


Modul CRUD JFrame | X RPL

Created by	 Biru Business
Files & media	https://github.com/birugh/java-crud

Pendahuluan

CRUD adalah singkatan dari **Create, Read, Update, Delete**, yang merupakan operasi dasar dalam pengelolaan database. Dalam modul ini, kita akan membuat aplikasi CRUD menggunakan **Java (JFrame)** dan **MySQL**.

Membuat Database

Buka MySQL dan jalankan perintah berikut untuk membuat database:

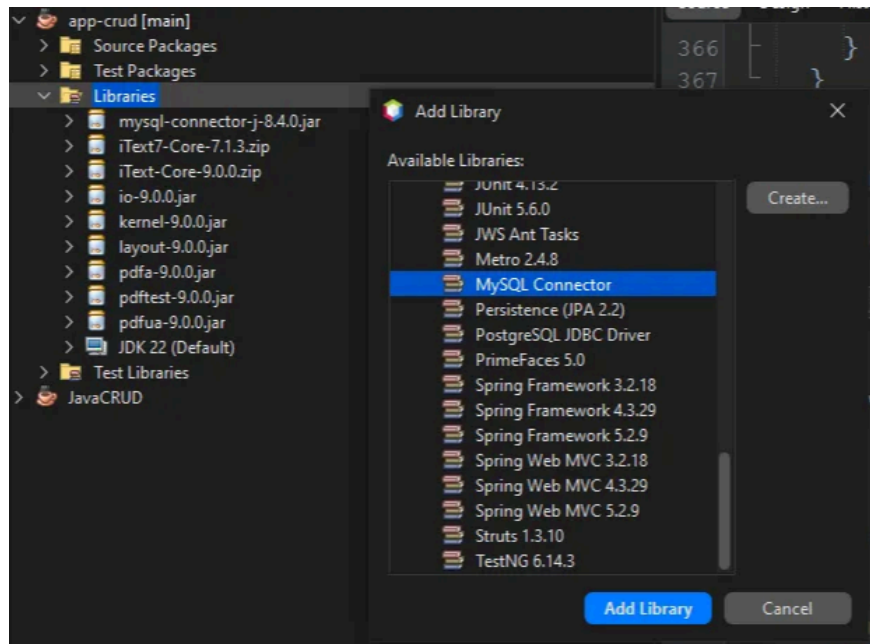
```
CREATE DATABASE db_sekolah;
```

Kemudian, jalankan perintah berikut untuk membuat tabel siswa:

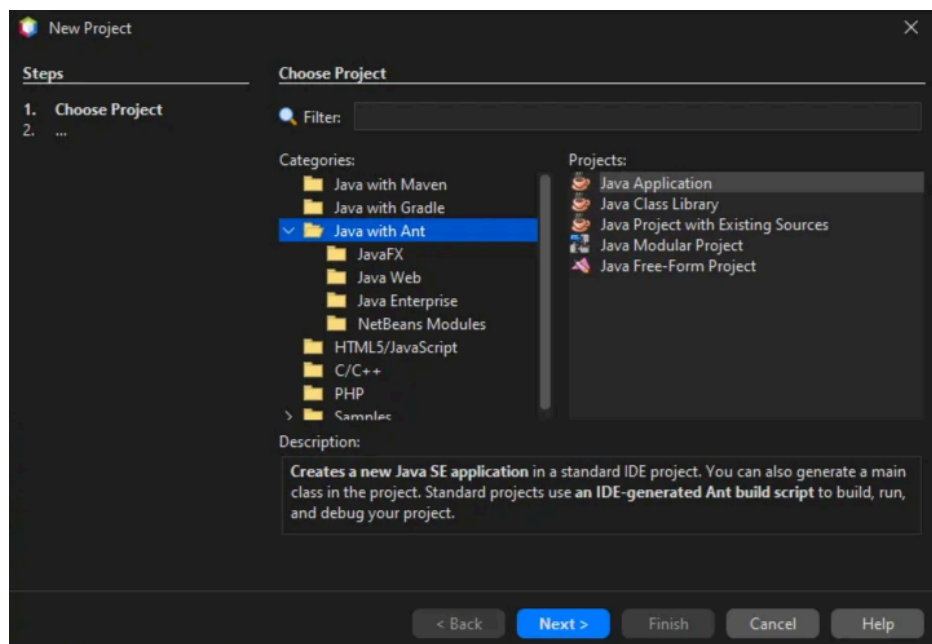
```
CREATE TABLE tbl_account(  
    id_user INT AUTO_INCREMENT PRIMARY KEY,  
    name VARCHAR(100),  
    email VARCHAR(100),  
    password VARCHAR(100)  
);
```

