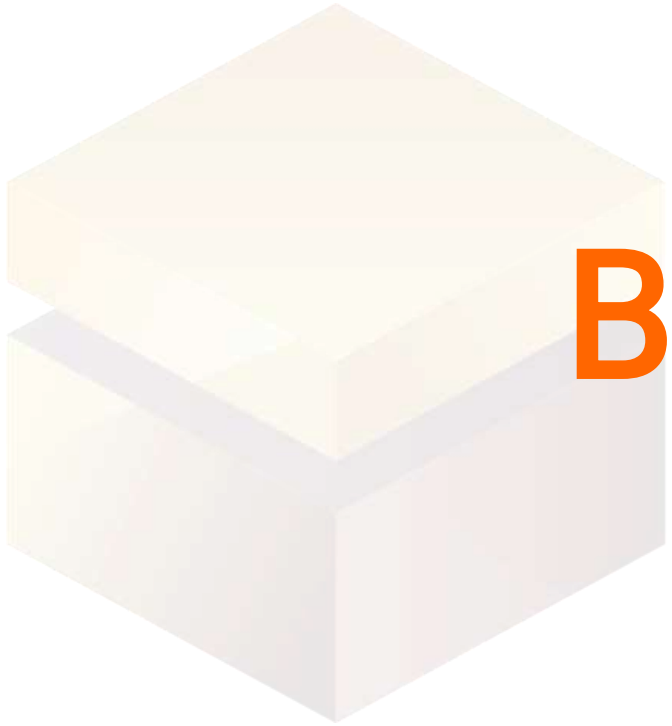


BUỔI 4 - JDBC

CYBERSOFT.EDU.VN

Nội dung

❑ JDBC

- ✓ Giới thiệu về JDBC.
- ✓ Cấu hình JDBC.
- ✓ Tạo lớp kết nối CSDL sử dụng JDBC.
- ✓ Truy vấn dữ liệu sử dụng JDBC.

❑ Mô hình MVC

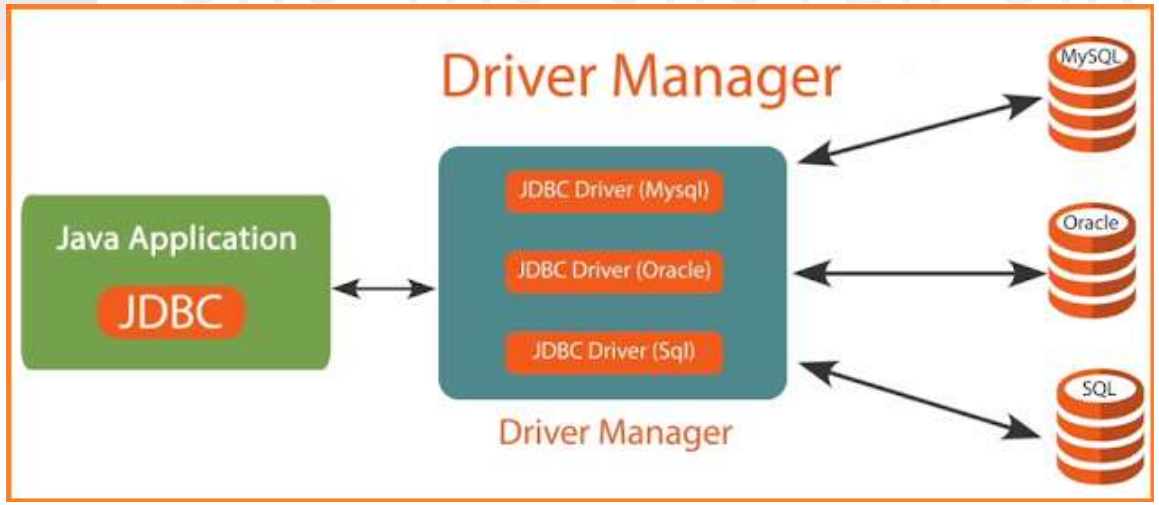
- ✓ Tổng quan về mô hình MVC.
- ✓ Cấu trúc dự án theo mô hình MVC.

