Caso

*En **SI Andalucía** han hecho un considerable esfuerzo por formar a **Víctor** y están deseosos de empezar a rentabilizarlo poniéndolo a colaborar con el grupo de desarrollo en algunas de las aplicaciones que están desarrollando para algunos clientes. Por su parte **Víctor** está agradecido de la formación recibida, que sabe que ha sido muy buena, y deseoso de empezar a trabajar en aplicaciones reales, para demostrar que la confianza que depositaron en él mereció la pena.*
*Por otra parte, **José** tiene pendiente la realización de una pequeña aplicación de gestión para el Ayuntamiento, y está embarcado en varios proyectos que no le permiten comenzarla todavía. No es una aplicación muy complicada, ni corre mucha prisa su entrega, pero le pregunta a **María** y a **Carmen**, que han seguido más de cerca la formación de **Víctor**, si creen que está ya preparado para encargarse él sólo del desarrollo de esa aplicación a partir de las especificaciones...*
Ambas están de acuerdo en que podría desarrollarla sin problemas, pero con una limitación: Todavía no ha aprendido a conectar las aplicaciones Java con las Bases de Datos, tendría que desarrollarla usando ficheros.
***José** les dice que es preferible que los datos de la aplicación se guarden en una base de datos, porque eso da más flexibilidad al permitir que en el futuro otras aplicaciones puedan acceder a esos mismos datos, ya que no estarán ligados a los formatos de la aplicación que ahora desarrollen. Por otro lado, también se consigue que la aplicación sea independiente de los datos. Si en el futuro el Ayuntamiento decide cambiar su Sistema Gestor de Bases de Datos, la aplicación seguirá siendo válida, y sólo habrá que cambiar las dos líneas de código que registran el driver o controlador JDBC y que establecen la conexión con la nueva base de datos. El resto de la aplicación seguirá funcionando exactamente igual.*
***Víctor** dice que hace un rato que se ha perdido... Bases de datos, SGBD, driver JDBC... ¿De qué están hablando?*
*Evidentemente, **María** y **Carmen** saben que **José** tiene razón, y que van a tener que explicarle a **Víctor** algunas cosas al respecto antes de que pueda encargarse del desarrollo de la aplicación de forma que pueda implementarla usando Bases de Datos, así que **Carmen** empieza por el principio, explicándole lo que es una Base de Datos y porqué son tan útiles...*
*Y **Víctor** la sigue, porque ya sabe que como programador, no se acostará ningún día sin haber aprendido algo nuevo...*

Introducción

Ya has estudiado que un programa informático es una secuencia de instrucciones que operan sobre unos datos de entrada para producir como resultado unos datos de salida. También has visto en unidades anteriores como los lenguajes de programación proveen de diferentes estructuras de datos para manipular y tratar la información.

Habitualmente las aplicaciones informáticas, sobre todo a nivel empresarial, manejan una gran cantidad de **datos**.

Piensa por ejemplo en una aplicación para gestionar la contabilidad de una empresa o la gestión de la información de cuentas de una entidad bancaria.

Esas grandes cantidades de datos necesitan ser almacenadas de forma permanente en algún dispositivo de almacenamiento masivo y deben ser gestionadas por programas especialmente diseñados para ello.

Java, como los demás lenguajes de programación, es una gran herramienta para manipular datos, pero en cambio no provee una manera simple y eficaz de almacenar y tratar grandes cantidades de datos. Por ello debe trabajar conjuntamente con otras aplicaciones que sí estén especializadas en esas tareas. Esas aplicaciones informáticas son los Sistemas Gestores de Bases de Datos (SGBD).

El estudio en profundidad de las bases de datos y los gestores de bases de datos es amplio y cae fuera del ámbito de este módulo profesional y será objeto de estudio en otros módulos profesionales del currículo del ciclo formativo que estás cursando, concretamente se da una introducción dentro del módulo

profesional de "**Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión**" y se aborda con bastante detalle y de forma muy práctica dentro del módulo de "**Desarrollo de Aplicaciones en Entornos de Cuarta Generación y con Herramientas Case**".

No obstante, es un tema importantísimo y sin el cual no sería posible la realización de ningún proyecto informático de envergadura. No obstante, no debes preocuparte si alguno de los conceptos no te quedan del todo claros, ya que esos otros módulos completarán sobradamente tu formación en Bases de Datos.

Considera esta unidad como una introducción al acceso a bases de datos desde Java, de forma que ya podamos, desde este módulo de introducción a la programación, afrontar el desarrollo de aplicaciones informáticas más o menos profesionales.

Comenzaremos introduciendo los conceptos básicos sobre bases de datos y gestores de bases de datos y después lo aplicaremos de forma práctica al lenguaje de programación Java.

Bases de datos y Sistemas de Gestión de Bases de Datos.

Posiblemente más de una vez hayas oído hablar de Bases Datos y de Sistemas de Gestión de Bases de Datos, o de Sistemas Gestores de Bases de Datos, que es otra forma de decir lo segundo. Pues vamos a darte una idea de lo que se está hablando:

- Una **base de datos** es un conjunto de datos que tienen relación entre sí y que están almacenados con una cierta estructura que permite su manipulación. Es decir, añadir nuevos datos, modificar los existentes, borrar datos o consultarlos.
- Un **sistema de gestión de bases de datos** (SGBD) es una aplicación informática que permite realizar las operaciones anteriores sobre una base de datos, de una forma rápida y sistemática. **No hay que confundir base de datos con gestor de base de datos.**

Normalmente una base de datos pretende modelar una situación del mundo real para permitir su tratamiento informático. Esto es, reproducir las informaciones del contexto que se pretende tratar, así como su estructura. A esto se le llama **modelo de base de datos**.

A lo largo del tiempo han sido propuestos diferentes modelos, los más importantes son:

- **Modelo jerárquico**: Utiliza un esquema de representación basado en estructuras de tipo árbol jerárquico. En la actualidad no se utiliza.
- **Modelo en red**: Es una mejora del anterior en el cual se establecen relaciones no necesariamente jerárquicas. Como el anterior hoy día está en desuso.
- **Modelo relacional**: Es el más utilizado en la actualidad, representa la información por medio del concepto matemático de relación.
- **Modelo orientado a objetos**: Es el más reciente y trata de representar la información por medio de objetos, lo que lo hace especialmente interesante en combinación con lenguajes de programación como Java.

