

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/332753019>

PHP PDO MYSQL

Book · September 2017

CITATIONS

0

READS

266

1 author:



[Oya Suryana](#)

Universitas Kuningan

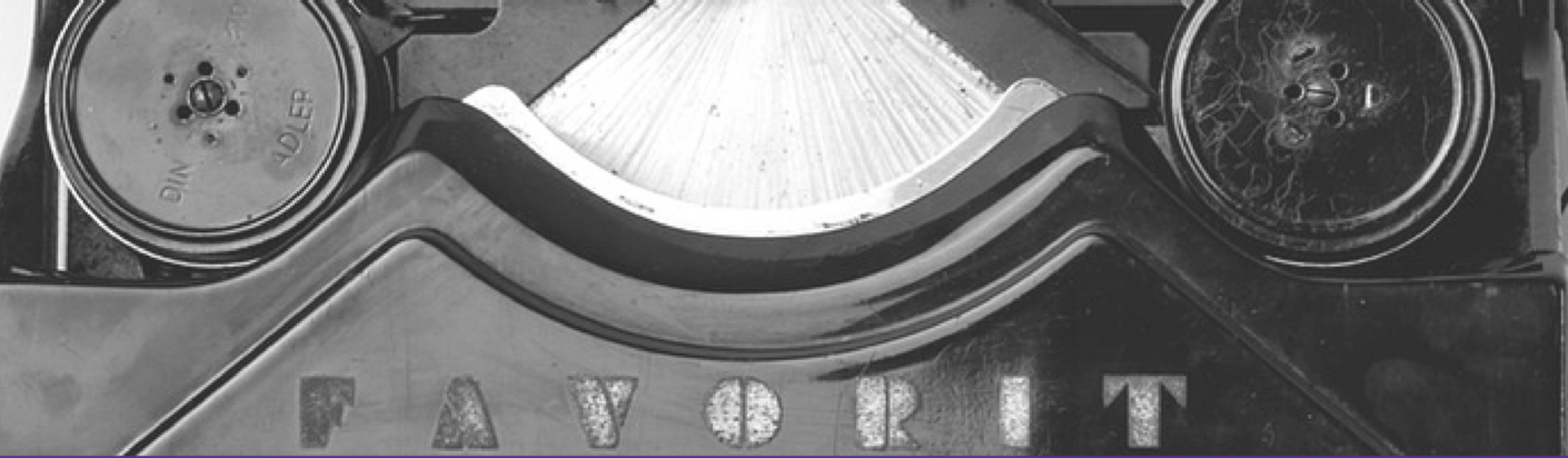
4 PUBLICATIONS 0 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



PHP PDO MySQL [View project](#)



PHP PDO MYSQL



think
write
run

OYA SURYANA, M.KOM

Pengantar

Alhamdulillah penulis panjatkan kehadirat Allah SWT yang telah memberikan kesehatan sehingga penulis bisa menyelesaikan buku panduan penggunaan PHP PDO MYSQL.

Buku panduan ini ditujukan bagi para pelajar dan mahasiswa serta sebagai panduan bagi para praktisi yang banyak bergelut dengan database dalam mengembangkan aplikasi berbasis web. Buku ini secara struktur merupakan terjemahan dari panduan dari website phpdelusions.net tentang php pdo, namun penulis susun ulang dengan memberikan contoh yang sedikit berbeda, namun tidak merubah esensi tujuan dari pada pembelajaran PHP PDO.

Buku panduan ini juga penulis persembahkan untuk :

1. Istri tercinta, Rika Widianingsih, S.P.d.
2. Anak-anaku Muhammad Ridwan Farhan, Abiq Sabiqul Khoir dan Zaki Nur Fatah
3. Rekan-rekan kerja tenaga pengajar di Program Keahlian Rekayasa Perangkat Lunak SMK Negeri 2 Kuningan
4. Peserta didik di Program Keahlian Rekayasa Perangkat Lunak SMK Negeri 2 Kuningan

Akhir kata penulis ucapkan terima kasih atas kesempatannya untuk menggunakan buku panduan ini dalam pembelajaran maupun pengembangan project aplikasi, mohon maaf apabila dalam penyusunan buku ini terdapat banyak kesalahan.

Kuningan, Oktober 2017
Penulis

Oya Suryana, M.Kom.

