	14. UPDATE H	

Se sabe que la operación 14 falló por un error de integridad referencial.

¿Qué datos (A,B, ...) se garantiza que se hayan insertado/modificado (estén en la BD) al acabar todas las operaciones? (2 puntos)

JUNIO 2020

1.1: Señala la afirmación correcta con respecto a la comunicación entre JDBC y el DBMS:

- A.Siempre necesita una capa intermedia a la que se traducen las peticiones.
- B.Nunca necesita una capa intermedia.
- C.Necesita una capa intermedia sólo cuando el DBMS disponga de un procesador de solicitudes JDBC.
- D.Necesita una capa intermedia sólo cuando el DBMS no disponga de un procesador de solicitudes JDBC.

1.2: Señala la afirmación correcta con respecto a la comunicación entre ODBC y el DBMS:

- A.Siempre necesita una capa intermedia a la que se traducen las peticiones.
- B.Nunca necesita una capa intermedia.
- C.Necesita una capa intermedia sólo cuando el DBMS disponga de un procesador de solicitudes ODBC.
- D.Necesita una capa intermedia sólo cuando el DBMS no disponga de un procesador de solicitudes ODBC.

2.1: PreparedStatement puede usarse para:

- A.Sólo para mejorar la planificación de carga del gestor.
- B.Sólo para evitar SQL injection.
- C.Ni para mejorar la planificación del gestor, ni para evitar SQL injection.
- D.Tanto para mejorar la planificación del gestor, como para evitar SQL injection.

2.2: Mediante el acceso a los metadatos de la base de datos podemos obtener:

- A.Los nombres de las columnas de una tabla.
- B.La versión del driver JDBC que se está usando.
- C.Ni los nombres de las columnas de una tabla, ni la versión del driver JDBC que se está usando.
- D.Tanto los nombres de las columnas de una tabla, como la versión del driver JDBC que se está usando.

2.3: ¿Cuál de los siguientes métodos de la clase Connection retorna un entero?

- A.execute()
- B.executeQuery()
- C.executeUpdate()
- D.executeStatement()

2.4: ¿Cuál de los siguientes métodos de la clase Connection retorna un ResultSet?

- A.execute()
- B.executeQuery()
- C.executeUpdate()
- D.executeStatement()

3.1: ¿Cuántos bloques de transacción hay en el siguiente código si el autocommit está activado?

- A.4
- B.5
- C.6
- D.8

```
INSERT  
START TRANSACTION  
INSERT  
INSERT  
COMMIT  
DELETE  
START TRANSACTION  
DELETE  
INSERT  
ROLLBACK  
INSERT  
INSERT
```

3.2: ¿Cuántos bloques de transacción hay en el siguiente código si el autocommit está activado?

- A.4
- B.5
- C.6
- D.7

```
INSERT  
DELETE  
START TRANSACTION  
INSERT  
INSERT  
START TRANSACTION  
DELETE  
INSERT  
ROLLBACK  
DELETE
```

3.3: ¿Cuántos bloques de transacción hay en el siguiente código si el autocommit está desactivado?

- A.3
- B.4
- C.6
- D.8

```
INSERT  
INSERT  
ROLLBACK  
DELETE  
START TRANSACTION  
DELETE  
INSERT  
ROLLBACK  
INSERT  
INSERT  
COMMIT
```

3.4: ¿Cuántos bloques de transacción hay en el siguiente código si el autocommit está desactivado?

- A.2
- B.3
- C.6
- D.8

```
INSERT  
INSERT  
INSERT  
ROLLBACK  
INSERT  
DELETE  
INSERT  
START TRANSACTION  
DETELE  
COMMIT
```

4.1: ¿Cuántas operaciones del siguiente código modifican el contenido de la base de datos si el autocommit está activado?

- A.4
- B.5
- C.6
- D.8

```
INSERT  
START TRANSACTION  
INSERT  
INSERT  
COMMIT  
DELETE  
START TRANSACTION  
DELETE  
INSERT  
ROLLBACK  
INSERT  
INSERT
```

4.2: ¿Cuántas operaciones del siguiente código modifican el contenido de la base de datos si el autocommit está activado?

