

¿Qué ofrece Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

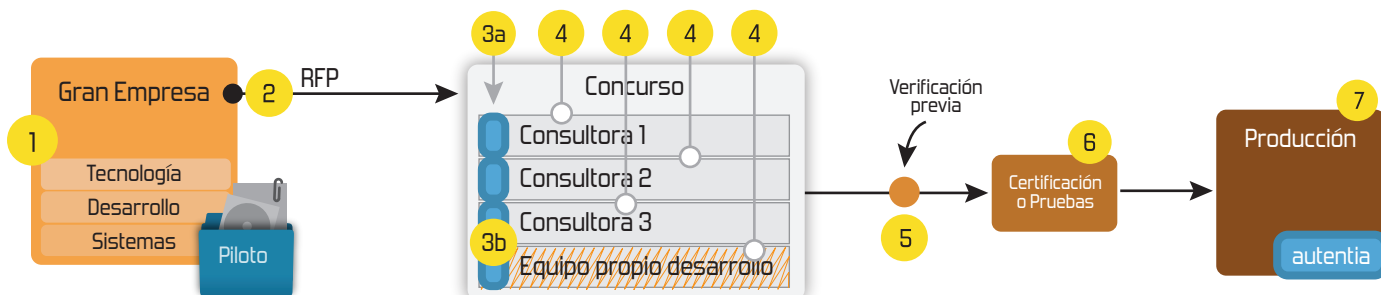
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces,
HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

Control de autenticación y
acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)



E-mail:

Contraseña:

Deseo registrarme
He olvidado mis datos de acceso

[Inicio](#) [Quiénes somos](#) [Tutoriales](#) [Formación](#) [Comparador de salarios](#) [Nuestro libro](#) [Charlas](#) [Más](#)



Estás en: [Inicio](#) [Tutoriales](#) Tutorial básico de bases de datos en Java mediante JDBC

DESARROLLADO POR:
 Francisco Ferri Pérez

Consultor tecnológico de desarrollo de proyectos informáticos.

Desarrollador de proyectos informáticos, Microsoft Certified IT Professional - Enterprise Administrator

Puedes encontrarme en [Autentia](#): Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

Anuncios Google

[Java](#)

[Test Java Code](#)

[Online Java Class](#)

[Java SMS API](#)

Fecha de publicación del tutorial: 2009-02-26



Share |

[Regístrate para votar](#)

Tutorial básico de bases de datos en Java mediante JDBC.

Previo y recomendado

<http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=>

Este tutorial continua a [Introducción a bases de datos SQL](#).

Se recomienda leer posteriormente [Buscar tutoriales de Hibernate](#).

Otra alternativa recomendada es [Introducción a bases de datos NoSQL](#).

Driver JDBC - Accediendo a la base de datos mediante Java Database Connectivity

Cada propietario de base de datos implementa un driver JDBC que podemos utilizar en nuestras aplicaciones java. Normalmente habrá un driver por versión y tipo de base de datos, puesto que no siempre se cumple la compatibilidad hacia versiones anteriores.

El driver JDBC es el más básico de que dispone Java para acceder a la base de datos, por lo tanto utilizarlo implica:

- **Manejar manualmente las conexiones**, nos estamos arriesgando a no cerrar correctamente las conexiones a la base de datos, y un sin fin de problemas derivados.

Nota: En el caso de aplicaciones web, el servidor de aplicaciones puede que implemente un pool de conexiones automático.

Por ejemplo en el caso de Apache Tomcat 6, colocando el driver JDBC en el directorio lib del propio Tomcat, no en nuestra aplicación, y configurando el fichero de contexto (context.xml) de nuestra aplicación (o también directamente la configuración global del servidor Tomcat - server.xml) podremos utilizar su pool de conexiones por defecto.

- **Cada vez que escribamos una query SQL vamos a tener que escribirla completamente**, puesto que cualquier sentencia Insert, Update o Select requerirá escribir un gran número de campos.

- **Manejar manualmente las relaciones**, teniendo en cuenta el orden de inserción o modificación de tablas y columnas. Esto puede no parecer un gran problema, pero se complica exponencialmente en modelos de datos complejos.

- **El SQL lo escribiremos para la versión y tipo de nuestra base de datos**, por lo tanto, este código estará completamente acoplado, sin poder ser utilizado sin ser rehecho con otra base de datos.