Daftar Isi

Pengantar.....	i
Daftar Isi.....	ii
Bahan dan Alat.....	iv
A. Mengapa PDO ?.....	1
B. Kesalahan para pemula !.....	1
C. Kelebihan PDO.....	1
D. Koneksi ke MySQL	2
E. Menjalankan Query dengan PDO::query()	4
F. Prepared statements. Perlindungan dari SQL injections.....	4
G. Metode Binding	6
H. Bagian-Bagian Query Yang Bisa Dibinding.....	8
I. Prepared Statement Untuk Multi Eksekusi	8
J. Menjalankan perintah, INSERT, UPDATE atau DELETE.....	9
a. DELETE	10
b. INSERT	11
c. SELECT	12
d. UPDATE	13
K. Mengambil Data Dengan Fungsi foreach().....	14
L. Mengambil Data Dengan Fungsi Fetch.....	15
M. Mengambil Data Dengan Fungsi fetchColumn()	21
N. Mengambil Data Dengan Format Lain Yang Berbeda, (fetchAll).....	22
a. Mendapatkan data berupa array.....	23
b. Mendapatkan data berupa array satu dimensi	24
c. Mendapatkan data berupa Pasangan Key=>Value	24
d. Mendapatkan baris data yang diindex dengan field unik.....	25

e. Mendapatkan baris data yang digrup berdasarkan field tertentu	27
O. Mengambil Satu Record Data (Single Row Fetch)	28
P. Menghitung Jumlah Data Dengan PDO	30
Q. Affected Rows Dan Insert id	31
R. Prepared Statement dalam Klausa Like	32
S. Prepared Statement dalam Klausa IN	33
T. Permasalahan dengan Klausa LIMIT	37
U. Transactions	40
V. Menjalankan Stored Procedures di PDO	42
W. Penanganan Error dengan Exceptions	44
Daftar Pustaka	
Penulis	

Bahan dan Alat

Dalam mempelajari buku panduan ini penulis siapkan database dan file-file php dan file SQL yang bisa anda download di blog milik penulis yaitu di :

<http://ozs.web.id/download/>

A. Mengapa PDO ?

Hal pertama yang harus dipikirkan adalah mengapa harus PDO ? perlu anda ketahui fungsi-fungsi mysql dalam php sekarang sudah ketinggalan zaman dan out of date alias sudah tidak didukung lagi oleh PHP. Fungsi mysql sudah tidak mendukung konsep modern sebuah database seperti *prepared statement*, *stored procedures*, *transaction*, dan lain-lain meskipun mysql menyediakan metode escaping paramater seperti *mysql_real_escape_string* dan metode concate sql string merupakan metode yang sudah ketinggalan zaman. Masalah lainnya adalah fungsi mysql sudah tidak mendapat perhatian lagi dari pengembang hal ini berakibat pada rentannya keamanan database atau bahkan sudah tidak dapat digunakan lagi pada MySQL versi terbaru (dan hal ini terjadi di PHP versi 7 bahkan 5 yang sudah tidak mensupport mysql function).

B. Kesalahan para pemula !

Berdasarkan pengalaman penulis banyak beberapa yang sedang belajar PHP lebih memilih PHP versi lama yang tetap mensupport MySQL Function (bahkan dosen, guru dan praktisi) dengan alasan script-scriptnya tidak jalan ketika menjalankan proses CRUD dengan MySQL. Hal ini terlihat mereka lebih menyenangi XAMPP versi 1.2.7 daripada menginstall XAMPP versi terbaru dimana PHP nya sudah tidak support lagi MySQL Function tetapi beralih ke MySQLi dan PDO. Jika anda tidak tertarik dengan PHP PDO maka MySQLi Function menjadi pilihan anda yang sedang belajar dari pada memaksakan membangun aplikasi web yang rentan karena masalah tidak di-support-nya lagi MySQL Function.