- A.4
- B.5
- C.6
- D.7

```
INSERT  
DELETE  
START TRANSACTION  
INSERT  
INSERT  
START TRANSACTION  
DELETE  
INSERT  
ROLLBACK  
DELETE
```

4.3: ¿Cuántas operaciones del siguiente código modifican el contenido de la base de datos si el autocommit está desactivado?

- A.3
- B.4
- C.5
- D.7

```
INSERT
INSERT
ROLLBACK
DELETE
START TRANSACTION
DELETE
INSERT
ROLLBACK
INSERT
INSERT
COMMIT
```

4.4: ¿Cuántas operaciones del siguiente código modifican el contenido de la base de datos si el autocommit está desactivado?

- A.3
- B.4
- C.5
- D.7

```
INSERT
INSERT
INSERT
ROLLBACK
INSERT
DELETE
INSERT
START TRANSACTION
DETELE
COMMIT
```

5.1: En la práctica de JDBC, si se ejecuta el método de coronapokerus(), sin ejemplares previamente infectados y con una duración de 4 días, ¿cuántos ejemplares quedarán infectados tras la llamada? Para ello, debe tenerse en cuenta que getEjemplarRandom() retornará los ejemplares en el orden siguiente (se supone, por simplicidad, que los ejemplares se identifican inequívocamente con un identificador único): 3, 5, 4, 3, 4, 6, 1, 2, 1, 7, 9, 1, 3.

- A.4
- B.5
- C.6
- D.8

5.2: En la práctica de JDBC, si se ejecuta el método de coronapokerus(), sin ejemplares previamente infectados y con una duración de 4 días, ¿cuántos ejemplares quedarán infectados tras la llamada? Para ello, debe tenerse en cuenta que getEjemplarRandom() retornará los ejemplares en el orden siguiente (se supone, por simplicidad, que los ejemplares se identifican inequívocamente con un identificador único): 3, 3, 4, 1, 4, 6, 1, 2, 1, 7, 9, 1, 3.

- A.3
- B.4
- C.5
- D.8

5.3: En la práctica de JDBC, si se ejecuta el método de coronapokerus(), sin ejemplares previamente infectados y con una duración de 4 días, ¿cuántos ejemplares quedarán infectados tras la llamada? Para ello, debe tenerse en cuenta que getEjemplarRandom() retornará los ejemplares en el orden siguiente (se supone, por simplicidad, que los ejemplares se identifican inequívocamente con un identificador único): 3, 5, 4, 1, 4, 6, 9, 2, 1, 7, 9, 1, 3.

- A.4
- B.5
- C.7
- D.8

5.4: En la práctica de JDBC, si se ejecuta el método de coronapokerus(), sin ejemplares previamente infectados y con una duración de 4 días, ¿cuántos ejemplares quedarán infectados tras la llamada? Para ello, debe tenerse en cuenta que getEjemplarRandom() retornará los ejemplares en el orden siguiente (se supone, por simplicidad, que los ejemplares se identifican inequívocamente con un identificador único): 3, 3, 4, 4, 3, 6, 9, 2, 1, 7, 9, 1, 3.

A.2

B.4

C.5

D.8

6.1: En la práctica de JDBC, si se ejecuta el método de coronapokerus(), sin ejemplares previamente infectados y con una duración de 5 días, ¿cuántos ejemplares quedarán infectados tras la llamada? Para ello, debe tenerse en cuenta que getEjemplarRandom() retornará los ejemplares en el orden siguiente (se supone, por simplicidad, que los ejemplares se identifican inequívocamente con un identificador único), provocando el ejemplar 6 un NullPointerException(): 3, 5, 4, 3, 4, 6, 1, 2, 1, 7, 9, 1, 3, 2, 2, 8.

A.2

B.3

C.8

D.16

6.2: En la práctica de JDBC, si se ejecuta el método de coronapokerus(), sin ejemplares previamente infectados y con una duración de 5 días, ¿cuántos ejemplares quedarán infectados tras la llamada? Para ello, debe tenerse en cuenta que getEjemplarRandom() retornará los ejemplares en el orden siguiente (se supone, por simplicidad, que los ejemplares se identifican inequívocamente con un identificador único), provocando el ejemplar 1 un NullPointerException(): 3, 5, 4, 1, 4, 6, 1, 2, 1, 7, 9, 1, 3, 2, 2, 8.

