LAPORAN PRAKTIKUM PBO

PEKAN 3 CONECT ECLIPSE KE DATABASE PHP

DAN TUGAS 3

Oleh :

ARYA PRATAMA HENDRI

NIM 2411533008

MATA KULIAH PBO

DOSEN PENGAMPU : NURFIAH , S.ST,M.KOM. . ,

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

PADANG, SEPTEMBER 2025

1. Tujuan

Tujuan praktikum ini adalah agar mahasiswa mampu:

1. Membuat tabel user pada database MySQL.
2. Membuat koneksi Java dengan database MySQL.
3. Membuat tampilan GUI CRUD user menggunakan JFrame.
4. Membuat dan mengimplementasikan **interface** dalam Java.
5. Membuat fungsi **DAO (Data Access Object)** dan menggunakannya pada aplikasi.
6. Mengimplementasikan fungsi **CRUD** (Create, Read, Update, Delete) dengan konsep Pemrograman Berorientasi Objek (OOP).

---

2. Alat dan Bahan

- Laptop/komputer dengan **JDK** dan **Eclipse IDE**.
- **XAMPP** (Apache + MySQL).
- **MySQL Connector/J** (driver untuk koneksi Java ke MySQL).
- Database **phpMyAdmin**.

---

3. Dasar Teori

1. **XAMPP**
   Merupakan paket software open source yang terdiri dari Apache, MySQL, PHP, dan Perl. Digunakan untuk development berbasis localhost.
2. **MySQL**
   Sistem manajemen database relasional (RDBMS) open source yang digunakan untuk menyimpan, mengelola, dan mengambil data dalam bentuk tabel.
3. **MySQL Connector/J**
   Driver yang menghubungkan aplikasi Java dengan MySQL. Berfungsi membuka koneksi, mengirimkan query SQL, menerima hasil, dan menutup koneksi.
4. **DAO (Data Access Object)**
   Objek yang menyediakan antarmuka abstrak terhadap method database.
   Tujuannya:
   - Memisahkan logika akses data dengan logika bisnis.
   - Memudahkan pemeliharaan dan meningkatkan reusabilitas.
5. **Interface Java**
   Kumpulan method abstrak yang wajib diimplementasikan oleh kelas yang menggunakannya.

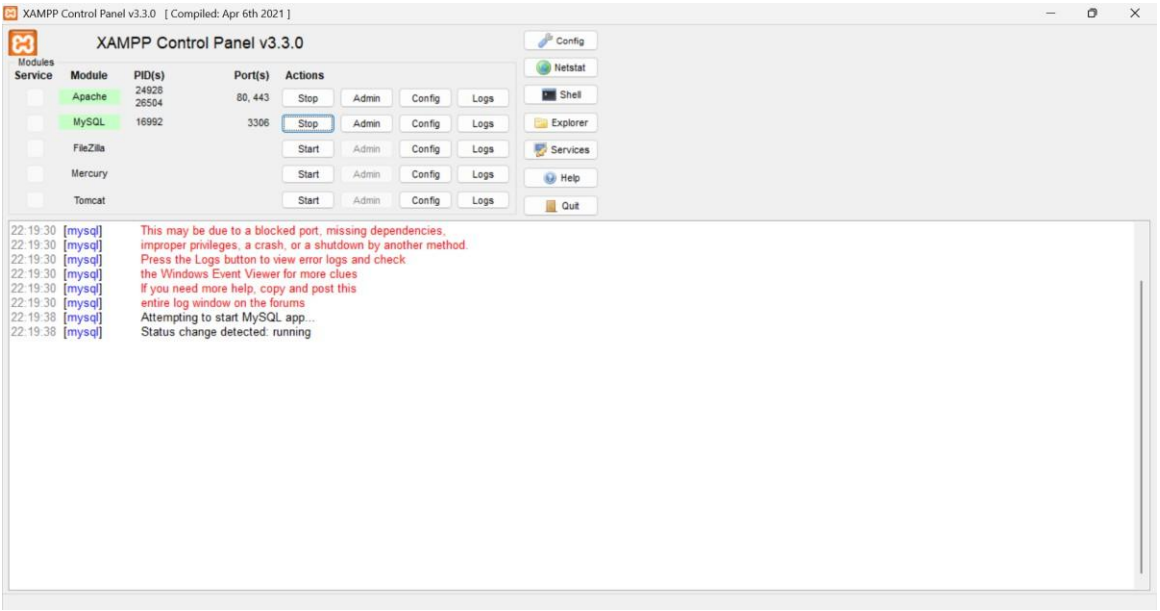
6. **CRUD**
   Operasi dasar dalam aplikasi database:
   - Create → menambahkan data baru.
   - Read → menampilkan data.
   - Update → mengubah data.
   - Delete → menghapus data.

