# Acceso a base de datos con JDBC

La API de JDBC fue diseñada para mantener las cosas simple. Esto significa que JDBC hace mas fácil las tareas con base de datos del día a día.

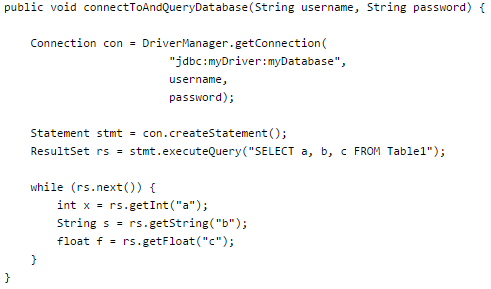
## Introducción a JDBC

La API de JDBC es una API de Java que puede acceder cualquier tipo de datos tabulares, especialmente aquellos almacenados en bases de datos relacionales.

JDBC te ayuda a escribir aplicaciones Java que realizan estas 3 actividades:

* Conectarse a una fuente de datos, como lo es una base de datos
* Realizar consultas y actualizaciones a la base de datos
* Obtener y procesar los resultas recibidos de la base de datos en respuesta a tu consulta

El siguiente fragmento de código, muestra un simple ejemplo de las 3 actividades anteriores:



Este corto fragmento de código, crea un objeto de DriverManager para conectar a un driver de base de datos y acceder a la base de datos, crea un objeto de Statement que transporta tu consulta en SQL a la base de datos; crea un objeto de ResultSet que obtiene el resultado de tu consulta, y ejecuta un bucle while, que obtiene y muestra esos resultados.

## Componentes de JDBC

JDBC incluye los siguientes cuatro componentes:

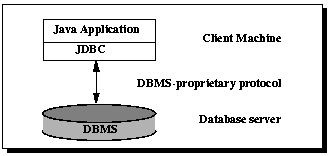
* La API de JDBC: La API de JDBC provee acceso programático desde Java a datos relacionales. Usando la API de JDBC, las aplicaciones puede ejecutar sentencias SQL, obtener resultados y propagar cambios a la fuente de datos utilizada. La API de JDBC también puede interactuar con múltiples fuentes de datos en un entorno distribuido y heterogéneo.
* JDBC Driver Manager: La clase DriverManager define objetos que pueden conectar aplicaciones Java a un driver JDBC. DriverManager ha sido tradicionalmente la columna vertebral de la arquitectura JDBC
* Suite de pruebas de JDBC: La suite de pruebas de JDBC, te ayuda a determinar que driver JDBC pueden ejecutar tu programa. Estas pruebas no son exhaustivas, pero ejecutan gran parte de las características mas importantes en la API de JDBC
* Puente JDBC-ODBC: El puente de software de Java provee acceso JDBC vía drivers ODBC. Debes notar que necesitas cargar el código del driver ODBC en cada máquina cliente que utilizará el driver

## Arquitectura JDBC

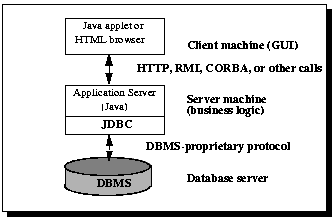
### Modelos de procesamiento de 2 y 3 capas

La API de JDBC soporta los modelos de 2 y 3 capas para el acceso a base de datos.

En el modelo de 2 capas, una aplicación interactúa directamente con la fuente de datos. Esto requiere un driver de JDBC que se pueda comunicar con la fuente de datos específica que esté siendo accedida. Los comandos del usuario son entregados a la base de datos u otra fuente de datos, y los resultados son enviados de vuelta al usuario. La fuente de datos puede estar en otra máquina a la cual el usuario se conecta a través de la red. Esto es conocido como una configuración cliente – servidor, donde la máquina del usuario es el cliente, y la máquina en donde se localiza la fuente de datos es el servidor.