C. Kelebihan PDO

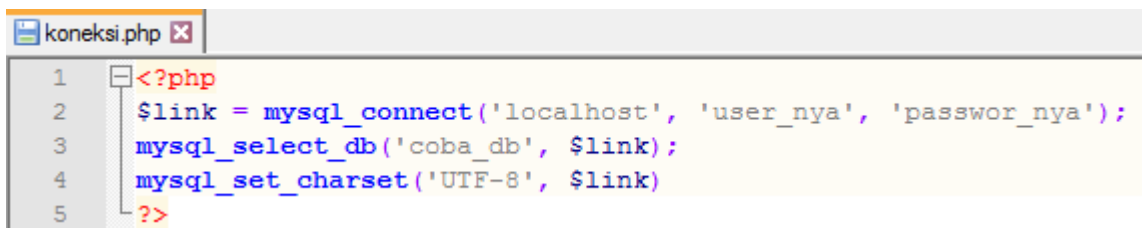
PDO memiliki antarmuka yang jauh lebih bagus, Anda akan menjadi lebih produktif, kode yang lebih *secure* dan *clean*. PDO juga memiliki driver yang berbeda untuk vendor database SQL yang berbeda yang memungkinkan Anda dengan mudah menggunakan vendor lain tanpa harus mempelajari kembali antarmuka yang berbeda, maksudnya dengan sekali membuat query maka query itu bisa untuk MySQL, Postgree, Oracle atau Database yang lain tanpa mengubah query (meskipun Anda harus belajar sedikit berbeda mungkin SQL) yang harus anda lakukan adalah mengubah database driver-nya. Pada Fungsi MySQL untuk mengamankan query adalah dengan menggabungkan string yang lolos setelah pengecekan, namun di PDO cukup dengan mem-*binding* parameter yang merupakan cara yang lebih mudah dan aman untuk mengamankan query. Binding parameter memungkinkan peningkatan kinerja saat memanggil query SQL yang sama berkali-kali dengan parameter yang sedikit berbeda. PDO juga memiliki banyak metode penanganan error. Masalah terbesar yang ada pada fungsi mysql_* adalah tidak memiliki penanganan yang konsisten, atau tidak ada penanganan sama sekali ! Dengan PDO dengan menggunakan *exception mode*, Anda bisa

mendapatkan penanganan kesalahan yang konsisten yang akan menghemat banyak waktu untuk melacak masalah.

PHP terbaru secara default langsung mengaktifkan PDO, namun Anda memerlukan dua ekstensi untuk dapat menggunakan PDO: PDO, dan driver untuk database yang ingin Anda gunakan seperti php_pdo_firebird jika akan mengakses firebird, php_pdo_mysql jika akan mengakses MySQL, php_pdo_odbc jika akan mengakses mysql melalui ODB, php_pdo_pgsql dan lainnya yang bisa anda konfigurasi ulang di file php.ini (file konfigurasi php).

D. Koneksi ke MySQL

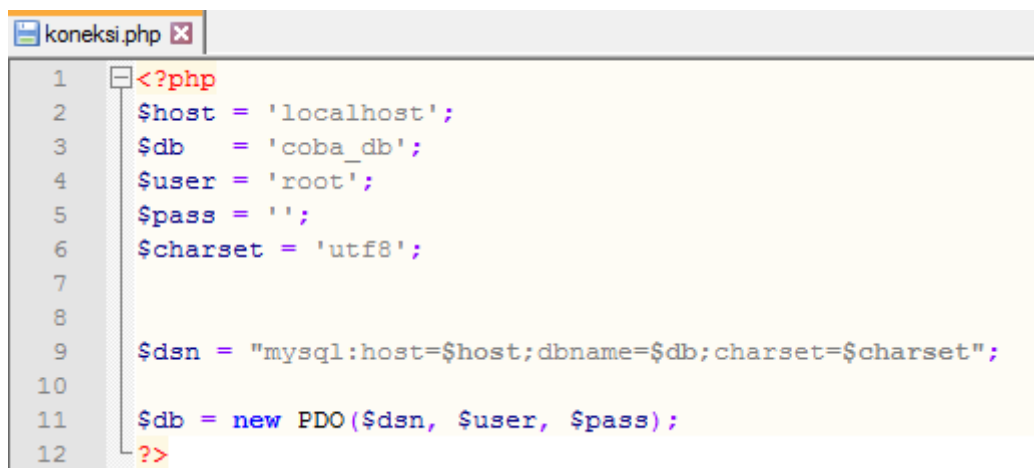
Langkah pertama untuk berkomunikasi dengan database adalah melakukan koneksi, cara lama dengan mysql function untuk membuat koneksi adalah :



```
1 <?php
2 $link = mysql_connect('localhost', 'user_nya', 'passwor_nya');
3 mysql_select_db('coba_db', $link);
4 mysql_set_charset('UTF-8', $link)
5 ?>
```