JDBC

- **JDBC** (Java Database Connectivity) là một **thư viện chuẩn** được viết bằng ngôn ngữ Java, **giúp ứng dụng Java thực hiện kết nối, làm việc với CSDL**.
- Cho phép ứng dụng thực hiện các thao **tác truy xuất**, **update dữ liệu** với CSDL quan hệ bằng việc **sử dụng các câu lệnh SQL**.
  - ✓ Tạo kết nối Database
  - ✓ Tạo câu lệnh SQL
  - ✓ Thực thi câu lệnh SQL
  - ✓ Sửa và xem kết quả trên Database

JDBC

- **DriverManager**: Quản lý danh sách các Driver.
- **Driver**: Liên kết các kết nối tới CSDL.
- **Connection**: Để quản lý các kết nối.
- **Statement**, **PreparedStatement**, **CallableStatement**: Dùng để thực thi các câu lệnh SQL.
- **ResultSet**: Biểu diễn tập kết quả trong CSDL khi sử dụng câu lệnh SELECT.
- **SQLExeption**: Lớp xử lý lỗi ngoại lệ.



Interface thực thi truy vấn

- **JDBC** cung cấp ba dạng interface để gửi câu lệnh SQL đến cơ sở dữ liệu và thực thi truy vấn là **PreparedStatement**, **CallableStatement** và **Statement**.
  - ✓ **Statement** được sử dụng để **thực thi các truy vấn SQL thông thường**, chủ yếu được sử dụng cho các câu lệnh không có tham số như **SELECT**, **CREATE**, **ALTER**, **DROP**, ...
  - ✓ **PreparedStatement** chuyên dùng để **thực hiện các truy vấn được tham số hóa**. Chúng được biên dịch và lưu trữ vào cuối cơ sở dữ liệu, do đó khá nhanh để thực hiện lặp lại.
  - ✓ **CallableStatement** dùng để **thực thi hoặc gọi sử dụng stored procedures**.

PreparedStatement

- **PreparedStatement** là interface cung cấp giao diện để **gửi câu lệnh** và **thực hiện truy vấn** đến database.
- **PreparedStatement** sử dụng **trình giữ chỗ có ký hiệu là dấu hỏi (?)** hay nói cách khác là tham số hóa câu truy vấn. Lập trình viên sẽ truyền dữ liệu dạng tham số vào cho các dấu hỏi (?).
- Khi bạn sử dụng **PreparedStatement**, truy vấn được biên dịch lần đầu tiên nhưng sau đó nó được **lưu trữ tại máy chủ cơ sở dữ liệu**, làm cho lần chạy tiếp theo nhanh hơn.

```
String query = "DELETE FROM roles WHERE id = ?";
PreparedStatement statement = conn.prepareStatement(query);
statement.setInt(1, id);
```

Bảng so sánh

Statement	PreparedStatement	CallableStatement
Nó được sử dụng để thực hiện các truy vấn SQL bình thường. (truy vấn 1 lần)	Được sử dụng để thực hiện các truy vấn SQL được tham số hóa hoặc động. (Truy vấn lặp và thay đổi giá trị)	Nó được sử dụng sử dụng để gọi stored procedures.
Nó được ưa thích khi một truy vấn SQL cụ thể chỉ được thực hiện một lần.	Nó được ưa thích khi một truy vấn cụ thể được thực hiện nhiều lần.	Nó được ưa thích khi gọi stored procedures và functions.
Giao diện này chủ yếu được sử dụng cho các câu lệnh DDL như CREATE, SELECT khôn có tham số, ALTER, DROP, v.v.	Nó được sử dụng cho bất kỳ loại truy vấn SQL nào sẽ được thực hiện nhiều lần.	Nó được sử dụng sử dụng để gọi stored procedures.
Hiệu suất của giao diện này rất thấp.	Hiệu suất của giao diện này tốt hơn giao diện Statement (khi được sử dụng để thực hiện nhiều truy vấn giống nhau).	Hiệu suất của giao diện này cao.