PARA SABER MÁS
Puedes encontrar una definición más detallada de estos conceptos, así como enlaces interesantes a otros conceptos relacionados con Bases de Datos en el siguiente enlace:
Definición y Conceptos relacionados con Bases de Datos **[Versión en caché]**
Aquí encontrarás de forma esquematizada las principales características de los manejadores de bases de datos Sql Server, Oracle e Informix para que los conozcas y vayas familiarizándote con ellos.
Manejadores de Bases de Datos **[Versión en caché]**

Bases de datos relacionales

*Aunque **Carmen** se ha centrado en explicarle a **Víctor** el modelo de **bases de datos relacional**, por ser el más usado y extendido, también le ha contado que existen otros modelos que han sido muy usados en su momento, como el modelo jerárquico y el modelo en red, y también le ha contado que cada vez más toma fuerza un nuevo modelo orientado a objetos, que parece que en el futuro posiblemente sea el más extendido.*
***Víctor** no conoce todos esos modelos, pero se alegra de que se use el modelo relacional, porque le ha parecido bastante simple y fácil de entender, ya que a fin de cuentas, consiste en representar los datos mediante tablas, en las que cada columna es un campo o atributo, y cada fila representa a un registro de las entidades que se quieren representar.*
*Es como hacerse una agenda a lápiz y papel, también se representa en una tabla, salvando las distancias. En poco tiempo ha entendido los conceptos que **Carmen** le explica, y tiene claro que la madre del cordero está en definir bien las tablas que van a formar parte de la base de datos y la forma de relacionar unas con otras. En ese punto espera contar con la ayuda y la experiencia de **María**, que habitualmente se encarga de hacer esta parte del diseño, para cuando vaya a comenzar con su aplicación para el Ayuntamiento.*

Ya sabemos que el modelo de datos que más se utiliza en la actualidad es el modelo relacional. Vamos a ver con más detalle en qué consiste, puesto que nos será imprescindible para entender el resto de la unidad didáctica que estás estudiando.

Volvemos a insistir en que éste es un tema muy amplio que ocuparía por sí solo mucho más que una unidad didáctica, por lo tanto se pasarán por alto muchos detalles. No obstante te recomendamos la lectura de algunos enlaces interesantes para ampliar tus conocimientos sobre el tema.

Modelo relacional (I)

¿Qué características presenta este modelo para que sea el más usado y el más extendido por el momento?

El modelo relacional **utiliza el concepto matemático de relación**, **que está basado en la teoría de conjuntos, para representar la información y su estructura**.

- Gráficamente **podemos imaginar una relación como una tabla**.
- Dicha tabla está formada por **filas y columnas**.
- **Cada fila almacena información sobre un elemento** del contexto que se pretende modelizar.
- **Las columnas de la tabla, también llamadas atributos, representan los datos que se conocen de ese elemento**.

Por ejemplo, supongamos que queremos representar información sobre el conjunto de trabajadores de una empresa. **De cada trabajador nos interesan unos determinados datos o atributos**, como son:

- Nombre,
- Fecha de nacimiento,
- Centro de trabajo y
- Salario.

Éstas serán las columnas de la tabla, y cada fila representará a una persona distinta. El conjunto de las filas representará la información disponible sobre todas las personas. Cada columna tendrá un nombre que representará su contenido y será único.

Cada atributo tendrá un tipo de dato concreto, a esto se le llama dominio del atributo.

Es decir, el rango de valores permitido para los valores de cada columna. En el ejemplo anterior se tendrían:

- **Nombre**: Dominio - Caracteres (cadenas de caracteres).
- **Fecha de nacimiento**: Dominio - Fechas válidas.
- **Centro de trabajo**: Dominio - Números enteros del 0 al 99.

- **Salario**: Dominio - Números enteros positivos.

Modelo relacional (II)

Una base de datos relacional suele constar de varias tablas que están relacionadas entre sí. Siguiendo con el ejemplo anterior podríamos añadir otra tabla que representase la información sobre los distintos centros de trabajo de la empresa: Centro de trabajo, dirección, teléfono.

Hay que hacer notar que las tablas estarían relacionadas por medio del campo común centro de trabajo. Lo que quiere decir que **no podrían existir filas en la tabla de trabajadores con un centro de trabajo inexistente en la tabla de centros de trabajo**.

PARA SABER MÁS
Aquí encontrarás una breve introducción al modelo relacional que te servirá para ir familiarizándote con el mismo.
Introducción al Modelo Relacional **[Versión en caché]**
En este enlace podrás profundizar un poco en el modelo relacional.
El modelo Relacional **[Versión en caché]**

Lenguaje SQL

*Cuando **Carmen** le explica a **Víctor** que para operar con la base de datos se usa un lenguaje distinto de Java que se llama SQL, y que tiene su propia sintaxis, éste casi se muere del susto. ¡Todavía no controla totalmente Java y ahora tiene que comenzar a aprender otro lenguaje más! **Carmen** le tranquiliza.*
Es cierto que es otro lenguaje más, pero que se usa para unas operaciones muy concretas y con una sintaxis bastante sencilla. Además, como las operaciones se repiten con relativa frecuencia, es suficiente con que disponga de algunos programas de ejemplo para que fijándose en ellos y usándolos como una especie de "plantilla" pueda hacer cualquier operación necesaria sobre cualquier Base de Datos de cualquiera de sus aplicaciones.
***Víctor** desconfía un poco, pero sabe que puede contar con **Carmen**, y eso le tranquiliza. A fin de cuentas ella hace poco que estudió bases de datos y SQL en el instituto, y sin embargo a él le consta que se desenvuelve con soltura integrando las consultas en SQL dentro de sus aplicaciones Java que usan bases de datos, ya que en más de una ocasión ha visto como **Jesús** la felicitaba por su trabajo.*

Ya hemos visto cómo representa los datos y sus relaciones el modelo relacional, pero no debemos olvidar que la manipulación de estos datos se hará por medio de un **programa** llamado sistema **gestor de bases de datos relacionales** (SGBDR). Se necesita comunicar de alguna forma normalizada con el SGBDR para indicarle que acciones queremos que lleve a cabo sobre los datos almacenados.

¿Cómo podremos pedirle al SGBDR que nos proporcione en cada momento la información que nos interese de la Base de Datos?

Se utiliza el lenguaje SQL **(Structured Query Language) para interactuar con el SGBDR**.

SQL es un lenguaje no **procedimental** en el cual se le indica al SGBDR **qué** queremos obtener y **no cómo hacerlo**. Será una parte del SGBDR la que analice nuestra orden y la lleve a cabo.

El estudio exhaustivo de SQL nos llevaría mucho más que una unidad didáctica y será objeto de estudio en otros módulos de este ciclo formativo. Pero como resulta imprescindible para poder continuar haremos una pequeña introducción de él.

El lenguaje SQL está compuesto por comandos o instrucciones, cláusulas, operadores y funciones.

Estos elementos se combinan para manipular las bases de datos.

Los comandos SQL se pueden dividir en dos grandes grupos:

- **Los que se utilizan para definir las estructuras de datos**, llamados comandos **DDL** (Data Definition Language).
- **Los que se utilizan para manipular las estructuras** llamados **DML** (Data Manipulation Language).

En la siguiente tabla puedes encontrar algunos de los comandos SQL más importantes, con indicación de sin son DDL o DML. También se muestra la sintaxis de cada comando.