---

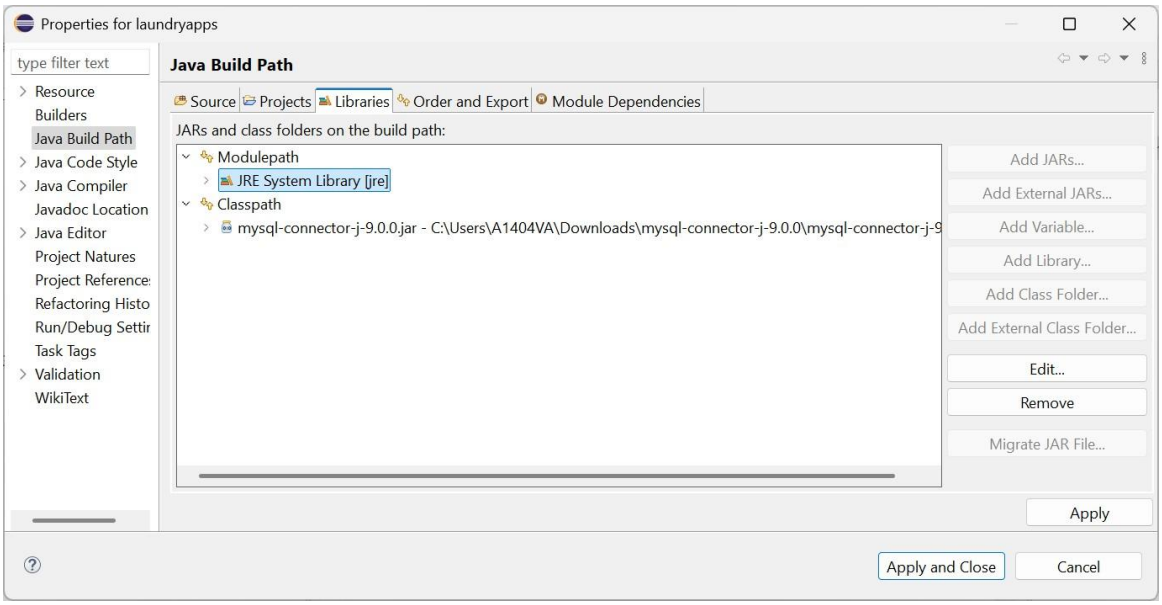
4. Langkah Kerja

4.1 Instalasi dan Persiapan

1. Install **XAMPP** dan jalankan Apache serta MySQL.

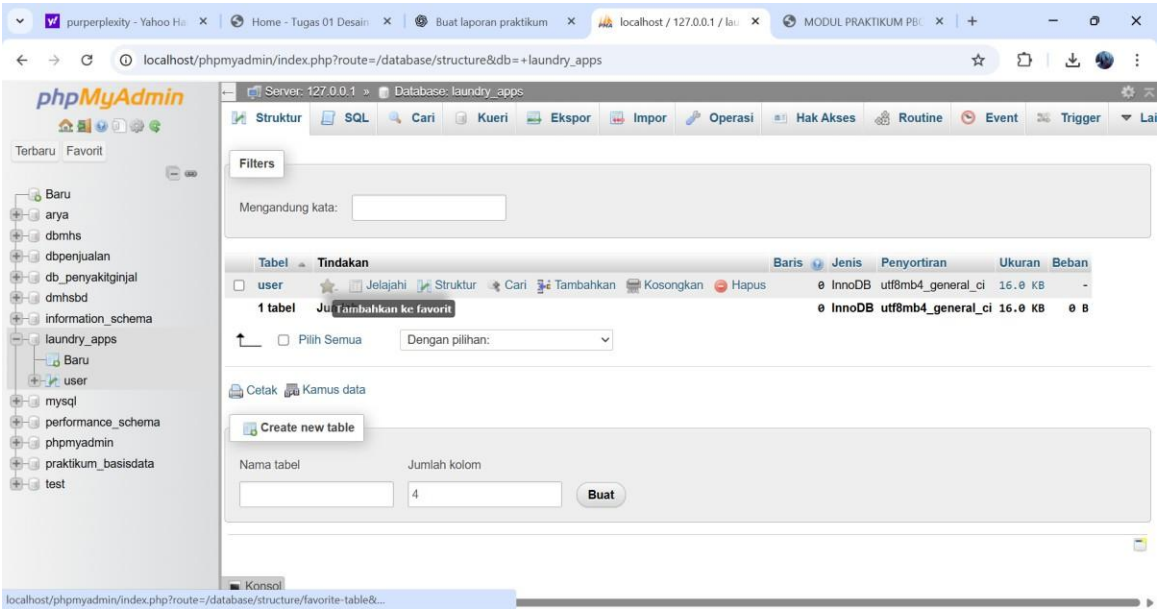


2. Tambahkan **MySQL Connector/J** ke