

teknologi  
informasi  
UNIVERSITAS MUHAMMADIYAH YOGYAKARTA

# Modul Praktikum

# Pengembangan Aplikasi Basisdata

Apriliya Kurnianti S.T., M.Eng



## Daftar Isi

<b>Stored Procedure dalam Aplikasi.....</b>	<b>1</b>
Tujuan Praktikum .....	1
Alat dan Bahan.....	1
Langkah-Langkah Praktikum .....	1

## Stored Procedure dalam Aplikasi

### Tujuan Praktikum

Memahami penggunaan Stored Procedure dalam sebuah aplikasi

### Alat dan Bahan

- SQL Server (Minimal SQL Server Express)
- SQL Server Management Studio (SSMS)
- Visual Studio (Minimal 2019 dengan .NET Framework)
- C# Windows Forms (WinForms)

### Langkah-Langkah Praktikum

1. Gunakan database UniversityDB
2. Buat store procedure pada sql server

```
-- Stored Procedure untuk Menambah Data Mahasiswa
CREATE PROCEDURE AddMahasiswa
    @NIM CHAR(11),
    @Nama VARCHAR(50),
    @Email VARCHAR(100),
    @Telepon VARCHAR(13),
    @Alamat VARCHAR(255)
AS
BEGIN
    INSERT INTO Mahasiswa (NIM, Nama, Email, Telepon, Alamat)
    VALUES (@NIM, @Nama, @Email, @Telepon, @Alamat);
END;
```

### Penjelasan:

1. CREATE PROCEDURE AddMahasiswa

CREATE PROCEDURE: Perintah SQL ini digunakan untuk membuat sebuah stored procedure (prosedur tersimpan) baru di dalam database. Prosedur ini akan disimpan dalam database dan dapat dipanggil kembali untuk mengeksekusi sekumpulan query SQL.

AddMahasiswa: Nama dari prosedur yang akan dibuat. Nama prosedur ini bisa digunakan untuk memanggil prosedur ini saat melakukan operasi di database.

2. Parameter yang Didefinisikan

Di dalam prosedur, terdapat lima parameter yang didefinisikan dengan nama dan tipe data tertentu:

@NIM CHAR(11): Parameter ini digunakan untuk menerima input berupa nomor induk mahasiswa (NIM), dengan tipe data CHAR(11), yang berarti NIM harus memiliki panjang tetap 11 karakter.

@Nama VARCHAR(50): Parameter ini menerima input nama mahasiswa, dengan tipe data VARCHAR(50), yang berarti nama mahasiswa dapat memiliki panjang hingga 50 karakter.

@Email VARCHAR(100): Parameter ini menerima input email mahasiswa, dengan tipe data VARCHAR(100), yang berarti email mahasiswa dapat memiliki panjang hingga 100 karakter.

@Telepon VARCHAR(13): Parameter ini menerima input nomor telepon mahasiswa, dengan tipe data VARCHAR(13), yang berarti nomor telepon dapat memiliki panjang hingga 13 karakter.

@Alamat VARCHAR(255): Parameter ini menerima input alamat mahasiswa, dengan tipe data VARCHAR(255), yang berarti alamat dapat memiliki panjang hingga 255 karakter.

### 3. Bagian AS

Setelah mendeklarasikan parameter-parameter yang diterima oleh prosedur, bagian AS mengindikasikan dimulainya blok kode yang berisi perintah-perintah SQL yang akan dijalankan ketika prosedur dipanggil.

### 4. BEGIN dan END

BEGIN dan END: Menandai awal dan akhir dari blok pernyataan SQL yang akan dijalankan. Semua perintah SQL yang berada di dalam BEGIN dan END adalah bagian dari prosedur ini.

### 5. INSERT INTO Mahasiswa

INSERT INTO adalah perintah SQL yang digunakan untuk menambahkan data baru ke dalam sebuah tabel. Dalam hal ini, data yang dimasukkan ke dalam tabel Mahasiswa. Mahasiswa adalah nama tabel tempat data mahasiswa akan disimpan.

### 6. VALUES (@NIM, @Nama, @Email, @Telepon, @Alamat)

VALUES: Menyatakan nilai-nilai yang akan dimasukkan ke dalam tabel Mahasiswa. Nilai-nilai ini akan diambil dari parameter yang diterima oleh prosedur:

@NIM: Nilai untuk kolom NIM.

@Nama: Nilai untuk kolom Nama.

@Email: Nilai untuk kolom Email.

@Telepon: Nilai untuk kolom Telepon.

@Alamat: Nilai untuk kolom Alamat.

```
-- Stored Procedure untuk Menghapus Data Mahasiswa berdasarkan NIM
CREATE PROCEDURE DeleteMahasiswa
    @NIM CHAR(11)
AS
BEGIN
    DELETE FROM Mahasiswa WHERE NIM = @NIM;
END;
```

### Penjelasan:

#### 1. CREATE PROCEDURE DeleteMahasiswa

CREATE PROCEDURE: Perintah SQL ini digunakan untuk membuat sebuah stored procedure (prosedur tersimpan) baru di dalam database. Prosedur ini akan disimpan dalam database dan dapat dipanggil kembali untuk mengeksekusi sekumpulan query SQL.

DeleteMahasiswa: nama dari prosedur yang akan dibuat. Nama prosedur ini bisa digunakan untuk memanggil prosedur ini saat melakukan operasi di database.

#### 2. Parameter yang Didefinisikan

@NIM CHAR(11): Parameter ini digunakan untuk menerima input berupa NIM (Nomor Induk Mahasiswa), dengan tipe data CHAR(11), yang berarti NIM harus memiliki panjang tetap 11 karakter.

CHAR(11) memastikan bahwa NIM yang dimasukkan memiliki panjang yang tepat, yaitu 11 karakter, dan tidak bisa lebih atau kurang.

#### 3. Bagian AS

Setelah mendeklarasikan parameter-parameter yang diterima oleh prosedur, bagian AS mengindikasikan dimulainya blok kode yang berisi perintah-perintah SQL yang akan dijalankan ketika prosedur dipanggil.

#### 4. BEGIN dan END

BEGIN dan END: Menandai awal dan akhir dari blok pernyataan SQL yang akan dijalankan. Semua perintah SQL yang berada di dalam BEGIN dan END adalah bagian dari prosedur ini.

#### 5. DELETE FROM Mahasiswa WHERE NIM = @NIM

DELETE FROM Mahasiswa: Perintah DELETE digunakan untuk menghapus baris-baris data dari tabel. Dalam hal ini, DELETE digunakan untuk menghapus data dari tabel Mahasiswa.