Accede a la tabla de comandos SQL

A continuación tienes algunos ejemplos del uso de los comandos de la tabla anterior:

- Listar los nombres de los empleados que trabajan en el centro 1345.
  ```
  SELECT Nombre FROM TRABAJADORES WHERE Centro de trabajo = 1345
  ```
- Modificar el salario de 'Alfonso Bonillo Sierra' aumentándolo en 200 €.
  ```
  UPDATE TRABAJADORES SET Salario = Salario + 200 WHERE Nombre = 'Alfonso Bonillo Sierra'
  ```
- Insertar en la tabla TRABAJADORES el empleado 'José Miguel González Pérez', nacido el 08/02/1975, que trabaja en el centro de trabajo número 1345 con un sueldo de 1750 €.
  ```
  INSERT INTO TRABAJADORES VALUES ('José Miguel González Pérez', '08/02/1975', 1345, 1750)
  ```

PARA SABER MÁS
En este enlace encontrarás de una manera breve pero interesante la historia del SQL
Historia del SQL **[Versión en caché]**
Aquí encontrarás en forma esquematizada lo visto en este punto profundizando un poco más en las sentencias DDL y DML.
El lenguaje SQL. Sentencias DDL y DML **[Versión en caché]**

MySQL

__Carmen__ le cuenta a __María__ que tras un par de días, __Víctor__ ha asimilado bien los conceptos necesarios de Bases de Datos y SQL, mostrando cierta soltura a la hora de hacer operaciones de consulta, modificación, borrado e inserción sobre las tablas de la base de datos. Ha venido trabajando con el SGBD que tiene instalado __María__ para hacer pruebas en su ordenador, pero __Carmen__ piensa que estaría bien que aprendiera a instalar y configurar tanto el servidor de BD como todo el software básico que maneja. A __María__ le parece buena idea, ya que a fin de cuentas, cuando le instale las aplicaciones al cliente, tendrá que configurarle también el SGBD y la propia Base de Datos.
Le han preguntado a José qué SGBD concreto piensa que se usará para la aplicación del Ayuntamiento. __José__ dice que van a usar MySQL por ser libre, reduciendo los costes globales de implantación de la aplicación y por ser extremadamente fácil de instalar y configurar. Al mismo tiempo es un sistema potente, que cubre sobradamente todas las necesidades que puede tener para gestionar bases de datos de tamaño medio como la que previsiblemente van a usar allí, e incluso bases de datos de gran tamaño. Por último, otra de sus ventajas es la existencia de abundante documentación en Internet tanto en inglés como en castellano, al ser un SGBD ampliamente extendido. __Carmen__ toma nota, ya que cuando le diga a __Víctor__ que tiene que practicar instalando MySQL éste seguramente le preguntará porqué, y le tendrá que contar todos esos motivos....

Una vez que conoces qué es una base de datos relacional, qué elementos utiliza y cómo interactuar con ellos por medio del lenguaje SQL, es el momento de instalar un SGBDR en tu ordenador y empezar a practicar todo lo aprendido hasta ahora. Mejor verlo todo probando y practicando, ¿verdad?

En el mercado existen multitud de SGBDR comerciales y de código fuente libre. Para nuestras prácticas utilizaremos uno muy extendido en la actualidad y que además puede descargarse y usarse libremente, nos referimos a MySQL.

MySQL es un producto que ha evolucionado mucho desde sus primeras versiones y que se utiliza ampliamente en la actualidad. Es un SGBD que **sigue el modelo relacional**, es decir estructura los datos en forma de tablas, y para realizar operaciones sobre los datos **utiliza el lenguaje SQL**. Por eso es necesario que conozcas y practiques los comandos que se han introducido en el apartado anterior.

Lo mejor es que pasemos directamente a descargar, instalar y empezar a usar MySQL para que comprendas bien todos los conceptos que hemos explicado sobre bases de datos, SGBDR, modelo relacional y SQL. Te facilitamos a continuación unas animaciones que te ayudarán en tus primeros pasos en el uso de MySQL.

- Lo primero que debes hacer es descargar la última versión de MySQL de su página web. También es muy interesante que descargues dos aplicaciones que aunque no son imprescindibles si serán de gran ayuda, mysql-administrator y mysql-querytool.

ZONA DE DESCARGA
La siguiente web es la página oficial de MySQL, de la que podrás descargarte todo el software necesario para esta unidad (MySQL, Mysql-Administrator, MySQL-QueryTool y el Conector JDBC para MySQL, llamado Connector-J)
Web oficial de MySQL

Encontrarás en las siguientes animaciones explicaciones que te guiarán por el proceso de descarga e instalación de todo ese software que te va a resultar necesario.

- Debes instalar en tu equipo las aplicaciones descargadas. Primero el SGBDR MySQL y después las utilidades de administración y manipulación de tablas.

Animaciones de instalación:

- Deberías familiarizarte con las aplicaciones instaladas, para ello lo mejor es crear una base de datos (schema) y en ella una tabla, después **añadir (INSERT), modificar (UPDATE) y borrar (DELETE) filas (rows)** y probar distintos comando de **selección (SELECT).**

Te facilitamos una animación con un ejemplo propuesto que después utilizaremos en esta misma unidad didáctica.

- Por último es interesante que te fijes en que MySQL es un SGBDR multiusuario y que permite que varios usuarios accedan a las bases de datos con diferentes niveles de privilegios. A continuación tienes una animación que explica como crear un usuario y darle permisos para trabajar con la base de datos del ejemplo anterior.

Animación de creación de usuario y asignación de privilegios:

PARA SABER MÁS
Aquí tienes un enlace a la Zona para Desarrolladores del sitio de MySQL. Encontrarás todo lo relacionado con este SGBD, eso sí, en inglés.
Zona de Desarrollo de MySQL **[Versión en caché]**
En estos enlaces encontrarás manuales de MySQL en castellano, escoge el que prefieras. Hay muchos más en Internet.
Manual de MySql en castellano (I) **[Versión en caché]**
Manual de MySql en castellano (II) **[Versión en caché]**

Acceso a bases de datos desde Java

Llegamos ahora a la parte más interesante de esta unidad. Vamos a conectar con bases de datos utilizando Java. No te preocupes, para usar bases de datos con Java o conectar nuestras aplicaciones con MySQL no hay que "aprender japonés", es bastante sencillo.

Necesitamos previamente conocer algunos detalles sobre cómo enlazar nuestros programas escritos en Java con sistemas gestores de bases de datos de otros fabricantes. Para nuestras prácticas utilizaremos MySQL, pero se actuaría de la misma forma con otros SGBD.

Nuestro objetivo es escribir programas Java que utilicen los datos que están almacenados en un SGBD y para ello debemos conocer que es JDBC.