Library MySQL Connector

Jangan lupa untuk tambahkan Library "MySQL Connector" agar bisa terhubung ke database.



Jika tidak ada folder Libraries pada package atau proyeknya, maka buat ulanglah proyeknya dengan jenis **Java with Ant**



Membuat Tampilan GUI dengan JFrame

Buat JFrame Form bernama **CRUDForm.java**. Tambahkan elemen-elemen berikut:

Koneksi Database di Java

Buat kelas **DatabaseConnection.java** untuk menghubungkan aplikasi dengan database.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConnection {
    private static final String URL = "jdbc:mysql://localhost:3306/db_sekolah";
    private static final String USER = "root";
    private static final String PASSWORD = "";

    public static Connection getConnection() {
        try {
            return DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

Reset Input Text Field

Buat method untuk reset input dari text field:

```
private void ResetInput() {  
    txtId.setText("");  
    txtName.setText("");  
    txtEmail.setText("");  
    txtPassword.setText("");  
    txtId.requestFocus();  
}
```

Menampilkan Data di JTable

Buat method untuk menampilkan data dari database ke JTable:

```
private void LoadData() {  
    Connection conn = DatabaseConnection.getConnection();  
    DefaultTableModel model = (DefaultTableModel) tblAccount.getModel();  
    model.setRowCount(0);  
  
    if (conn != null) {  
        try {  
            String query = "SELECT * FROM tbl_account";  
            PreparedStatement pst = conn.prepareStatement(query);  
            ResultSet rs = pst.executeQuery();  
  
            while (rs.next()) {  
                Object[] row = {  
                    rs.getString("id_user"),  
                    rs.getString("name"),  
                    rs.getString("email"),  
                    rs.getString("password")  
                };  
                model.addRow(row);  
            }  
  
            rs.close();  
        }  
    }  
}
```

```

        pst.close();
        conn.close();
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
    }
} else {
    JOptionPane.showMessageDialog(this, "Koneksi database gagal!");
}
}

```

Menambahkan Data (Create)

Buat method untuk menyimpan data dari text field ke database:

```

private void InsertData() {
    Connection conn = DatabaseConnection.getConnection();
    if (conn != null) {
        try {
            String query = "INSERT INTO tbl_account (id_user, name, email, passw
            PreparedStatement pst = conn.prepareStatement(query);
            pst.setString(1, txtId.getText());
            pst.setString(2, txtName.getText());
            pst.setString(3, txtEmail.getText());
            pst.setString(4, txtPassword.getText());

            int rowsInserted = pst.executeUpdate();
            if (rowsInserted > 0) {
                JOptionPane.showMessageDialog(this, "Data berhasil ditambahkan!
                LoadData();
            }

            pst.close();
            conn.close();
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
            ResetInput();
        }
    } else {

```

```

        JOptionPane.showMessageDialog(this, "Koneksi database gagal!");
    }
}

```

Mengedit Data (Update)