Gambar 1

Cara yang baru menggunakan PDO adalah dengan membuat objek baru. Konstruktor PDO memiliki empat parameter yaitu DSN, username, password, and array untuk menampung opsi dari driver database, jika dengan PDO MYSQL maka koneksi diatas dibuat dengan cara seperti berikut :



```
1 <?php
2 $host = 'localhost';
3 $db = 'coba_db';
4 $user = 'root';
5 $pass = '';
6 $charset = 'utf8';
7
8
9 $dsn = "mysql:host=$host;dbname=$db;charset=$charset";
10
11 $db = new PDO($dsn, $user, $pass);
12 ?>
```

Gambar 2

Catatan : Jika Anda mendapat error tentang charset, pastikan Anda menambahkan parameter charset ke DSN. Menambahkan charset ke DSN sangat penting untuk alasan keamanan, kebanyakan contoh yang akan Anda lihat di sekitar membiarkannya keluar. PASTIKAN UNTUK TERMASUK CHARSET !, bagaimana mengetahui jenis-jenis charset, Anda bisa melihatnya melalui halaman awal

phpmyadmin .



Gambar 3

Perhatikan kembali contoh koneksi php pdo diatas, setiap parameter dipisahkan dengan tanda titik koma, pada contoh diatas hanya tiga parameter yang digunakan yaitu DSN, User, dan Password, sedangkan Opsi Driver tidak dipakai, jika akan menggunakan Opsi Driver maka contohnya sebagai berikut :

```
1 <?php
2 $host = 'localhost';
3 $db = 'coba_db';
4 $user = 'root';
5 $pass = '';
6 $charset = 'utf8';
7
8
9 $dsn = "mysql:host=$host;dbname=$db;charset=$charset";
10
11 $opsi = [
12     PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
13     PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
14     PDO::ATTR_EMULATE_PREPARES => false,
15 ];
16
17 $pdo = new PDO($dsn, $user, $pass, $opsi);
18 ?>
```

Gambar 4

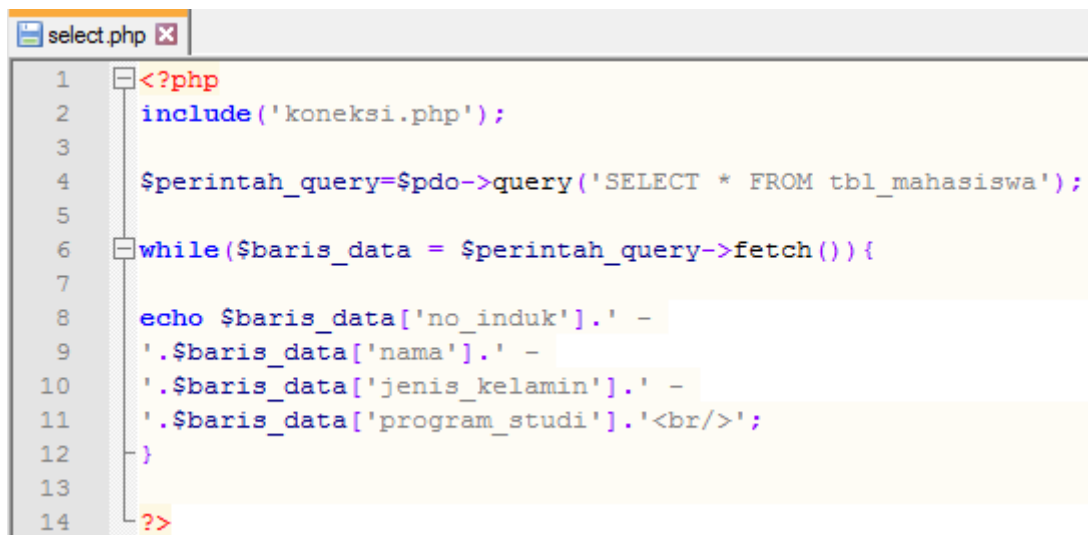
Dengan semua variabel yang disebutkan dengan benar seperti diatas, Anda kini memiliki sebuah koneksi dengan PDO yang tersimpan dalam variabel \$pdo. Catatan penting untuk para pengguna yang masih menggunakan mysql function :

1. Tidak seperti fungsi mysql_* yang lama, yang dapat digunakan di manapun dalam kode program, instance PDO disimpan dalam variabel biasa, yang berarti dapat diakses dalam didalam sebuah fungsi.
2. Koneksi hanya dibuat sekali ! Tidak ada koneksi dalam setiap fungsi. Tidak terhubung di setiap kelas konstruktor. Jika tidak, maka akan terdapat beberapa koneksi yang dibuat, yang pada akhirnya akan membunuh performance server database Anda. Dengan demikian, PDO tunggal harus dibuat dan digunakan didalam skrip program secara keseluruhan.