WHERE NIM = @NIM: Kondisi WHERE digunakan untuk menentukan baris mana yang akan dihapus. Dalam hal ini, hanya baris yang memiliki nilai NIM yang sesuai dengan parameter @NIM yang akan dihapus.

@NIM adalah parameter yang diterima oleh prosedur ini, yang diisi dengan NIM mahasiswa yang ingin dihapus.

-- Stored Procedure untuk Memperbarui Data Mahasiswa

```
CREATE PROCEDURE UpdateMahasiswa
    @NIM CHAR(11),
    @Nama VARCHAR(50),
    @Email VARCHAR(100),
    @Telepon VARCHAR(13),
    @Alamat VARCHAR(255)
AS
BEGIN
    UPDATE Mahasiswa
    SET
        Nama = COALESCE(NULLIF(@Nama, ''), Nama),
        Email = COALESCE(NULLIF(@Email, ''), Email),
        Telepon = COALESCE(NULLIF(@Telepon, ''), Telepon),
        Alamat = COALESCE(NULLIF(@Alamat, ''), Alamat)
    WHERE NIM = @NIM;
END;
```

### Penjelasan:

#### 1. CREATE PROCEDURE UpdateMahasiswa

CREATE PROCEDURE: Perintah ini digunakan untuk membuat prosedur tersimpan (stored procedure) baru di dalam database. Prosedur ini akan disimpan di dalam database dan dapat dipanggil untuk melakukan operasi secara otomatis tanpa menulis query SQL berulang kali.

UpdateMahasiswa: Nama prosedur yang digunakan untuk mengidentifikasi dan memanggil prosedur ini.

#### 2. Parameter yang Didefinisikan

Prosedur UpdateMahasiswa memiliki lima parameter:

@NIM CHAR(11): Parameter ini menerima nilai NIM mahasiswa, yang memiliki panjang tetap 11 karakter (menggunakan CHAR(11)).

@Nama VARCHAR(50): Parameter ini menerima nilai Nama mahasiswa, dengan panjang maksimum 50 karakter (VARCHAR(50)).

@Email VARCHAR(100): Parameter ini menerima nilai Email mahasiswa, dengan panjang maksimum 100 karakter (VARCHAR(100)).

@Telepon VARCHAR(13): Parameter ini menerima nilai Telepon mahasiswa, dengan panjang maksimum 13 karakter (VARCHAR(13)).

@Alamat VARCHAR(255): Parameter ini menerima nilai Alamat mahasiswa, dengan panjang maksimum 255 karakter (VARCHAR(255)).

### 3. Bagian AS

Setelah mendeklarasikan parameter-parameter yang diterima, bagian AS menunjukkan dimulainya blok kode SQL yang berisi perintah yang akan dijalankan ketika prosedur dipanggil.

### 4. BEGIN dan END

BEGIN dan END: Menandakan awal dan akhir dari blok pernyataan SQL yang akan dieksekusi dalam prosedur ini. Semua query SQL yang ada di dalamnya adalah bagian dari prosedur ini.

### 5. UPDATE Mahasiswa

UPDATE Mahasiswa: Perintah UPDATE digunakan untuk memperbarui data di dalam tabel Mahasiswa. Dalam hal ini, data mahasiswa yang sesuai dengan NIM yang diberikan akan diperbarui.

### 6. SET dengan COALESCE dan NULLIF

SET: Digunakan untuk menetapkan nilai baru ke kolom-kolom yang ada dalam tabel Mahasiswa.

COALESCE(NULLIF(@Nama, ''), Nama):

NULLIF(@Nama, ''): Fungsi NULLIF membandingkan @Nama dengan string kosong (''). Jika @Nama adalah string kosong, maka hasilnya adalah NULL, dan jika tidak, @Nama tetap dipertahankan.

COALESCE(NULLIF(@Nama, ''), Nama): Fungsi COALESCE memeriksa hasil dari NULLIF. Jika hasilnya NULL (yaitu @Nama adalah string kosong), maka kolom Nama akan tetap menggunakan nilai lama dari tabel Mahasiswa. Jika @Nama tidak kosong, maka Nama akan diperbarui dengan @Nama.

Proses yang sama berlaku untuk kolom Email, Telepon, dan Alamat.

Dengan cara ini, hanya kolom yang diterima dengan nilai non-kosong yang akan diperbarui. Jika tidak ada nilai yang diberikan (misalnya, parameter kosong), nilai kolom tersebut akan tetap seperti sebelumnya.

#### 7. WHERE NIM = @NIM

WHERE: Digunakan untuk menentukan baris mana yang akan diperbarui. Dalam hal ini, baris yang akan diperbarui adalah yang memiliki NIM yang cocok dengan parameter @NIM.

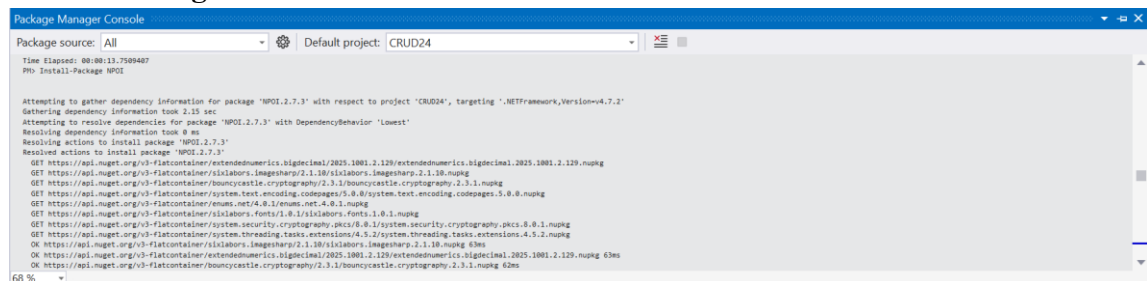
Hanya satu mahasiswa (atau lebih jika NIM bukan PRIMARY KEY atau tidak unik) yang akan diperbarui, berdasarkan NIM yang diterima sebagai parameter.

