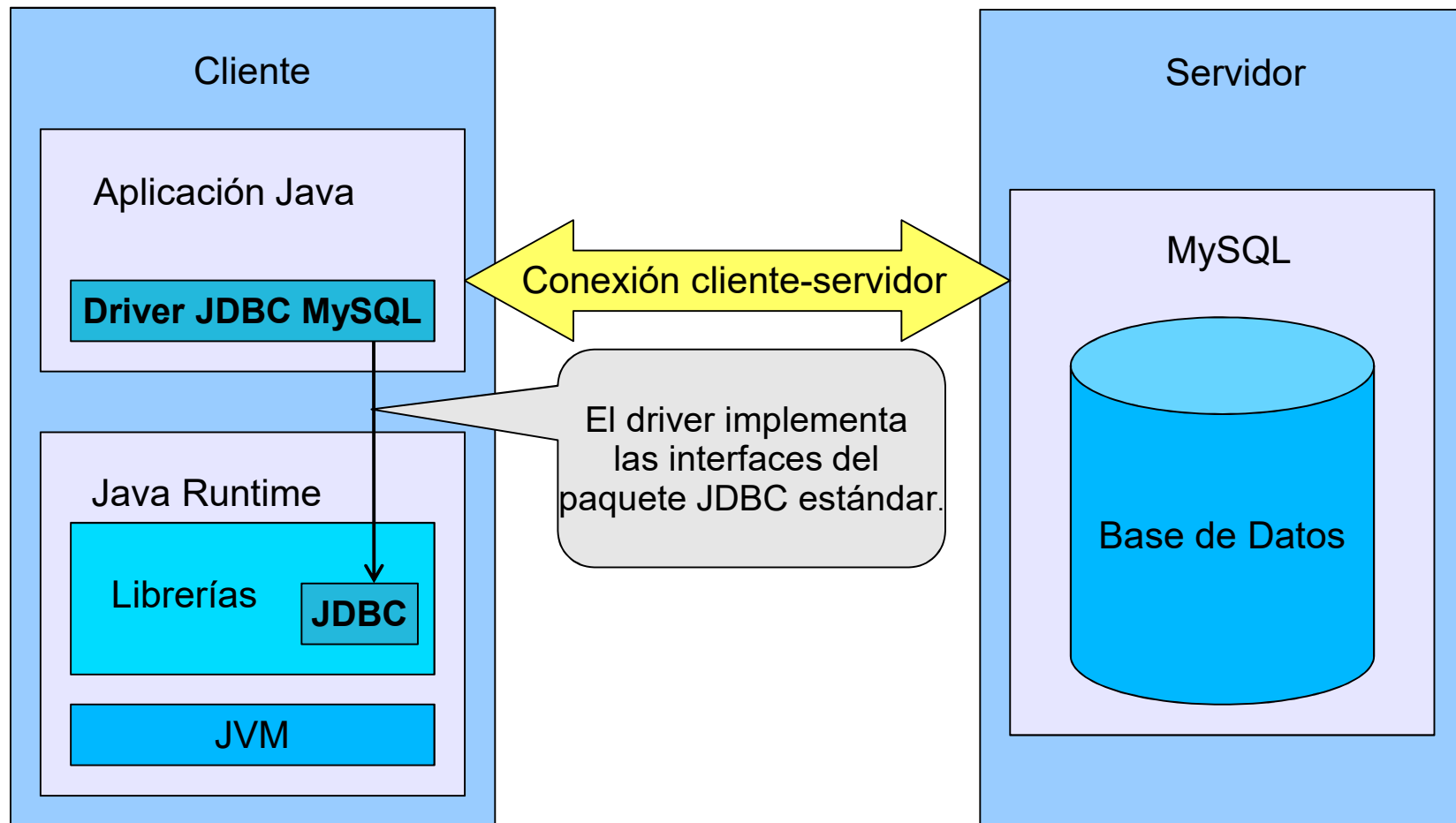
