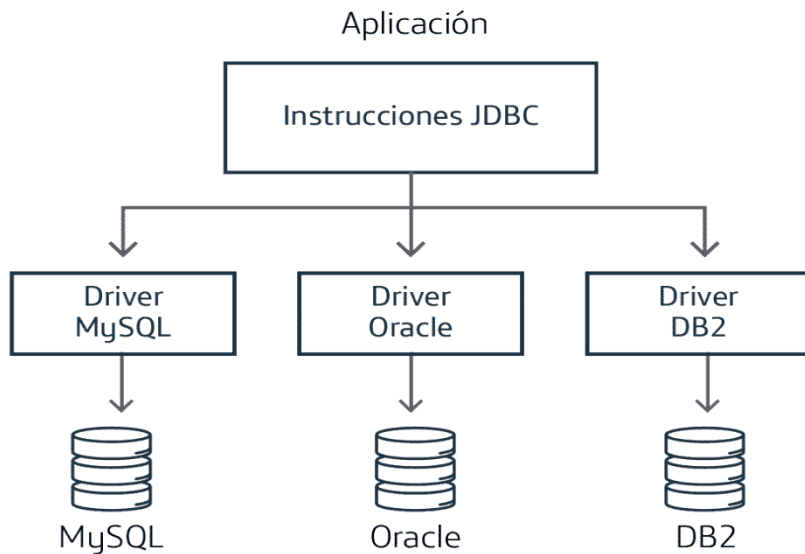


Acceso a datos con JDBC

Fundamentos JDBC

➤ Conjunto de clases e interfaces que permiten a una aplicación Java acceder a cualquier base de datos a través de un driver




El API JDBC

- Se encuentra en el paquete `java.sql`
- Entre las principales clases e interfaces están:
 - **DriverManager.** Proporciona un método estático para poder obtener conexiones contra la base de datos.
 - **Connection.** Representa una conexión contra la base de datos. La obtención de una conexión es un paso previo para poder operar contra la misma.
 - **Statement.** A través de este objeto podemos enviar consultas SQL a la base de datos.
 - **PreparedStatement.** Es una versión alternativa de **Statement**, con la que podemos precompilar consultas SQL antes de enviarlas a la BD.
 - **ResultSet.** Cuando una consulta devuelve resultados (caso de las instrucciones **Select**), la manipulación de los mismos se realiza a través de un objeto **ResultSet**

Pasos para operar contra una BD

➤ El proceso para realizar operaciones contra una BD es:

1. Carga del driver
 2. Establecimiento de la conexión con la BD
 3. Ejecución de la consulta SQL
 4. Manipulación de resultados, si procede
 5. Cierre de la conexión
- 

Carga del driver

➤ El driver es una librería .jar que se incluye dentro del classpath de la aplicación.

➤ Deberá ser cargado en memoria mediante:

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

➤ Desde JDBC 4 no es necesario realizar esta operación

Establecimiento de la conexión

- La conexión con la base de datos se establece a través del método `getConnection()` de `DriverManager`, que devuelve un objeto `Connection`:

```
Connection con=DriverManager.getConnection(String cadena, String user, String pwd);
```

```
Connection con=DriverManager.getConnection(String cadena, Properties prop);
```

- La cadena de conexión tiene el siguiente formato:

```
jdbc:<subprotocolo>:subname
```

- Donde *subprotocolo* es el tipo de base de datos y *subname* depende de la base de datos. Ejemplos:

```
jdbc:mysql://localhost:3306/mydata
```

```
jdbc:oracle:thin:@localhost:1521/servicedata
```

```
jdbc:db2://localhost:50000/datasets
```

Ejecución de consulta SQL

➤ Para ejecutar una consulta SQL se utilizan los objetos **Statement** o **PreparedStatement**:

Statement

```
String sql="insert ito tabla(col1,col2) values(40,'www')";  
Statement st=con.createStatement();  
st.execute(sql);
```

PreparedStatement

Posición de los
parámetros, el
primero es el 1

```
String sql="insert ito tabla(col1,col2) values(?,?)";  
PreparedStatement st=con.prepareStatement(sql);  
st.setInt(1,40);  
st.setString(2,"www");  
st.execute();
```

➤ En el caso de una consulta de selección, se debe obtener el objeto **ResultSet** para acceder a los registros:

```
String sql="Select * from data";  
Statement st=con.createStatement();  
st.execute(sql);  
ResultSet rs=st.getResultSet();
```

```
String sql="Select * from data";  
Statement st=con.createStatement();  
ResultSet rs=st.executeQuery(sql);
```

Manipulación de resultados

➤ Para acceder a los registros empleamos los siguientes métodos de ResultSet:

- `boolean next()`. Se desplaza al siguiente registro, si no hay ninguno devolverá false:

```
//recorre todas las filas
while(rs.next()){
    ...
}
```

- `xxx getXxx(int col)`. Métodos para obtener el valor de la columna indicada. La posición de la primera columna es la 1. xxx es el nombre del tipo Java (`getInt`, `getString`,...)
- `xxx getXxx(String col)`. Igual que el anterior, utilizando el nombre de la columna.

Cierre de la conexión

➤ Las conexiones deben cerrarse cuando no van a ser utilizadas:

- Utilizando el método `close()` de `Connection`:

```
try{
    Connection con=....
    ...
}
finally{
    con.close();
}
```

- Mediante un `try` con recursos:

```
try(Connection con=...){
    ...
} //se cierra automáticamente al abandonar el try
```