3. Lakukan modifikasi pada code
4. Install NPOI via NuGet Package Manager Console

- a. Buka NuGet Package Manager Console:
- b. Di Visual Studio, buka Tools > NuGet Package Manager > Package Manager Console.
- c. Install EPPlus:

Di Package Manager Console, ketikkan perintah berikut untuk menginstal EPPlus:

#### Install-Package NPOI

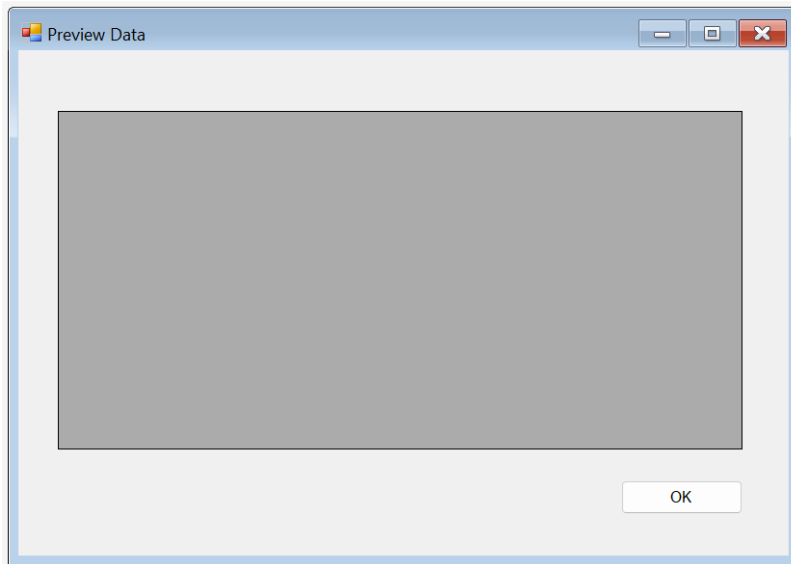


```
Package Manager Console
Package source: All Default project: CRUD24
Time Elapsed: 00:00:13.7509487
PM> Install-Package NPOI

Attempting to gather dependency information for package 'NPOI.2.7.3' with respect to project 'CRUD24', targeting '.NETFramework,Version=v4.7.2'
Gathering dependency information took 2.15 sec
Attempting to resolve dependencies for package 'NPOI.2.7.3' with DependencyBehavior 'Lowest'
Resolving dependency information took 0 ms
Resolving actions to install package 'NPOI.2.7.3'
Resolved actions to install package 'NPOI.2.7.3'
GET https://api.nuget.org/v3-flatcontainer/extendednnumeric.bigdecimal/2025.1001.2.129/extendednnumeric.bigdecimal.2025.1001.2.129.nupkg
GET https://api.nuget.org/v3-flatcontainer/extendednnumeric.bigdecimal/2025.1001.2.129/extendednnumeric.bigdecimal.2025.1001.2.129.nupkg
GET https://api.nuget.org/v3-flatcontainer/bouncycastle.cryptography/2.3.1/bouncycastle.cryptography.2.3.1.nupkg
GET https://api.nuget.org/v3-flatcontainer/system.text.encoding.codepages/5.0.0/system.text.encoding.codepages.5.0.0.nupkg
GET https://api.nuget.org/v3-flatcontainer/system.text.encoding.codepages/5.0.0/system.text.encoding.codepages.5.0.0.nupkg
GET https://api.nuget.org/v3-flatcontainer/system.security.cryptography.pkcs/8.0.1/system.security.cryptography.pkcs.8.0.1.nupkg
GET https://api.nuget.org/v3-flatcontainer/system.security.cryptography.pkcs/8.0.1/system.security.cryptography.pkcs.8.0.1.nupkg
GET https://api.nuget.org/v3-flatcontainer/system.threading.tasks.extensions/4.5.2/system.threading.tasks.extensions.4.5.2.nupkg
OK https://api.nuget.org/v3-flatcontainer/extendednnumeric.bigdecimal/2025.1001.2.129/extendednnumeric.bigdecimal.2025.1001.2.129.nupkg 63ms
OK https://api.nuget.org/v3-flatcontainer/extendednnumeric.bigdecimal/2025.1001.2.129/extendednnumeric.bigdecimal.2025.1001.2.129.nupkg 63ms
OK https://api.nuget.org/v3-flatcontainer/bouncycastle.cryptography/2.3.1/bouncycastle.cryptography.2.3.1.nupkg 62ms
```

- d. Tambahkan 1 form lagi, beri nama PreviewForm untuk melihat data sebelum di import dari file excel.





##### 5. Code untuk menambah data menggunakan stored procedure:

```
//Fungsi untuk menambahkan data (CREATE)
1 reference
private void BtnTambah(object sender, EventArgs e)
{
    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        try
        {
            if (txtNIM.Text == "" || txtNama.Text == "" || txtEmail.Text == "" || txtTelepon.Text == "")
            {
                MessageBox.Show("Harap isi semua data!", "Peringatan", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }

            conn.Open();
            using (SqlCommand cmd = new SqlCommand("AddMahasiswa", conn))
            {
                cmd.CommandType = CommandType.StoredProcedure;

                // Menambahkan parameter untuk stored procedure
                cmd.Parameters.AddWithValue("@NIM", txtNIM.Text.Trim());
                cmd.Parameters.AddWithValue("@Nama", txtNama.Text.Trim());
                cmd.Parameters.AddWithValue("@Email", txtEmail.Text.Trim());
                cmd.Parameters.AddWithValue("@Telepon", txtTelepon.Text.Trim());
                cmd.Parameters.AddWithValue("@Alamat", txtAlamat.Text.Trim());

                int rowsAffected = cmd.ExecuteNonQuery();
                if (rowsAffected > 0)
                {
                    MessageBox.Show("Data berhasil ditambahkan!", "Sukses", MessageBoxButtons.OK, MessageBoxIcon.Information);
                    LoadData();
                    ClearForm(); //Auto Clear setelah tambah data
                }
                else
                {
                    MessageBox.Show("Data tidak berhasil ditambahkan!", "Kesalahan", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error: " + ex.Message, "Kesalahan", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
}
```

##### Penjelasan:

- Deklarasi BtnTambah (Event Handler):

BtnTambah adalah metode yang akan dipanggil saat tombol Tambah diklik. Ini adalah event handler untuk event Click pada tombol BtnTambah.

- Membuka Koneksi ke Database:
  - `using (SqlConnection conn = new SqlConnection(connectionString));`  
Membuka koneksi ke database menggunakan SqlConnection. Koneksi ini akan dipakai untuk mengirimkan query ke database. Koneksi menggunakan string koneksi yang sudah didefinisikan di connectionString.
  - using memastikan bahwa objek conn akan dibersihkan dan ditutup secara otomatis setelah selesai digunakan.
- Validasi Input:
  - `if (txtNIM.Text == "" || txtNama.Text == "" || txtEmail.Text == "" || txtTelepon.Text == "")`  
Mengecek apakah ada kolom input yang kosong pada txtNIM, txtNama, txtEmail, atau txtTelepon.
  - Jika ada kolom yang kosong, MessageBox akan muncul untuk memberi peringatan kepada pengguna dengan pesan "Harap isi semua data!" dan proses akan dihentikan dengan return.
- Membuka Koneksi dan Menyiapkan Stored Procedure:
  - `conn.Open();`  
Membuka koneksi yang telah didefinisikan sebelumnya.
  - `using (SqlCommand cmd = new SqlCommand("AddMahasiswa", conn))`  
Membuat perintah SqlCommand untuk mengeksekusi prosedur tersimpan (stored procedure) AddMahasiswa pada database. Stored procedure ini akan dijalankan di database.
  - `cmd.CommandType = CommandType.StoredProcedure;`  
Menentukan bahwa SqlCommand yang digunakan adalah stored procedure, bukan query biasa.
- Menambahkan Parameter untuk Stored Procedure:
  - `cmd.Parameters.AddWithValue("@NIM", txtNIM.Text.Trim());`  
Menambahkan parameter untuk stored procedure. Nilai parameter @NIM diambil dari txtNIM.Text, yang merupakan input yang dimasukkan pengguna ke dalam TextBox.