3. Suatu hal yang sangat penting untuk mengatur charset melalui DSN – merupakan satu-satunya cara yang tepat karena memberitahu PDO yang charset akan digunakan.

E. Menjalankan Query dengan PDO::query()

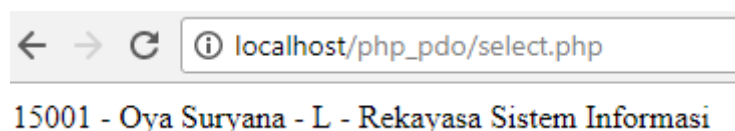
Terdapat dua cara untuk menjalankan query, jika tidak ada variable yang akan digunakan dalam query (tanpa klausa where), Anda dapat langsung menggunakan metode PDO::query(). Metode tersebut akan menjalankan query dan menghasilkan nilai balik berupa object yang serupa dengan fungsi mysql_query. Untuk jelasnya perhatikan contoh berikut :



```
1 <?php
2 include('koneksi.php');
3
4 $perintah_query=$pdo->query('SELECT * FROM tbl_mahasiswa');
5
6 while($baris_data = $perintah_query->fetch()){
7
8     echo $baris_data['no_induk'].' -
9     '.$baris_data['nama'].' -
10    '.$baris_data['jenis_kelamin'].' -
11    '.$baris_data['program_studi'].'<br/>';
12 }
13
14 ?>
```

Gambar 5

Jika script tersebut dijalankan maka akan tampak hasil berikut :



localhost/php_pdo/select.php

15001 - Oya Suryana - L - Rekayasa Sistem Informasi

Gambar 6

F. Prepared statements. Perlindungan dari SQL injections

Ini merupakan alasan utama mengapa anda harus berpindah hati dari fungsi mysql tercinta ke PHP Data Object, Prepared statement merupakan cara yang tepat untuk menjalankan query yang mengandung klausa where.

Jika menggunakan fungsi mysql untuk menampilkan data misal data mahasiswa berjenis kelamin laki-laki jurusan Akuntansi maka query nya adalah :

```

1 <?php
2 $jenis_kelamin='L';
3 $progam_studi='Akuntansi';
4 $sql = "SELECT * FROM tbl_mahasiwa
5         WHERE jenis_kelamin = '$jenis_kelamin'
6         AND program_studi='$progam_studi'";
7 ?>

```

Gambar 7

Jika akan diubah ke PDO, maka script diatas akan menjadi :

```

$sql = 'SELECT * FROM tbl_mahasiswa
        WHERE jenis_kelamin=?
        AND program_studi=?';

```

Gambar 8

Atau bisa juga seperti berikut

```

$sql = "SELECT * FROM tbl_mahasiswa
        WHERE jenis_kelamin =:jk
        AND program_studi =:prodi";

```

Gambar 9

Adapun script lengkapnya adalah sebagai berikut jika menggunakan cara pertama :