Buat method untuk mengedit data dari text field ke database:

```

private void UpdateData() {
    Connection conn = DatabaseConnection.getConnection();
    if (conn != null) {
        try {
            String query = "UPDATE tbl_siswa SET name=?, email=?, password=? WHERE id=?";
            PreparedStatement pst = conn.prepareStatement(query);

            pst.setString(1, txtName.getText());
            pst.setString(2, txtEmail.getText());
            pst.setString(3, txtPassword.getText());
            pst.setString(4, txtId.getText());

            int rowsUpdated = pst.executeUpdate();
            if (rowsUpdated > 0) {
                JOptionPane.showMessageDialog(this, "Data berhasil diperbarui!");
                LoadData();
                ResetInput();
            }

            pst.close();
            conn.close();
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
        }
    } else {
        JOptionPane.showMessageDialog(this, "Koneksi database gagal!");
    }
}

```

Menghapus Data (Delete)

Buat method untuk menghapus data dari JTable & Database:

```
private void DeleteData() {
    Connection conn = DatabaseConnection.getConnection();
    if (conn != null) {
        try {
            String query = "DELETE FROM tbl_siswa WHERE id_user=?";
            PreparedStatement pst = conn.prepareStatement(query);
            pst.setString(1, txtId.getText());

            int confirm = JOptionPane.showConfirmDialog(this, "Apakah Anda y
            if (confirm == JOptionPane.YES_OPTION) {
                int rowsDeleted = pst.executeUpdate();
                if (rowsDeleted > 0) {
                    JOptionPane.showMessageDialog(this, "Data berhasil dihapus!"
                    LoadData();
                } else {
                    JOptionPane.showMessageDialog(this, "ID tidak ditemukan!");
                }
            }

            pst.close();
            conn.close();
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
            ResetInput();
        }
    } else {
        JOptionPane.showMessageDialog(this, "Koneksi database gagal!");
    }
}
```

Menampilkan Data Saat Klik di JTable

```
private void TakeData() {
    int selectedRow = tblAccount.getSelectedRow();

    if (selectedRow != -1) {
        txtId.setText(tblAccount.getValueAt(selectedRow, 0).toString());
        txtName.setText(tblAccount.getValueAt(selectedRow, 1).toString());
        txtEmail.setText(tblAccount.getValueAt(selectedRow, 2).toString());
        txtPassword.setText(tblAccount.getValueAt(selectedRow, 3).toString());
    }
}
```

Langkah Terakhir

Panggil semua method yang sudah dibuat, ke Event yang sesuai.

```
public CRUDForm() {
    initComponents();
    LoadData();
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
Generated Code

private void btnSubmitActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    InsertData();
}

private void btnResetActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    ResetInput();
}

private void btnUpdateActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    UpdateData();
}

private void btnDeleteActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    DeleteData();
}

private void tblAccountMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    TakeData();
}
```

Penjelasan dari setiap line code

DatabaseConnection

```
// Ini adalah kode yang memungkinkan program kita berbicara dengan databa
import java.sql.Connection; // Untuk membuat jalur ke database.
```



```

import java.sql.DriverManager; // Untuk menghubungkan program dengan dat
import java.sql.SQLException; // Untuk menangkap kesalahan jika koneksi ga

// Kita membuat kelas bernama DatabaseConnection.
public class DatabaseConnection {

    // Alamat tempat database kita berada.
    // "jdbc:mysql://" Menandakan kita menggunakan MySQL sebagai database
    // "localhost" berarti database ada di komputer kita sendiri.
    // "3306" adalah gerbang (port) tempat MySQL berjalan.
    // "db_sekolah" adalah nama database yang ingin kita gunakan.
    private static final String URL = "jdbc:mysql://localhost:3306/db_sekolah";

    // Nama pengguna untuk masuk ke database (biasanya "root" kalau lokal).
    private static final String USER = "root";

    // Kata sandi untuk masuk ke database (kosong karena default MySQL tidak
    private static final String PASSWORD = "";

    // Fungsi untuk membuat koneksi ke database.
    public static Connection getConnection() {
        try {
            // Mencoba (try) menyambungkan ke database menggunakan informasi
            return DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (SQLException e) {
            // Jika gagal terhubung, tampilkan pesan kesalahan di layar.
            e.printStackTrace();
            return null; // Beri tahu program kalau koneksi gagal.
        }
    }
}

```

Ringkasan

Bayangkan **database adalah sebuah rumah**, dan program kita adalah **orang yang ingin masuk ke rumah itu**.