JDBC (I)

Víctor *estaba un poco asustado porque imaginaba lo complicado que debería ser escribir el código necesario para que una aplicación Java pudiera conectar a la base de datos y traducir sentencias Java a consultas SQL, y luego hacer conversiones de los datos leídos desde el formato de la base de datos a los tipos propios de Java, con un montón de conversiones, casting explícitos y cosas similares. Menos mal que se lo comentó a* ***María****, que disipó todas las dudas sin más que enseñarle un ejemplo, y mientras se lo enseñaba, tuvo lugar más o menos el siguiente diálogo:*
Víctor:*"Debe ser complicado esto de trabajar con Bases de Datos que no tienen nada que ver con Java, ¿verdad?".*
María*: "Al contrario,* ***Víctor****, es tremendamente sencillo porque Java ha pensado las cosas para que sea fácil y flexible. Usando la tecnología JDBC sólo necesitas dos simples sentencias Java para tener tu aplicación conectada a la base de datos. Mira las que son en el ejemplo, para MySQL, y que luego* ***Carmen*** *te explique los detalles. ".*

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

miConexion=DriverManager.getConnection("jdbc:mysql://localhost:3306/biblioteca",

"root","123456");
```

Víctor*: "No parece difícil, ni nada rebuscado. Pero una vez conectado, hacer las consultas en SQL desde Java debe ser tremendamente complicado".*
María*: "Ni lo pienses. Sólo hay que pasarle como parámetro la consulta SQL escrita entre comillas, como un literal String a un método apropiado de Java, y sólo tienes dos métodos para ello,* ***`executeQuery()`*** *para pasarle sentencias SQL que hacen consultas y* ***`executeUpdate()`*** *para sentencias SQL que realicen modificaciones sobre la base de datos como inserciones, borrados, etc."*
Víctor*: "Pero será complicado adaptar los tipos de datos de la base de datos a los tipos de datos que usa Java".*
María*: "Menos todavía. Java proporciona una amplia variedad de tipos de datos de forma que hay tipos compatibles con cualquier tipo de cualquier gestor de bases de datos, ya que éstos suelen ser más o menos estándares también".*
Víctor*: "Me está resultando interesante eso de JDBC. ¿De verdad no hay nada más que las cuatro cosas que me has contado?".*
María*: "Bueno, algo más hay, como instalar el conector, que es tan fácil como descargarlo y copiarlo en la carpeta apropiada, y alguna cosilla más, pero nada complicado. Ahora bien, tendrás que estudiarlo tú con la ayuda de* ***Carmen*** *y con los ejemplos que os he preparado".*

¿Qué es JDBC y qué relación tiene con Java y con las bases de datos?

JDBC son las siglas de Java DataBase Connectivity.
Es una especificación de un conjunto de clases y métodos de operación que permiten a cualquier programa Java acceder a sistemas de bases de datos de forma homogénea.

Para ello, entre el programa Java y el SGBD se interpone un elemento llamado **controlador (driver) JDBC**. Este controlador es el que **implementa la funcionalidad de todas las clases de acceso a datos y proporciona la comunicación entre el API JDBC y el SGBD**. La misión del controlador será traducir comandos del API JDBC al protocolo nativo del SGBD.

Con JDBC y los controladores JDBC **se independiza a las aplicaciones Java del SGBD** concreto que se esté utilizando, de forma que Java se "entiende" con el controlador JDBC y nunca con el SGBD directamente. Esto tiene muchas ventajas, en concreto **se podría cambiar de SGBD y de controlador JDBC sin tener que cambiar el código del programa Java y este seguiría funcionando**. Si queremos cambiar el SGBD que empleamos lo único que deberemos hacer es reemplazar el antiguo controlador por el nuevo, y seremos capaces de conectarnos al nuevo SGBD.

PARA SABER MÁS
Cada sistema Gestor de Bases de datos suele proporcionar en su web el conector JDBC para permitir el acceso a aplicaciones Java. En el siguiente enlace puedes visitar la Base de datos que la empresa Sun, desarrolladora de Java, nos proporciona con todas las APIS que soportan los drivers JDBC para multitud de SGBD diferentes. Puedes consultar la lista detallada.
Base de Datos de APIs que soportan JDBC **[Versión en caché]**

JDBC (II)

Otra cosa que hay que tener en cuenta, es que los controladores JDBC aceptan SQL como lenguaje de consulta y manipulación de datos, por eso es tan importante que te familiarices con SQL y sus comandos.

El API JDBC soporta dos modelos de acceso a datos (modelo de dos capas y modelo de tres capas).

- **Modelo de dos capas:** En este caso la conexión entre la aplicación Java y el SGBD se hace de forma directa, lo que quiere decir que el controlador JDBC debe residir en el sistema local donde se ejecute la aplicación Java. En cambio el SGBD puede estar en otra máquina conectada en red. Ésta es la configuración más sencilla y la que usaremos en nuestras prácticas.
- **Modelo de tres capas:** En este modelo de acceso a las bases de datos, las instrucciones SQL son enviadas a un servidor intermedio entre la máquina que ejecuta la aplicación Java y la máquina que ejecuta el SGBD. Es un modelo más complejo de implementar que el de 2 capas, pero tiene la ventaja de que el nivel intermedio mantiene en todo momento el control del tipo de operaciones que se realizan con la base de datos, y además, los controladores JDBC no tienen que residir en la máquina cliente, lo cual libera al usuario de la instalación de cualquier tipo de controlador.

PARA SABER MÁS
Puedes encontrar más información sobre estos dos modelos en el siguiente enlace.
Modelos de acceso a datos del API JDBC **[Versión en caché]**

Tipos de controladores JDBC

Parece que los controladores JDBC son una potente herramienta para conseguir que nuestras aplicaciones accedan a los datos sin que tengamos que preocuparnos en absoluto de los detalles de

almacenamiento del SGBD concreto que se utilice. Eso sin duda hace que nuestra aplicación sea más fácilmente adaptable, al ser independiente de los datos. ¿Pero existe un único tipo de conector JDBC? Parece que no...

Existen cuatro categorías de controladores JDBC, cada una de ellas se diferencia de las demás por su forma de operar.

- **Controlador tipo I, puente JDBC-ODBC**
  Se trata de un tipo de controladores diseñado para aprovechar las conexiones existentes que utilicen tecnología **ODBC**.
  ODBC **(Open Database Connectivity**) es una tecnología de Microsoft que funciona con una filosofía similar a la de JDBC, de manera que permite acceder desde las aplicaciones a las bases de datos gestionadas por un SGBD cualquiera, siempre que se disponga de un controlador ODBC adecuado. La desventaja es que es una tecnología muy ligada a los sistemas operativos de Microsoft.
  Los controladores JDBC de tipo I permiten acceder desde Java a ODBC y desde ahí al SGBD.
  - **Como ventaja** se tiene el **disponer de forma inmediata de las conexiones establecidas ya para ODBC**
  - **Como desventajas se tiene su lentitud al interponer un nuevo eslabón en la cadena entre Java y el SGBD y además limitan su uso a sistemas Microsoft,** por lo que hay que descartar su uso salvo que no se tenga otra solución. Por ejemplo servirían para acceder a una base de datos Microsoft Access desde un programa Java.
- **Controlador tipo II, API Nativo**
  Se basa en una librería escrita en código nativo para acceder a la base de datos**.**