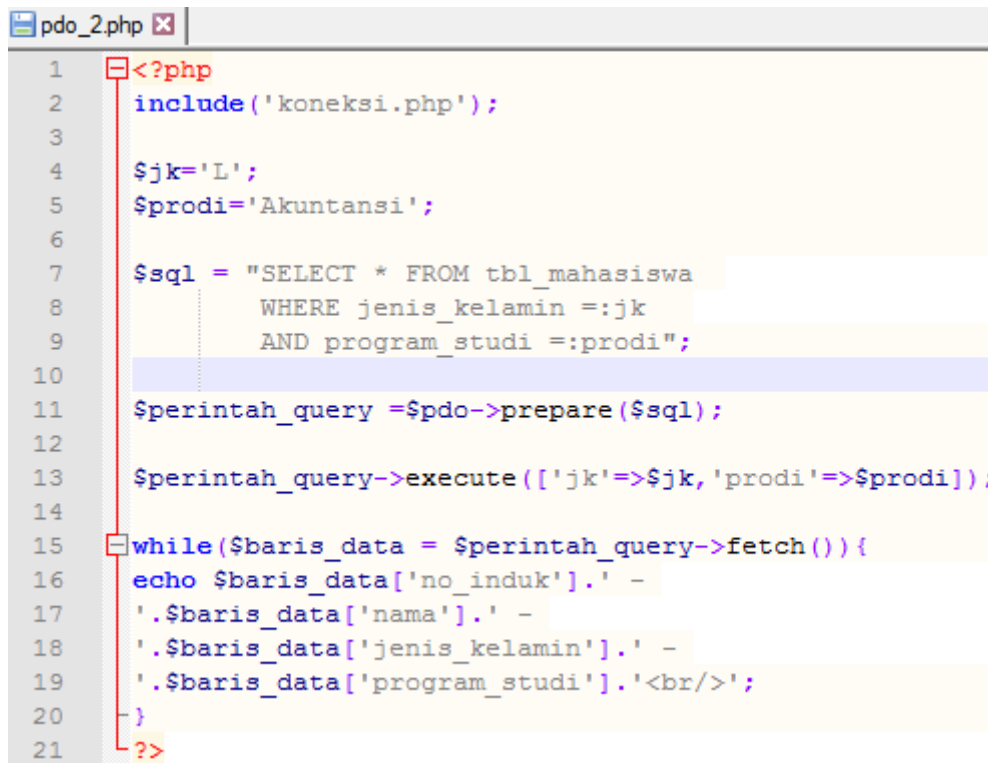
```

1 <?php
2 include('koneksi.php');
3
4 $jk='L';
5 $prodi='Akuntansi';
6
7 $sql = 'SELECT * FROM tbl_mahasiswa
8         WHERE jenis_kelamin=?
9         AND program_studi=?';
10
11 $perintah_query = $pdo->prepare($sql);
12 $perintah_query->execute([$jk,$prodi]);
13
14 while($baris_data = $perintah_query->fetch()){
15     echo $baris_data['no_induk'].' -
16     '.$baris_data['nama'].' -
17     '.$baris_data['jenis_kelamin'].' -
18     '.$baris_data['program_studi'].'<br/>';
19 }
20 ?>

```

Gambar 10

Dan jika menggunakan cara kedua maka script lengkapnya adalah sebagai berikut :



```

1  <?php
2  include('koneksi.php');
3
4  $jk='L';
5  $prodi='Akuntansi';
6
7  $sql = "SELECT * FROM tbl_mahasiswa
8         WHERE jenis_kelamin =:jk
9         AND program_studi =:prodi";
10
11 $perintah_query = $pdo->prepare($sql);
12
13 $perintah_query->execute(['jk'=>$jk, 'prodi'=>$prodi]);
14
15 while($baris_data = $perintah_query->fetch()){
16     echo $baris_data['no_induk'].' -
17     '.$baris_data['nama'].' -
18     '.$baris_data['jenis_kelamin'].' -
19     '.$baris_data['program_studi'].'<br/>';
20 }
21 ?>

```

Gambar 11

Seperti yang anda lihat cara pertama (gambar 10) klausa where diisi dengan tanda tanya (?) sebagai pengganti variabel pada mysql function biasa, penempatan tanda tanya menentukan urutan dalam pengisian parameter pada fungsi execute pada contoh gambar sembilan tanda tanya pertama mewakili variabel input jenis kelamin , tanda tanya kedua mewakili variabel input program studi, untuk itu perhatikan pada saat menjalankan fungsi execute jangan sampai terbalik memanggil variabel. Fungsi execute sendiri diisi dengan data yang bertipe array.

Berbeda dengan cara kedua seperti tampak pada gambar 11 di variabel \$sql terlihat setiap klausa where diisi dengan variabel namun bukan variabel PHP, ini adalah variabel binding dimana variabel diawali dengan tanda titik dua, dan fungsi execute sendiri sama diisi dengan data bertipe array namun disini harus menyebutkan index array yaitu variabel binding yang disebut dalam statement query beserta value array-nya disini bebas menyimpan urutan anggota array yang penting index array nya sama dengan variabel binding.

Ini adalah kelebihan dari PHP PDO dibanding fungsi mysql_* dimana PHP PDO lebih aman dengan adanya binding seperti ini maka SQL Injections dapat dicegah.

G. Metode Binding

Parsing data kedalam fungsi execute () (seperti yang ditunjukkan di atas) dianggap sebagai metode default dan yang paling mudah. Bila metode ini digunakan, semua nilai akan dianggap sebagai string

(kecuali nilai NULL, yang akan dikirim ke kueri seperti, yaitu sebagai SQL NULL), namun hal tersebut rata-rata tidak akan menimbulkan masalah.