- Trim() digunakan untuk menghapus spasi yang tidak diperlukan di awal dan akhir input.
  - Parameter lain (@Nama, @Email, @Telepon, @Alamat) ditambahkan dengan cara yang sama, mengambil nilai dari TextBox masing-masing.
- Menjalankan Stored Procedure:
- `int rowsAffected = cmd.ExecuteNonQuery();`  
Menjalankan ExecuteNonQuery untuk mengeksekusi stored procedure yang telah disiapkan. Fungsi ini digunakan ketika query yang dijalankan tidak mengembalikan data (seperti SELECT), melainkan hanya mempengaruhi data (seperti INSERT, UPDATE, atau DELETE).
  - rowsAffected akan berisi jumlah baris yang terpengaruh oleh operasi (dalam hal ini, jumlah baris yang berhasil ditambahkan ke tabel Mahasiswa).
- Menangani Hasil Eksekusi:
- Jika rowsAffected lebih besar dari 0, berarti data berhasil ditambahkan ke dalam database. Maka, MessageBox dengan pesan "Data berhasil ditambahkan!" akan ditampilkan, dan proses LoadData() serta ClearForm() akan dijalankan untuk memperbarui tampilan dan mengosongkan kolom input.
  - Jika rowsAffected adalah 0 (tidak ada baris yang terpengaruh), maka MessageBox akan menampilkan pesan "Data tidak berhasil ditambahkan!".
- Menangani Error dengan catch:
- `catch (Exception ex)`  
Jika terjadi kesalahan saat menjalankan query atau membuka koneksi, error tersebut akan ditangani di sini.
  - `MessageBox.Show("Error: " + ex.Message, "Kesalahan",  
MessageBoxButtons.OK, MessageBoxIcon.Error);`  
Menampilkan MessageBox dengan pesan error yang sesuai.

```

//Fungsi untuk menghapus data (DELETE)
1 reference
private void BtnHapus(object sender, EventArgs e)
{
    if (dgvMahasiswa.SelectedRows.Count > 0)
    {
        DialogResult confirm = MessageBox.Show("Yakin ingin menghapus data ini?", "Konfirmasi", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
        if (confirm == DialogResult.Yes)
        {
            using (SqlConnection conn = new SqlConnection(connectionString))
            {
                try
                {
                    string nim = dgvMahasiswa.SelectedRows[0].Cells["NIM"].Value.ToString();
                    conn.Open();
                    using (SqlCommand cmd = new SqlCommand("DeleteMahasiswa", conn))
                    {
                        cmd.CommandType = CommandType.StoredProcedure;
                        cmd.Parameters.AddWithValue("@NIM", nim);

                        int rowsAffected = cmd.ExecuteNonQuery();
                        if (rowsAffected > 0)
                        {
                            MessageBox.Show("Data berhasil dihapus!", "Sukses", MessageBoxButtons.OK, MessageBoxIcon.Information);
                            LoadData();
                            ClearForm(); // o Auto Clear setelah hapus data
                        }
                        else
                        {
                            MessageBox.Show("Data tidak ditemukan atau gagal dihapus!", "Kesalahan", MessageBoxButtons.OK, MessageBoxIcon.Error);
                        }
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Error: " + ex.Message, "Kesalahan", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
        }
    }
    else
    {
        MessageBox.Show("Pilih data yang akan dihapus!", "Peringatan", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}

```

### Penjelasan:

- Deklarasi BtnTambah (Event Handler):  
 BtnTambah adalah metode yang akan dipanggil saat tombol Tambah diklik. Ini adalah event handler untuk event Click pada tombol BtnTambah.
- Membuka Koneksi ke Database:
  - `using (SqlConnection conn = new SqlConnection(connectionString))`  
 Membuka koneksi ke database menggunakan SqlConnection. Koneksi ini akan dipakai untuk mengirimkan query ke database. Koneksi menggunakan string koneksi yang sudah didefinisikan di connectionString.
  - using memastikan bahwa objek conn akan dibersihkan dan ditutup secara otomatis setelah selesai digunakan.
- Validasi Input:
  - `if (txtNIM.Text == "" || txtNama.Text == "" || txtEmail.Text == "" || txtTelepon.Text == "")`  
 Mengecek apakah ada kolom input yang kosong pada txtNIM, txtNama, txtEmail, atau txtTelepon.

- Jika ada kolom yang kosong, MessageBox akan muncul untuk memberi peringatan kepada pengguna dengan pesan "Harap isi semua data!" dan proses akan dihentikan dengan return.
- Membuka Koneksi dan Menyiapkan Stored Procedure:
  - `conn.Open();`  
Membuka koneksi yang telah didefinisikan sebelumnya.
  - `using (SqlCommand cmd = new SqlCommand("AddMahasiswa", conn))`  
Membuat perintah SqlCommand untuk mengeksekusi prosedur tersimpan (stored procedure) AddMahasiswa pada database. Stored procedure ini akan dijalankan di database.
  - `cmd.CommandType = CommandType.StoredProcedure;`  
Menentukan bahwa SqlCommand yang digunakan adalah stored procedure, bukan query biasa.
- Menambahkan Parameter untuk Stored Procedure:
  - `cmd.Parameters.AddWithValue("@NIM", txtNIM.Text.Trim());`  
Menambahkan parameter untuk stored procedure. Nilai parameter @NIM diambil dari txtNIM.Text, yang merupakan input yang dimasukkan pengguna ke dalam TextBox.
  - Trim() digunakan untuk menghapus spasi yang tidak diperlukan di awal dan akhir input.
  - Parameter lain (@Nama, @Email, @Telepon, @Alamat) ditambahkan dengan cara yang sama, mengambil nilai dari TextBox masing-masing.
- Menjalankan Stored Procedure:
  - `int rowsAffected = cmd.ExecuteNonQuery();`  
Menjalankan ExecuteNonQuery untuk mengeksekusi stored procedure yang telah disiapkan. Fungsi ini digunakan ketika query yang dijalankan tidak mengembalikan data (seperti SELECT), melainkan hanya mempengaruhi data (seperti INSERT, UPDATE, atau DELETE).
  - rowsAffected akan berisi jumlah baris yang terpengaruh oleh operasi (dalam hal ini, jumlah baris yang berhasil ditambahkan ke tabel Mahasiswa).
- Menangani Hasil Eksekusi:

- Jika rowsAffected lebih besar dari 0, berarti data berhasil ditambahkan ke dalam database. Maka, MessageBox dengan pesan "Data berhasil ditambahkan!" akan ditampilkan, dan proses LoadData() serta ClearForm() akan dijalankan untuk memperbarui tampilan dan mengosongkan kolom input.
  - Jika rowsAffected adalah 0 (tidak ada baris yang terpengaruh), maka MessageBox akan menampilkan pesan "Data tidak berhasil ditambahkan!".
- Menangani Error dengan catch:
- `catch (Exception ex)`  
Jika terjadi kesalahan saat menjalankan query atau membuka koneksi, error tersebut akan ditangani di sini.
  - `MessageBox.Show("Error: " + ex.Message, "Kesalahan", MessageBoxButtons.OK, MessageBoxIcon.Error);`  
Menampilkan MessageBox dengan pesan error yang sesuai.

```
//Fungsi untuk mengubah data (UPDATE)
1 reference
private void BtnUbah(object sender, EventArgs e)
{
    if (dgvMahasiswa.SelectedRows.Count > 0)
    {
        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            try
            {
                conn.Open();
                using (SqlCommand cmd = new SqlCommand("UpdateMahasiswa", conn))
                {
                    cmd.CommandType = CommandType.StoredProcedure;

                    // Menambahkan parameter untuk stored procedure
                    cmd.Parameters.AddWithValue("@NIM", txtNIM.Text.Trim());
                    cmd.Parameters.AddWithValue("@Nama", txtNama.Text.Trim());
                    cmd.Parameters.AddWithValue("@Email", txtEmail.Text.Trim());
                    cmd.Parameters.AddWithValue("@Telepon", txtTelepon.Text.Trim());
                    cmd.Parameters.AddWithValue("@Alamat", txtAlamat.Text.Trim());

                    int rowsAffected = cmd.ExecuteNonQuery();
                    if (rowsAffected > 0)
                    {
                        MessageBox.Show("Data berhasil diperbarui!", "Sukses", MessageBoxButtons.OK, MessageBoxIcon.Information);
                        LoadData();
                        ClearForm(); // Auto Clear setelah update data
                    }
                    else
                    {
                        MessageBox.Show("Data tidak ditemukan atau gagal diperbarui!", "Kesalahan", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message, "Kesalahan", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
    }
    else
    {
        MessageBox.Show("Pilih data yang akan diubah!", "Peringatan", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
```

### Penjelasan:

- BtnUbah - Event Handler untuk Tombol Ubah:

- BtnUbah adalah event handler yang menangani klik tombol Ubah.
- Ketika tombol Ubah ditekan, prosedur ini dijalankan untuk memperbarui data mahasiswa yang telah dipilih dari DataGridView.
- Memeriksa Baris yang Dipilih:
  - `if (dgvMahasiswa.SelectedRows.Count > 0):`  
 Memeriksa apakah ada baris yang dipilih di DataGridView (dgvMahasiswa). Jika ada baris yang dipilih, proses selanjutnya dilanjutkan untuk memperbarui data.
  - `SelectedRows.Count > 0` memastikan bahwa pengguna memilih baris di DataGridView sebelum melakukan operasi pembaruan.
- Membuka Koneksi ke Database:
  - `using (SqlConnection conn = new SqlConnection(connectionString)):`  
 Membuka koneksi ke database menggunakan SqlConnection yang terhubung dengan connectionString. Koneksi ini akan digunakan untuk menjalankan query SQL terhadap database.
  - `using` memastikan bahwa objek koneksi ditutup dan dibersihkan secara otomatis setelah selesai digunakan.
- Membuka Koneksi dan Menyiapkan Stored Procedure:
  - `conn.Open();`  
 Membuka koneksi ke database.
  - `using (SqlCommand cmd = new SqlCommand("UpdateMahasiswa", conn)):`  
 Membuat SqlCommand untuk menjalankan stored procedure UpdateMahasiswa di database.
  - `cmd.CommandType = CommandType.StoredProcedure;`  
 Menyatakan bahwa perintah yang akan dijalankan adalah stored procedure dan bukan query SQL biasa.
- Menambahkan Parameter untuk Stored Procedure:
  - `cmd.Parameters.AddWithValue("@NIM", txtNIM.Text.Trim());`  
 Menambahkan parameter @NIM ke SqlCommand, yang nilainya diambil dari txtNIM.Text (kolom NIM yang diinputkan oleh pengguna pada form).

- Trim() digunakan untuk menghapus spasi ekstra di awal dan akhir nilai input.
- Parameter lainnya (@Nama, @Email, @Telepon, @Alamat) ditambahkan dengan cara yang sama. Setiap nilai diambil dari kolom TextBox pada form dan digunakan untuk memperbarui data mahasiswa.
- Menjalankan Stored Procedure:
  - `int rowsAffected = cmd.ExecuteNonQuery();`  
Perintah ExecuteNonQuery digunakan untuk menjalankan stored procedure. Fungsi ini mengembalikan jumlah baris yang terpengaruh oleh operasi (dalam hal ini, jumlah baris yang berhasil diperbarui).
  - Jika rowsAffected lebih dari 0, berarti pembaruan data berhasil dilakukan.
- Menangani Hasil Eksekusi:
  - Jika rowsAffected > 0, berarti data berhasil diperbarui. Proses berikutnya:
    - `MessageBox.Show("Data berhasil diperbarui!", "Sukses", MessageBoxButtons.OK, MessageBoxIcon.Information);` Menampilkan MessageBox yang memberi tahu pengguna bahwa data berhasil diperbarui.
    - `LoadData();` Memanggil metode LoadData() untuk memperbarui tampilan DataGridView dengan data terbaru setelah pembaruan.
    - `ClearForm();` Memanggil metode ClearForm() untuk mengosongkan kolom input form agar siap untuk memasukkan data baru.
  - Jika rowsAffected <= 0, berarti pembaruan gagal dilakukan (kemungkinan karena NIM yang dimasukkan tidak ditemukan atau tidak ada perubahan). MessageBox akan menampilkan pesan kesalahan.
- Menangani Error dengan catch:
  - `catch (Exception ex)`  
Menangani kemungkinan kesalahan yang terjadi selama eksekusi query atau pembukaan koneksi.
  - `MessageBox.Show("Error: " + ex.Message, "Kesalahan", MessageBoxButtons.OK, MessageBoxIcon.Error);`



Jika terjadi error, pesan error akan ditampilkan kepada pengguna dengan detail kesalahan dari exception yang terjadi.