Cách hoạt động của JDBC

❑ **Các bước thực JDBC thi**

1. Tạo kết nối tới Database.
2. Gửi SQL query đến database sử dụng JDBC driver tương ứng.
3. JDBC driver kết nối đến database.
4. Thực thi câu lệnh query để lấy kết quả trả về (số bản ghi lấy được, số bản ghi được update/delete).
5. Gửi dữ liệu đến ứng dụng thông qua Driver Manager.
6. Xử lý dữ liệu trả về.
7. Đóng (giải phóng) kết nối đến database.

Download thư viện

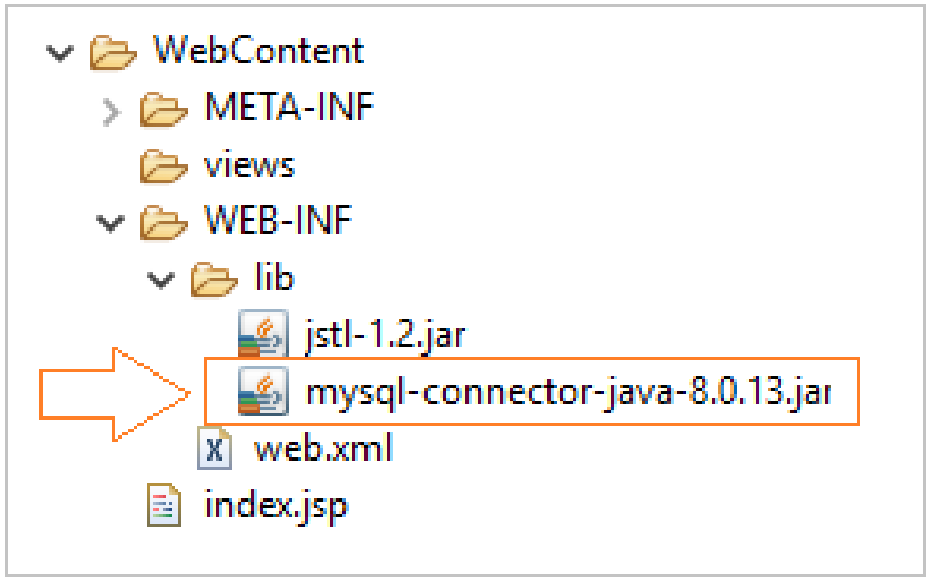
- Download https://mvnrepository.com/artifact/mysql/mysql-connector-java/8.0.13

Tải thư viện

License	GPL 2.0
Categories	MySQL Drivers
Organization	Oracle Corporation
HomePage	http://dev.mysql.com/doc/connector-j/en/
Date	(Sep 27, 2018)
Files	jar (2.0 MB) View All
Repositories	Central
Used By	3,563 artifacts

Tải về

Kéo vào dự án



Tạo lớp kết nối database

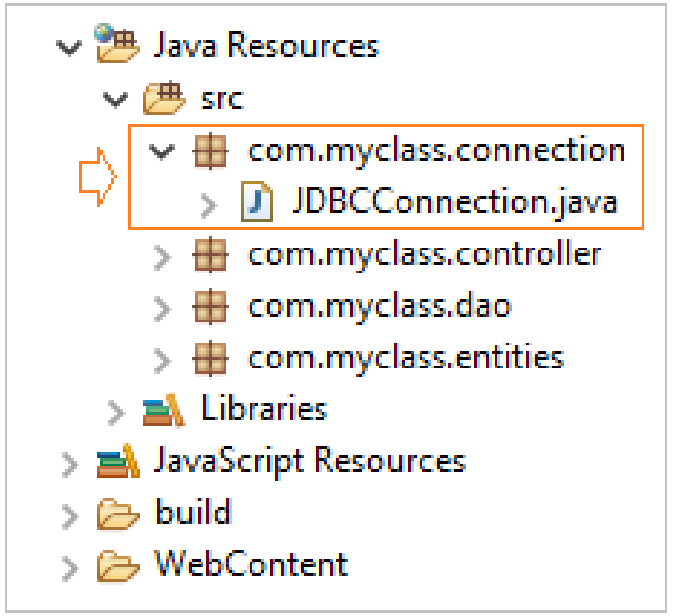
Lớp kết nối database

```java
public class JDBCConnection {

    // database
    private final static String url = "jdbc:mysql://localhost:3306/crm_app";
    private final static String username = "root"; // Tên đăng nhập mysql
    private final static String password = "123456"; // Mật khẩu đăng nhập

    public static Connection getConnection() {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            return DriverManager.getConnection(url, username, password);
        } catch (ClassNotFoundException e) {
            System.out.println("Không tìm thấy Driver!");
            e.printStackTrace();
        }
        catch (SQLException e) {
            System.out.println("Không tìm thấy db!");
            e.printStackTrace();
        }
        return null;
    }
}
```

