Bab 6 | Database Connection

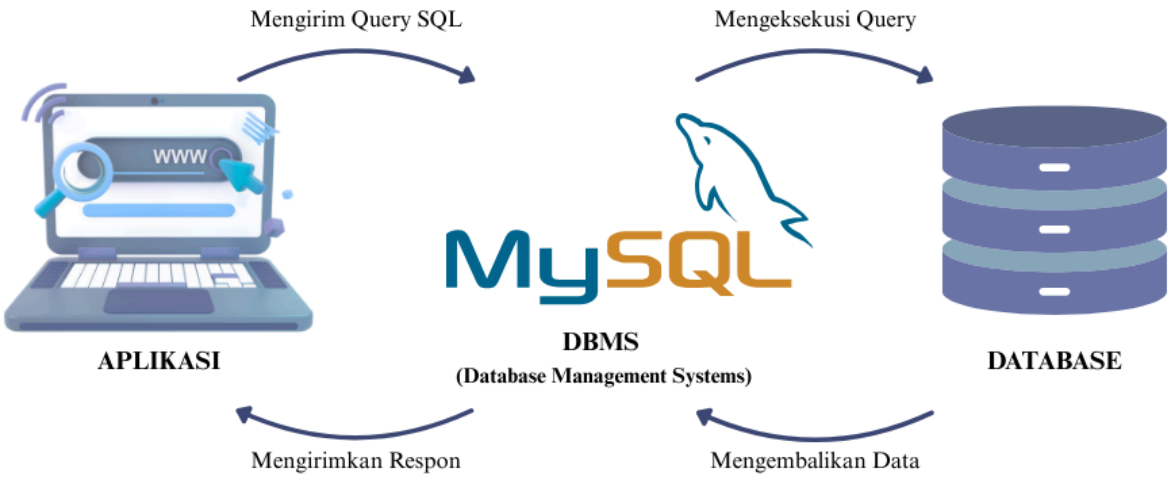
6.1 Pengenalan Database MySQL

6.1.1 Apa itu Database & DBMS?

Bayangkan sebuah perpustakaan besar yang terorganisir dengan baik. Perpustakaan itu adalah Database, sebuah kumpulan data yang disimpan secara sistematis dan terstruktur. Data ini biasanya diatur dalam bentuk tabel-tabel yang saling berhubungan, mirip seperti rak-rak buku yang rapi.

Nah, untuk menemukan, menambah, atau mengatur buku di perpustakaan, Anda memerlukan seorang pustakawan. Pustakawan inilah yang kita sebut DBMS (Database Management System).

Secara sederhana, DBMS adalah perangkat lunak (software) yang berfungsi sebagai perantara antara pengguna (atau aplikasi) dengan database. Ia yang menerjemahkan perintah kita agar database bisa mengerti. MySQL adalah salah satu contoh DBMS yang paling populer di dunia.



6.1.2 Kenapa MySQL Populer di Aplikasi Web?

MySQL menjadi favorit di kalangan pengembang web karena beberapa alasan kuat:

- Open-Source & Gratis: Kamu bisa menggunakannya tanpa biaya lisensi.
- Cepat dan Ringan: Dikenal memiliki performa tinggi bahkan untuk aplikasi berskala besar.
- Dukungan Luas: Hampir semua penyedia hosting mendukung MySQL, dan komunitasnya sangat besar, jadi mudah mencari solusi jika ada masalah.

- Integrasi Sempurna dengan PHP: PHP dan MySQL sering disebut sebagai "pasangan duet" dalam pengembangan web karena kemudahan integrasinya.

6.2 Menggunakan Fungsi MySQLi untuk Koneksi

Untuk mulai berkomunikasi dengan MySQL, langkah pertama adalah membuat koneksi. Ada dua cara populer menggunakan MySQLi: Prosedural dan Object-Oriented (OOP).

6.2.1 Menggunakan `mysqli_connect()` (Prosedural)

Ini adalah cara yang paling umum dan mudah dipahami untuk pemula. Kita menggunakan satu fungsi utama, `mysqli_connect()`, untuk membuat koneksi.

Parameter:

- `$hostname`: Alamat server database (biasanya `localhost`).
- `$username`: Nama pengguna database (default `root`).
- `$password`: Kata sandi pengguna (kosongkan jika tidak ada).
- `$database`: Nama database yang ingin digunakan.

```php
<?php

// Parameter koneksi database
$hostname = "localhost";
$username = "root";
$password = "";
$database = "akademik_db"; // Ganti dengan nama databasemu

// Membuat koneksi
$conn = mysqli_connect($hostname, $username, $password, $database);

// Memeriksa koneksi
if (!$conn) {
    // Jika koneksi gagal, hentikan script dan tampilkan pesan error
    die("Koneksi gagal: " . mysqli_connect_error());
}

?>
```

Catatan : Fungsi `die()` akan menghentikan eksekusi script dan menampilkan pesan. Ini sangat penting untuk debugging awal.

