BASES DE DATOS

Ingeniería Informática

Matemáticas e Informática

BASES DE DATOS

Acceso a Bases de datos en Java

BASES DE DATOS

Contacto con Prof. Alejandro Rodríguez

Email: alejandro.rg@upm.es

Fase 3: Crear y ejecutar consulta

```
private void init() throws Exception {
    String drv = "com.mysql.jdbc.Driver";
    Class.forName(drv);

    String serverAddress = "localhost:3306";
    String db = "sakila";
    String user = "bd";
    String pass = "bdupm";
    String url = "jdbc:mysql://" + serverAddress + "/" + db;
    Connection conn = DriverManager.getConnection(url, user, pass);
    System.out.println("Conectado a la base de datos!");

    Statement st = conn.createStatement();
    ResultSet rs = st.executeQuery("SELECT * FROM actor");
    System.out.println("Query ejecutada!");
}
```

Ejecutamos resulta pero no manipulamos ResultSet

Fase 3: Crear y ejecutar consulta

Ver ejemplo – EJ3

Fase 4: Obtener resultados (I)

```
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery("SELECT * FROM actor");
System.out.println("Query ejecutada!");

while (rs.next()) {
    int id = rs.getInt("actor_id");
    String firstName = rs.getString("first_name");
    String lastName = rs.getString(3); // tercera columna. Empiezan en 1, no en 0
    Date lastUpdate = rs.getDate("last_update");

    System.out.println("Actor:");
    System.out.println("\tID: " + id);
    System.out.println("\tName: " + firstName);
    System.out.println("\tLast name: " + lastName);
    System.out.println("\tLast update: " + lastUpdate.toString());
}
```

Leemos el ResultSet.

Fase 4: Obtener resultados (I)

Ver ejemplo – EJ4

Fase 4: Obtener resultados (I)

Métodos de ResultSet	Tipo Java devuelto
getInt	int
getLong	long
getFloat	float
getDouble	double
getBoolean	boolean
getString	String
getDate	java.sql.Date
getTime	java.sql.Time
getTimestamp	java.sql.Timestamp

Fase 5: Liberar recursos

```
rs.close();
st.close();
conn.close();
```

Fase 5: Liberar recursos

¿Debemos cerrar la conexión (conn.close) tras cada consulta? ¿Por qué?

Fase 5: Liberar recursos

¿Debemos cerrar la conexión (conn.close) tras cada consulta? ¿Por qué?

No, es ineficiente si se van a ejecutar varias cosas.

Posibles errores

Errores más comunes:

- Driver no cargado (no se encuentra en classpath).
- Fallo de conexión.
- No existe la base de datos.
- Error de sintaxis en sentencia SQL.
- Errores de permisos.
- Violación reglas integridad referencial.

¿Qué/como debemos comprobar para solucionar cada error?

Posibles errores

Cualquier de esos errores producirá una excepción.

En función del tipo de excepción o de los datos que nos de la misma (códigos de error, por ejemplo) podemos saber el error concreto.

Manejo de excepciones

```
try{ ... // las 5 fases descritas anteriormente}

catch(SQLException se) {
            //se.printStackTrace();
            System.out.println("Mensaje errorr : "+se.getMessage());
            System.out.println("Codigo error : "+se.getErrorCode());
            System.out.println("Estado SQL : "+se.getSQLState());
    }
 catch (ClassNotFoundException e) {
             e.printStackTrace();   //No se encuentra el driver//
             }
  catch(Exception e) {
             System.out.println("Se produjo un error inesperado: "+e.getMessage());
             }
   finally {
            try {
                        System.out.println("Ejecuto finally \n");
                        if (resultSet != null) resultSet.close();
                        if (statement != null) statement.close();
                        if (connection != null) connection.close();
            }
            catch (SQLException e)
            { e.printStackTrace(); }
            }
    }
}
```