Thư mục chứa



Truy vấn lấy dữ liệu

```java
List<Role> roles = new ArrayList<Role>();
try {
    // Mở kết nối đến database
    Connection conn = JDBCConnection.getConnection();
    String query = "SELECT * FROM roles";
    // Tạo truy vấn SQL đến database sử dụng PreparedStatement
    PreparedStatement statement = conn.prepareStatement(query);
    ResultSet res = statement.executeQuery(); // Thực thi truy vấn
    while(res.next()) {
        Role role = new Role();
        role.setId(res.getInt("id"));
        role.setName(res.getString("name"));
        role.setDescription(res.getString("description"));
        roles.add(role);
    }
    conn.close(); // Đóng kết nối
}
catch (Exception e) {
    e.printStackTrace();
}
```

Truy vấn thêm mới dữ liệu

```
try {
    // Mở kết nối đến database
    Connection conn = JDBCConnection.getConnection();
    String query = "INSERT INTO roles (name, description) VALUES (?, ?)";
    // Tạo truy vấn SQL đến database sử dụng PreparedStatement
    PreparedStatement statement = conn.prepareStatement(query);
    statement.setString(1, "ROLE_TEST"); // Truyền giá trị vào dấu hỏi thứ nhất
    statement.setString(2, "Test thêm mới"); // Truyền giá trị vào dấu hỏi thứ hai
    int result = statement.executeUpdate();
    conn.close(); // Đóng kết nối
}
catch (Exception e) {
    e.printStackTrace();
}
```

Xây dựng Servlet

- Áp dụng kiến thức ở bài trước, tận dụng **multi url** chúng ta sẽ xây dựng servlet như sau:

✓ **Bước 1**: Tạo RoleServlet với các url tương ứng có thể truy cập.

✓ **Bước 2**: Dựa vào url xây dựng các phương thức tương ứng thực hiện thêm,sửa, xóa và lấy dữ liệu.

RoleServlet

```java
@WebServlet(name = "RoleServlet",
    urlPatterns = {"/role", "/role/add", "/role/edit", "/role/delete" })
public class RoleServlet extends HttpServlet{
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) ..
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) ..

    private void getList(HttpServletRequest req, HttpServletResponse resp) ..
    private void getAdd(HttpServletRequest req, HttpServletResponse resp) ..
    private void getEdit(HttpServletRequest req, HttpServletResponse resp) ..
    private void getDelete(HttpServletRequest req, HttpServletResponse resp) ..

    private void postAdd(HttpServletRequest req, HttpServletResponse resp) ..
    private void postEdit(HttpServletRequest req, HttpServletResponse resp) ..
}
```

Hàm doGet()

```java
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
    String action = req.getServletPath();
    switch (action) {
    case "/role":
        getList(req, resp);
        break;
    case "/role/add":
        getAdd(req, resp);
        break;
    case "/role/edit":
        getEdit(req, resp);
        break;
    case "/role/delete":
        getDelete(req, resp);
        break;
    default:
        break;
    }
}
```

Hàm doPost()

```
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
    String action = req.getServletPath();
    switch (action) {
    case "/role/add":
        postAdd(req, resp);
        break;
    case "/role/edit":
        postEdit(req, resp);
        break;
    default:
        break;
    }
}
```