Meskipun tidak akan menimbulkan masalah, terkadang lebih baik mengatur tipe data secara eksplisit. Kemungkinan kasus yang akan terjadi adalah:

1. Adanya Klausa Limit dalam query yang tidak bertipe string, atau klausa sql lainnya yang tidak dapat menerima data string
2. Query yang kompleks yang dapat dipengaruhi oleh operan / data yang salah
3. Jenis kolom yang tidak umum seperti BIGINT dan BOOLEAN yang membutuhkan operan data yang tepat.

Jadi pertimbangkan kembali apakah tipe data anda string semua ?? jika ya maka anda cukup menggunakan fungsi execute diatas dengan mem-binding data seperti diatas jika tidak maka tinggalkan cara diatas karena saya yakin operan data anda tidak akan seluruhnya bertipe string, bisa saja bertipe INT, DECIMAL, FLOAT atau SPATIAL.

Jika anda memutuskan untuk membinding data dengan metode yang lain maka ada dua pilihan metode binding data yaitu bindValue() dan bindParam();

```
1 <?php
2 include('koneksi.php');
3
4 $jk='L';
5 $prodi='Akuntansi';
6
7 $sql = "SELECT * FROM tbl_mahasiswa
8       WHERE jenis_kelamin =:jk
9       AND program_studi =:prodi";
10
11 $perintah_query = $pdo->prepare($sql);
12
13 $perintah_query->bindParam(':jk',$jk);
14 $perintah_query->bindParam(':prodi',$prodi);
15
16 $perintah_query->execute();
17
18 while($baris_data = $perintah_query->fetch()){
19     echo $baris_data['no_induk'].' -
20     '.$baris_data['nama'].' -
21     '.$baris_data['jenis_kelamin'].' -
22     '.$baris_data['program_studi'].'<br/>';
23 }
24 ?>
```

Gambar 12

Apabila contoh pada gambar 10 dan gambar 11 dilakukan binding data terhadap variable jenis

kelamin dan program studi, maka akan tampak pada gambar 12, perhatikan baris 13 dan 14 pada gambar 11 tersebut, dan perhatikan fungsi execute pada baris 16 bandingkan fungsi execute pada gambar 9 dan gambar 10, apa perbedaanya ?

H. Bagian-Bagian Query Yang Bisa Dibinding

Sangat penting untuk memahami bagian-bagian mana saja dari query yang bisa dibinding dan bagian mana yang tidak bisa dibinding, faktanya hanya operan yang berupa string dan numeric yang bisa dibinding, jadi selama data yang anda miliki berupa numeric atau quoted string ITU BISA DIBINDING. Untuk kebanyakan kasus anda tidak dapat menggunakan PDO prepare untuk identifier, comma-separated list, atau bagian dari query string, seperti klausa LIKE dibawah maka value tidak bisa dibinding

```
$stmt = $pdo->prepare("SELECT * FROM table WHERE name LIKE '???'")
```

Gambar 13

Sepintas proses binding diatas tampak benar, tapi jika dijalankan maka akan terjadi error, nah disinilah kita harus tahu bagian mana yang bisa dibinding atau tidak.

I. Prepared Statement Untuk Multi Eksekusi

Terkadang Anda dapat menggunakan prepared statement untuk beberapa eksekusi dari query yang disiapkan. Ini sedikit lebih cepat daripada melakukan query yang sama berulang-ulang. Sebagai contoh perhatikan database yang penulis miliki.

no_induk	nama	program_studi	jenis_kelamin	kelas
15001	Oya Suryana	Rekayasa Sistem Informasi	L	
15002	M. Ridwan Farhan	Akuntansi	L	
15003	Abiq Sabiqul Khoir	Teknik Informatika	L	
15004	Zaki Nur Fatah	Sistem Informasi	L	
15005	Rika Widiningsih	Pendidikan Pancasila	P	

Gambar 14

Dari gambar diatas penulis akan meng-update kelas secara sekaligus dengan multi eksekusi, maka scriptnya adalah :

```

1  <?php
2  include('koneksi.php');
3  // siapkan array dengan index adalah no_induk
4  // dan value adalah kelas
5  $kelas = [
6      '15001' => 'A',
7      '15002' => 'A',
8      '15003' => 'B',
9      '15004' => 'B',
10     '15005' => 'C'
11 ];
12
13 // siapkan perintah SQL
14 $perintah_query = $pdo->prepare('UPDATE tbl_mahasiswa
15     SET kelas = ? WHERE no_induk = ?');
16
17 //Loop dan lakukan execute query ketika loop
18 foreach ($kelas as $no_induk => $kelas)
19 {
20     $perintah_query->execute([$kelas, $no_induk]);
21 }
22 ?>