6.2.2 Menggunakan OOP (`new mysqli`)

Pendekatan Object-Oriented Programming (OOP) lebih modern dan terstruktur. Kita membuat sebuah *objek* koneksi dari *class* `mysqli`.

```php
<?php

// Parameter koneksi database
$hostname = "localhost";
$username = "root";
$password = "";
$database = "akademik_db"; // Ganti dengan nama databasemu

// Membuat objek koneksi baru
$conn = new mysqli($hostname, $username, $password, $database);

// Memeriksa koneksi
if ($conn->connect_error) {
    // Jika koneksi gagal, hentikan script dan tampilkan pesan error
    die("Koneksi gagal: " . $conn->connect_error);
}

?>
```

6.3 Menjalankan Query SQL melalui PHP

Setelah koneksi berhasil, kita bisa mengirimkan perintah SQL (disebut *query*) ke database menggunakan fungsi `mysqli_query()`.

Sintaks Dasar: `mysqli_query($koneksi, "PERINTAH_SQL_ANDA");`

Contoh-contoh Query:

- **`INSERT` (Menambah Data)**

```php
<?php
require 'koneksi.php'; // Hubungkan ke database

$sql = "INSERT INTO mahasiswa (nama, nim, jurusan) VALUES ('Liayra', '123456',
'Sistem Informasi')";

if (mysqli_query($conn, $sql)) {
```

```
    echo "Data baru berhasil ditambahkan!";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

- **`SELECT` (Mengambil Data)**

```
<?php

require 'koneksi.php'; // Hubungkan ke database

$sql = "SELECT id, nama, jurusan FROM mahasiswa";

$result = mysqli_query($conn, $sql);

?>
```

- **`UPDATE` (Mengubah Data)**

```
<?php
require 'koneksi.php'; // Hubungkan ke database

$sql = "UPDATE mahasiswa SET jurusan='Sistem Informasi' WHERE id=1";

if (mysqli_query($conn, $sql)) {
    echo "Data berhasil diupdate!";
} else {
    echo "Error updating record: "  . mysqli_error($conn);
}
?>
```

- **`DELETE` (Menghapus Data)**

```
<?php
require 'koneksi.php'; // Hubungkan ke database

$sql = "DELETE FROM mahasiswa WHERE id=3";

if (mysqli_query($conn, $sql)) {
    echo "Data berhasil dihapus!";
```

```
} else {
    echo "Error deleting record: "  . mysqli_error($conn);
}
?>
```

6.4 Mencegah Serangan SQL Injection dengan Prepared Statement

6.4.1 Apa itu SQL Injection?

SQL Injection adalah teknik serangan di mana penyerang menyisipkan kode SQL berbahaya ke dalam input aplikasi (misalnya, form login atau pencarian). Jika aplikasi tidak aman, kode jahat ini akan dieksekusi oleh database.

Contoh Sederhana: Bayangkan query login: `SELECT * FROM users WHERE username = '$username' AND password = '$password'`. Jika penyerang memasukkan username `' OR '1'='1`, query-nya akan menjadi: `SELECT * FROM users WHERE username = '' OR '1'='1' AND password = '...'` . Karena `'1'='1'` selalu benar, query ini akan mengembalikan semua user dan penyerang bisa login tanpa password!

6.4.2 Prepared Statements

Prepared Statements adalah cara paling efektif untuk mencegah SQL Injection. Konsepnya memisahkan perintah SQL dari data yang diinput user, sehingga input selalu dianggap data biasa, bukan bagian dari perintah SQL.

Alur Kerjanya:

1. **Prepare:** Kirim "template" query SQL ke database dengan penanda (`?`) untuk data.
2. **Bind:** Ikat variabel PHP yang berisi data user ke penanda tersebut, dan tentukan tipe datanya (string, integer, dll).
3. **Execute:** Jalankan query yang sudah aman.

- **Contoh `INSERT` (Menambah Data) dengan Prepared Statement:**

```
<?php
require 'koneksi.php';

// 1. Prepare: Buat template query dengan penanda (?)
$sql = "INSERT INTO mahasiswa (nama, nim, jurusan) VALUES (?, ?, ?)";
$stmt = mysqli_prepare($conn, $sql);
```

```php
// Data dari user
$nama_mahasiswa = "Liayra";
$nim_mahasiswa = "654321";
$jurusan_mahasiswa = "Sistem Informasi";