PreparedStatement

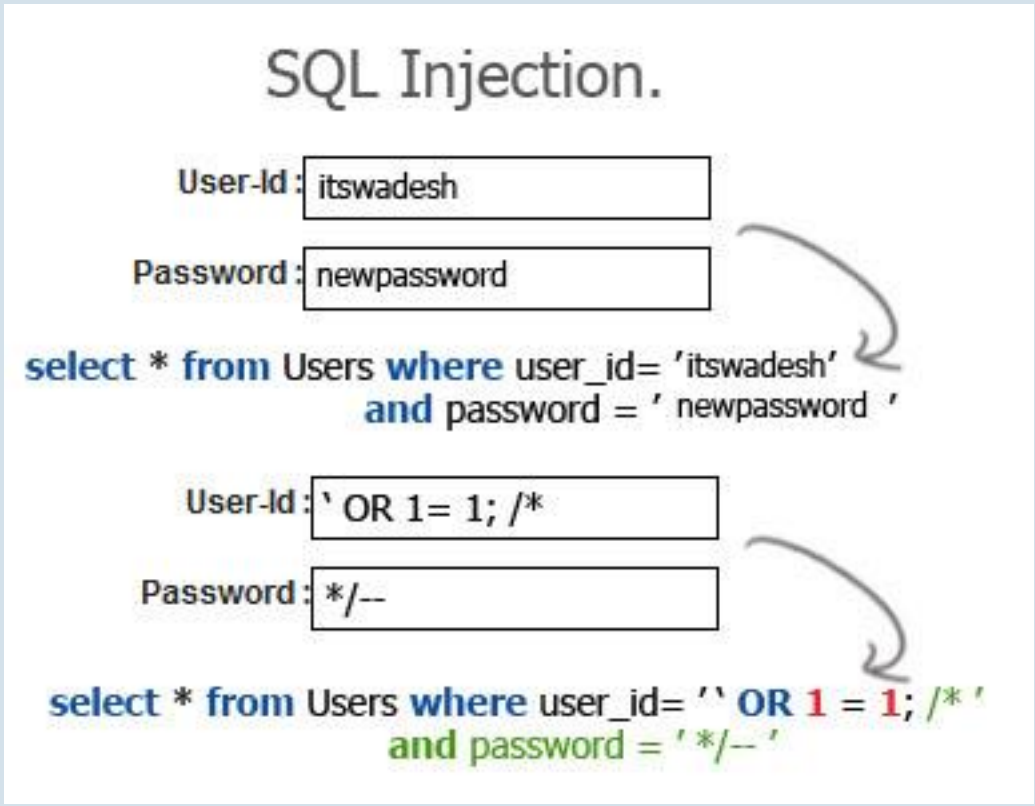
Cada vez que enviamos una consulta al SGBD, éste:

- La analiza sintácticamente (Query Processor)
- Construye un plan para ejecutarla (Query Optimizer)

PreparedStatement

- Si tenemos un bucle donde repetidamente lanzamos la misma query con diferentes parámetros es ineficiente usar la clase Statement.
- Es mejor usar para estas situaciones PreparedStatement.
- Además evita los ataques por inyección de código SQL en Java.

SQL Injection



Fuente: http://bit.ly/1NSAxh1

SQL Injection

```
public ArrayList<String> getActors(boolean stB, String lname) throws Exception {

    ResultSet rs = null;
    Statement st = null;


        String query = "select first_name, last_name from sakila.actor where last_name ='" + lname + "'";
        System.out.println("Executing with Statement");
        System.out.println("-> Query to execute: " + query);
        st = conn.createStatement();
        rs = st.executeQuery(query);
```

SQL Injection

Vamos a ver el ejemplo en código:
https://youtu.be/yjNDRhie5p8

Insert

```
String nombres[] = { "Clara", "Dani", "Edward", "Denzel" };
String apellidos[] = { "Lago", "Rovira", "Norton", "Washington" };

// No hace falta pasar ID porque es autoincremental
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()));
    int res = pst.executeUpdate();
    System.out.println("Insertado correctamente? " + ((res == 1)?"Si":"No"));
}
System.out.println("Query ejecutada!");

pst.close();
conn.close();
```

Insertar varios registros usando PreparedStatement: más eficiente, evitamos SQL Injection.

Insert

Ver ejemplo – EJ2_4

Update (I)

```
Statement st = conn.createStatement();

int result = st.executeUpdate("UPDATE actor SET first_name = 'Daniel' WHERE first_name = 'Dani'");

System.out.println("Número de filas afectadas: " + result);
System.out.println("Query ejecutada!");
```

Actualización de registros mediante UPDATE.