Hàm getList()

```
private void getList(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

    List<Role> roles = new ArrayList<Role>();
    try {
        // Mở kết nối đến database
        Connection conn = JDBCConnection.getConnection();
        String query = "SELECT * FROM roles";
        // Tạo truy vấn SQL đến database sử dụng PreparedStatement
        PreparedStatement statement = conn.prepareStatement(query);
        ResultSet res = statement.executeQuery(); // Thực thi truy vấn
        while(res.next()) {
            Role role = new Role();
            role.setId(res.getInt("id"));
            role.setName(res.getString("name"));
            role.setDescription(res.getString("description"));
            roles.add(role);
        }
        conn.close(); // Đóng kết nối
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    req.getRequestDispatcher("/views/role/index.jsp").forward(req, resp);
}
```

Hàm getAdd()

```
private void getAdd(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

    req.setAttribute("role", new Role());
    req.getRequestDispatcher("/views/role/add.jsp").forward(req, resp);
}
```

Hàm getEdit()

```
private void getEdit(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
    int id = Integer.parseInt(req.getParameter("id"));
    Role role = new Role();
    try {
        // Mở kết nối đến database
        Connection conn = JDBCConnection.getConnection();
        String query = "SELECT * FROM roles WHERE id = ?";
        // Tạo truy vấn SQL đến database sử dụng PreparedStatement
        PreparedStatement statement = conn.prepareStatement(query);
        statement.setInt(1, id);
        ResultSet res = statement.executeQuery(); // Thực thi truy vấn
        while(res.next()) {
            role.setId(res.getInt("id"));
            role.setName(res.getString("name"));
            role.setDescription(res.getString("description"));
        }
        conn.close(); // Đóng kết nối
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    req.setAttribute("role", role);
    req.getRequestDispatcher("/views/role/edit.jsp").forward(req, resp);
}
```

Hàm getDelete()

```
private void getDelete(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
    int id = Integer.parseInt(req.getParameter("id"));
    try {
        // Mở kết nối đến database
        Connection conn = JDBCConnection.getConnection();
        String query = "DELETE FROM roles WHERE id = ?";
        // Tao truy vấn SQL đến database sử dụng PreparedStatement
        PreparedStatement statement = conn.prepareStatement(query);
        statement.setInt(1, id);
        statement.executeUpdate(); // Thực thi truy vấn
        conn.close(); // Đóng kết nối
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    resp.sendRedirect(req.getContextPath() + "/role");
}
```

Hàm postAdd()

```
private void postAdd(HttpServletRequest req, HttpServletResponse resp)
        throws IOException {

    String name = req.getParameter("name");
    String description = req.getParameter("description");
    try {
        // Mở kết nối đến database
        Connection conn = JDBCConnection.getConnection();
        String query = "INSERT INTO roles (name, description) VALUES (?, ?)";
        // Tạo truy vấn SQL đến database sử dụng PreparedStatement
        PreparedStatement statement = conn.prepareStatement(query);
        statement.setString(1, name); // Truyền giá trị vào dấu hỏi thứ nhất
        statement.setString(2, description); // Truyền giá trị vào dấu hỏi thứ hai
        statement.executeUpdate();
        conn.close(); // Đóng kết nối
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    resp.sendRedirect(req.getContextPath() + "/role");
}
```

Hàm postEdit()

```
private void postEdit(HttpServletRequest req, HttpServletResponse resp)
        throws IOException {

    int id = Integer.parseInt(req.getParameter("id"));
    String name = req.getParameter("name");
    String description = req.getParameter("description");

    try {
        // Mở kết nối đến database
        Connection conn = JDBCConnection.getConnection();
        String query = "UPDATE roles name = ?, description = ? WHERE id = ?";
        // Tạo truy vấn SQL đến database sử dụng PreparedStatement
        PreparedStatement statement = conn.prepareStatement(query);
        statement.setString(1, name); // Truyền giá trị vào dấu hỏi thứ nhất
        statement.setString(2, description); // Truyền giá trị vào dấu hỏi thứ hai
        statement.setInt(3, id);
        statement.executeUpdate();
        conn.close(); // Đóng kết nối
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    resp.sendRedirect(req.getContextPath() + "/role");
}
```

Tách lớp truy vấn database

- Trong các dự án thực tế, để dễ dàng bảo trì và nâng cấp hệ thống thì chúng ta sẽ **tách phần truy vấn database ra thành các class riêng**.