- Jika Tidak Ada Baris yang Dipilih:
  - else: Jika tidak ada baris yang dipilih di DataGridView, maka MessageBox akan muncul dengan pesan "Pilih data yang akan diubah!" untuk memberitahukan pengguna agar memilih baris yang ingin diubah.

## 6. code untuk PreviewForm.cs

```
4 references
public partial class PreviewForm : Form
{
    private string connectionString = "Data Source=DESKTOP-RAM20FI\\APRILIVA;Initial Catalog=OrganisasiMahasiswa;Integrated Security=True";

    // Konstruktor menerima DataTable dan menampilkan data di DataGridView
    1 reference
    public PreviewForm(DataTable data)
    {
        InitializeComponent();
        // Menetapkan data source DataGridView ke DataTable yang diterima
        dgvPreview.DataSource = data;
    }

    // Event ketika form dimuat
    1 reference
    private void PreviewForm_Load(object sender, EventArgs e)
    {
        // Optional: Sesuaikan DataGridView jika perlu
        dgvPreview.AutoResizeColumns(); // Menyesuaikan ukuran kolom
    }

    // Event ketika tombol OK ditekan
    1 reference
    private void button1_Click(object sender, EventArgs e)
    {
        // Menanyakan kepada pengguna jika mereka ingin mengimpor data
        DialogResult result = MessageBox.Show("Apakah Anda ingin mengimpor data ini ke database?", "Konfirmasi", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

        if (result == DialogResult.Yes)
        {
            // Mengimpor data dari DataGridView ke database
            ImportDataToDatabase();
        }
    }

    private bool ValidateRow(DataRow row)
    {
        string nim = row["NIM"].ToString();

        // Validasi NIM (misalnya, harus berjumlah 11 karakter)
        if (nim.Length != 11)
        {
            MessageBox.Show("NIM harus terdiri dari 11 karakter.", "Kesalahan Validasi", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return false;
        }

        // Jika perlu, tambahkan validasi lain sesuai dengan kebutuhan (misalnya, pola tertentu untuk NIM)

        return true;
    }
}
```

```

private void ImportDataToDatabase()
{
    try
    {
        DataTable dt = (DataTable)dgvPreview.DataSource;

        foreach (DataRow row in dt.Rows)
        {
            // Validasi setiap baris sebelum diimpor
            if (!ValidateRow(row))
            {
                // Jika validasi gagal, lanjutkan ke baris berikutnya
                continue; // Lewati baris ini jika tidak valid
            }

            string query = "INSERT INTO Mahasiswa (NIM, Nama, Email, Telepon, Alamat) VALUES (@NIM, @Nama, @Email, @Telepon, @Alamat)";
            using (SqlConnection conn = new SqlConnection(connectionString))
            {
                conn.Open();
                using (SqlCommand cmd = new SqlCommand(query, conn))
                {
                    cmd.Parameters.AddWithValue("@NIM", row["NIM"]);
                    cmd.Parameters.AddWithValue("@Nama", row["Nama"]);
                    cmd.Parameters.AddWithValue("@Email", row["Email"]);
                    cmd.Parameters.AddWithValue("@Telepon", row["Telepon"]);
                    cmd.Parameters.AddWithValue("@Alamat", row["Alamat"]);
                    cmd.ExecuteNonQuery();
                }
            }
        }

        MessageBox.Show("Data berhasil diimpor ke database.", "Sukses", MessageBoxButtons.OK, MessageBoxIcon.Information);
        this.Close(); // Tutup PreviewForm setelah data diimpor
    }
    catch (Exception ex)
    {
        MessageBox.Show("Terjadi kesalahan saat mengimpor data: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

## Penjelasan:

### 1. Deklarasi Kelas dan Koneksi ke Database

- PreviewForm adalah kelas Form yang digunakan untuk menampilkan pratinjau data dan memberikan pilihan untuk mengimpor data ke dalam database.
- connectionString adalah string koneksi yang digunakan untuk terhubung ke database Organisasi Mahasiswa menggunakan SQL Server. Ini mencakup nama server, nama database, dan metode keamanan yang digunakan untuk autentikasi.

### 2. Konstruktor PreviewForm

- Konstruktor menerima parameter DataTable yang berisi data yang akan dipratinjau.
- dgvPreview.DataSource = data;  
Menetapkan DataGridView (dgvPreview) sebagai sumber data dari DataTable yang diterima, sehingga data yang diberikan akan ditampilkan dalam DataGridView.

### 3. Event PreviewForm\_Load

- PreviewForm\_Load adalah event yang terjadi saat form pertama kali dimuat.
- dgvPreview.AutoSizeColumnsMode();

digunakan untuk menyesuaikan lebar kolom di DataGridView secara otomatis agar sesuai dengan panjang data yang ditampilkan.

#### **4. Event button1\_Click (Tombol OK)**

- button1\_Click adalah event handler untuk tombol OK (button1).
- Ketika tombol OK ditekan, akan muncul pesan konfirmasi menggunakan MessageBox.Show.
- Jika pengguna memilih Yes, maka ImportDataToDatabase akan dipanggil untuk mengimpor data ke database.

#### **5. Validasi Baris Data**

- ValidateRow adalah metode untuk memvalidasi setiap baris data dalam DataTable sebelum diimpor ke database.
- Dalam contoh ini, validasi dilakukan untuk kolom NIM. NIM harus memiliki panjang 11 karakter.
- Jika validasi gagal (misalnya, NIM tidak berjumlah 11 karakter), maka akan menampilkan pesan kesalahan dan mengembalikan nilai false. Jika validasi berhasil, mengembalikan nilai true.

#### **6. Mengimpor Data ke Database**

- ImportDataToDatabase adalah metode yang digunakan untuk mengimpor data dari DataGridView ke dalam database.
- DataTable dt = (DataTable)dgvPreview.DataSource;: Mengambil data yang ada di DataGridView (dgvPreview) dan mengonversinya menjadi DataTable.
- foreach (DataRow row in dt.Rows): Iterasi melalui setiap baris data dalam DataTable.
- Sebelum memasukkan data ke database, setiap baris divalidasi menggunakan ValidateRow. Jika baris tidak valid, proses continue akan dilewati dan baris tersebut tidak akan diproses lebih lanjut.
- INSERT INTO Mahasiswa: Perintah INSERT INTO digunakan untuk memasukkan data mahasiswa ke dalam tabel Mahasiswa.
  - ✓ cmd.Parameters.AddWithValue(...): Parameter untuk NIM, Nama, Email, Telepon, dan Alamat diambil dari setiap baris DataRow.

✓ `cmd.ExecuteNonQuery()`: Menjalankan perintah INSERT untuk memasukkan data ke dalam database.

- Setelah data berhasil dimasukkan, `MessageBox` akan memberi tahu pengguna bahwa data telah berhasil diimpor.
- `this.Close()`:: Menutup `PreviewForm` setelah data diimpor.

## 7. Menangani Error

Jika terjadi kesalahan selama proses impor data (misalnya koneksi ke database gagal atau ada masalah dengan query), akan ada `MessageBox` yang menampilkan pesan kesalahan dengan rincian error yang terjadi.

## 8. Tambahkan code pada `Form1.cs` (sesuaikan dengan nama form anda)