  El controlador traduce las llamadas JDBC a llamadas al código de la librería nativa, ésta tiene que ser proporcionada por los desarrolladores del SGBD.
  - **Ventaja:Este tipo de controladores más eficienteque los de tipo I**, ya que hay menos capas entre el programa Java y el SGBD.
  - **Desventaja:** Sin embargo **sigue teniendo el problema de la portabilidad al depender del código nativo.**
- **Controlador tipo III, JDBC-Net**
  En este caso el controlador se comunica con un servidor intermedio que se encuentra entre el cliente y el SGBD.
  El servidor intermedio se encarga de traducir las llamadas al API JDBC al protocolo específico de la base de datos.
  - **Ventajas:**
    - **No se requiere ningún tipo de código nativo en el cliente**, por lo que **la portabilidad** de la aplicación está garantizada.
    - El controlador es tecnología **100% Java**.
    - Además si el servidor intermedio **es capaz de traducir las llamadas JDBC a protocolos específicos de diferentes SGBD**, **podremos emplear un mismo controlador para comunicarnos con diferentes SGBD**. Esto lo convierte en el controlador más versátil de todos.
  - **Desventajas:**
    - No obstante se trata de un **controlador complejo**,
    - que **no es común** encontrar en sistemas simples,
    - estando reservado a **arquitecturas sofisticadas donde incluso se utilicen varios SGBD** al mismo tiempo.
- **Controlador tipo IV, Protocolo nativo**
  Los controladores de tipo IV **traducen directamente las llamadas al API JDBC al protocolo nativo del SGBD.**
  - **Ventaja:Son los controladores que tienen mejor rendimiento y también emplea tecnología 100% Java.**
  - **Desventaja**: **Está más ligado al SGBD que empleemos que los controladores de tipo III** JDBC-Net, donde el uso del servidor intermedio nos daba una gran flexibilidad a la hora de cambiar de base de datos.

Obtención e instalación del controlador JDBC.

Debemos obtener el controlador JDBC adecuado al SGBD que vayamos a utilizar y además debemos fijarnos en el tipo. Siempre preferiremos los controladores de tipo III o IV. Normalmente los fabricantes de SGBD proveen de controladores JDBC que podemos descargar de Internet. También existen controladores desarrollados por terceras partes que pueden ser de código libre o comerciales.

Como para nuestras prácticas vamos a utilizar MySQL debemos hacernos con un controlador para este SGBD, y además tenemos que instalarlo en nuestro equipo. La obtención del controlador es fácil, sólo tenemos que visitar la página web de MySQL indicada en el apartado Zona de Descarga del apartado 3.3. y buscar el enlace de descarga del controlador JDBC.

Una vez descargado comprobaremos que **además del controlador tendremos una serie de archivos de documentación, programas Java de ejemplo y el código fuente del controlador.**

La instalación es muy sencilla: **basta con copiar el fichero del controlador JDBC en el directorio ext de nuestra instalación del JDK**. A partir de ese momento ya podemos empezar a utilizarlo. En los apartados siguientes veremos cómo podemos acceder a las bases de datos desde Java.

El API de Java para acceso a bases de datos java.sql

Ya hemos descargado e instalado el controlador JDBC. ¿Qué más tengo que hacer para poder conectarme a la base de datos y poder consultar y modificar su información desde mi aplicación?

El paquete `java.sql` es el API de Java para acceso a bases de datos. **Deberemos importarlo (también se dice "registrarlo") en todas las aplicaciones Java que desarrollemos para acceder a bases de datos**. En este paquete están todas las clases e interfaces necesarios.

Toda aplicación que acceda a una base de datos empleando el API JDBC debe realizar una serie de pasos:

1. **Registrar el controlador JDBC.**
2. **Establecer una conexión con la base de datos.**
3. **Ejecutar las instrucciones SQL.**
4. Si las instrucciones SQL devuelven datos **procesar los datos devueltos.**
5. Por último hay que **liberar los recursos de la conexión.**

En este apartado vamos a conocer cómo realizar cada uno de los pasos anteriormente descritos. Para ello nos serviremos de la base de datos MySQL de ejemplo que creamos en el apartado anterior dedicado a MySQL. Como recordarás la base de datos se llamaba biblioteca y constaba de una tabla llamada libro. En esta tabla se almacenaba información de libros con los siguientes atributos: título, autor, género y precio. Deberías disponer de esta base de datos funcionando y con algunos datos de prueba en la tabla libro para poder seguir los ejemplos que veremos a continuación.

PARA SABER MÁS
En este enlace tienes disponible una buena referencia de los paquetes básicos de la API de Java.
Bibliotecas de la API de Java **[Versión en caché]**
Aquí encontrarás una muy buena referencia para crear aplicaciones JDBC completas.
Acceso a Bases de Datos con Java (JDBC) **[Versión en caché]**

Registrar el controlador JDBC

Cargar o registrar el controlador que queremos utilizar es muy sencillo y sólo implica una línea de código. Deberemos consultar la documentación del controlador que vamos a utilizar para conocer el nombre de la

clase que hay que emplear. En el caso del controlador para MySQL es "**com.mysql.jdbc.Driver**", es decir, la clase **Driver** que está en el paquete **com.mysql.jdbc** del conector que hemos descargado, y que habrás observado que no era más que una librería empaquetada en un fichero .jar. La línea de código necesaria sería:

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

Hay que asegurarse de que el archivo .jar que contiene el controlador JDBC para nuestro SGBD está incluido en el CLASSPATH que emplea nuestra máquina virtual, o bien está instalado en el directorio ext del JRE de nuestra instalación del JDK.

Quizás te ayude a recordar el lugar donde debes colocarlo el hecho de que el controlador que viene en el fichero .jar no es más que una **librería externa** al JDK, y que si la metemos dentro de nuestro **entorno de ejecución (JRE)** no habrá problemas con el CLASSPATH, ya que el entorno de ejecución estará incluido en el CLASSPATH.

Por tanto, dentro de la carpeta de instalación de nuestro JDK (C:\Archivos de programa\Java\jdk1.5.0_04, por ejemplo), **debemos buscar la carpeta de instalación del JRE** (C:\Archivos de programa\Java\jdk1.5.0_04\jre), **y dentro la carpeta de librerías** (C:\Archivos de programa\Java\jdk1.5.0_04\jre\lib), **y dentro de ésta la carpeta de librerías externas** (C:\Archivos de programa\Java\jdk1.5.0_04\jre\lib\ext).

Ya tenemos formada toda nuestra ruta, dentro de la que debemos colocar el conector o controlador JDBC para MySQL, que es el fichero mysql-connector-java-3.1.12-bin.jar o el nombre que tenga en la última versión del mismo que te hayas descargado.

Una vez cargado el controlador, es posible hacer una conexión al SGBD.

En los siguientes enlaces puedes encontrar un ejemplo presentado primero mediante una demo que te destaca los aspectos que queremos resaltar en esta primera ejecución de una aplicación que accede a bases de datos, y su código fuente después para que lo descargues y hagas las pruebas que estimes oportunas.

Descarga el proyecto AccesoBDatos

Establecer una conexión con la base de datos

El Driver ya está registrado. Ahora podemos establecer una conexión al SGBD y a la base de datos que deseemos. Para ello se utiliza la clase **DriverManager** y el método adecuado:

```
static Connection getConnection(String url, String user,String password)
```

Este método intenta establecer conexión con la base de datos que le indiquemos en el campo **url** empleando para ello el controlador que hemos registrado anteriormente. Si tiene éxito devuelve un objeto **Connection** que es el que utilizaremos para comunicar con la base de datos.

Hay que explicar que la formación del **url** dependerá del controlador JDBC que se utilice y que habrá que consultar la documentación suministrada con él. En concreto en el controlador que estamos utilizando para MySQL la sintaxis del **url** es:

```
"jdbd:mysql://<Servidor MySQL> : <puerto comunicación> / <base de datos>"
```

Por otra parte el **user** y el **password** identificarán el usuario y la contraseña con el que accederemos al SGBD, para mejorar la seguridad. **El valor de usuario y contraseña deberá estar registrado en el servidor MySQL al que nos conectemos.**

Veamos la sentencia que quedaría en nuestro ejemplo ejemplo, con una conexión al servidor MySQL alojado en nuestro propio equipo (jdbc:mysql://localhost) establecida la comunicación por el puerto :3306 del equipo, a la base de datos alojada en el mismo llamada biblioteca, conectándonos como usuario root con contraseña 123456:

```
miConexion=DriverManager.getConnection("jdbc:mysql://localhost:3306/biblioteca",
"root","123456");
```

Ejecutar las instrucciones SQL

Dijimos que para consultar la base de datos se usaba un lenguaje propio para los SGBD, llamado SQL. ¿Quiere eso decir que las consultas a la Base de datos no las hago usando el lenguaje Java?

La respuesta es que ni si, ni no. Usamos SQL, que no es Java, para las consultas a la base de datos, pero esas sentencias SQL van "embebidas" en otras sentencias especiales que sí son propias de Java. Digamos que las consultas SQL las escribimos como parámetros de algunos métodos Java que reciben el String con el texto de la consulta SQL.

Para ejecutar las instrucciones SQL se utilizan los objetos de tipo **`Statement`** o PreparedStatement, dependiendo de si queremos ejecutar SQL o SQL precompilado. La diferencia entre los dos métodos es que en el primer caso se pasan al controlador JDBC las instrucciones SQL cómo un String, mientras que en el caso de utilizar un objeto **`PreparedStatement`** se puede parametrizar la instrucción SQL proporcionando más flexibilidad a la hora de programar. En cualquier caso se utilizan los métodos definidos en la interface **Connection.**

1. Para ejecutar la consulta se utiliza el método **`executeUpdate(String sql)`** para instrucciones tipo **`UPDATE`**, **`DELETE`**, **`INSERT`** o instrucciones de tipo DDL, que son las que de alguna forma modifican la base de datos.
2. Para las instrucciones de tipo **`SELECT`**, que obtienen datos de la base de datos, pero que no la modifican, empleamos el método **`ResulSet executeQuery(String sql)`**. Hay que hacer notar que en este último caso el método de ejecución del comando SQL devuelve un objeto de tipo **`ResultSet`** que sirve para contener el resultado del comando **`SELECT`**, y que nos permitirá su procesamiento.

Procesamiento de datos devueltos por una instrucción SELECT (I)

__Víctor__ le plantea a __Carmen__ una duda, y es que él sabe que cuando se hace una consulta a una base de datos que puede tener miles de registros. Algunas consultas pueden devolver un conjunto de registros bastante grande, que puede resultar difícil de manejar desde el programa, ya que por norma general tendremos que manejar esos datos registro a registro.
A estas alturas está seguro de que Java tendrá una buena solución para esto, pero le gustaría saber cuál es. Carmen le comenta que las consultas devuelven "un __ResulSet__", que viene a ser una versión reducida de la tabla de la base de datos, en la que cada columna es un campo, y cada fila un registro, pero sólo con la información que hemos pedido al consultar. Además, sobre esa tabla tenemos disponibles toda una serie de métodos que nos permiten movernos hacia delante y hacia atrás en las filas, y obtener la información de una fila, campo a campo. Resulta fácil y manejable, incluso más que manejar un array típico de Java.

- Ya hemos visto en el apartado anterior que cuando ejecutamos una instrucción **`SELECT`** utilizando el método **`executeQuery()`** se obtiene un objeto de tipo **`ResultSet`**. Este objeto tiene una serie de métodos que sirven para recorrerlo y para extraer los datos que ha generado la ejecución del **`SELECT`**.
- Para obtener los datos de los atributos de cada fila del **`ResultSet`** se tienen los métodos **`getXXX (<nombre atributo>)`**, donde **`XXX`** es el tipo de dato que se quiere utilizar. Hay que tener en

cuenta los tipos de dato utilizados en la tabla de la base de datos para utilizar un tipo de dato Java que sea compatible. A continuación tienes una tabla donde se especifican esas correspondencias.

Métodos de `ResultSet`	Integer	Float	Double	Decimal	Numeric	Char	Date	Time	Varchar
`getByte()`	x	x	x	x	x	x			
`getShort()`	x	x	x	x	x	x			
`getInt()`	x	x	x	x	x	x			
`getLong()`	x	x	x	x	x	x			
`getFloat()`	x	x	x	x	x	x			
`getDouble()`	x	x	x	x	x	x			
`getBigDecimal()`	x	x	x	x	x	x			
`getBoolean()`	x	x	x	x	x	x			
`getString()`	x	x	x	x	x	x	x	x	x
`getBytes()`							x		
`getDate()`						x		x	x
`getTime()`						x			x
`getTimestamp()`						x		x	x
`getAsciiStream()`						x	x		
`getUnicodeStream()`						x	x		
`getBinaryStream()`							x		
`getObject()`	x	x	x	x	x	x	x	x	x

PARA SABER MÁS
Para una información más detallada acerca de los métodos getXXX() puedes usar el siguiente enlace, concretamente el apartado "Recuperar Valores desde una Hoja de Resultados" o directamente consultar la documentación del API de Java para el interface ResultSet:
Acceso a Bases de Datos JDBC [Versión en Caché]
Documentación de la API de Java [Versión en caché]

Procesamiento de datos devueltos por una instrucción SELECT (II)

En cuanto a la forma de recorrer un `ResultSet` el método más utilizado es `next()`. Este avanza una fila en el `ResultSet` y devuelve un valor lógico verdadero (true) o falso (false) indicando si se ha llegado al final del `ResultSet` o no.