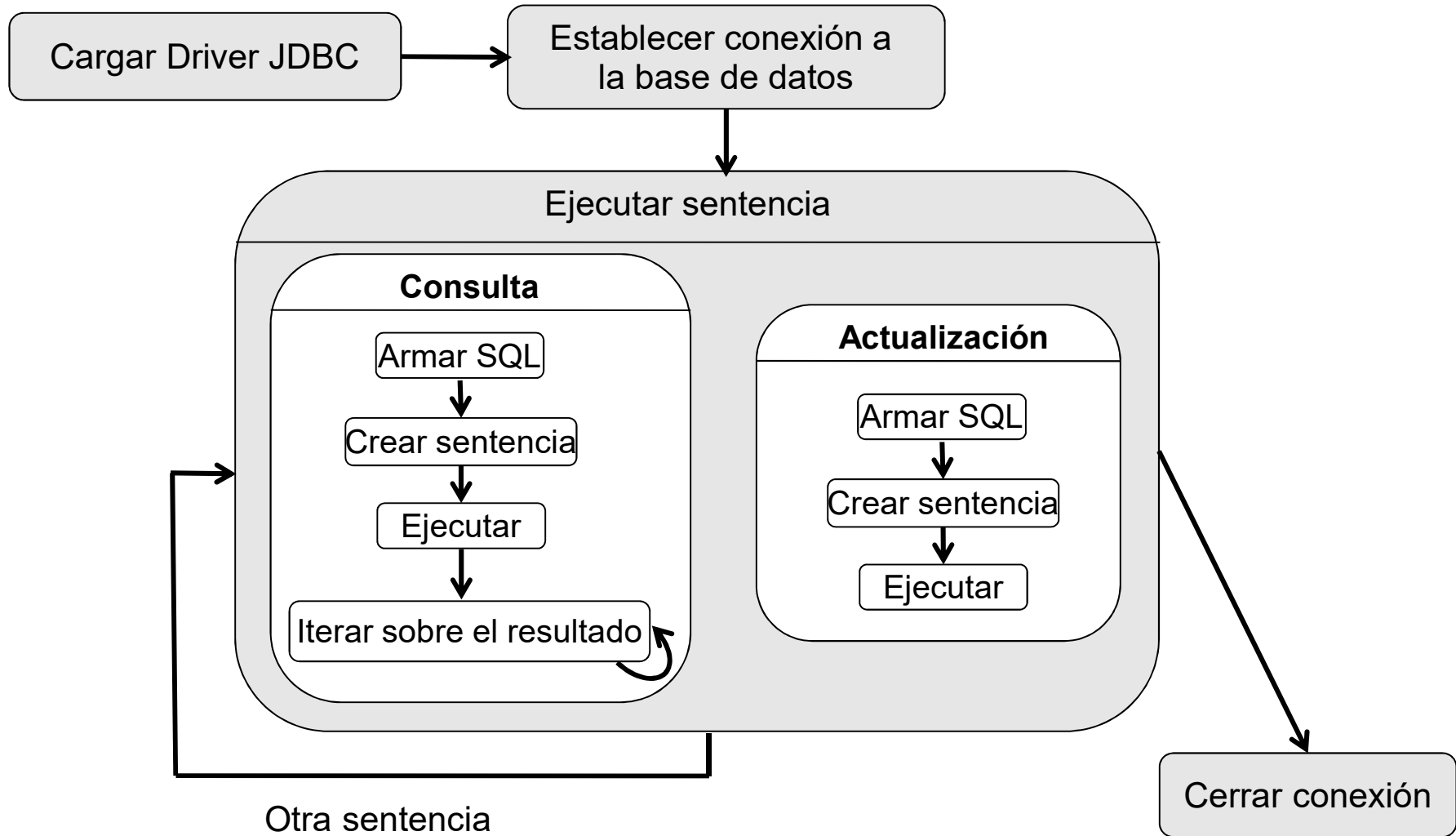