dalam proyek Eclipse melalui *Build Path*.

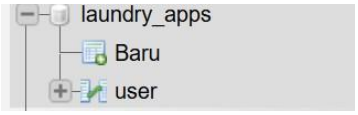


4.2 Pembuatan Database dan Tabel

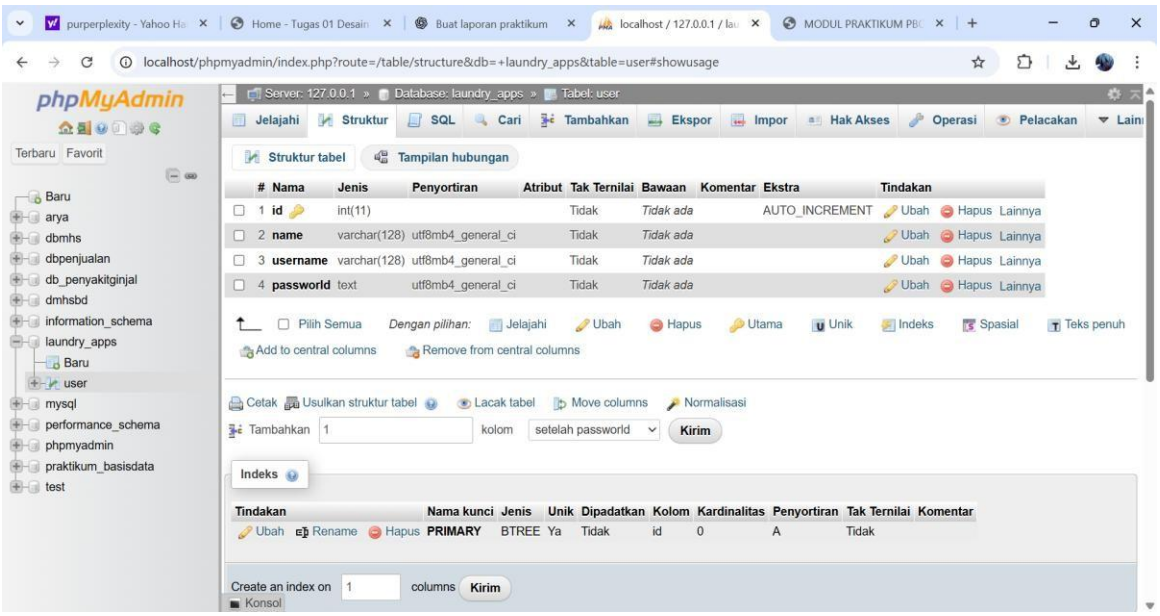
1. Buka **phpMyAdmin** pada http://localhost/phpmyadmin.