1. **Kelas** `DatabaseConnection` → Bertugas menghubungkan aplikasi ke database.

2. **Konstanta** `URL` , `USER` , dan `PASSWORD` → Berisi informasi koneksi ke database.
3. **Method** `getConnection()` → Menghubungkan aplikasi ke database dan menangani error jika gagal.
4. `DriverManager.getConnection(URL, USER, PASSWORD)` → Memproses koneksi ke MySQL menggunakan informasi yang telah diberikan.
5. **Jika koneksi gagal** → Program menampilkan error dengan `e.printStackTrace();` agar mudah diketahui penyebabnya.

LoadData

```
// Fungsi ini digunakan untuk mengambil dan menampilkan data dari database
private void LoadData() {
    // Membuat koneksi ke database.
    Connection conn = DatabaseConnection.getConnection();

    // Mengambil model tabel dari tampilan GUI (tabel di aplikasi kita).
    DefaultTableModel model = (DefaultTableModel) tblAccount.getModel();

    // Menghapus semua data lama yang ada di tabel sebelum menampilkan data
    model.setRowCount(0);

    // Jika koneksi ke database berhasil...
    if (conn != null) {
        try {
            // Membuat perintah SQL (query) untuk mengambil semua data dari tabel
            String query = "SELECT * FROM tbl_account";

            // Menyiapkan perintah SQL agar bisa dikirim ke database.
            PreparedStatement pst = conn.prepareStatement(query);

            // Menjalankan perintah SQL dan menyimpan hasilnya.
            ResultSet rs = pst.executeQuery();

            // Perulangan ini berjalan selama masih ada data di dalam database.
            while (rs.next()) {
                // Mengambil data dari setiap kolom di database.
            }
        }
    }
}
```

```

        Object[] row = {
            rs.getString("id_user"), // Mengambil data dari kolom "id_user".
            rs.getString("name"),    // Mengambil data dari kolom "name".
            rs.getString("email"),   // Mengambil data dari kolom "email".
            rs.getString("password") // Mengambil data dari kolom "password".
        };

        // Menambahkan data yang sudah diambil ke dalam tabel aplikasi.
        model.addRow(row);
    }

    // Menutup hasil query setelah selesai digunakan.
    rs.close();

    // Menutup perintah SQL setelah selesai digunakan.
    pst.close();

    // Menutup koneksi ke database agar tidak membebani sistem.
    conn.close();
} catch (SQLException e) {
    // Jika terjadi kesalahan, tampilkan pesan error kepada pengguna.
    JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
}
} else {
    // Jika koneksi ke database gagal, beri tahu pengguna.
    JOptionPane.showMessageDialog(this, "Koneksi database gagal!");
}
}
}

```

Ringkasan

Bayangkan **database adalah rak buku** di perpustakaan, dan **tabel di aplikasi adalah meja tempat kita meletakkan buku**.

- `Connection conn = DatabaseConnection.getConnection();` → Kita membuka pintu perpustakaan.
- `String sql = "SELECT * FROM tbl_account";` → Kita meminta semua buku dari rak "tbl_account".

- `while (rs.next())` → Kita membaca buku satu per satu.
- `model.addRow(row);` → Kita meletakkan buku di meja (menampilkan data di tabel aplikasi).
- `conn.close();` → Setelah selesai, kita keluar dari perpustakaan agar tidak penuh sesak.