A.2

B.3

C.8

D.16

6.3: En la práctica de JDBC, si se ejecuta el método de coronapokerus(), sin ejemplares previamente infectados y con una duración de 5 días, ¿cuántos ejemplares quedarán infectados tras la llamada? Para ello, debe tenerse en cuenta que getEjemplarRandom() retornará los ejemplares en el orden siguiente (se supone, por simplicidad, que los ejemplares se identifican inequívocamente con un identificador único), provocando el ejemplar 1 un NullPointerException(): 3, 5, 4, 5, 4, 6, 2, 8, 1, 7, 9, 1, 3, 2, 2, 8.

A.5

B.6

C.8

D.16

6.4: En la práctica de JDBC, si se ejecuta el método de coronapokerus(), sin ejemplares previamente infectados y con una duración de 5 días, ¿cuántos ejemplares quedarán infectados tras la llamada? Para ello, debe tenerse en cuenta que getEjemplarRandom() retornará los ejemplares en el orden siguiente (se supone, por simplicidad, que los ejemplares se identifican inequívocamente con un identificador único), provocando el ejemplar 9 un NullPointerException(): 3, 5, 4, 5, 8, 6, 2, 8, 1, 7, 9, 1, 3, 2, 2, 8.

A.5

B.6

C.8

D.16

JUNIO 2020 - FINAL

1.1: Señala la afirmación correcta:

- A.Cualquier aplicación que use ODBC podrá acceder a cualquier DBMS.
- B.Cualquier aplicación que use ODBC podrá acceder a cualquier DBMS que implemente un módulo de peticiones ODBC.
- C.ODBC sólo permite conectarnos a DBMS de Microsoft.
- D.ODBC sólo permite conectarnos a DBMS que se ejecuten sobre Linux.

1.2: Señala la afirmación correcta:

- A.ODBC necesita una capa de cliente intermedia, mientras que JDBC no.
- B.JDBC necesita una capa de cliente intermedia, mientras que ODBC no.
- C.Ni ODBC, ni JDBC necesitan una capa de cliente intermedia.
- D.Tanto ODBC, como JDBC necesitan una capa de cliente intermedia.

1.3: Señala la afirmación correcta:

- A.Para usar JDBC es necesario descargar el driver correspondiente al DBMS que se va a usar.
- B.Cualquier aplicación podrá acceder a cualquier DBMS que implemente un módulo de peticiones JDBC.
- C.JDBC sólo permite conectarnos a DBMS de Microsoft.
- D.JDBC sólo permite conectarnos a DBMS que se ejecuten sobre Linux.

1.4: Señala la afirmación correcta:

- A.Es posible usar JDBC directamente desde Python.
- B.Es posible usar ODBC directamente desde C.
- C.Es posible usar JDBC directamente desde C.
- D.Ninguna de las otras afirmaciones es cierta.

2.1: ¿Qué método de Class se usa para cargar el driver de JDBC?

- A.loadDriver()
- B.getConnection()
- C.forName()
- D.searchDriver()

2.2: ¿En qué clase se encuentra el método getConnection()?

- A.Connection
- B.Class
- C.DriverManager
- D.Statement

2.3: ¿Qué excepción de las siguientes puede surgir al cargar el driver de JDBC?

- A.ClassNotFoundException
- B.DriverNotFoundException
- C.SQLException
- D.IncorrectOriginException

2.4: ¿Qué ocurre si intento acceder con una combinación de usuario y contraseña correctos a un esquema que no existe en el DMBS al que me conecto?

- A.Se produce una SQLException indicando que no puedo conectarme al servidor.
- B.Se retorna null al intentar obtener la conexión.
- C.Se produce una DatabaseNotFoundException.
- D.Se produce una SQLException indicando que no puedo conectarme a esa base de datos.

3.1: ¿Cuántos bloques de transacción hay en el siguiente código si el autocommit está activado?

- A.7
- B.8
- C.9
- D.10

```
DELETE  
INSERT  
START TRANSACTION  
INSERT  
SELECT  
COMMIT  
DELETE  
START TRANSACTION  
DELETE  
SELECT  
INSERT  
ROLLBACK  
INSERT  
INSERT
```

3.2: ¿Cuántos bloques de transacción hay en el siguiente código si el autocommit está activado?