2. Buat database dengan nama **laundry_apps**.

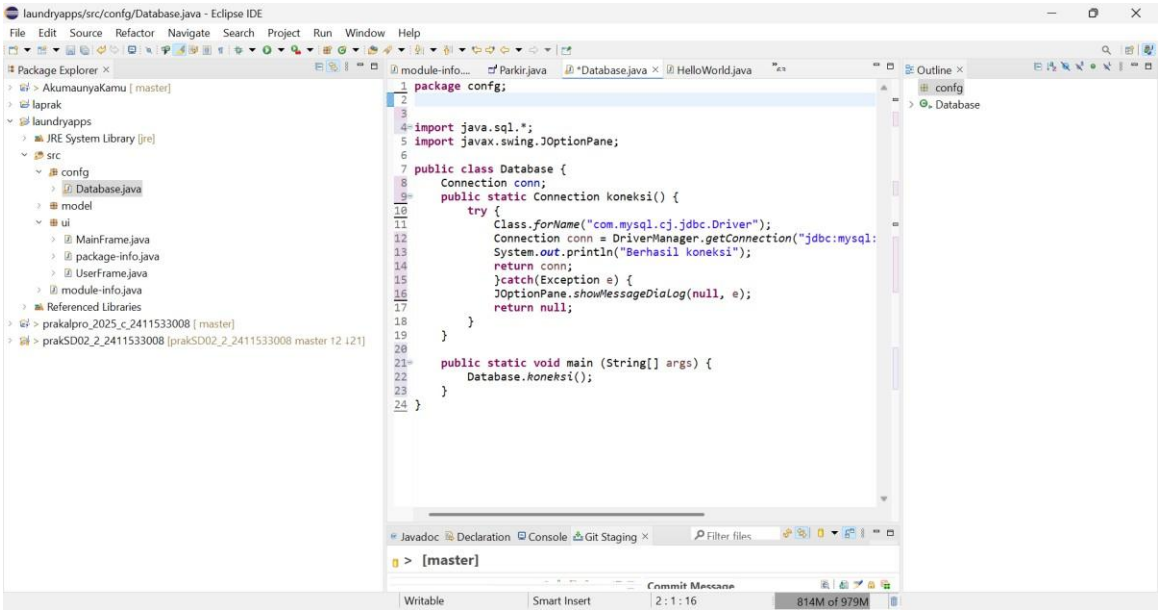


3. Buat tabel user dengan struktur:
   - id (INT, Auto Increment, Primary Key)
   - name (VARCHAR)
   - username (VARCHAR)
   - password (VARCHAR).