```
// Event untuk memilih file dan mempreview data
1 reference
private void button1_Click(object sender, EventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog();
    openFileDialog.Filter = "Excel Files|*.xlsx;*.xlsm";
    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        string filePath = openFileDialog.FileName;
        PreviewData(filePath); // Display preview before importing
    }
}
```

### Penjelasan:

1. `private void button1_Click(object sender, EventArgs e):`
  - merupakan event handler untuk tombol `button1`. Ketika tombol ini ditekan, metode `button1_Click` akan dijalankan.
  - Event handler ini digunakan untuk memilih file Excel dan mempratinjau data yang ada di dalamnya sebelum mengimpor ke database.
2. `OpenFileDialog openFileDialog = new OpenFileDialog();`
  - `OpenFileDialog` adalah sebuah objek yang menyediakan dialog untuk memilih file di komputer.
  - `openFileDialog` adalah instance dari `OpenFileDialog` yang akan digunakan untuk menampilkan jendela dialog untuk memilih file.
3. `openFileDialog.Filter = "Excel Files|*.xlsx;*.xlsm";`
  - Filter digunakan untuk membatasi jenis file yang ditampilkan dalam dialog pemilihan file.
  - Dalam hal ini, filter diatur untuk hanya menampilkan file dengan ekstensi `.xlsx` dan `.xlsm` (file Excel).

4. `if (openFileDialog.ShowDialog() == DialogResult.OK):`
  - `ShowDialog()` menampilkan dialog untuk memilih file.
  - Jika pengguna memilih file dan mengklik OK, maka `ShowDialog()` akan mengembalikan `DialogResult.OK`, dan blok kode di dalam `if` akan dijalankan.
  - Jika pengguna membatalkan atau menutup dialog, `ShowDialog()` akan mengembalikan nilai lain (bukan `DialogResult.OK`), dan kode di dalam `if` tidak akan dijalankan.
5. `string filePath = openFileDialog.FileName;;`
  - `FileName` adalah properti dari `openFileDialog` yang menyimpan path lengkap dari file yang dipilih oleh pengguna.
  - `filePath` menyimpan path file Excel yang dipilih, yang nantinya akan digunakan untuk membuka dan membaca data dari file tersebut.
6. `PreviewData(filePath);:`
  - Setelah file dipilih, metode `PreviewData` dipanggil dengan `filePath` sebagai parameter.
  - `PreviewData` adalah metode yang akan digunakan untuk membuka file Excel dan menampilkan data di dalamnya di form (misalnya, di dalam `DataGridView` untuk pratinjau).

```

// Method untuk menampilkan preview data di DataGridView
1 reference
private void PreviewData(string filePath)
{
    try
    {
        using (var fs = new FileStream(filePath, FileMode.Open, FileAccess.Read))
        {
            IWorkbook workbook = new XSSFWorkbook(fs); // Membuka workbook Excel
            ISheet sheet = workbook.GetSheetAt(0); // Mendapatkan worksheet pertama
            DataTable dt = new DataTable();

            // Membaca header kolom
            IRow headerRow = sheet.GetRow(0);
            foreach (var cell in headerRow.Cells)
            {
                dt.Columns.Add(cell.ToString());
            }

            // Membaca sisa data
            for (int i = 1; i <= sheet.LastRowNum; i++) // Lewati baris header
            {
                IRow dataRow = sheet.GetRow(i);
                DataRow newRow = dt.NewRow();
                int cellIndex = 0;
                foreach (var cell in dataRow.Cells)
                {
                    newRow[cellIndex] = cell.ToString();
                    cellIndex++;
                }
                dt.Rows.Add(newRow);
            }

            // Membuka PreviewForm dan mengirimkan DataTable ke form tersebut
            PreviewForm previewForm = new PreviewForm(dt);
            previewForm.ShowDialog(); // Tampilkan PreviewForm
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error reading the Excel file: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

### Penjelasan:

1. private void PreviewData(string filePath):
  - PreviewData adalah metode yang bertanggung jawab untuk membuka file Excel, membaca data, dan menampilkannya dalam PreviewForm (sebuah form yang menampilkan pratinjau data Excel).
  - Parameter filePath adalah path lengkap ke file Excel yang dipilih oleh pengguna, yang akan dibuka dan diproses.
2. Membuka File Excel:
  - FileStream digunakan untuk membuka file Excel dalam mode baca (FileAccess.Read).
  - XSSFWorkbook digunakan untuk membuka file Excel dengan format .xlsx (XSSF) menggunakan NPOI library.
  - workbook.GetSheetAt(0) mengakses worksheet pertama dalam workbook. Nilai 0 menunjukkan indeks sheet pertama (karena indeks dimulai dari 0).
3. Membaca header row