- A.6
- B.7
- C.8
- D.9

```
INSERT  
DELETE  
START TRANSACTION  
INSERT  
SELECT  
INSERT  
START TRANSACTION  
DELETE  
INSERT  
ROLLBACK  
DELETE  
INSERT
```

3.3: ¿Cuántos bloques de transacción hay en el siguiente código si el autocommit está desactivado?

- A.3
- B.4
- C.6
- D.9

```
INSERT  
SELECT  
INSERT  
ROLLBACK  
DELETE  
START TRANSACTION  
DELETE  
INSERT  
ROLLBACK  
INSERT  
INSERT  
SELECT  
COMMIT
```

3.4: ¿Cuántos bloques de transacción hay en el siguiente código si el autocommit está desactivado?

- A.2
- B.3
- C.6
- D.10

```
INSERT  
INSERT  
SELECT  
ROLLBACK  
INSERT  
DELETE  
INSERT  
START TRANSACTION  
DELETE  
SELECT  
COMMIT
```

4.1: ¿Cuántas operaciones del siguiente código modifican el contenido de la base de datos si el autocommit está activado?

- A.4
- B.5
- C.6
- D.8

```
INSERT  
START TRANSACTION  
INSERT  
SELECT  
COMMIT  
DELETE  
START TRANSACTION  
DELETE  
INSERT  
ROLLBACK  
INSERT  
INSERT
```

4.2: ¿Cuántas operaciones del siguiente código modifican el contenido de la base de datos si el autocommit está activado?

- A.4
- B.5
- C.6
- D.7

```
INSERT
DELETE
START TRANSACTION
INSERT
SELECT
START TRANSACTION
DELETE
INSERT
SELECT
ROLLBACK
DELETE
```

4.3: ¿Cuántas operaciones del siguiente código modifican el contenido de la base de datos si el autocommit está desactivado?

- A.1
- B.3
- C.5
- D.7

```
INSERT
INSERT
ROLLBACK
DELETE
START TRANSACTION
DELETE
INSERT
ROLLBACK
SELECT
SELECT
COMMIT
```

4.4: ¿Cuántas operaciones del siguiente código modifican el contenido de la base de datos si el autocommit está desactivado?

- A.2
- B.3
- C.4
- D.7

```
INSERT
SELECT
INSERT
ROLLBACK
INSERT
DELETE
SELECT
START TRANSACTION
DELETE
COMMIT
```

5.1: ¿A qué fila apunta un ResultSet antes de ejecutar ninguna operación sobre él?

- A.A la primera.
- B.Antes de la primera.
- C.A después de la primera.
- D.A la última.

5.2: ¿Qué ocurre si se realiza un next() sobre un ResultSet que ya apunta a la última fila?

- A.Retorna false.
- B.Se produce un NullPointerException.
- C.Retorna null.
- D.Retorna 0.

5.3: ¿Cuántas llamadas a next() se pueden hacer sobre un ResultSet antes de que se produzca una excepción?

- A.Tantas como filas tenga.
- B.Tantas como filas tenga, más una.
- C.Tantas como filas tenga, menos una.
- D.Tantas como se quiera.

5.4: ¿Cuántas llamadas a next() hay que hacer sobre un ResultSet para que retorne false?

- A.Tantas como filas tenga.
- B.Tantas como filas tenga, más una.
- C.Tantas como filas tenga, menos una.
- D.Nunca retornará false.

6.1: ¿Sobre un objeto de qué clase se puede ejecutar el método getColumnName()?

- A.ResultSet
- B.ResultSetMetaData
- C.DatabaseMetaData
- D.Connection

6.2: ¿Sobre un objeto de qué clase se puede ejecutar el método getDriverName()?

- A.ResultSet
- B.ResultSetMetaData
- C.DatabaseMetaData
- D.Connection

6.3: Queremos insertar la fecha actual en una base de datos, para ello tenemos ya la siguiente línea (que se ejecuta sin problemas): `long currentTime = System.currentTimeMillis();` ¿Cómo deberíamos obtener la fecha de forma que la variable resultante se pueda insertar directamente en la base de datos en una columna de tipo DATE?