No obstante, existe toda una serie de métodos útiles para moverse por el ResultSet, hacia delante y hacia atrás.

En la siguiente tabla tienes algunos de los más usados.

Debes tener en cuenta que el ResulSet que devuelve la consulta viene a ser como la tabla de la BD, pero sólo con las columnas que seleccionemos en el SELECT y con las filas que cumplan la condición especificada en la cláusula WHERE del SELECT de nuestra consulta. Pero tiene algunas particularidades:

- Antes del primer registro de la base de datos añade una fila vacía.
- Lo mismo hace después del último registro.
- Todas las filas, que se corresponden con los registros de la base de datos, están numeradas.
- Siempre hay un puntero "imaginario" señalando una de ellas.
- Inicialmente el puntero estará delante del primer registro.
- Podremos moverlo mediante los métodos de la siguiente tabla hacia delante y hacia atrás para situarlo en el registro que nos interese.

- Sólo se puede recuperar la información del registro que está señalado por ese puntero imaginario.

Métodos más comunes para navegar por un objeto ResulSet

Tipo devuelto	Método	Utilidad
boolean	`absolute(int row)`	Sitúa el cursor en la fila cuyo número se especifique como parámetro en la llamada. Devuelve verdadero si el puntero queda dentro del ResulSet.
void	`afterLast()`	Sitúa el cursor al final del ResulSet, justo detrás de la última fila de datos.
void	`beforeFirst()`	Sitúa el cursor al principio del ResulSet, justo delante de la primera fila de datos.
void	`close()`	Cierra y libera todos los recursos usados por el ResulSet. No es imprescindible usarlo, ya que el ResultSet se cierra automáticamente al cerrar el Statement a partir del cual se creó.
boolean	`first()`	Mueve el cursor a la primera fila de datos del ResulSet, si existe una primera fila a la que moverse, devolviendo verdadero. Si no puede por estar vacío el ResulSet, devuelve falso.
boolean	`isAfterLast()`	Devuelve verdadero si el puntero está al final del ResulSet, detrás de la última fila de datos.
boolean	`isBeforeFirst()`	Devuelve verdadero si el puntero está al principio del ResulSet, antes de la priemera fila de datos.
boolean	`isFirst()`	Devuelve verdadero si el puntero está en la primera fila de datos.
boolean	`isLast()`	Devuelve verdadero si el puntero está en la última fila de datos.
boolean	`last()`	Mueve el puntero a la última fila de datos, y devuelve verdadero si ha sido posible. Devolverá falso si el ResulSet está vacío y no hay una última fila de datos a la que moverse.
boolean	`next()`	Mueve el puntero a la fila posterior en el ResultSet, si existe una siguiente fila , devolviendo verdadero. Si no es posible por no existir dicha fila (estamos ya al final del ResulSet), devolverá falso.
boolean	`previous()`	Mueve el puntero a la fila anterior en el ResultSet, si existe una fila anterior, devolviendo verdadero. Si no es posible por no existir dicha fila, (estamos ya al principio del ResulSet) devolverá falso.
int	`getRow()`	Devuelve el número de fila en la que está situado el puntero. La primera fila es la 1, la segunda es la 2 y así sucesivamente. Si no hay ninguna fila en el ResultSet, devuelve el valor cero.
boolean	`Relative(int rows)`	Mueve el puntero el número de filas especificado como parámetro desde la posición de la fila actual. Si es positivo, hacia delante, y si es negativo hacia atrás. Devuelve verdadero si todo ha ido bien, y falso si nos hemos salido del ResulSet.

Puedes encontrar la información completa sobre los métodos del interfaz ResultSet consultando la documentación de la API de Java.

En el último apartado de la unidad, dedicado a darte ejemplos, tienes tanto una demo como el enlace al código del proyecto RecorrerTablaBD que te proporcionamos para que lo pruebes y lo uses de guía en tus programas. En él se hace uso de algunos de esos métodos para recorrer un `ResulSet` que en este caso se obtiene tras una consulta que incluye todos los libros de la tabla libros de la base de datos biblioteca que venimos usando como ejemplo.

Liberar los recursos de la conexión

¿Qué pasará si muchos usuarios establecen conexiones con la base de datos y no se preocupan de cerrarlas cuando terminen de usarlas?

Posiblemente que llegará un momento en el que la base de datos no podrá aceptar que nuevos usuarios se conecten, porque tiene sus conexiones reservadas, aunque realmente no estén siendo usadas.

Una vez que hayamos terminado de usar la conexión a la base de datos, hay que liberar los recursos que se estaban consumiendo en dicha conexión.

Hay que liberar el **`ResultSet`** si lo hubiéramos utilizado, el objeto **`Statement`** y el objeto **`Connection`**. Todos estos objetos disponen de un método **`close()`** para liberar los recursos consumidos.

A continuación tienes a modo de ejemplo las sentencias que liberan los recursos de la conexión con la base de datos biblioteca en uno de los programas que te proporcionamos. En este caso las sentencias necesarias se han incluido en un método cada una, con fines de documentación, más que nada, y al mismo tiempo para capturar dentro de esos métodos posibles excepciones. No se ha cerrado el ResulSet porque como se ha comentado en la tabla del apartado anterior, se cierra automáticamente al cerrar el objeto Statement que lo creó. Aquí llevas el código:

```
miConexBD=new MiConexionBD();
     miConexBD.inicializarConexion();
     Connection conn=miConexBD.getMiConexion();
     stmt= conn.createStatement();
//...
     public void close(ResultSet rs){
        if(rs !=null){
            try{
               rs.close();
            }
            catch(Exception e){}
        }
     }
    /**Este método establece el cierre de la sentencia Statement usada para la consulta. *
     public void close (java.sql.Statement stmt){
        if(stmt !=null){
            try{
               stmt.close();
            }
            catch(Exception e){      /*Capturada pero no tratada*/          }
        }
```