Java y MySQL

. JDBC: Java DataBase Connectivity



Ciclo de ejecución



Clase DriverManager (Paquete **java.sql**)

- **DriverManager**: clase estática (no requiere instanciación de objetos) que administra los drivers jdbc disponibles para iniciar conexiones.
 - **Connection getConnection(String url)**: intenta iniciar una conexión a una base de datos según los parámetros especificados en el url. Generalmente el string de conexión tiene el siguiente formato:
jdbc:<driver>:<propiedades de la conexión>
Para MySQL:
jdbc:mysql://<servidor>:<puerto>/<base_datos>?<parametros>
Por ejemplo: jdbc:mysql://localhost:3306/batallas?&user=barco&password=pwbarco
 - **Connection getConnection(String url, String usuario, String clave)**: idem al anterior pero por compatibilidad y seguridad, el usuario y la clave de acceso, son parámetros individuales.
 - **setLoginTimeout(int segundos)**: configura la cantidad de segundos de espera para intentar establecer la próxima conexión a una base de datos.

Clase Connection (Paquete **java.sql**)

- **Connection**: interfaz para implementar una sesión cliente-servidor con una base de datos.
 - **Statement** **createStatement()**: crea una nueva sentencia para ejecutar código SQL en forma directa en el servidor a través de la conexión.
 - **PreparedStatement** **prepareStatement(String sql)**: crea una sentencia *preparada* con una estructura predeterminada dada por parámetros, para luego enviar los datos efectivos.
 - **boolean isValid(int timeout)**: verifica que la conexión está abierta y disponible para ejecutar una operación. Es necesaria para determinar si la conexión de red aun permanece activa desde la ejecución del último SQL.
 - **close()**: cierra la conexión y libera los recursos utilizados.
 - **setAutoCommit(boolean autoCommit)**, **commit()**, **rollback()**: utilizados para el manejo de transacciones en la conexión actual.

Conexión JDBC a MySQL

```
// Intento de conexión a una base de datos
String servidor = "localhost:3306";
String baseDatos = "batallas";
String usuario = "admin_batallas";
String clave = "pwbatallas";
String url = "jdbc:mysql://" + servidor + "/" + baseDatos +
            "?serverTimezone=America/Argentina/Buenos_Aires";

java.sql.Connection cnx;
try
{
    cnx = java.sql.DriverManager.getConnection(url, usuario, clave);
}
catch (java.sql.SQLException ex)
{
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
}
```

Clase Statement (Paquete **java.sql**)

- **Statement**: se utiliza para ejecutar una sentencia SQL en base a un string estático, ya sea un comando o una consulta.
 - **boolean execute(String sql)**: ejecuta cualquier tipo de SQL. Si es una consulta se debe recuperar el resultado mediante el método **ResultSet getResultSet()**;
 - **int executeUpdate(String sql)**: sólo para comandos de actualización de datos (insert, delete, update) o configuración dinámica de la sesión (transacciones, concurrencia, etc).
 - **ResultSet executeQuery(String sql)**: sólo para consultas que retornan un resultado en filas o registros.
 - **addBatch(String sql), int[] executeBatch(), clearBatch()**: permiten ejecutar una secuencia de comandos enviados en un lote.
 - **void setQueryTimeout(int seconds)**: impone un límite de espera por el resultado de la ejecución de la sentencia.
 - **close()**: cierra la sentencia liberando los recursos utilizados.

Clase ResultSet (Paquete `java.sql`)

- **ResultSet**: contiene el conjunto resultado de una consulta SQL, estructurado en filas y columnas, con el comportamiento de un iterador.
 - **boolean next()**: avanza el índice interno del iterador a la próxima fila. Retorna false si no hay mas filas.
 - **String getString(int columnIndex)**, **String getString(String columnLabel)**: permiten recuperar los valores de las columnas como un String, según su posición en la fila (la primer columna es 1) o mediante su nombre respectivamente.
 - **int getInt(...)**, **long getLong(...)**, **float getFloat(...)**, **double getDouble(...)**, **boolean getBoolean(...)**, **Date getDate(...)**, **Timestamp getTimestamp(...)**, **Object getObject(...)**, etc...: una función para cada tipo de dato (java) de las columnas.
 - **boolean wasNull()**: verifica si el último valor recuperado de una columna correspondía al valor NULL de SQL.
 - **boolean previous()**, **boolean first()**, **boolean last()**, **boolean absolute(int row)**, **boolean relative(int rows)**: funciones para navegar en el conjunto resultado.
 - **ResultSetMetaData getMetaData()**: para recuperar los meta-datos (cantidad de columnas, tipos, ...) del conjunto resultado y de las columnas.