// 2. Bind
mysqli_stmt_bind_param($stmt, "sss", $nama_mahasiswa, $nim_mahasiswa,
$jurusan_mahasiswa);

// 3. Execute
if (mysqli_stmt_execute($stmt)) {
    echo "Data mahasiswa baru berhasil ditambahkan dengan aman!";
} else {
    echo "Error: " . mysqli_stmt_error($stmt);
}

mysqli_stmt_close($stmt);
mysqli_close($conn);
?>
```

Kunci Keamanan: Metode ini aman karena perintah SQL (`INSERT...`) dikirim dan "dikunci" terlebih dahulu oleh database. Data (`Ariana Grande`, dll.) dikirim sesudahnya dan hanya dianggap sebagai teks biasa, bukan bagian dari perintah. Kode SQL jahat pun tidak akan pernah dieksekusi.

Memahami Tipe Data (`"sss"`): Kode pada `mysqli_stmt_bind_param()` wajib ada untuk menentukan tipe data. Kode ini dipakai untuk *"mengikat"* variabel PHP ke placeholder `?` dalam query Prepared Statement.

- `s` - untuk string (teks)
- `i` - untuk integer (angka bulat)
- `d` - untuk double (angka desimal)
- `b` - untuk blob (file/data biner)

Catatan : (`"sss"`) pada contoh kode di atas menunjukkan bahwa ketiga variabel (`$nama_mahasiswa`, `$nim_mahasiswa`, `$jurusan_mahasiswa`) semuanya bertipe string (teks).

Pentingnya `mysqli_stmt_close()`: Menutup *statement* akan langsung membebaskan memori di server database. Ini adalah praktik yang baik untuk menjaga performa aplikasi, terutama yang memiliki banyak pengunjung.

- **Contoh `SELECT` (Mengambil Data) dengan Prepared Statement:**

```php
<?php
require 'koneksi.php';

// Data pencarian dari user
$jurusan_dicari = "Sistem Informasi";

// 1. Prepare
$sql = "SELECT id, nama, nim FROM mahasiswa WHERE jurusan = ?";
$stmt = mysqli_prepare($conn, $sql);

// 2. Bind
mysqli_stmt_bind_param($stmt, "s", $jurusan_dicari);

// 3. Execute
mysqli_stmt_execute($stmt);

// 4. Get Result
$result = mysqli_stmt_get_result($stmt);

while ($row = mysqli_fetch_assoc($result)) {
    echo "ID: " . $row['id'] . " - Nama: " . $row['nama'] . " - NIM: " . $row['nim'] . "<br>";
}

mysqli_stmt_close($stmt);
mysqli_close($conn);
?>
```

Peran `mysqli_stmt_get_result()`: Setelah dieksekusi, hasil dari query `SELECT` masih "tersimpan" di dalam statement. Fungsi `mysqli_stmt_get_result()` menarik hasil query SELECT agar bisa diproses dengan `mysqli_fetch_assoc()`.

Aturan Praktis: Setiap kali ada variabel atau input dari pengguna yang masuk ke dalam query SQL (terutama di klausa `WHERE`, `ORDER BY`, atau `LIMIT`), wajib menggunakan Prepared Statements.

6.5 Mengelola Hasil Query

Setelah menjalankan query `SELECT`, kita perlu mengambil dan menampilkan datanya.

- `mysqli_num_rows($result)`: Fungsi ini digunakan untuk menghitung jumlah baris data yang dihasilkan oleh query. Berguna untuk mengecek apakah data ditemukan atau tidak.

- `mysqli_fetch_assoc($result)`: Fungsi ini mengambil satu baris data dari hasil query dan mengubahnya menjadi *associative array* (indeksnya berupa nama kolom). Setiap kali dipanggil, ia akan berpindah ke baris berikutnya.

Contoh Menampilkan Data dalam Tabel HTML

```
<?php
require 'koneksi.php';

$sql = "SELECT id, nama, nim, jurusan FROM mahasiswa ORDER BY nama ASC";
$result = mysqli_query($conn, $sql);
?>

<!DOCTYPE html>
<html>
<head>
    <title>Daftar Mahasiswa</title>
    <style>
        table, th, td { border: 1px solid black; border-collapse: collapse; padding: 8px; }
        th { background-color: #f2f2f2; }
    </style>
</head>
<body>

    <h2>Data Mahasiswa</h2>

    <?php
    // Cek apakah ada data yang ditemukan
    if (mysqli_num_rows($result) > 0) {
        echo "<table>";
        echo "<tr><th>ID</th><th>Nama
Mahasiswa</th><th>NIM</th><th>Jurusan</th></tr>";