Realicemos operaciones CRUD (Create, Read, Update, y Delete) simples con JDBC

Una simple conexión a la base de datos

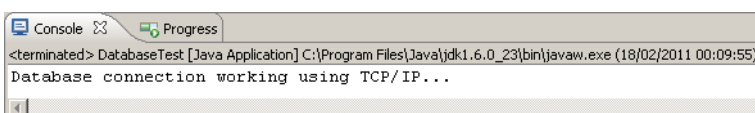
```
Connection conn = null;
```

```
final String url = "jdbc:mysql://localhost:3306/"; // Nuestra cadena de conexión
final String dbName = "bookshopdb"; // Nombre de nuestra base de datos
final String driver = "com.mysql.jdbc.Driver";
final String userName = "root";
final String password = "password";
```

```
try {
    Class.forName(driver).newInstance();
    conn = DriverManager.getConnection(url + dbName, userName, password);

    if (!conn.isClosed())
        System.out.println("Database connection working using TCP/IP...");
} catch (Exception e) {
    System.err.println("Exception: " + e.getMessage());
} finally {
    try {
        if (conn != null)
            conn.close();
    } catch (SQLException e) {
    }
}
```

El resultado de su ejecución



Catálogo de servicios Autentia

Últimas Noticias

- XIV Charla Autentia - ZK - Vídeos y Material
- Hablando de coaching ágil, milagro nocturno y pruebas de vida
- XIII Charla Autentia - AOS y TDD - Vídeos y Material
- Las metodologías ágiles como el catalizador del cambio
- XIV Charla Autentia - ZK

Histórico de NOTICIAS

Últimos Tutoriales

- Introducción a bases de datos SQL en Java.
- Introducción a bases de datos NoSQL (Not Only SQL)
- Informes dinámicos con DynamicJasper
- Banners animados: Cómo realizar animaciones en CSS3
- Pruebas de integración del envío de Email con el soporte de Spring.

Últimos Tutoriales del Autor

- Introducción a bases de datos SQL en Java.
- Introducción a bases de datos NoSQL (Not Only SQL)
- ZK - Añadir versiones de ZK al ZK Studio en Eclipse y cambiarle la versión de ZK a un proyecto.
- ZK - Instalar Studio en Eclipse
- ZK - ¿Cómo crear tu primer proyecto con ZK?

Síguenos a través de:



Últimas ofertas de empleo

- 2010-10-11 Comercial - Ventas - SEVILLA.
- 2010-08-30 Otras - Electricidad - BARCELONA.
- 2010-08-24

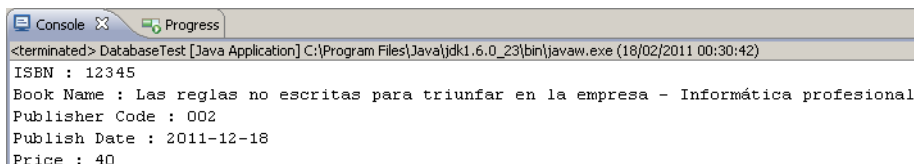
Un simple consulta

```
PreparedStatement stmt = conn.prepareStatement("SELECT * FROM BOOK WHERE ISBN = ?");
stmt.setString(1, "12345");

ResultSet rs = stmt.executeQuery();
while (rs.next()) {
    System.out.println("ISBN : " + rs.getString("ISBN"));
    System.out.println("Book Name : " + rs.getString("BOOK_NAME"));
    System.out.println("Publisher Code : " + rs.getString("PUBLISHER_CODE"));
    System.out.println("Publish Date : " + rs.getDate("PUBLISH_DATE"));
    System.out.println("Price : " + rs.getInt("PRICE"));
    System.out.println();
}

rs.close();
stmt.close();
```

El resultado de su ejecución



```
<terminated> DatabaseTest [Java Application] C:\Program Files\Java\jdk1.6.0_23\bin\javaw.exe (18/02/2011 00:30:42)
ISBN : 12345
Book Name : Las reglas no escritas para triunfar en la empresa - Informática profesional
Publisher Code : 002
Publish Date : 2011-12-18
Price : 40
```

Una simple actualización de un registro

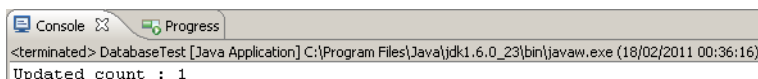
```
PreparedStatement stmt = conn.prepareStatement("UPDATE BOOK SET BOOK_NAME = ? WHERE ISBN = ?");
stmt.setString(1, "ZK Developer's Guide");
stmt.setString(2, "67890");

int count = stmt.executeUpdate();

System.out.println("Updated count : " + count);

stmt.close();
```

El resultado de su ejecución



```
<terminated> DatabaseTest [Java Application] C:\Program Files\Java\jdk1.6.0_23\bin\javaw.exe (18/02/2011 00:36:16)
Updated count : 1
```