4.3 Pembuatan Koneksi Database

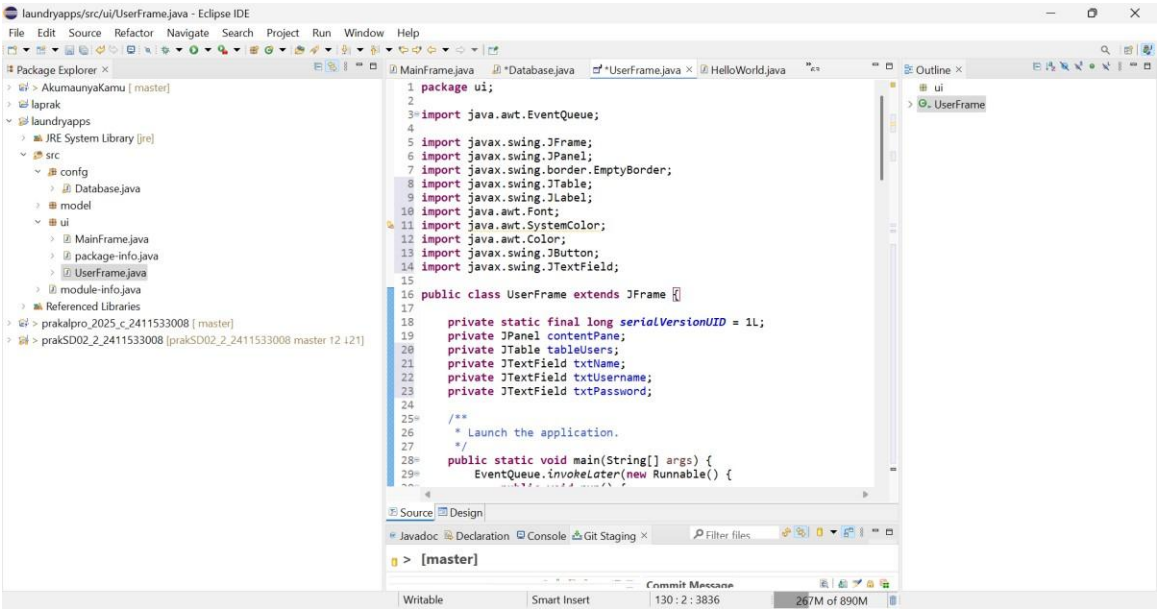
- Buat package config → class Database.
- Gunakan Connection dari java.sql.* untuk menghubungkan aplikasi Java dengan MySQL.



Penjelasan : ▪ Import java.sql.* digunakan untuk import seluruh fungsi-fungsi SQL ▪ Line 8 membuka method Connection dengan nama koneksi, yang mana method ini akan digunakan untuk membuka koneksi ke database ▪ Line 10-13 membuat koneksi database, jika koneksi berhasil maka akan mengembalikkan nilai Connection ▪ Line 15-16 jika koneksi gagal maka akan ditampilkan pesan error menggunakan JOptionPane

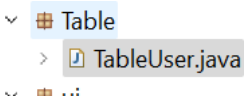
4.4 Membuat GUI CRUD User

- Buat package ui → class UserFrame.
- Tambahkan komponen:
    - JTextField → txtName, txtUsername, txtPassword.
    - JButton → btnSave, btnUpdate, btnDelete, btnCancel.
    - JTable → tableUsers.



4.5 Membuat Table Model

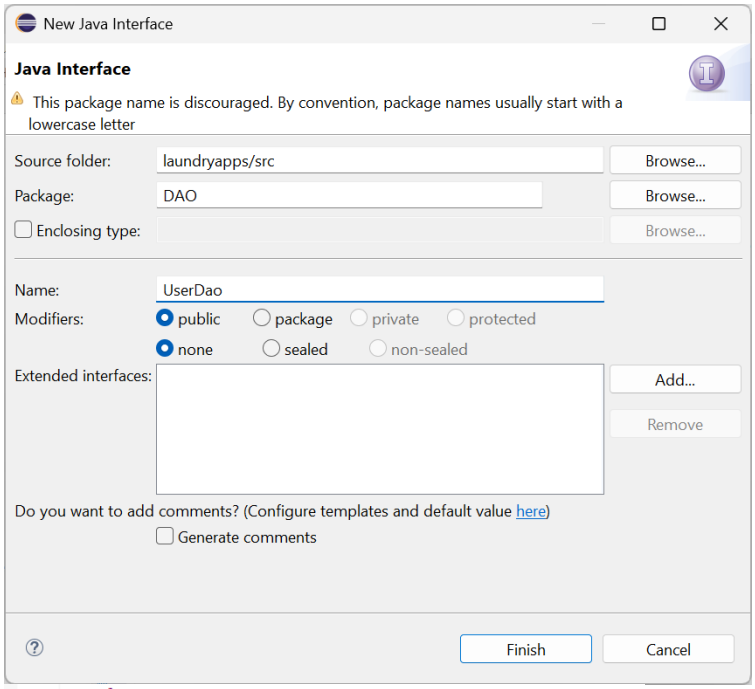
- Buat package table → class TableUser.