- `sheet.GetRow(0)` membaca baris pertama (header) dalam worksheet.
  - `foreach (var cell in headerRow.Cells)`: Setiap cell di baris header dibaca dan nilainya digunakan untuk membuat kolom dalam DataTable (`dt.Columns.Add`).
  - Dengan cara ini, kolom di DataTable akan dinamai sesuai dengan nilai yang ada di header file Excel.
4. Membaca data dari baris berikutnya:
- `for (int i = 1; i <= sheet.LastRowNum; i++)`: Mulai dari baris kedua (mengabaikan header yang ada di baris pertama), setiap baris akan dibaca.
  - `sheet.GetRow(i)`: Mengambil data dari baris ke-i.
  - `DataRow newRow = dt.NewRow();`: Membuat baris baru untuk DataTable.
  - `foreach (var cell in dataRow.Cells)`: Untuk setiap cell dalam baris, nilai sel tersebut ditambahkan ke DataRow baru.
  - `newRow[cellIndex] = cell.ToString();`: Nilai setiap cell diubah menjadi string dan dimasukkan ke dalam kolom yang sesuai di DataRow.
  - `dt.Rows.Add(newRow);`: Menambahkan DataRow ke dalam DataTable.
5. Menampilkan Preview dalam Form:
- Setelah semua data dibaca dan dimasukkan ke dalam DataTable, sebuah instance dari PreviewForm dibuat, dan DataTable yang berisi data dari Excel diteruskan sebagai parameter.
  - `previewForm.ShowDialog();` menampilkan form PreviewForm sebagai dialog modal. Form ini akan menampilkan data yang telah dipratinjau.
6. Penanganan Error:
- Jika ada kesalahan saat membaca file Excel (misalnya file tidak ditemukan, format tidak valid, dll.), pengecualian akan ditangkap oleh blok catch.
  - `MessageBox.Show` akan menampilkan pesan error yang memberikan informasi tentang masalah yang terjadi.

```

// Event ketika form pertama kali dimuat
1 reference
private void Form1_Load(object sender, EventArgs e)
{
    LoadData();
}

// Fungsi untuk menampilkan data di DataGridView
5 references
private void LoadData()
{
    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        try
        {
            conn.Open();
            string query = "SELECT NIM AS [NIM], Nama, Email, Telepon, Alamat FROM Mahasiswa";
            SqlDataAdapter da = new SqlDataAdapter(query, conn);
            DataTable dt = new DataTable();
            da.Fill(dt);

            dgvMahasiswa.AutoGenerateColumns = true;
            dgvMahasiswa.DataSource = dt;
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error: " + ex.Message, "Kesalahan", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
}

```

### Penjelasan:

#### 1. Event Form1\_Load

- Form1\_Load adalah event handler yang dipicu saat form pertama kali dimuat (ketika aplikasi dibuka atau form ditampilkan).
- Dalam event ini, LoadData() dipanggil untuk mengisi DataGridView dengan data dari database.
- sender adalah objek yang memicu event (dalam hal ini adalah form itu sendiri), dan e berisi data event (misalnya, informasi mengenai form yang dimuat).

#### 2. Fungsi LoadData

##### a. using (SqlConnection conn = new SqlConnection(connectionString)):

- Membuka koneksi ke database menggunakan SqlConnection.
- connectionString berisi informasi yang diperlukan untuk menghubungkan aplikasi dengan database, seperti nama server, nama database, dan metode autentikasi yang digunakan.
- using memastikan bahwa conn akan ditutup dan dibersihkan secara otomatis setelah blok kode selesai dieksekusi, sehingga koneksi tidak akan terbuka lebih lama dari yang diperlukan.

##### b. conn.Open();:



- Open() digunakan untuk membuka koneksi ke database.
  - Tanpa conn.Open(), tidak ada operasi database yang dapat dilakukan.
- c. `string query = "SELECT NIM AS [NIM], Nama, Email, Telepon, Alamat FROM Mahasiswa";`
- query berisi SQL query yang akan dieksekusi pada database.
  - Query ini memilih kolom NIM, Nama, Email, Telepon, dan Alamat dari tabel Mahasiswa.
  - Alias AS [NIM] digunakan untuk memastikan nama kolom tetap sesuai dengan yang diinginkan di hasil query.
- d. `SqlDataAdapter da = new SqlDataAdapter(query, conn);`
- SqlDataAdapter adalah objek yang digunakan untuk mengambil data dari database dan mengisi objek DataTable dengan hasil query.
  - `da.Fill(dt);` akan menjalankan query, mendapatkan data dari database, dan mengisinya ke dalam DataTable (dt).
- e. `DataTable dt = new DataTable();`
- DataTable adalah objek yang menyimpan data dalam bentuk tabel (baris dan kolom) yang bisa diakses dengan mudah.
  - Di sini, dt digunakan untuk menyimpan data yang diambil dari query SQL.
- f. `da.Fill(dt);`
- Fill digunakan untuk mengisi DataTable (dt) dengan hasil dari query yang dijalankan oleh SqlDataAdapter.
- g. `dgvMahasiswa.AutoGenerateColumns = true;`
- AutoGenerateColumns adalah properti dari DataGridView (dalam hal ini dgvMahasiswa) yang menentukan apakah kolom-kolom di DataGridView akan otomatis dibuat berdasarkan nama kolom dalam DataTable.
  - Dengan AutoGenerateColumns = true, kolom-kolom dalam DataGridView akan dibuat otomatis sesuai dengan kolom yang ada di DataTable yang telah diisi.
- h. `dgvMahasiswa.DataSource = dt;`
- Menetapkan DataGridView (dgvMahasiswa) untuk menampilkan data yang ada di dalam DataTable (dt).

- Data dari DataTable akan ditampilkan dalam DataGridView, sehingga pengguna dapat melihat daftar mahasiswa yang ada di dalam tabel Mahasiswa di database.
- i. catch (Exception ex):
- Blok catch digunakan untuk menangani pengecualian (error) yang mungkin terjadi selama eksekusi query.
  - Jika terjadi kesalahan (misalnya, koneksi ke database gagal atau query tidak valid), maka pesan kesalahan akan ditampilkan kepada pengguna.
- j. `MessageBox.Show("Error: " + ex.Message, "Kesalahan", MessageBoxButtons.OK, MessageBoxIcon.Error);`
- Jika terjadi pengecualian, pesan MessageBox akan menampilkan detail pesan kesalahan yang terjadi selama eksekusi query.
  - `ex.Message` berisi deskripsi kesalahan yang terjadi.

Hasil akhir :

### 1. Tampilan form 1

NIM	Nama	Email
20140140001	Krisna Nuresa Go...	Krisna.Nuresa.2
20180140076	Rafa R	Rafa@umy.ac.id
20210000000	Davit Putra Harto...	Davit2@umy.ac
20210000002	Alwan Fauzi Wah...	Alwan2@umy.ac
20210000003	Dina Wulan Ratn...	Dina2@umy.ac
20210140001	Beatrix Devanti A...	Beatrix@umy.ac
20210140002	Aldi Septiyanto	Aldi@umy.ac.id
20210140003	Bagus Satrio Sa...	Bagus@umy.ac
20210140004	Ghani Zumar Ra...	Ghani@umy.ac
20210140005	Nalendra Satria S...	Nalendra@umy
20210140006	Arfan Pridaksana...	Arfan@umy.ac.id

### 2. Tampilan form 2 setelah menekan tombol «import Data»

Preview Data

	NIM	Nama	Email	Telepon	Alamat
▶	20250120099	Icis	icis@umy.ac.id	08789876789	Sleman
*					

OK