Update (I)

Ver ejemplo – EJ2_5

Update (II)

```
Statement st = conn.createStatement();

int result = st.executeUpdate("UPDATE actor SET actor_id = 1 WHERE first_name = 'Daniel'");

System.out.println("Número de filas afectadas: " + result);
System.out.println("Query ejecutada!");
```

```
Conectado a la base de datos!
Error al conectar a la BD: Duplicate entry '1' for key 'PRIMARY'
```

¿Qué ocurre? ¿Qué significa ese error?

Update (II)

Ver ejemplo – EJ2_6

Delete (I)

```
Statement st = conn.createStatement();

int result = st.executeUpdate("DELETE FROM actor where actor_id = '5'");

System.out.println("Número de filas afectadas: " + result);
System.out.println("Query ejecutada!");
```

```
Cannot delete or update a parent row: a foreign key constraint fails (`sakila`.`film_actor`, CONSTRAINT
`fk_film_actor_actor` FOREIGN KEY (`actor_id`) REFERENCES `actor` (`actor_id`) ON UPDATE CASCADE)
```

¿Qué ocurre? ¿Qué significa ese error?

Delete (I)

Ver ejemplo – EJ2_7

Delete (II)

```
Statement st = conn.createStatement();

int result = st.executeUpdate("DELETE FROM rental where rental_id = '1'");

System.out.println("Número de filas afectadas: " + result);
System.out.println("Query ejecutada!");
```

Delete (II)

Ver ejemplo – EJ2_8

Que clases y métodos usar

executeQuery	*Select*
executeUpdate	*Insert, Update, Delete*
PreparedStatement	*Cuando se realiza una operación varias veces*

Clase DatabaseMetaData

```
DatabaseMetaData dbMet = conn.getMetaData();

System.out.println("Tipo de BD: " + dbMet.getDatabaseProductName());
System.out.println("Versión: " + dbMet.getDatabaseProductVersion());
System.out.println("Información del driver:");
System.out.println("\tNombre: " + dbMet.getDriverName());
System.out.println("\tVersión: " + dbMet.getDriverVersion());
```

Clase DatabaseMetaData

Ver ejemplo – EJ2_13

Estructura de tablas

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

System.out.println("Numero de columnas: " + rsmd.getColumnCount());

for (int i = 1; i < rsmd.getColumnCount(); i++) {
    System.out.println("Columna " + i + ": " + rsmd.getColumnName(i));
    System.out.println("Etiqueta " + i + ": " + rsmd.getColumnLabel(i));
    System.out.println("Tipo de columna " + i + ": " + rsmd.getColumnTypeName(i));
}
```

También podemos obtener información de la estructura de tablas usando la clase ResultSetMetaData.

Estructura de tablas

Ver ejemplo – EJ2_14

Funciones de tiempo en Java-SQL (I)

```
long msActuales = System.currentTimeMillis(); //tiempo actual en milisegundos
java.sql.Date fechaEnSQLDate = new java.sql.Date(msActuales);
java.sql.Time horaEnSQLDate = new java.sql.Time(msActuales);
java.sql.Timestamp sqlTimestamp = new java.sql.Timestamp(msActuales);

//otra forma sin usar System.currentTimeMillis y usando objetos Date de java.util
java.sql.Date fechaEnSQLDate2 = new java.sql.Date(new java.util.Date().getTime());

System.out.println(fechaEnSQLDate.toString());
System.out.println(horaEnSQLDate.toString());
System.out.println(sqlTimestamp.toString());

System.out.println(fechaEnSQLDate2.toString());
```

Podemos convertir tipos de fecha en Java a SQL de forma sencilla. Tenemos los tipos Date, Time y TimeStamp como básicos.

Info sobre Date y Timestamp: http://stackoverflow.com/questions/409286/should-i-use-field-datetime-or-timestamp

Funciones de tiempo en Java-SQL (I)

Ver ejemplo – EJ2_15

Funciones de tiempo en Java-SQL (II)

```
Calendar now = Calendar.getInstance();

java.util.Date dateHoy = now.getTime();

System.out.println("Date from Calendar: " + dateHoy.toString());

long msActuales = dateHoy.getTime();

java.sql.Date fechaEnSQLDate = new java.sql.Date(msActuales);
java.sql.Time horaEnSQLDate = new java.sql.Time(msActuales);
java.sql.Timestamp sqlTimestamp = new java.sql.Timestamp(msActuales);