        // Looping untuk menampilkan setiap baris data
        while ($row = mysqli_fetch_assoc($result)) {
            echo "<tr>";
            echo "<td>" . $row['id'] . "</td>";
            echo "<td>" . $row['nama'] . "</td>";
            echo "<td>" . $row['nim'] . "</td>";
            echo "<td>" . $row['jurusan'] . "</td>";
            echo "</tr>";
        }
```

```
        echo "</table>";
    } else {
        echo "Tidak ada data mahasiswa.";
    }

    // Tutup koneksi setelah selesai
    mysqli_close($conn);
    ?>

</body>
</html>
```

6.6 Menutup Koneksi ke Database

Sangat penting untuk selalu menutup koneksi database setelah semua pekerjaan selesai. Kenapa?

- **Menghemat Sumber Daya:** Setiap koneksi yang terbuka memakan memori di server. Menutupnya akan melepaskan memori tersebut.
- **Keamanan:** Mengurangi risiko koneksi disalahgunakan.

Caranya sangat mudah, cukup panggil fungsi `mysqli_close($conn);`.

```
<?php
// ... semua kode PHP Anda di sini ...

// Baris terakhir sebelum script selesai
mysqli_close($conn);
?>
```

6.7 Pembuatan Aplikasi Web Sederhana

6.7.1 Membangun Formulir Pendaftaran Pengguna

`register.html`

```
<!DOCTYPE html>
<html>
 <head>
  <title>Formulir Pendaftaran Pengguna</title>
 </head>
 <body>
  <h1>Formulir Pendaftaran Pengguna</h1>
  <form method="post" action="daftar.php">
   <label for="username">Username:</label>
   <input type="text" id="username" name="username" required>
   <label for="password">Password:</label>
   <input type="password" id="password" name="password" required>

   <button type="submit">Daftar</button>
  </form>
 </body>
</html>
```

6.7.2 Validasi Data dan Penyimpanan ke Database

`daftar.php`

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $username = $_POST["username"];
  $password = password_hash($_POST["password"], PASSWORD_DEFAULT);

  // Lakukan validasi data (misalnya, cek apakah username sudah ada)
  // Jika data valid, simpan data ke database

  $server = "localhost";
  $db_username = "root";
  $db_password = "";
  $db_name = "nama_database";

  $conn = new mysqli($server, $db_username, $db_password, $db_name);

  if ($conn->connect_error) {
```

```
        die("Koneksi gagal: " . $conn->connect_error);
    }

    $sql = "INSERT INTO pengguna (username, password) VALUES ('$username',
'$password')";

    if ($conn->query($sql) === TRUE) {
        echo "Pendaftaran berhasil!";
    } else {
        echo "Error: " . $sql . "<br>" . $conn->error;
    }

    $conn->close();
}

?>
```

6.7.3 Membuat Halaman Masuk (Login) dan Sistem Autentikasi Sederhana

`login.html`

```
<!DOCTYPE html>
<html>
  <head>
    <title>Formulir Masuk</title>
  </head>
  <body>
    <h1>Formulir Masuk</h1>
    <form method="post" action="masuk.php">
      <label for="username">Username:</label>
      <input type="text" id="username" name="username" required>
      <label for="password">Password:</label>
      <input type="password" id="password" name="password" required>

      <button type="submit">Masuk</button>
    </form>
</body>
</html>
```

masuk.php

```php
<?php
session_start();
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $username = $_POST["username"];
  $password = $_POST["password"];

  // Saring data pengguna dari database dan verifikasi kata sandi

  $server = "localhost";
  $db_username = "root";
  $db_password = "";
  $db_name = "nama_database";

  $conn = new mysqli($server, $db_username, $db_password, $db_name);

  if ($conn->connect_error) {
    die("Koneksi gagal: " . $conn->connect_error);
  }

  $sql = "SELECT username, password FROM pengguna WHERE
username='$username'";
  $result = $conn->query($sql);

  if ($result->num_rows == 1) {
    $row = $result->fetch_assoc();
    if (password_verify($password, $row["password"])) {
      $_SESSION["username"] = $username;
      header("Location: halaman_pengguna.php");
      exit();
    } else {
      echo "Autentikasi gagal. Silakan coba lagi.";
    }
  } else {
    echo "Autentikasi gagal. Silakan coba lagi.";
  }

  $conn->close();
}

?>
```

6.7.4 Mengimplementasikan Halaman Pengguna dengan Akses Terbatas

`halaman_pengguna.html`

```php
<?php
session_start();
if (isset($_SESSION["username"])) {
    $username = $_SESSION["username"];
    echo "Halo, $username! Ini halaman pengguna.";
} else {
    echo "Anda harus masuk terlebih dahulu.";
}
?>
```