- A.java.sql.Date fecha = new java.sql.Date(currentTime);
- B.java.sql.Time fecha = new java.sql.Time(currentTime);
- C.java.util.Date fecha = new java.util.Date(currentTime);
- D.java.util.Time fecha = new java.util.Time(currentTime);

6.4: Queremos insertar la hora actual en una base de datos, para ello tenemos ya la siguiente línea (que se ejecuta sin problemas): `long currentTime = System.currentTimeMillis();` ¿Cómo deberíamos obtener la hora de forma que la variable resultante se pueda insertar directamente en la base de datos en una columna de tipo TIME?

- A.java.sql.Date fecha = new java.sql.Date(currentTime);
- B.java.sql.Time fecha = new java.sql.Time(currentTime);
- C.java.util.Date fecha = new java.util.Date(currentTime);
- D.java.util.Time fecha = new java.util.Time(currentTime);

7.1: Señala la afirmación incorrecta.

- A.El tipo BLOB de SQL permite almacenar objetos binarios.
- B.El tipo BINARY de SQL permite almacenar objetos binarios.
- C.El tipo BLOB de SQL ocupa siempre un tamaño fijo en la base de datos.
- D.El tipo BINARY de SQL ocupa siempre un tamaño fijo en la base de datos.

7.2: ¿Qué tipos de columna podrían usarse para almacenar una imagen de 60 KB en una base de datos?

- A.Sólo podría usarse LONGBLOB.
- B.TINYBLOB o BLOB.
- C.Podrían usarse LONGBLOB o MEDIUMBLOB, pero no BLOB.
- D.LONGBLOB, MEDIUMBLOB o BLOB.

7.3: Se van a almacenar emojis con flechas en blanco y negro en la base de datos, con una resolución de 15x15 píxeles y usando 1 bit por píxel (tamaño máximo de cada imagen de 225 bytes). ¿Qué tipo podría usarse en la columna de la base de datos que las almacene?

- A.Sólo TINYBLOB.
- B.Sólo BLOB.
- C.Sólo LONGBLOB.
- D.Tanto TINYBLOB, como BLOB, como LONGBLOB.

7.4: ¿Qué tipos de columna podrían usarse para almacenar una imagen de 6 MB en una base de datos?

- A.Sólo podría usarse LONGBLOB.
- B.TINYBLOB o BLOB.
- C.Podrían usarse LONGBLOB o MEDIUMBLOB, pero no BLOB.
- D.LONGBLOB, MEDIUMBLOB o BLOB.

8.1: En el siguiente código en Java se imprime el contenido de una tabla llamada “ejemplar”, la cual tiene las siguientes columnas: id (de tipo INT), apodo (de tipo VARCHAR(20)), nivel (de tipo INT) y sexo (de tipo CHAR(1)).La variable conn es un objeto de tipo Connection y se asume que contiene la conexión sin error a la base de datos que contiene la tabla citada. ¿Qué error hay en el código?

```
private void consultarDatos() {
    Statement st = null;
    ResultSet rs = null;

    try {
        st = conn.createStatement();
        String query = "SELECT * FROM ejemplar";
        rs = st.executeQuery(query);

        while (rs.next()) {
            int nivel = rs.getInt("nivel");
            char sexo = rs.getChar("sexo");
            System.out.println("Nivel: " + nivel + ", Sexo: " + sexo);
        }
    } catch (SQLException e) {
        System.out.println("Error.");
    } finally {
        try {
            if (rs != null) rs.close();
            if (st != null) st.close();
        } catch (SQLException e) {
            System.out.println("Error");
        }
    }
}
```

- A.Debería usarse un PreparedStatement en lugar de un Statement.
- B.No existe la función getChar() para la clase ResultSet.
- C.No se capturan todas las excepciones.
- D.El código es correcto tal y como se muestra.

8.2: En el siguiente código en Java se imprime el contenido de una tabla llamada “ejemplar”, la cual tiene las siguientes columnas: id (de tipo INT), apodo (de tipo VARCHAR(20)), nivel (de tipo INT) y sexo (de tipo CHAR(1)).La variable conn es un objeto de tipo Connection y se asume que contiene la conexión sin error a la base de datos que contiene la tabla citada. ¿Qué error hay en el código?

```
private void consultarDatos() {
    Statement st = null;
    ResultSet rs = null;

    try {
        st = conn.createStatement();
        String query = "SELECT * FROM ejemplar";
        rs = st.executeQuery(query);

        while (rs.next()) {
            int nivel = rs.getInt("nivel");
            char sexo = rs.getString("sexo").charAt(0);
            System.out.println("Nivel: " + nivel + ", Sexo: " + sexo);
        }
    } catch (SQLException e) {
        System.out.println("Error.");
    } finally {
        if (rs != null) rs.close();
        if (st != null) st.close();
    }
}
```

- A.Debería usarse un PreparedStatement en lugar de un Statement.
- B.Debería usarse el método getChar() en lugar de getString() para obtener la columna “sexo”.
- C.No se capturan todas las excepciones.
- D.El código es correcto tal y como se muestra.