❖ Các bước thực hiện:

✓ **Bước 1**: Tạo package **com.myclass.dao** (package này sẽ chứa các class chứa các nghiệp vụ truy vấn database).

✓ **Bước 2**: Tạo lớp **RoleDao**, khai báo các phương thức thêm, sửa, xóa, lấy dữ liệu, sau đó chuyển code truy vấn database từ servlet đưa vào các phương thức này.

✓ **Bước 3**: Khởi tạo một đối tượng **RoleDao** ở servlet và gọi các phương thức của đối tượng vừa tạo để sử dụng.

Hàm findAll()

```
public List<Role> findAll(){
    List<Role> roles = new ArrayList<Role>();
    try {
        Connection conn = JDBCConnection.getConnection();
        String query = "SELECT * FROM roles";
        PreparedStatement statement = conn.prepareStatement(query);
        ResultSet res = statement.executeQuery();
        while(res.next()) {
            Role role = new Role();
            role.setId(res.getInt("id"));
            role.setName(res.getString("name"));
            role.setDescription(res.getString("description"));
            roles.add(role);
        }
        conn.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return roles;
}
```

Hàm findById()

```java
public Role findById(int id) {
    Role role = new Role();
    try {
        Connection conn = JDBCConnection.getConnection();
        String query = "SELECT * FROM roles WHERE id = ?";
        PreparedStatement statement = conn.prepareStatement(query);
        statement.setInt(1, id);
        ResultSet res = statement.executeQuery();
        while(res.next()) {
            role.setId(res.getInt("id"));
            role.setName(res.getString("name"));
            role.setDescription(res.getString("description"));
        }
        conn.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return role;
}
```

Hàm insert()

```java
public int insert(Role role) {
    int result = -1;
    try {
        Connection conn = JDBCConnection.getConnection();
        String query = "INSERT INTO roles(name, description) VALUES (?, ?)";
        PreparedStatement statement = conn.prepareStatement(query);
        statement.setString(1, role.getName());
        statement.setString(2, role.getDescription());
        result = statement.executeUpdate();
        conn.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return result;
}
```

Hàm update()

```
public int update(Role role) {
    int result = -1;
    try {
        Connection conn = JDBCConnection.getConnection();
        String query = "UPDATE roles SET name = ?, description = ? WHERE id = ?";
        PreparedStatement statement = conn.prepareStatement(query);
        statement.setString(1, role.getName());
        statement.setString(2, role.getDescription());
        statement.setInt(3, role.getId());
        result = statement.executeUpdate();
        conn.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return result;
}
```

Hàm delete()