InsertData

```
// Fungsi ini digunakan untuk menambahkan data baru ke dalam database.
private void InsertData() {
    // Membuka koneksi ke database.
    Connection conn = DatabaseConnection.getConnection();

    // Jika koneksi ke database berhasil...
    if (conn != null) {
        try {
            // Membuat perintah SQL untuk menambahkan data ke dalam tabel `tbl`
            String query = "INSERT INTO tbl_account (id_user, name, email, passw

            // Menyiapkan perintah SQL agar bisa dikirim ke database.
            PreparedStatement pst = conn.prepareStatement(query);

            // Mengisi tanda tanya (?) dalam perintah SQL dengan data dari inputan
            pst.setString(1, txtId.getText()); // Mengambil ID dari inputan pengguna
            pst.setString(2, txtName.getText()); // Mengambil nama dari inputan pengguna
            pst.setString(3, txtEmail.getText()); // Mengambil email dari inputan pengguna
            pst.setString(4, txtPassword.getText()); // Mengambil password dari inputan pengguna

            // Menjalankan perintah SQL untuk menyimpan data ke database.
            int rowsInserted = pst.executeUpdate();

            // Jika ada data yang berhasil ditambahkan...
            if (rowsInserted > 0) {
                // Tampilkan pesan bahwa data berhasil ditambahkan.
                JOptionPane.showMessageDialog(this, "Data berhasil ditambahkan!");

                // Panggil fungsi LoadData() agar tabel di aplikasi diperbarui dengan
```

```

        LoadData();
    }

    // Menutup perintah SQL setelah selesai digunakan.
    pst.close();

    // Menutup koneksi ke database agar tidak membebani sistem.
    conn.close();
} catch (SQLException e) {
    // Jika terjadi kesalahan, tampilkan pesan error kepada pengguna.
    JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());

    // Memanggil fungsi ResetInput() untuk menghapus data di kolom input
    ResetInput();
}
} else {
    // Jika koneksi ke database gagal, beri tahu pengguna.
    JOptionPane.showMessageDialog(this, "Koneksi database gagal!");
}
}
}

```

Ringkasan

Bayangkan **database adalah rak buku** di perpustakaan, dan **form input adalah buku yang akan ditambahkan ke rak**.

- `Connection conn = DatabaseConnection.getConnection();` → Kita membuka pintu perpustakaan.
- `String query = "INSERT INTO tbl_account (id_user, name, email, password) VALUES (?, ?, ?, ?)"` → Kita menyiapkan formulir untuk menambahkan buku baru ke rak.
- `pst.setString(1, txtId.getText());` → Kita menuliskan ID buku di formulir.
- `pst.setString(2, txtName.getText());` → Kita menuliskan nama buku di formulir.
- `pst.setString(3, txtEmail.getText());` → Kita menuliskan email pemilik buku di formulir.
- `pst.setString(4, txtPassword.getText());` → Kita menuliskan password untuk buku tersebut.
- `int rowsInserted = pst.executeUpdate();` → Kita menyerahkan formulir ke petugas perpustakaan, dan mereka menambahkan buku ke rak.

- `LoadData();` → Setelah buku ditambahkan, kita memperbarui daftar buku di meja.
- `conn.close();` → Setelah selesai, kita keluar dari perpustakaan agar tidak penuh sesak.