8.3: En el siguiente código en Java se imprime el contenido de una tabla llamada “ejemplar”, la cual tiene las siguientes columnas: id (de tipo INT), apodo (de tipo VARCHAR(20)), nivel (de tipo INT) y sexo (de tipo CHAR(1)). La variable conn es un objeto de tipo Connection y se asume que contiene la conexión sin error a la base de datos que contiene la tabla citada. ¿Qué error hay en el código?

```

private void consultarDatos() {
    Statement st = null;
    ResultSet rs = null;

    try {
        st = conn.createStatement();
        String query = "SELECT * FROM ejemplar";
        rs = st.executeQuery(query);

        while (rs.next()) {
            int nivel = rs.getInt("nivel");
            char sexo = rs.getString("sexo").charAt(0);
            System.out.println("Nivel: " + nivel + ", Sexo: " + sexo);
        }
    } catch (SQLException e) {
        System.out.println("Error.");
    } finally {
        try {
            if (rs != null) rs.close();
            if (st != null) st.close();
        } catch (SQLException e) {
            System.out.println("Error");
        }
    }
}

```

- A.Debería usarse un PreparedStatement en lugar de un Statement.
- B.Debería usarse el método getChar() en lugar de getString() para obtener la columna “sexo”.
- C.No se capturan todas las excepciones.
- D.El código es correcto tal y como se muestra.

8.4: En el siguiente código en Java se imprimen algunos datos de una tabla llamada “ejemplar”, la cual tiene las siguientes columnas: id (de tipo INT), apodo (de tipo VARCHAR(20)), nivel (de tipo INT) y sexo (de tipo CHAR(1)). La variable conn es un objeto de tipo Connection y se asume que contiene la conexión sin error a la base de datos que contiene la tabla citada. ¿Qué error hay en el código?

```

private void consultarDatos(String apodo) {
    Statement st = null;
    ResultSet rs = null;

    try {
        st = conn.createStatement();
        String query = "SELECT * FROM ejemplar WHERE apodo=" + apodo;
        rs = st.executeQuery(query);

        while (rs.next()) {
            int nivel = rs.getInt("nivel");
            char sexo = rs.getString("sexo").charAt(0);
            System.out.println("Nivel: " + nivel + ", Sexo: " + sexo);
        }
    } catch (SQLException e) {
        System.out.println("Error.");
    } finally {
        try {
            if (rs != null) rs.close();
            if (st != null) st.close();
        } catch (SQLException e) {
            System.out.println("Error");
        }
    }
}

```

- A.Debería usarse un PreparedStatement en lugar de un Statement.
- B.Debería usarse el método getChar() en lugar de getString() para obtener la columna “sexo”.
- C.No se capturan todas las excepciones.
- D.El código es correcto tal y como se muestra.

9.1: En el siguiente código en Java se carga una imagen en la columna “imagen” (de tipo LONGBLOB) de la tabla “especie”. La columna “nombre” está definida como UNIQUE en dicha tabla. La variable conn es un objeto de tipo Connection y se asume que contiene la conexión sin error a la base de datos que contiene la tabla citada. ¿Qué error hay en el código?

```

private void establecerImagen(String nombre, String filename) {
    PreparedStatement pst = null;
    String query = "UPDATE especie SET imagen = ? WHERE nombre = ?";
    FileInputStream fis = null;

    try {
        File f = new File(filename);
        fis = new FileInputStream(f);
        pst = conn.prepareStatement(query);

        pst.setBinaryStream(1, fis, (int)fis.length());
        pst.setString(2, nombre);
        pst.executeUpdate();
    } catch (SQLException e) {
        System.out.println("Error");
    } catch (FileNotFoundException e) {
        System.out.println("Error");
    } finally {
        try {
            if (fis != null) fis.close();
        } catch (IOException e) {
            System.out.println("Error");
        }
        try {
            if (pst != null) pst.close();
        } catch (SQLException e) {
            System.out.println("Error");
        }
    }
}

```

A.No es necesario un FileInputStream, se debería hacer el setBinaryStream() con un objeto de tipo File directamente.