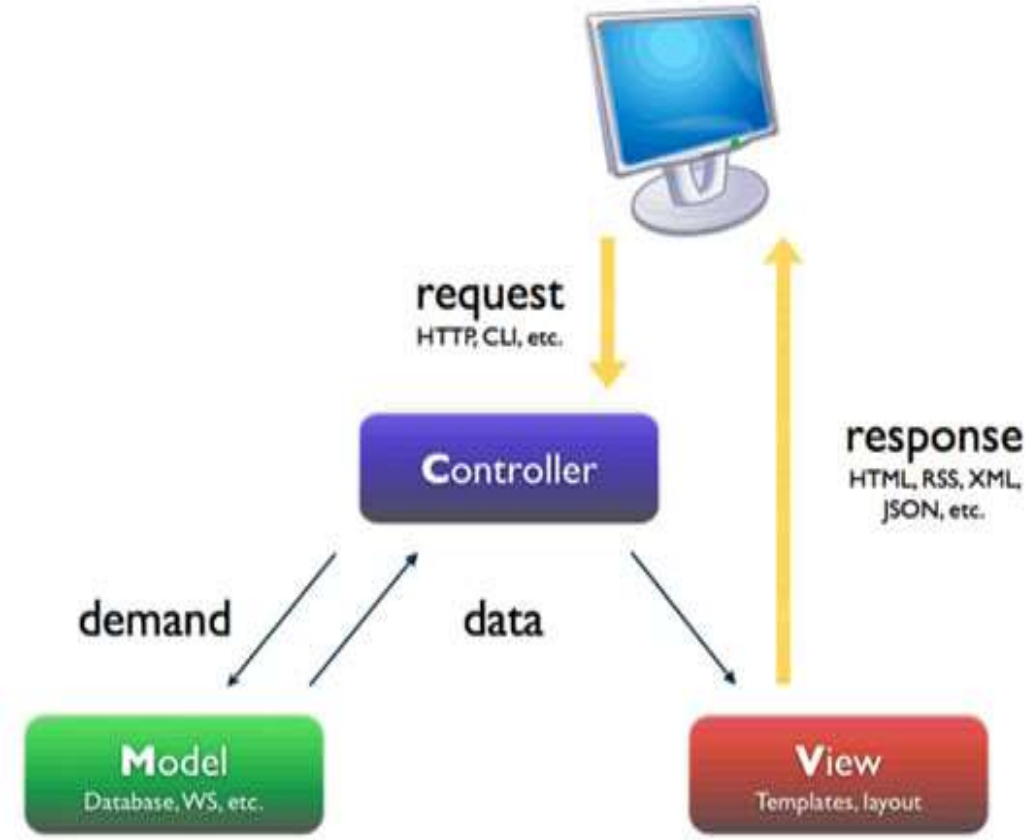
```
public int delete(int id) {
    int result = -1;
    try {
        Connection conn = JDBCConnection.getConnection();
        String query = "DELETE FROM roles WHERE id = ?";
        PreparedStatement statement = conn.prepareStatement(query);
        statement.setInt(1, id);
        result = statement.executeUpdate();
        conn.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return result;
}
```

Mô hình MVC

- Mô hình **MVC** (Model - View – Controller) là một **kiến trúc phần mềm** hay **mô hình thiết kế** được sử dụng trong kỹ thuật phần mềm.
- Mô hình **MVC** giúp lập trình viên **tách ứng dụng thành các phần khác nhau** và **mỗi thành phần có nhiệm vụ riêng biệt** độc lập với các thành phần khác.

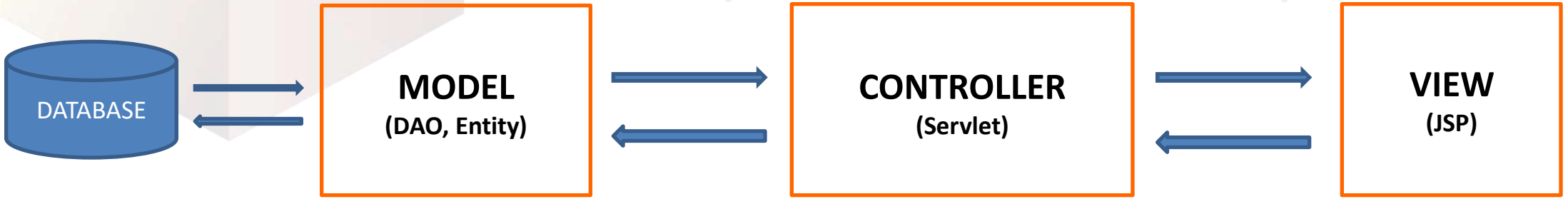
❖ **3 thành phần chính**:

- Model
- View
- Controller



Mô hình MVC

- **Model**: Tương tác và truy xuất dữ liệu đến database (cơ sở dữ liệu), xử lý các logic nghiệp vụ (business).
- **View**: Giao diện mà người dùng có thể nhìn thấy, thường thì view chỉ có một nhiệm vụ duy nhất là hiển thị dữ liệu.
- **Controller**: Nó có nhiệm vụ điều khiển, điều hướng và tương tác giữa tầng Model và View.



Tổng kết

❑ JDBC

- ✓ Giới thiệu về JDBC.
- ✓ Cấu hình JDBC.
- ✓ Tạo lớp kết nối CSDL sử dụng JDBC.
- ✓ Truy vấn dữ liệu sử dụng JDBC.

❑ Mô hình MVC

- ✓ Tổng quan về mô hình MVC.
- ✓ Cấu trúc dự án theo mô hình MVC.