- Fungsinya mengambil data dari database untuk ditampilkan ke JTable.

4.6 Membuat Fungsi DAO

1. Buat package DAO → interface UserDAO.



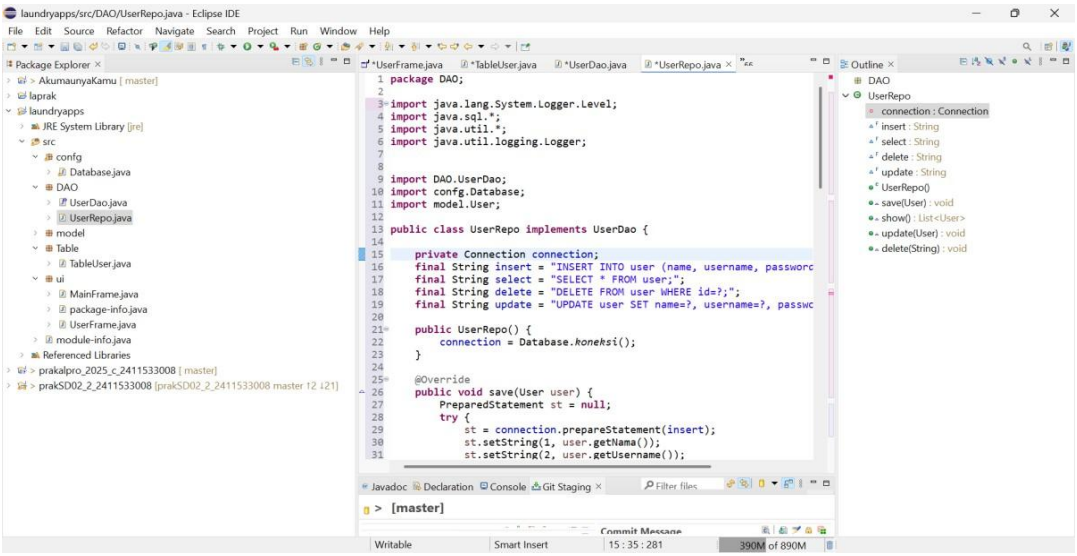

```java
package DAO;

import java.util.List;
import model.User;

public interface UserDao {
    void save(User user);
    public List<User> show();
    public void delete(String id);
    public void update(User user);
}
```

2. Method utama:
    - save()
    - show()
    - update()
    - delete().

3. Implementasikan interface dengan class UserRepo.



- import java.util.Date; → digunakan agar bisa memakai tipe data **Date** untuk menyimpan tanggal.
- public class Order → mendefinisikan sebuah **kelas** bernama Order.
private → artinya hanya bisa diakses dari dalam kelas, menjaga **enkapsulasi**.
Atribut ini merepresentasikan data sebuah order:

id → ID order (nomor unik pesanan).
id_costumer → ID pelanggan.
id_service → ID layanan/jasa yang dipilih.
id_user → ID pegawai/kasir yang menangani.
total → jumlah biaya pesanan.
tanggal → tanggal order dibuat.
tanggal_selesai → tanggal pesanan selesai.
status → status order (misalnya: "Proses", "Selesai").
status_pembayaran → status pembayaran (misalnya: "Lunas", "Belum Bayar").