En el modelo de 3 capas, los comandos son enviados a una capa intermedia de servicios, y son estos los que envían los comandos a la fuente de datos. La fuente de datos procesa los comandos y envía el resultado de vuelta a la capa intermedia, que entonces envía la respuesta al usuario.



Con las empresas incrementando el uso del lenguaje de programación Java para escribir código de lado servidor, la API de JDBC está siendo usada mas y mas en la capa intermedia de una arquitectura de 3 capas. Algunas de las características que hacen de JDBC una tecnología de servidor, son su soporte para el “connection pooling”, transacciones distribuidas y conjunto de filas desconectado.

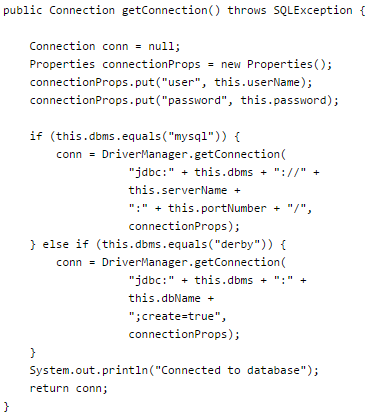
## Estableciendo una Conexión

Primero, necesitas establecer una conexión a la fuente de datos que quieras utilizar. Una fuente de datos puede ser un RDBMS u otra fuente de datos con un driver de JDBC correspondiente. Normalmente una aplicación JDBC conecta a la fuente de datos utilizando una de dos clases:

* DriverManager: Esta clase conecta una aplicación a una fuente de datos, que es especificada a través de una URL. Cuando esta clase intenta por primera vez establecer la conexión, automáticamente carga cualquier driver JDBC que encuentre en el class path.
* DataSource: Esta interface es preferida sobre DriverManager porque permite que los detalles acerca de la fuente de datos sean transparentes para tu aplicación.

## Usando la clase DriverManager

Conectar a tu DBMS con la clase DriverManager implica invocar al método DriverManager.getConnection. El siguiente ejemplo establece una conexión a una base de datos:



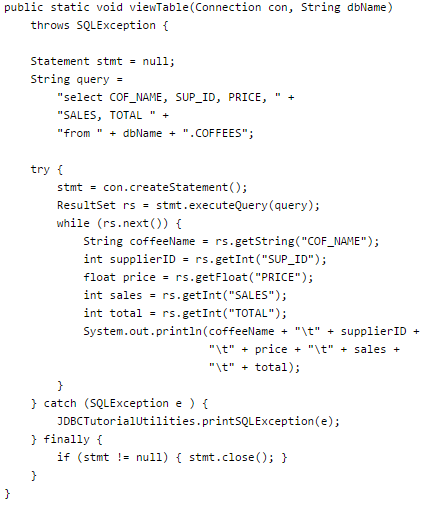
El método DriverManager.getConnection establece una conexión a la base de datos. Este método requiere una URL, la cual varía dependiendo de tu DBMS. La siguientes son ejemplos de URLs:

* MySQL: jdbc:mysql://localhost:3306/
* Java DB: jdbc:derby:testdb;create=true

Este método específica el usuario y contraseña necesarios para acceder al DBMS con un objeto Properties. El método retorna un objeto Connection, que representa la conexión al DBMS.

## Obteniendo y modificando valores a partir de Result Sets

El siguiente método imprime el contenido de la tabla COFFEES, y demuestra el uso del objeto ResultSet:



Un objeto ResultSet es una tabla de datos representando un conjunto de resultados de una base de datos, que usualmente es generado al ejecutar una sentencia que consulta a la base de datos.

## Actualizando filas de objetos ResultSet

Un objeto ResultSet por default no puede ser actualizado, solamente puedes mover el cursor al siguiente elemento. Sin embargo, puedes crear objetos ResultSet que puede ser consultados hacia adelante o a una posición específica, y actualizados.

El siguiente método multiplica la columna PRICE de cada fila por el argumento percentage:

