Universidad
Europea de Madrid

LAUREATE INTERNATIONAL UNIVERSITIES

SQL. DEFINICIÓN DE DATOS

LDD/LMD EMPOTRADO: JDBC

Índice

Presentación

Los sistemas de base de datos (SBD) están compuestos de la base de datos (BD), del sistema de gestión de la base de datos (SGBD) que permite gestionarla y administrarla y de las aplicaciones a través de las cuales los usuarios interactúan con el SGBD. Hemos aprendido a utilizar las sentencias del lenguaje del SGBD (en nuestro caso SQL) que establecen cómo pedirle al SGBD las operaciones que queremos realizar sobre la base de datos.

Normalmente, los usuarios que interactúan con el SBD no tienen por qué conocer SQL, por lo que se utilizan aplicaciones. Éstas ocultan al usuario las sentencias SQL que hay que generar y enviar al SGBD para realizar las operaciones que el usuario requiere: consultar información, introducir información nueva y eliminar información. De esta forma, el usuario sólo percibirá que rellena un formulario y la aplicación transformará esta acción en una sentencia INSERT que envía al SGBD para su ejecución.

Las aplicaciones de usuario se realizan con un lenguaje de alto nivel. Los lenguajes de alto nivel ofrecen un mecanismo que permite introducir sentencias SQL dentro del código de alto nivel, enviárselas al SGBD para que las ejecute y obtener de éste los resultados. Este mecanismo se llama lenguaje empotrado.

En este tema aprenderás a:

- Utilizar Java como lenguaje de alto nivel.
- Usar JDBC como lenguaje empotrado.

LDD/LMD empotrado

El acceso a las BD por parte de lenguajes de propósito general, se hace necesario porque:

- No todas las consultas pueden expresarse en SQL.
- SQL no puede llevar a cabo acciones no declarativas (impresión de informes, interacción con un usuario, envío de resultados a una interfaz gráfica, etc.).

Se puede usar SQL dentro de lenguajes como C, C++, Java, Pascal, ADA, y otros.

Organización general

El usuario no interactúa directamente con el SGBD sino que lo hace a través de una aplicación que hace de interfaz. Esta aplicación llevará incrustada sentencias SQL, que definen las operaciones que se quieren realizar. Esas sentencias son enviadas al SGBD que interactúa con la BD aplicándole la operación especificada.

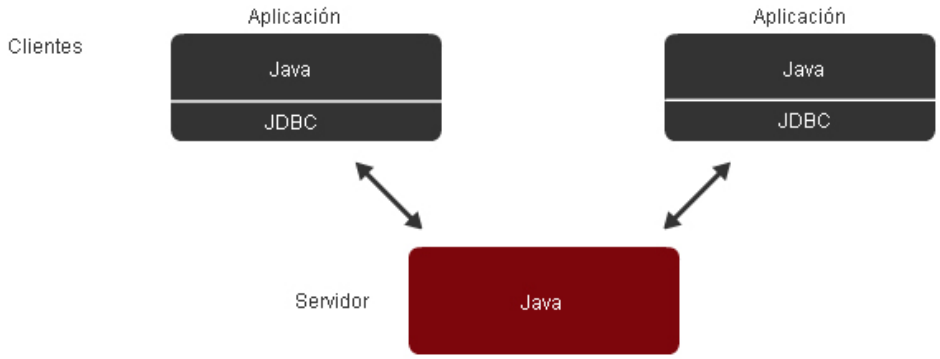
Existen diferentes maneras de trabajar con bases de datos en SQL dependiendo del lenguaje de programación de propósito general empleado, entre otras:

- SQL/RT: lenguaje C
- JDBC: lenguaje Java

JDBC (Java Database Connectivity)

JDBC (Java Database Connectivity) es una API orientada a la conexión con bases de datos remotas para aplicaciones desarrolladas en Java. Está compuesto por un conjunto de **clases** que permiten conectar la aplicación a la base de datos e interactuar con el SGBD para enviarle las sentencias SQL y recuperar los resultados obtenidos de la ejecución de las mismas.

JDBC trabaja sobre una **arquitectura Cliente/Servidor**. La base de datos está situada en un **servidor** donde también está instalado el SGBD que la administra. Por otro lado está el **host del usuario**, parte cliente, que tiene instalada la aplicación Java con JDBC que utilizará para definir las operaciones que quiere realizar sobre la base de datos. La aplicación envía al servidor las peticiones que realiza el cliente en forma de sentencias SQL y el servidor responderá al cliente con los resultados proporcionados por el SGBD: tablas resultado de las consultas, códigos de error, etc.

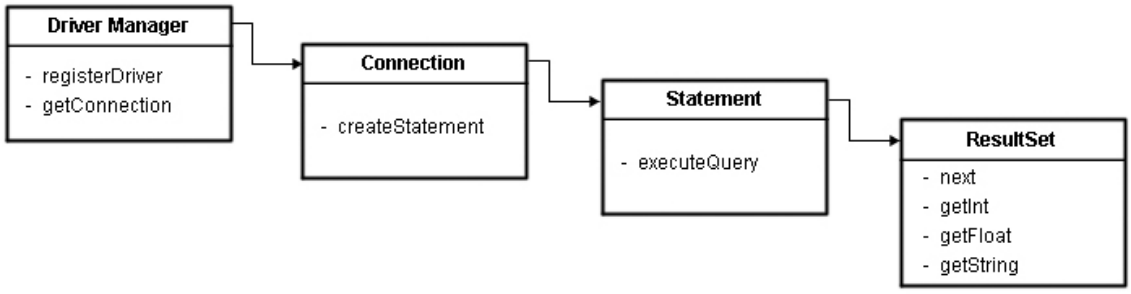


JDBC dispone de dos tipos de **clases** del SGBD que se esté utilizando (Oracle, MySQL, Microsoft SQL-Server, etc.):

- Un conjunto de **clases comunes**.
- Otro conjunto de **clases específicas**.

JDBC. Presentación de las clases

Las clases básicas de las que está compuesto JDBC, junto con sus métodos, se muestran en el esquema:



Donde:

- **DriverManager**. Es la clase gestora de los controladores de las aplicaciones en Java. Cada SGBD tendrá su propio controlador. Con esta clase especificamos qué controlador y, por tanto, qué SGBD vamos a utilizar (método registerDriver) y establecemos una conexión con la BD (método getConnection).
- **Método executeQuery**. Nos permite pasar la consulta al SGBD y pedir que se ejecute.
- **Clase ResulSet**. El resultado de executeQuery es una instancia de la clase ResultSet que representa una TABLA con el resultado de la consulta. Hay varios **métodos** para moverse por esa tabla.

En general, toda aplicación Java, tendrá que realizar los siguientes pasos: (1) Establecer la conexión de la Bd. (2) Solicitar información. (3) Acceder a la información solicitada. (4) Cerrar la conexión.

métodos

- **next**: para avanzar en la tabla.
- **getString**: para leer cadenas de char.
- **getFloat**: para leer números reales.
- **getInt**: para leer números enteros.

Paso 1: establecer la conexión con la BD

En general, **toda aplicación Java** tendrá que realizar estos pasos, que se desarrollan en las siguientes pantallas.

Para establecer la conexión con la base de datos hay que:

- **Cargar el controlador del SGBD específico** utilizando el método registerDriver de la clase DriverManager. Esto permite establecer una comunicación entre la aplicación y la BD en el SGBD específico utilizado. Las siguientes partes de código realizan esta función para Oracle y MySQL respectivamente:

```
DriverManager.registerDriver(new
oracle.jdbc.driver.OracleDriver());
DriverManager.registerDriver(new com.mysql.jdbc.Driver());
```

- **Establecer una conexión a la BD** con el método getConnection de la clase DriverManager. El resultado será un objeto de tipo Connection que representa el canal de comunicación entre la aplicación y la BD a través del cual enviaremos las sentencias SQL a ejecutar y recibiremos las respuestas del SGBD:

```
getConnection(url.servidor, usuario, password) → Connection
```

La parte de código que realiza esta operación para Oracle es:

```
Connection conexion =
DriverManager.getConnection(jdbc:oracle:thin:@host:puerto:oracle
_sid, usuario, clave);
```

Ejemplo

Explicación del código

Ejemplo

```
Connection conexion = DriverManager.getConnection
(jdbc:oracle:thin:@10.2.9.1:1521:ESP, M31SIST-G20, grupo20);
```

En detalle

Explicación del código

Donde:

Host: dirección del servidor con el que se desea conectar, por ejemplo, aurora.esi.uem.es ó 10.2.9.1.

Puerto: Puerto TCP a través del cual se establece la conexión. Normalmente 1521.

Oracle_sid: identificador de la BD Oracle con la que se conectará.

Usuario: nombre de usuario con el que se va a conectar a la BD.

Clave: clave de acceso para el usuario especificado anteriormente.

Paso 2: enviar al SGBD la sentencia a ejecutar

Para enviar al SGBD la sentencia a ejecutar hay que realizar dos pasos:

- Crear una sentencia. Para ello utilizamos el método createStatement de la clase Connection:

```
Connection.createStatement() →Statement
```

El siguiente fragmento de código crea una sentencia sobre el objeto conexión, creado en el apartado anterior.

```
Statement senteciaSQL = conexion.createStatement();
```

- Ejecutar la sentencia SQL y recoger el resultado.Si la sentencia que queremos enviar es una consulta select, utilizamos el método executeQuery de la clase Statement. Este método devuelve la tabla resultado de la consulta que se almacena en un objeto de clase Resulset.

```
Statement.executeQuery(ConsultaSQL) → ResultSet
```

El siguiente código realiza esto: primero crea un objeto de clase ResultSet y después almacena en éste el resultado de ejecutar la sentencia SQL. Como se puede comprobar, la sentencia a ejecutar es un string que se pasa como argumento a executeQuery.

```
ResulSet resultado = sentenciaSQL.executeQuery (select * from
clientes where saldo > 1000);
```

Si lo que queremos es realizar un insert, delete, create table, drop table ó create index, utilizaremos el método executeUpdate que devuelve un entero que indica el número de filas afectadas por la modificación.

```
Statement.executeUpdate(ConsultaModificaciónSQL) → int
```

Método executeUpdate

Ejemplo

```
Connection conexion = DriverManager.getConnection
(jdbc:oracle:thin:@10.2.9.1:1521:ESP, M31SIST-G20, grupo20);
```

Paso 3: acceder a la información obtenida

Una vez que la consulta se ha ejecutado y disponemos del resultado en el objeto de clase ResultSet podemos utilizar los métodos:

- ResultSet.next() → boolean. Pasa a la siguiente tupla, y devuelve false si hemos llegado al final de la tabla.
- ResultSet.get<TIPO_*X*> (int numColumna) → TIPO_*X*. Permite almacenar el dato de la columna <numColumna> en una variable del mismo tipo. Para especificar la columna del dato que queremos recuperar podemos utilizar el número de columna ó el nombre de la misma pasándoselo a este método como un argumento de tipo string.

```
ResultSet.get<TIPO_X> (String nombreColumna) →TIPO_ X.
```

El siguiente trozo del código recoge el valor de la segunda columna del objeto **resultado** que contenía la tabla, consecuencia de ejecutar la sentencia SQL ***select * from clientes where saldo > 1000***. La tabla **clientes** tiene el formato clientes (dni, nombre, dirección).

```
String s = resultado.getString(2);
String s = resultado.getString("nombre");
```

Paso 4: cerrar la conexión

Una vez que hemos realizado todo el proceso que necesitábamos debemos cerrar de forma explícita el objeto de tipo *ResultSet* donde almacenamos la tabla **resultado**, la sentencia a través de la cual enviamos la consulta al SGBD y la conexión que establecimos con la base de datos. Para ello utilizamos el método *close* de estas clases:

```
ResultSet.close();
Statement.close();
Connection.close();
```