Menyediakan **struktur data pesanan** dengan berbagai informasi penting (ID, layanan, pelanggan, total, status, dll).
Menjaga data tetap aman dengan **enkapsulasi** (menggunakan getter dan setter).
Mempermudah **inisialisasi data** dengan constructor.
Memberikan cara praktis untuk **menampilkan informasi** pesanan melalui tampilkanData().

TUGAS 3

Fungsi CRUD untuk Pelanggan (Customer)

1. **Membuat DAO (Data Access Object)**

- Buat sebuah **package** dengan nama DAO.
- Di dalam package tersebut, tambahkan sebuah **interface** baru bernama CustomerDAO.
- Pada interface ini, definisikan method utama yang wajib diimplementasikan pada class turunannya, yaitu:
- save untuk menyimpan data,
- show untuk menampilkan data,
- delete untuk menghapus data,
- update untuk memperbarui data

Ketik kodenya:

```
package DAO;

import confg.Database;
import model.Costumer;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

public class CostumerRepo implements CostumerDAO {

    private final Connection connection;
    private static final String INSERT = "INSERT INTO costumer (id, nama, alamat, nomor_hp)
VALUES (?,?,?,?)";
    private static final String SELECT = "SELECT * FROM costumer";
    private static final String UPDATE = "UPDATE costumer SET nama=?, alamat=?,
nomor_hp=? WHERE id=?";
    private static final String DELETE = "DELETE FROM costumer WHERE id=?";

    public CostumerRepo() {
        connection = Database.koneksi();
    }

    @Override
    public void save(Costumer costumer) {
        PreparedStatement st = null;
        try {
            st = connection.prepareStatement(INSERT);
            st.setString(1, costumer.getId());
            st.setString(2, costumer.getNama());
            st.setString(3, costumer.getAlamat());
            st.setString(4, costumer.getNomorHp());
            st.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (st != null) st.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    @Override
    public List<Costumer> show() {
        List<Costumer> list = new ArrayList<>();
        Statement st = null;
        ResultSet rs = null;

        try {
            st = connection.createStatement();
            rs = st.executeQuery(SELECT);
```

```
        while (rs.next()) {
            Costumer costumer = new Costumer();
            costumer.setId(rs.getString("id"));
            costumer.setNama(rs.getString("nama"));
            costumer.setAlamat(rs.getString("alamat"));
            costumer.setNomorHp(rs.getString("nomor_hp"));
            list.add(costumer);
        }
    } catch (SQLException e) {
        Logger.getLogger(CostumerRepo.class.getName()).log(Level.SEVERE, null, e);
    } finally {
        try {
            if (rs != null) rs.close();
            if (st != null) st.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    return list;
}

@Override
public void update(Costumer costumer) {
    PreparedStatement st = null;
    try {
        st = connection.prepareStatement(UPDATE);
        st.setString(1, costumer.getNama());
        st.setString(2, costumer.getAlamat());
        st.setString(3, costumer.getNomorHp());
        st.setString(4, costumer.getId());
        st.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        try {
            if (st != null) st.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

@Override
public void delete(String id) {
    PreparedStatement st = null;
    try {
        st = connection.prepareStatement(DELETE);
        st.setString(1, id);
        st.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        try {
            if (st != null) st.close();
```

```
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Fungsin dari code diatas:
Class CostumerRepo adalah implementasi **DAO untuk entitas Customer** yang menyediakan fungsi **CRUD**:

- **Create (save)** → menambah data baru.
- **Read (show)** → membaca daftar customer.
- **Update (update)** → memperbarui data customer.
- **Delete (delete)** → menghapus data customer.

Buatkan interfacenya:

```
package DAO;

import model.Costumer;
import java.util.List;

public interface CostumerDAO {
    void save(Costumer costumer);
    List<Costumer> show();
    void update(Costumer costumer);
    void delete(String id);
}
```

2.Fungsi CRUD untuk Layanan (Service)
1) Membuat Fungsi DAO • Pada package bernama DAO • Buat class Interface baru dengan nama ServiceDAO, kemudian isikan dengan kode program berikut. Terdapat method save, show, delete dan update. Method pada class interface digunakan sebagai method utama yang wajib diimplementasikan pada class yang menggunakannya.