System.out.println(fechaEnSQLDate.toString());
System.out.println(horaEnSQLDate.toString());
System.out.println(sqlTimestamp.toString());
```

Funciones de tiempo en Java-SQL (II)

Ver ejemplo – EJ2_16

Funciones de tiempo en Java-SQL (III)

```
String query = "INSERT INTO actor (first_name, last_name, last_update) VALUES (?,?,?);";
PreparedStatement pst = conn.prepareStatement(query);

pst.setString(1, "Charlize");
pst.setString(2, "Theron");
pst.setDate(3, new java.sql.Date(System.currentTimeMillis()));
```

Insertamos fechas usando un PreparedStatement donde pasamos la fecha con un objeto java.sql.Date

En este ejemplo pasamos directamente la fecha actual.

Funciones de tiempo en Java-SQL (III)

Ver ejemplo – EJ2_17

Funciones de tiempo en Java-SQL (IV)

```
String query = "INSERT INTO actor (first_name, last_name, last_update) VALUES (?,?,?);";
PreparedStatement pst = conn.prepareStatement(query);

// con Date (deprecated)
java.util.Date fechaUtil = new java.util.Date("2013/01/15");
java.sql.Date fecha = new java.sql.Date(fechaUtil.getTime());

// con Calendar

Calendar cal = Calendar.getInstance();
cal.set(Calendar.YEAR, 2013);
cal.set(Calendar.MONTH, Calendar.MAY);
cal.set(Calendar.DAY_OF_MONTH, 15);
// fecha = new java.sql.Date(cal.getTime().getTime()); //descomentar esta línea para usar Calendar

pst.setString(1, "Scarlett");
pst.setString(2, "Johansson");
pst.setDate(3, fecha);
```

Misma dinámica pero usando una fecha específica tanto con Date como con Calendar.

Funciones de tiempo en Java-SQL (IV)

Ver ejemplo – EJ2_18

Funciones de tiempo en Java-SQL (V)

```
Calendar cal1 = Calendar.getInstance();
cal1.set(Calendar.YEAR, 2016);
cal1.set(Calendar.MONTH, Calendar.JUNE);
cal1.set(Calendar.DAY_OF_MONTH, 12);

Calendar cal2 = Calendar.getInstance();
cal2.set(Calendar.YEAR, 2014);
cal2.set(Calendar.MONTH, Calendar.DECEMBER);
cal2.set(Calendar.DAY_OF_MONTH, 26);

java.sql.Timestamp t1 = new java.sql.Timestamp(cal1.getTime().getTime());
java.sql.Timestamp t2 = new java.sql.Timestamp(cal2.getTime().getTime());

int compare = t1.compareTo(t2);

System.out.println("T1: " + t1.toString());
System.out.println("T2: " + t2.toString());
if (compare == 0) {
    System.out.println("Timestamp iguales");
}
if (compare > 0) {
    System.out.println("T1 es despues de T2");
}
if (compare < 0) {
    System.out.println("T1 es antes de T2");
}
```

Comparación de fechas.

Funciones de tiempo en Java-SQL (V)

Ver ejemplo – EJ2_19

Ficheros binarios en Java (I)

Modificamos la tabla “actor” para que podamos añadir ficheros binarios.

Crearemos una nueva columna que acepte el tipo LONGBLOB (Binary Large Object):

ALTER TABLE `sakila`.`actor` ADD COLUMN `picture` LONGBLOB NULL AFTER `last_update`;

Ficheros binarios en Java (I): Enviar

```
String query = "INSERT INTO actor (first_name, last_name, last_update, picture) VALUES (?,?,?,?);";
PreparedStatement pst = conn.prepareStatement(query);

pst.setString(1, "Megan");
pst.setString(2, "Fox");
pst.setDate(3, new java.sql.Date(System.currentTimeMillis()));

File file = new File("pics/meganfox.jpg");
FileInputStream fis = new FileInputStream(file);
pst.setBinaryStream(4, fis, (int)file.length());
pst.executeUpdate();

System.out.println("Añadido!");
```

Añadimos el fichero usando FileInputStream.

Ficheros binarios en Java (I): Enviar

Ver ejemplo – EJ2_20

Ficheros binarios en Java (II): Obtener