UpdateData

```
// Fungsi ini digunakan untuk mengubah data yang sudah ada di dalam database
private void UpdateData() {
    // Membuka koneksi ke database.
    Connection conn = DatabaseConnection.getConnection();

    // Jika koneksi ke database berhasil...
    if (conn != null) {
        try {
            // Membuat perintah SQL untuk memperbarui data di tabel `tbl_siswa`.
            // Data yang diubah adalah name, email, dan password berdasarkan id.
            String query = "UPDATE tbl_siswa SET name=?, email=?, password=? \

            // Menyiapkan perintah SQL agar bisa dikirim ke database.
            PreparedStatement pst = conn.prepareStatement(query);

            // Mengisi tanda tanya (?) dalam perintah SQL dengan data dari inputan
            pst.setString(1, txtName.getText()); // Mengambil nama baru dari inputan
            pst.setString(2, txtEmail.getText()); // Mengambil email baru dari inputan
            pst.setString(3, txtPassword.getText()); // Mengambil password baru dari inputan
            pst.setString(4, txtId.getText()); // Mengambil ID untuk menentukan

            // Menjalankan perintah SQL untuk memperbarui data di database.
            int rowsUpdated = pst.executeUpdate();

            // Jika ada data yang berhasil diperbarui...
            if (rowsUpdated > 0) {
                // Tampilkan pesan bahwa data berhasil diperbarui.
                JOptionPane.showMessageDialog(this, "Data berhasil diperbarui!");

                // Panggil fungsi LoadData() agar tabel di aplikasi diperbarui dengan
                LoadData();
            }
        }
    }
}
```

```

        // Panggil fungsi ResetInput() untuk mengosongkan input setelah data
        ResetInput();
    }

    // Menutup perintah SQL setelah selesai digunakan.
    pst.close();

    // Menutup koneksi ke database agar tidak membebani sistem.
    conn.close();
} catch (SQLException e) {
    // Jika terjadi kesalahan, tampilkan pesan error kepada pengguna.
    JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
}
} else {
    // Jika koneksi ke database gagal, beri tahu pengguna.
    JOptionPane.showMessageDialog(this, "Koneksi database gagal!");
}
}
}

```

Ringkasan

Bayangkan **database adalah rak buku** di perpustakaan, dan **form input adalah lembar perubahan data buku**.

- `Connection conn = DatabaseConnection.getConnection();` → Kita membuka pintu perpustakaan.
- `String query = "UPDATE tbl_siswa SET name=?, email=?, password=? WHERE id_user=?"` → Kita menyiapkan formulir perubahan data buku berdasarkan ID buku.
- `pst.setString(1, txtName.getText());` → Kita mengganti nama buku di formulir.
- `pst.setString(2, txtEmail.getText());` → Kita mengganti email pemilik buku di formulir.
- `pst.setString(3, txtPassword.getText());` → Kita mengganti password buku di formulir.
- `pst.setString(4, txtId.getText());` → Kita menentukan ID buku yang ingin diubah.
- `int rowsUpdated = pst.executeUpdate();` → Kita menyerahkan formulir ke petugas perpustakaan, dan mereka memperbarui buku di rak.
- `LoadData();` → Setelah buku diperbarui, kita memperbarui daftar buku di meja.

- `ResetInput();` → Setelah selesai, kita membersihkan formulir agar bisa digunakan lagi.
- `conn.close();` → Setelah selesai, kita keluar dari perpustakaan agar tidak penuh sesak.

DeleteData

```
// Fungsi ini digunakan untuk menghapus data dari database berdasarkan ID p
private void DeleteData() {
    // Membuka koneksi ke database.
    Connection conn = DatabaseConnection.getConnection();

    // Jika koneksi ke database berhasil...
    if (conn != null) {
        try {
            // Membuat perintah SQL untuk menghapus data berdasarkan id_user.
            String query = "DELETE FROM tbl_siswa WHERE id_user=?";

            // Menyiapkan perintah SQL agar bisa dikirim ke database.
            PreparedStatement pst = conn.prepareStatement(query);

            // Mengisi tanda tanya (?) dalam perintah SQL dengan ID pengguna yang
            pst.setString(1, txtId.getText());

            // Menampilkan kotak dialog konfirmasi untuk memastikan pengguna b
            int confirm = JOptionPane.showConfirmDialog(this, "Apakah Anda yak

            // Jika pengguna memilih "YA" (YES_OPTION), maka proses penghapus
            if (confirm == JOptionPane.YES_OPTION) {
                // Menjalankan perintah SQL untuk menghapus data di database.
                int rowsDeleted = pst.executeUpdate();

                // Jika ada data yang berhasil dihapus...
                if (rowsDeleted > 0) {
                    // Tampilkan pesan bahwa data berhasil dihapus.
                    JOptionPane.showMessageDialog(this, "Data berhasil dihapus!");

                    // Panggil fungsi LoadData() agar tabel di aplikasi diperbarui setela
```



```

        LoadData();
    } else {
        // Jika ID yang dimasukkan tidak ditemukan di database, tampilkan
        JOptionPane.showMessageDialog(this, "ID tidak ditemukan!");
    }
}

// Menutup perintah SQL setelah selesai digunakan.
pst.close();

// Menutup koneksi ke database agar tidak membebani sistem.
conn.close();
} catch (SQLException e) {
    // Jika terjadi kesalahan, tampilkan pesan error kepada pengguna.
    JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());

    // Memanggil fungsi ResetInput() untuk mengosongkan input setelah te
    ResetInput();
}
} else {
    // Jika koneksi ke database gagal, beri tahu pengguna.
    JOptionPane.showMessageDialog(this, "Koneksi database gagal!");
}
}
}