```
    }
   /** Este método  cierra la conexión a la base de datos */
    public void destroy(){
        if(miConexion !=null){
            try{
               miConexion.close();
            }
            catch(Exception e){      /*Capturada pero no tratada*/          }
    }
  }
/* Y ahora el método que invoca a los métodos que cierran los recursos de la conexión*/
   private void formWindowClosing(java.awt.event.WindowEvent evt) {
       miConexBD.close(stmt);
       miConexBD.destroy();
   }
```

Gestión de excepciones en JDBC

*Cuando **Víctor** le enseña a **José** la primera aplicación que ha hecho con bases de datos, para el Ayuntamiento, se encuentra con un problema, y es que la aplicación no funciona, no muestra los datos de la base de datos. **Víctor** no sabe porqué. José mira el tratamiento de las excepciones que ha hecho **Víctor**, en el que se limita a capturarlas para que no aborte la aplicación, pero no las trata. Le dice que el problema seguramente es que se está intentando acceder a una base de datos que no existe, o que el servidor MySQL no está arrancado, o que se ha intentado hacer alguna operación no permitida sobre la base de datos, como acceder con un usuario y contraseña no registrados.*
*Lo que pasa es que al producirse la excepción, además de capturarla debería haber añadido código en el manejador para que mostrara algún mensaje informando del error. Aparte de la información que pueda escribir directamente **Víctor**, le aconseja que use el método **`getMessage()`** de la clase **SQLException** para recoger y mostrar el mensaje de error que ha generado MySQL, lo que seguramente nos proporcionará una información más ajustada sobre lo que está fallando.*
Por lo demás, José le felicita por el buen trabajo que ha hecho, y le dice que ya es todo un programador, y que en cuanto termine ese detalle del tratamiento de errores, podrán dar por finalizada su primera aplicación comercial.

Cuando se escribe código Java para acceder a una base de datos externa gestionada por un SGBD pueden ocurrir situaciones de error que conviene tratar de forma adecuada, bien informando al usuario del error o incluso recuperando el control del programa.

Cuando se produce un error se lanza una excepción del tipo **`java.sql.SQLException`**.

- **Es importante que todas las operaciones de acceso a base de datos estén encerradas en un bloque `try-catch` que gestione las excepciones.**
- Los objetos del tipo **`SQLException`** tienen dos métodos muy útiles para obtener el código del error producido y el mensaje descriptivo del mismo, **`getErrorCode()`** y **`getMessage()`** respectivamente.

Sobre todo resulta útil **`getMessage()`** que imprime el mensaje de error asociado a la excepción que se ha producido, que aunque estará en inglés, nos puede ayudar a saber qué ha generado el error que

causó la excepción. `getErrorCode()`, por el contrario, nos devuelve sólo un número entero que representa el código de error asociado. Puede ser útil, pero requiere buscar en la documentación, lo cual es más engorroso.

Poniéndolo todo en un ejemplo

Bien, ya conocemos todo lo necesario para escribir nuestro primer programa Java con acceso a una base de datos externa. Utilizaremos como SGBD MySQL, y la **base de datos biblioteca** que tenemos creada con algunos datos introducidos en la **tabla libro**. El programa de ejemplo seguirá los pasos descritos en los apartados anteriores para producir una lista de los libros que tenemos con indicación de su título y autor.

Observa también la siguiente animación en la que destacamos los aspectos más significativos en los que te debes fijar al ejecutarlo.

Puedes probar el ejemplo anterior usando el enlace al código que te ponemos al final del apartado y comprobar que se obtiene la lista de los libros que hayas insertado en la tabla libro. Por ahora, nuestra aplicación sólo hace una consulta sobre la base de datos, pero por algo hay que empezar...

Te recomendamos que hagas tus propias pruebas modificando el programa anterior para producir otros resultados como:

- Lista de libros de género "Infantil".
- Lista de libros que cuestan más de 22 euros.
- Lista de libros que son del género "Aventuras" y su precio es menor de 30 euros.

Además de ejecutarlo, te recomendamos que leas con atención el código, identificando las principales sentencias que intervienen en la conexión, de entre las que hemos venido hablando hasta ahora.

Descarga el proyecto PrimerAccesoBD

Ejemplos de aplicaciones Java con acceso a bases de datos

En esta última sección de esta unidad dedicada al acceso a bases de datos desde Java, te vamos a presentar algunos ejemplos de operaciones bastantes habituales cuando se desarrollan aplicaciones de gestión. Se han realizado utilizando un interface gráfico, y se sigue utilizando la base de datos **biblioteca que tiene una tabla llamada libros.**

Te recomendamos que veas las animaciones en las que destacamos algunos aspectos en los que queremos que te fijes especialmente, y además de ejecutar los ejemplos, **leas atentamente los comentarios que se incluyen en el código, ya que con ellos pretendemos explicar algunas cosas sobre el funcionamiento en el mismo código**, para no alargar excesivamente los contenidos de la unidad.

Al mismo tiempo, estos ejemplos deben servirte como "plantillas" de las principales operaciones que deberás realizar sobre Bases de datos en futuras aplicaciones que tengas que desarrollar, así como en el proyecto final de la siguiente unidad.

Los ejemplos que te proponemos son los siguientes:

- **Seleccionar las filas que cumplen una condición:** Se solicita al usuario un género literario y se presentan los libros que pertenecen a ese género.

Descarga el proyecto ConsultaBD

- **Insertar filas en una tabla:** En este ejemplo se presenta al usuario un formulario donde puede rellenar los valores de los campos de la tabla libro, cuando se pulsa el botón "añadir" se lanza una instrucción INSERT contra la base de datos.

Descarga el proyecto InsertarYListarBD

- **Recorrer el contenido de una tabla:** Se presentan los datos de la primera fila de la tabla libro y con botones se puede recorrer adelante o atrás el listado completo, al mismo tiempo que se muestra el total de libros que incluye la consulta y el número de orden que ocupa el libro que estamos viendo en la ventana en cada momento.

Descarga el proyecto RecorrerTablaBD

__José__ ha estado esta mañana en el Ayuntamiento, y el informático de la casa le ha felicitado por la estupenda aplicación que le han hecho. __José__ le dice que todo el mérito es de __Víctor__, del empeño que ha puesto y de la buena disposición que ha mostrado en todo momento para aprender, lo que le ha llevado a convertirse en un estupendo profesional.
De vuelta a la oficina, ha hablado con __María__ y con __Jesús__, los otros socios de __SI Andalucía__, para contárselo, y para proponerles la contratación de __Víctor__ como programador de la empresa, con unas buenas condiciones laborales. También contratarán a __Carmen__, quien además de haber finalizado las prácticas de forma más que satisfactoria, y haber demostrado sus grandes conocimientos de programación, ha demostrado una gran habilidad y paciencia en la formación de __Víctor__. Como la empresa está recibiendo muchos encargos de trabajo, han pensado que __Carmen__ sería la persona ideal para formar a los nuevos trabajadores que tendrán que contratar, integrándolos en las técnicas, procedimientos y metodología de trabajo de la empresa.
Y deciden darles una sorpresa, aprovechando que es Viernes, y al cerrar la empresa los invitan a una fiesta, en la que además de darles la bienvenida como nuevos empleados de la plantilla, pasan un rato agradable junto a otros amigos comentando todo lo que han vivido en este intenso último año que han dedicado a mejorar su formación como programadores desarrollando aplicaciones, participando en la construcción de una nueva empresa y aprendiendo las peculiaridades de la programación y del lenguaje __Java__.