```
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();
System.out.println("Fichero guardado!");
```

Guardarmos el fichero con FileOutputStream.

Ficheros binarios en Java (II): Obtener

Ver ejemplo – EJ2_21

Actualizar BD a través de ResultSet (I)

```
Statement st = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
ResultSet rs = st.executeQuery("SELECT * FROM actor where actor_id >= 40 and actor_id <= 50;");
//para situarnos bien, cogemos los actores con ids entre 40 y 50 (no hay ningún Morgan)
// imprimo los nombres

System.out.println("Nombres antes!:\n");
while (rs.next()) {
    String name = rs.getString("first_name");
    System.out.println(name);
}
System.out.println();

rs.beforeFirst();

rs.absolute(2); //nos movemos a la fila 2 (2° resultado)
rs.updateString(2, "Morgan"); // actualizamos la columna 2 (first_name) de ese registro
rs.updateRow(); // actualizamos el registro
rs.beforeFirst(); // movemos el cursor al principio, justo antes del primer elemento

System.out.println("Nombres después!:\n");
while (rs.next()) {
    String name = rs.getString("first_name");
    System.out.println(name);
}
System.out.println();
```

Actualizamos un registro: UPDATE ROW.

Actualizar BD a través de ResultSet (I)

Ver ejemplo – EJ2_22

Actualizar BD a través de ResultSet (II)

```
Statement st = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
ResultSet rs = st.executeQuery("SELECT * FROM actor where actor_id >= 200");
//para situarnos bien, cogemos los actores con ids entre >= 200
// imprimo los nombres

System.out.println("Nombres antes!:\n");
while (rs.next()) {
    String id = rs.getString("actor_id");
    String name = rs.getString("first_name");
    System.out.println(id + ": " + name);
}
System.out.println();

rs.moveToInsertRow(); // nos movemos en 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();

rs.beforeFirst(); // movemos el cursor al principio, justo antes del primer elemento

System.out.println("Nombres después!:\n");
while (rs.next()) {
    String id = rs.getString("actor_id");
    String name = rs.getString("first_name");
    System.out.println(id + ": " + name);
}
System.out.println();
```

Actualizamos un registro: INSERT ROW.

Actualizar BD a través de ResultSet (II)

Ver ejemplo – EJ2_23

Actualizar BD a través de ResultSet (III)

```
Statement st = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
ResultSet rs = st.executeQuery("SELECT * FROM rental where rental_id >= 40 and rental_id <= 50;");
//para situarnos bien, cogemos las rentals con ids entre 40 y 50
// imprimo los nombres

System.out.println("Rentals antes!:\n");
while (rs.next()) {
    String id = rs.getString("rental_id");
    String customer = rs.getString("customer_id");
    System.out.println(id + ": " + customer);
}
System.out.println();

rs.absolute(4); // nos movemos a la fila que queremos borrar
rs.deleteRow();

rs.beforeFirst(); // movemos el cursor al principio, justo antes del primer elemento

System.out.println("Rentals después!:\n");
while (rs.next()) {
    String id = rs.getString("rental_id");
    String customer = rs.getString("customer_id");
    System.out.println(id + ": " + customer);
}
System.out.println();
```

Borramos un registro: DELETE ROW.

Actualizar BD a través de ResultSet (III)

Ver ejemplo – EJ2_24

Procedimientos almacenados (I)

```
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 oscar_winners(actor_id, film_id, type_oscar) VALUES (act_id, film_id, typePrize);
END IF;

COMMIT;

END
```

Procedimientos almacenados (I)

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

String serverAddress = "localhost:3306";
String db = "sakila";
String user = "bd";
String pass = "bdupm";
String url = "jdbc:mysql://" + serverAddress + "/" + db + "?noAccessToProcedureBodies=true";
conn = DriverManager.getConnection(url, user, pass);
conn.setAutoCommit(true);
System.out.println("Conectado a la base de datos!");
```

Dos formas de solucionar el acceso a procedimientos almacenados:

http://stackoverflow.com/questions/986628/cant-execute-a-mysql-stored-procedure-from-java

Procedimientos almacenados (I)

Ver ejemplo – EJ2_25

ArrayList para guardar datos (I)

```
// Declaración de un ArrayList de "String".
//Puede ser de cualquier otro Elemento u Objeto (float, Boolean, Object, ...)
ArrayList<String> nombreArrayList = new ArrayList<String>();

nombreArrayList.add("Elemento"); // Añade el elemento al ArrayList
nombreArrayList.add(n, "Elemento 2"); // Añade el elemento al ArrayList en la posición 'n'
nombreArrayList.size(); // Devuelve el numero de elementos del ArrayList

nombreArrayList.get(2); // Devuelve el elemento que esta en la posición '2' del ArrayList
nombreArrayList.contains("Elemento"); // Comprueba se existe del elemento ('Elemento')
                                      que se le pasa como parametro
nombreArrayList.indexOf("Elemento"); // Devuelve la posición de la primera ocurrencia
                                     ('Elemento') en el ArrayList

nombreArrayList.remove(5); // Borra el elemento de la posición '5' del ArrayList
nombreArrayList.remove("Elemento"); // Borra la primera ocurrencia del 'Elemento'
                                    que se le pasa como parametro.
nombreArrayList.clear(); //Borra todos los elementos de ArrayList

ArrayList arrayListCopia = (ArrayList) nombreArrayList.clone(); // Copiar un ArrayList
```

ArrayList para guardar datos (I)

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

String serverAddress = "localhost:3306";
String db = "sakila";
String user = "bd";
String pass = "bdupm";
String url = "jdbc:mysql://" + serverAddress + "/" + db + "?noAccessToProcedureBodies=true";
conn = DriverManager.getConnection(url, user, pass);
conn.setAutoCommit(true);
System.out.println("Conectado a la base de datos!");
```

Obtenemos los datos y los guardamos en un ArrayList de String y los imprimimos.

ArrayList para guardar datos (I)

Ver ejemplo – EJ2_26

ArrayList para guardar datos (II)

```
public EJ2_27() {
    try {
        ArrayList<Actor> nombresActores = getNombresActores();
        for (int i = 0; i < nombresActores.size(); i++) {
            Actor act = nombresActores.get(i);
            System.out.println(act.toString());
        }
    } catch (Exception e) {
        System.err.println("Error al conectar a la BD: " + e.getMessage());
    }
}

class Actor {
    private int id;
    private String firstName;
    private String lastName;
    private Date lastUpdate;

    public Actor(int id, String fn, String ln, Date lu) {
        this.id = id;
        this.firstName = fn;
        this.lastName = ln;
        this.lastUpdate = lu;
    }

    public String toString() {
        String ret = "\n";
        ret += "ID: " + id + "\n";
        ret += "First name: " + firstName + "\n";
        ret += "Last name: " + lastName + "\n";
        ret += "Last update: " + lastUpdate.toString() + "\n";
        ret += "\n";
        return ret;
    }
}
```

ArrayList para guardar datos (II)

```
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery("SELECT * FROM actor");
System.out.println("Query ejecutada!");

ArrayList<Actor> ret = new ArrayList<Actor>();
while (rs.next()) {
    int id = rs.getInt("actor_id");
    String firstName = rs.getString("first_name");
    String lastName = rs.getString(3); // tercera columna. Empiezan en                    // 1, no en 0
    Date lastUpdate = rs.getDate("last_update");
    ret.add(new Actor(id, firstName, lastName, lastUpdate));
}
rs.close();
st.close();
conn.close();
```

ArrayList para guardar datos (II)

Ver ejemplo – EJ2_27

Escritura en fichero como CSV

```
class ActorCSV {
    private int id;
    private String firstName;
    private String lastName;
    private Date lastUpdate;

    public ActorCSV(int id, String fn, String ln, Date lu) {
        this.id = id;
        this.firstName = fn;
        this.lastName = ln;
        this.lastUpdate = lu;
    }

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

Añadimos la opción de CSV.

Escritura en fichero como CSV

```
public EJ2_28() {
    try {
        ArrayList<ActorCSV> nombresActores = getNombresActores();
        BufferedWriter bW = new BufferedWriter(new FileWriter("actores.csv"));
        for (int i = 0; i < nombresActores.size(); i++) {
            ActorCSV act = nombresActores.get(i);
            bW.write(act.toCSV());
            bW.newLine();
        }
        bW.close();
    } catch (Exception e) {
        System.err.println("Error al conectar a la BD: " + e.getMessage());
    }
}
```

Escribimos los resultados en un fichero.

Escritura en fichero como CSV

Ver ejemplo – EJ2_28