B.Debería realizarse un executeQuery() en lugar de un executeUpdate().

C.Debe llamarse al método length() con el objeto de la clase File en lugar de con el de la clase FileInputStream.

D.Ninguno, el código es correcto.

10.1: ¿Cuál de los siguientes elementos podría activarse de forma automática al hacer una inserción en una tabla?

A. Sólo un procedimiento almacenado.

B. Sólo un trigger.

C. Tanto un procedimiento almacenado, como un trigger.

D. Ni un procedimiento almacenado, ni un trigger.

9.2: En el siguiente código en Java se carga una imagen en la columna “imagen” (de tipo LONGBLOB) de la tabla “especie”. La columna “nombre” está definida como UNIQUE en dicha tabla. La variable conn es un objeto de tipo Connection y se asume que contiene la conexión sin error a la base de datos que contiene la tabla citada. ¿Qué error hay en el código?

```

private void establecerImagen(String nombre, String filename) {
    PreparedStatement pst = null;
    String query = "UPDATE especie SET imagen = ? WHERE nombre = ?";

    try {
        File f = new File(filename);
        pst = conn.prepareStatement(query);

        pst.setBinaryStream(1, f, (int)f.length());
        pst.setString(2, nombre);
        pst.executeUpdate();
    } catch (SQLException e) {
        System.out.println("Error");
    } finally {
        try {
            if (pst != null) pst.close();
        } catch (SQLException e) {
            System.out.println("Error");
        }
    }
}

```

- A.No es posible llamar al método setBinaryStream() con un objeto de la clase File, es necesario un objeto de la clase FileInputStream.
- B.Debería realizarse un executeQuery() en lugar de un executeUpdate().
- C.La clase File no tiene un método length().
- D.Ninguno, el código es correcto.

10.2: ¿Cuál de los siguientes elementos queda almacenado en la base de datos?

- A.Sólo un procedimiento almacenado.
- B.Sólo un trigger.
- C.Tanto un procedimiento almacenado, como un trigger.
- D.Ni un procedimiento almacenado, ni un trigger.

9.3: En el siguiente código en Java se carga una imagen en la columna “imagen” (de tipo LONGBLOB) de la tabla “especie”. La columna “nombre” está definida como UNIQUE en dicha tabla. La variable conn es un objeto de tipo Connection y se asume que contiene la conexión sin error a la base de datos que contiene la tabla citada. ¿Qué error hay en el código?

```

private void establecerImagen(String nombre, String filename) {
    PreparedStatement pst = null;
    String query = "UPDATE especie SET imagen = ? WHERE nombre = ?";
    FileInputStream fis = null;

    try {
        File f = new File(filename);
        fis = new FileInputStream(f);
        pst = conn.prepareStatement(query);

        pst.setBinaryStream(1, fis, (int)f.length());
        pst.setString(2, nombre);
        pst.executeUpdate();
    } catch (SQLException e) {
        System.out.println("Error");
    } catch (FileNotFoundException e) {
        System.out.println("Error");
    } finally {
        try {
            if (fis != null) fis.close();
        } catch (IOException e) {
            System.out.println("Error");
        }
        try {
            if (pst != null) pst.close();
        } catch (SQLException e) {
            System.out.println("Error");
        }
    }
}

```

- A.No es necesario un FileInputStream, se debería hacer el setBinaryStream() con un objeto de tipo File directamente.
- B.Debería realizarse un executeQuery() en lugar de un executeUpdate().
- C.La clase File no tiene un método length().
- D.Ninguno, el código es correcto.

10.3: ¿Cuál de los siguientes elementos es el encargado de gestionar las peticiones que llegan mediante JDBC?

- A.Sólo un procedimiento almacenado.
- B.Sólo un trigger.
- C.Tanto un procedimiento almacenado, como un trigger.
- D.Ni un procedimiento almacenado, ni un trigger.