Consultas SQL

```
try
{
    // Se crea una sentencia jdbc para realizar la consulta
    java.sql.Statement stmt = cnx.createStatement();

    // Se prepara el string SQL de la consulta
    String sql = "SELECT nombre_barco, id, capitan FROM barcos";

    // Se ejecuta la sentencia y se recibe un resultado
    java.sql.ResultSet rs = stmt.executeQuery(sql);

    // Se recorre el resultado
    while (rs.next())
    {
        String nombreBarco = rs.getString("nombre_barco");
        int id = rs.getInt("id");
        String capitan = rs.getString("capitan");
    }
    rs.close();
    stmt.close();
}
catch (java.sql.SQLException ex) {}
```


Clase ResultSetMetaData (Paquete `java.sql`)

- **ResultSetMetaData**: permite obtener los tipos y propiedades de las columnas de un conjunto resultado (ResultSet):
 - **int getColumnCount()**: cantidad de columnas en el resultado.
 - **String getColumnLabel(int column)**, **String getColumnName(int column)**: recuperar el nombre modificado o el nombre real de una columna respectivamente.
 - **String getColumnClassName(int column)**: recuperar la clase de Java determinada como predefinida para el tipo de dato de una columna.
 - **int getColumnType(int column)**, **String getColumnTypeNames(int column)**: recuperar el tipo de dato SQL estándar y SQL específico de una columna respectivamente.
 - **int isNullable(int column)**, **boolean isAutoIncrement(int column)**: propiedades particulares de una columna.
 - **boolean isSearchable(int column)**: determina si es posible ejecutar un filtro en el WHERE de una consulta sobre la columna indicada.

Actualización de datos

- Comandos o sentencias de manipulación de datos (insert, update, delete) que no retornan un resultado.

```
try
{ // Se crea una sentencia jdbc para realizar la consulta
    java.sql.Statement stmt = cnx.createStatement();
    // Se prepara el string SQL de la inserción
    String sql = "INSERT INTO barcos (nombre_barco, id, capitan) " +
                 "VALUES ('Bismark', 22, 'Ernst Lindeman')";

    // Se ejecuta la inserción
    stmt.execute(sql);
    // Se retornan los recursos utilizados cerrando la sentencia
    stmt.close();
}
catch (java.sql.SQLException ex)
{ System.out.println("Mensaje: " + ex.getMessage()); // Mensaje
  //retornado por MySQL
  System.out.println("Código: " + ex.getErrorCode()); // Código de
  //error de MySQL
  System.out.println("SQLState: " + ex.getSQLState()); // Código de
  //error del SQL standart
}
```

Sentencias Preparadas

- Se utilizan cuando una misma sentencia (consulta o actualización) debe ejecutarse repetidamente con la misma estructura pero distintos valores.
- Más comunmente usado en inserciones de registros en masa.
- Acelera la ejecución al evitar la interpretación del SQL de manera individual. Una vez preparada la sentencia en el servidor, sólo se envían los datos efectivos.
- El método puede interpretarse como una pre-compilación de las sentencias para un posterior uso repetitivo. No sólo se ahorran ciclos de cpu en el servidor, sino también ancho de banda para la transmisión de comandos y datos.

```
String sql = "INSERT INTO barcos (nombre_barco, id, capitan)
VALUES (?, ?, ?)";
// Se crea un sentencia preparada
java.sql.PreparedStatement stmt = cnx.prepareStatement(sql);
// Se ligam los parámetros efectivos
stmt.setString(1, "Bismark");
stmt.setInt(2, 22);
stmt.setString(3, "Ernst Lindeman");
// Se ejecuta la inserción
stmt.executeUpdate();
// se cierra la sentencia
stmt.close();
```

Clase JTable (paquete `javax.swing.table`)

- Permite mostrar graficamente tablas con datos, permitiendo opcionalmente al usuario editar los datos.

The Header contains
Column labels

First Name	Last Name	Sport	# of Years	Vegetarian
Kathy	Smith	Snowboarding	5	<input type="checkbox"/>
John	Doe	Rowing	3	<input checked="" type="checkbox"/>
Sue	Black	Knitting	2	<input type="checkbox"/>
Jane	White	Speed reading	20	<input checked="" type="checkbox"/>
Joe	Brown	Swimming	10	<input type="checkbox"/>

Each Cell displays
a data item

Each Column displays
one type of data

- Cada tabla `JTable` usa un objeto `TableModel` para manejar y almacenar los datos.
- Si no se especifica ningun modelo de tabla, `JTable` utiliza por defecto el modelo `DefaultTableModel` que almacena los datos como vector de vectores.
- Para crear un modelo de tabla se debe implementar la interface `TableModel`. Generalmente se implementa extendiendo la clase `DefaultTableModel` o `AbstractTableModel`.

(Ver ejemplos de su uso en proyecto BatallasMVC)