```

Ringkasan

Bayangkan **database adalah rak buku** di perpustakaan, dan **form input adalah catatan daftar buku**.

- `Connection conn = DatabaseConnection.getConnection();` → Kita membuka pintu perpustakaan.
- `String query = "DELETE FROM tbl_siswa WHERE id_user=?"` → Kita menyiapkan formulir penghapusan buku berdasarkan ID.
- `pst.setString(1, txtId.getText());` → Kita menentukan buku mana yang akan dihapus dengan mengisi ID buku di formulir.
- `JOptionPane.showConfirmDialog(...)` → Petugas bertanya, "Apakah Anda yakin ingin menghapus buku ini?"

- **Jika pengguna memilih "YA"** → Buku akan dihapus dari rak.
- `LoadData();` → Setelah buku dihapus, kita memperbarui daftar buku di meja.
- **Jika ID tidak ditemukan** → Petugas memberi tahu, "Buku dengan ID ini tidak ada."
- **Jika ada error** → Kita membersihkan formulir agar bisa digunakan lagi.
- `conn.close();` → Setelah selesai, kita keluar dari perpustakaan agar tidak penuh sesak.

TakeData

```
// Fungsi ini digunakan untuk mengambil data dari tabel dan menampilkannya
private void TakeData() {
    // Mengambil nomor baris yang sedang dipilih di tabel.
    int selectedRow = tblAccount.getSelectedRow();

    // Jika ada baris yang dipilih (bukan -1)...
    if (selectedRow != -1) {
        // Mengambil data dari tabel berdasarkan baris yang dipilih, lalu mengisi
        txtId.setText(tblAccount.getValueAt(selectedRow, 0).toString()); // Mengi
        txtName.setText(tblAccount.getValueAt(selectedRow, 1).toString()); // Me
        txtEmail.setText(tblAccount.getValueAt(selectedRow, 2).toString()); // Me
        txtPassword.setText(tblAccount.getValueAt(selectedRow, 3).toString()); //
    }
}
```

Ringkasan dengan Contoh Sehari-hari

Bayangkan **tabel** adalah daftar nama siswa di papan tulis dan **form input** adalah buku catatan.

1. `tblAccount.getSelectedRow();` → Kita melihat baris mana yang sedang dipilih di papan tulis.
2. **Jika ada baris yang dipilih (`selectedRow != -1`)** → Artinya, kita menunjuk ke salah satu nama di daftar.
3. `tblAccount.getValueAt(selectedRow, 0).toString();` → Kita menyalin ID dari daftar ke buku catatan.

4. `tblAccount.getValueAt(selectedRow, 1).toString();` → Kita menyalin Nama dari daftar ke buku catatan.
5. `tblAccount.getValueAt(selectedRow, 2).toString();` → Kita menyalin Email dari daftar ke buku catatan.
6. `tblAccount.getValueAt(selectedRow, 3).toString();` → Kita menyalin Password dari daftar ke buku catatan.