```
import confg.Database;
import model.Service;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ServiceRepo implements ServiceDAO {

    private final Connection connection;
    private static final String INSERT = "INSERT INTO service (id, jenis, harga, status) VALUES (?,?,?,?)";
    private static final String SELECT = "SELECT * FROM service";
    private static final String UPDATE = "UPDATE service SET jenis=?, harga=?, status=? WHERE id=?";
    private static final String DELETE = "DELETE FROM service WHERE id=?";

    public ServiceRepo() {
        connection = Database.koneksi();
    }
```

```
@Override
public void save(Service service) {
    PreparedStatement st = null;
    try {
        st = connection.prepareStatement(INSERT);
        st.setString(1, service.getId());
        st.setString(2, service.getJenis());
        st.setDouble(3, service.getHarga());
        st.setString(4, service.getStatus());
        st.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        try {
            if (st != null) st.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

@Override
public List<Service> show() {
    List<Service> list = new ArrayList<>();
    Statement st = null;
    ResultSet rs = null;

    try {
        st = connection.createStatement();
        rs = st.executeQuery(SELECT);
        while (rs.next()) {
            Service service = new Service();
            service.setId(rs.getString("id"));
            service.setJenis(rs.getString("jenis"));
            service.setHarga(rs.getDouble("harga"));
            service.setStatus(rs.getString("status"));
            list.add(service);
        }
    } catch (SQLException e) {
        Logger.getLogger(ServiceRepo.class.getName()).log(Level.SEVERE, null, e);
    } finally {
        try {
            if (rs != null) rs.close();
            if (st != null) st.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    return list;
}

@Override
public void update(Service service) {
    PreparedStatement st = null;
```

```
        try {
            st = connection.prepareStatement(UPDATE);
            st.setString(1, service.getJenis());
            st.setDouble(2, service.getHarga());
            st.setString(3, service.getStatus());
            st.setString(4, service.getId());
            st.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (st != null) st.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    @Override
    public void delete(String id) {
        PreparedStatement st = null;
        try {
            st = connection.prepareStatement(DELETE);
            st.setString(1, id);
            st.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (st != null) st.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Class **ServiceRepo** adalah implementasi dari ServiceDAO yang menangani operasi **CRUD (Create, Read, Update, Delete)** pada tabel service di database dengan memanfaatkan koneksi JDBC.

Buatlah interfacenya:

```
package DAO;

import model.Service;
import java.util.List;

public interface ServiceDAO {
    void save(Service service);
    List<Service> show();
    void update(Service service);
    void delete(String id);
}
```

KESIMPULAN:

Praktikum ini bertujuan agar saya dapat memahami serta menerapkan pembuatan sistem CRUD (Create, Read, Update, Delete) pada data user dengan menggunakan bahasa pemrograman Java yang terintegrasi dengan database MySQL. Melalui kegiatan ini, saya mempelajari cara membuat tabel user di MySQL dan menghubungkannya ke aplikasi Java melalui koneksi database. Selain itu, saya juga belajar merancang antarmuka pengguna (GUI) untuk mendukung proses CRUD dengan mengimplementasikan konsep Pemrograman Berorientasi Objek (OOP). Tidak hanya itu, saya memperoleh pemahaman mengenai penerapan interface dan penggunaan Data Access Object (DAO) sebagai pendekatan yang lebih modular dan terstruktur dalam mengelola CRUD, sehingga sistem menjadi lebih efisien dan mudah dikembangkan di masa mendatang.