El fragmento que realiza esta operación es:

```
resultado.close();
senteciaSQL.close();
conexion.close();
```

Todos los métodos anteriores generan una excepción ante cualquier error:

```
SQLException
```

Documentos **Código completo**

Paso de argumentos y sentencias preparadas I

En la mayoría de los casos, la sentencia SQL a ejecutar dependerá de los valores que ha introducido el usuario a través del formulario que proporciona la aplicación. En estos casos el procedimiento es exactamente el mismo, salvo por la forma en que se genera y envía la consulta al SGBD (paso 2 del proceso descrito previamente). Para ello se utilizan **sentencias preparadas**.

Las sentencias preparadas permiten:

- Parametrización
- Mayor eficiencia

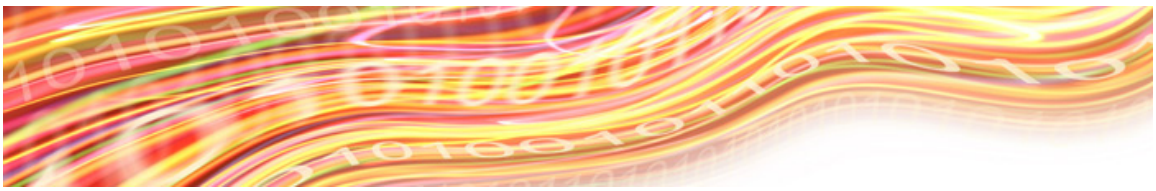
Se utiliza una subclase de Statement, llamada PreparedStatement.

```
Connection.prepareStatement(ConsultaSQLParametrizada) → PreparedStatement
```

Donde ConsultaSQLParametrizada es un string que contendrá la sentencia SQL a ejecutar donde hemos sustituido los parámetros a utilizar por el carácter ?.

Ejemplo:

```
PreparedStatement sentenciaSQL = conexion.prepareStatement
(“select * from ordenes where o < ? and cant < ?”);
```



Paso de argumentos y sentencias preparadas II

A continuación se deben indicar cuáles son los argumentos a utilizar en la sentencia anterior. Para ello utilizamos el método set<TIPO_X> de PreparedStatement que no devuelve nada.

```
set<TIPO_X>(posición, valor)
```

Donde:

- *Posición* indica el número de parámetro que es y *valor* el argumento que queremos darle.
- TIPO_X: puede ser String, int, Date, etc.

Ejemplos

```
sentenciaSQL.setInt(2, 100);
/*establece a 100 el argumento para cantidad (segundo ?) de la
sentencia preparada*/
sentenciaSQL.setInt("cant", 100);
```

La ejecución de una sentencia preparada se hace igual que con una normal, utilizando los métodos *executeQuery* y *executeUpdate* de PreparedStatement.

```
sent.executeQuery() → ResultSet

sent.executeUpdate() → int
```

Ejemplo

```
resultado = sentenciaSQL.executeQuery();
```

Código con paso de argumentos

Resumen

JDBC es un conjunto de clases que permiten ejecutar sentencias SQL dentro de una aplicación Java.
La estructura de la aplicación contiene los siguientes pasos:

1. Crear la conexión para lo cual debemos cargar el controlador del SGBD concreto que vayamos a utilizar (método registerDriver de la clase DriverManager) y establecer la conexión (método getConnection de la clase DriverManager).
2. Crear la sentencia SQL y ejecutarla. La sentencia puede ser normal ó preparada. Si es una sentencia normal debemos crear la sentencia (método createStatement de la clase Connection) y ejecutarla (método executeQuery ó executeUpdate de la clase Statement en función de si queremos una sentencia SELECT o otra de otro tipo). Si la sentencia es preparada los pasos son similares pero utilizando la clase PreparedStatement.
3. Manejar los datos recibidos. Para ello utilizamos los métodos next de la clase ResultSet (pasa a la siguiente fila de la tabla) ó get<tipo> de ResultSet (recupera el valor de una columna de esta fila).
4. Cerrar la conexión. Utilizando el método close de las clases ResultSet, Statement y Connection.