Una simple inserción/persistencia de un registro

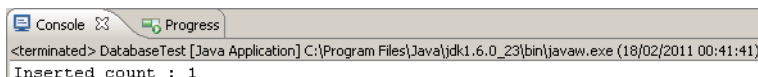
```
PreparedStatement stmt = conn.prepareStatement("INSERT INTO PUBLISHER (CODE, PUBLISHER_NAME, ADDRESS) VALUES (?, ?, ?)");
stmt.setString(1, "1111");
stmt.setString(2, "EDITORIAL AUTENTIABOOKS");
stmt.setString(3, "C/ Informática nº 196, Madrid");

int count = stmt.executeUpdate();

System.out.println("Inserted count : " + count);

stmt.close();
```

El resultado de su ejecución



```
<terminated> DatabaseTest [Java Application] C:\Program Files\Java\jdk1.6.0_23\bin\javaw.exe (18/02/2011 00:41:41)
Inserted count : 1
```

Un simple borrado

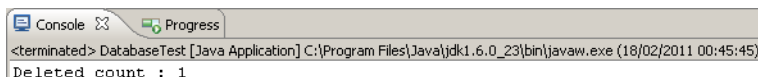
```
PreparedStatement stmt = conn.prepareStatement("DELETE FROM BOOK WHERE ISBN = ?");
stmt.setString(1, "12345");

int count = stmt.executeUpdate();

System.out.println("Deleted count : " + count);

stmt.close();
```

El resultado de su ejecución



```
<terminated> DatabaseTest [Java Application] C:\Program Files\Java\jdk1.6.0_23\bin\javaw.exe (18/02/2011 00:45:45)
Deleted count : 1
```

POJO's (Plain Old Java Objects) mediante JDBC

Para crear el modelo de objetos de dominio utilizamos JavaBeans, que son en realidad Plain Old Java Objects (POJO's), los cuales son clases que deben tener un constructor sin ningún tipo de argumento, y normalmente métodos públicos Getters y Setters para mapear todos sus atributos privados.

Se diferencian de los EJB's porque no implementan interfaces (en el caso de los EJB's javax.ejb). Veamos dos ejemplos.

```

package com.autentia.pojos;

public class Publisher {

    private String code;
    private String name;
    private String address;

    // Podemos omitirlo puesto que Java crea
    // este constructor por defecto
    public Publisher() {}

    // GETTERS Y SETTERS
    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }
}

public class Book {

    private String isbn;
    private String name;
    private Publisher publisher;
    private Date publishDate;
    private int price;
    private List<Chapter> chapters;

    // GETTERS Y SETTERS

```

```

package com.autentia.pojos;

public class Chapter {

    private int index;
    private String title;
    private int numOfPages;

    // GETTERS Y SETTERS
    public int getIndex() {
        return index;
    }

    public void setIndex(int index) {
        this.index = index;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public int getNumOfPages() {
        return numOfPages;
    }

    public void setNumOfPages(int numOfPages) {
        this.numOfPages = numOfPages;
    }
}

```

Utilizamos una clave ajena para referenciar PUBLISHER con la tabla BOOK, y además hay una relación many-to-one entre CHAPTERS y BOOKS.

Para modelar esto en los POJO's definimos listas de atributos, en vez de un simple atributo. Fíjate en el ejemplo del POJO de Book, que tiene una lista para los chapters.

Para manejar esta incompatibilidad entre modelo-relacional y objetos necesitamos un proceso que convierta de modelo relacional a objetos POJO's cuando recibimos la información.

Al código que realiza esta conversión por nosotros lo llamamos **ORM** (Object/Relational Mapping).

¿Entonces ya puedo hacerlo todo?

Como podemos deducir, a estas alturas del tutorial, es evidente que hay diferencias entre el modelo de objetos y el modelo relacional.

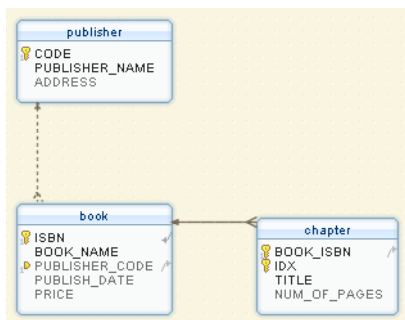
El modelo de objetos está basado en el análisis del negocio, y en consecuencia define el modelo del dominio.

El modelo relacional está basado en cómo está organizada la información, en columnas y filas.

Los frameworks ORM, como por ejemplo Hibernate, nos dan estrategias para abstraernos de las asociaciones, herencia y polimorfismos que no casan entre los modelos de objetos y relacionales.

ORM manual

Imaginemos que queremos mostrar los datos de las siguientes tablas de nuestro ejemplo si nun ORM, directamente con JDBC:



Cada una con sus correspondientes registros:

Tabla Books

book				
ISBN	BOOK_NAME	PUBLISHER_CODE	PUBLISH_DATE	PRICE
12345	Las reglas no escritas para triunfar en la empresa - Informática profesional	002	2011-12-18	40
67890	Si funciona, cámbialo	001	2011-12-18	30

Tabla Publisher

CODE	PUBLISHER_NAME	ADDRESS
001	AUTENTIA BOOKS	C\ Informática, Madrid
002	STARBOOK	C\ Servidores, Valencia

Tabla Chapters

BOOK_ISBN	IDX	TITLE	NUM_OF_PAGES
12345	1	Cap1	100

El método que tendríamos que crear sería tremendamente largo:

```
String isbn = "12345";

PreparedStatement stmt = conn.prepareStatement(
    "SELECT * FROM BOOK, PUBLISHER WHERE BOOK.PUBLISHER_CODE = PUBLISHER.CODE AND BOOK.ISBN = ?");
stmt.setString(1, isbn);
ResultSet rs = stmt.executeQuery();
Book book = new Book();
if (rs.next()) {
    book.setIsbn(rs.getString("ISBN"));
    book.setName(rs.getString("BOOK_NAME"));
    book.setPublishDate(rs.getDate("PUBLISH_DATE"));
    book.setPrice(rs.getInt("PRICE"));
    Publisher publisher = new Publisher();
    publisher.setCode(rs.getString("PUBLISHER_CODE"));
    publisher.setName(rs.getString("PUBLISHER_NAME"));
    publisher.setAddress(rs.getString("ADDRESS"));
    book.setPublisher(publisher);
}
rs.close();
stmt.close();

List<Chapter> chapters = new ArrayList<Chapter>();
stmt = conn.prepareStatement("SELECT * FROM CHAPTER WHERE BOOK_ISBN = ?");
stmt.setString(1, isbn);
rs = stmt.executeQuery();
while (rs.next()) {
    Chapter chapter = new Chapter();
    chapter.setIndex(rs.getInt("IDX"));
    chapter.setTitle(rs.getString("TITLE"));
    chapter.setNumOfPages(rs.getInt("NUM_OF_PAGES"));
    chapters.add(chapter);
}
book.setChapters(chapters);
rs.close();
stmt.close();
```

El resultado de la consulta es un objeto POJO "book" que contendría los siguientes datos:

```
BOOK[
  12345
  Las reglas no escritas para triunfar en la empresa
  Informática profesional
  PUBLISHER[002 - STARBOOK - C\ Servidores, Valencia]
  2011-12-18
  40
  CHAPTER[1 - Cap1 - 100]]
```

Del mismo modo, persistir Objetos POJO mediante JDBC, pero para que os hagáis una idea, tendríamos que crear una transacción, y dentro de esta recorrer las propiedades del objeto Book, para completar la sentencia INSERT, pero a su vez cuando una propiedad del objeto Book fuese un listado habría que repetir el proceso con bucles sobre las tablas/entidades correspondientes.

Conclusión

Utilizar JDBC implica construir y ejecutar repetidamente sentencias SELECT, INSERT, UPDATE y DELETE.

Por lo tanto:

Creamos mucho código que además estará muy acoplado a la base de datos que estemos usando.

Tenemos que iterar manualmente sobre las propiedades de objetos como ResultSet cada vez que consultemos algo en la base de datos.

A su vez es muy costoso crear PreparedStatements en cada caso por el mismo motivo de las iteraciones, pero en este caso sería sobre los POJO's para Inserts, Updates y Deletes.

Tendríamos que gestionar manualmente el orden de las inserciones, actualizaciones y borrados para que no hubiese problemas con la integridad referencial.

Espero que os sea de utilidad a los que empezáis

Me podéis encontrar en fferri@autentia.com y en twitter [@franciscoferri](https://twitter.com/franciscoferri)

¡Anímate y coméntanos lo que pienses sobre este **TUTORIAL**!

Puedes opinar o comentar cualquier sugerencia que quieras comunicarnos sobre este tutorial; con tu ayuda, podemos ofrecerte un mejor servicio.

Enviar comentario

(Sólo para usuarios registrados)

» **Regístrate** y accede a esta y otras ventajas «

COMENTARIOS



Esta obra está licenciada bajo [licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5](#)

Copyright 2003-2011 © All Rights Reserved | [Texto legal y condiciones de uso](#) | [Banners](#) | [Powered by Autentia](#) | [Contacto](#)