```

Gambar 15

Sebagai bukti maka hasil update adalah sebagai berikut (perhatikan field kelas)

no_induk	nama	program_studi	jenis_kelamin	kelas
15001	Oya Suryana	Rekayasa Sistem Informasi	L	A
15002	M. Ridwan Farhan	Akuntansi	L	A
15003	Abiq Sabiqul Khoir	Teknik Informatika	L	B
15004	Zaki Nur Fatah	Sistem Informasi	L	B
15005	Rika Widiningsih	Pendidikan Pancasila	P	C

Gambar 16

J. Menjalankan perintah, INSERT, UPDATE atau DELETE

Kembali ke focus materi, pada dasarnya tidak ada yang khusus dalam perintah query, begitu pula dalam PDO semuanya sama tidak menjadi masalah query apa yang anda jalankan.

Seperti yang telah dicontohkan dalam contoh-contoh diatas, yang harus anda persiapkan adalah sebuah perintah query yang disimpan dalam fungsi prepare kemudian dijalankan dengan fungsi execute. Untuk proses DELETE dan SELECT intinya sama dengan perintah SELECT, perbedaannya adalah (untuk query yang tidak menghasilkan data) dapat menggunakan metode berantai kemudian menjalankan fungsi execute disebelah kanan fungsi prepare.

a. DELETE

Untuk jelasnya perhatikan perintah DELETE dengan menggunakan metode unchaining (tidak berantai) berikut :

```
1 <?php
2 include('koneksi.php');
3
4 $no_induk='15001';
5
6 $sql='DELETE FROM tbl_mahasiswa WHERE no_induk=:nomor_induk';
7
8 $perintah_hapus=$pdo->prepare($sql);
9
10 $perintah_hapus->bindParam(':nomor_induk',$no_induk);
11
12 $perintah_hapus->execute();
13 ?>
```

Gambar 17

Perhatikan gambar 16 diatas, untuk menghapus data menggunakan metode unchaining terdiri dari tiga baris yaitu baris 8, 10 dan 12, untuk kasus yang tidak menghasilkan data/nilai balik maka bisa diubah menggunakan metode chaining (berantai) sehingga baris 8, 10, dan 12 bisa digabungkan menjadi seperti berikut :

```
1 <?php
2 include('koneksi.php');
3
4 $no_induk='15001';
5
6 $sql='DELETE FROM tbl_mahasiswa WHERE no_induk=:nomor_induk';
7
8 $pdo->prepare($sql)->execute(array('nomor_induk'=>$no_induk));
9 ?>
```

Gambar 18

Perhatikan baris 8 proses execute query dilakukan secara berantai, perhatikan pula proses binding data, tidak menggunakan bindParam tapi data yang dibinding menjadi array yang akan menjadi masukan dalam fungsi execute (perhatikan kembali gambar 11).

Namun jika ingin menampilkan dilayar berapa baris data yang terpengaruh akibat penghapusan data, maka metode chaining tidak bisa dilakukan, harus tetap menggunakan tiga baris perintah seperti pada gambar 16, dengan menambahkan baris 14 untuk menghitung data yang terhapus dan baris 15 untuk menampilkan hasilnya ke layar

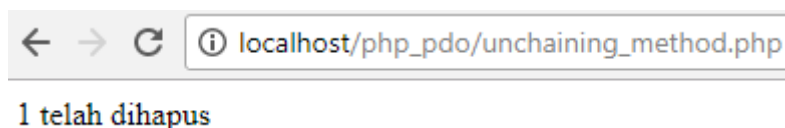
```

1  <?php
2  include('koneksi.php');
3
4  $no_induk='15001';
5
6  $sql='DELETE FROM tbl_mahasiswa WHERE no_induk=:nomor_induk';
7
8  $perintah_hapus=$pdo->prepare($sql);
9
10 $perintah_hapus->bindParam(':nomor_induk',$no_induk);
11
12 $perintah_hapus->execute();
13
14 $data_terhapus=$perintah_hapus->rowCount();
15
16 echo $data_terhapus.' telah dihapus';
17 ?>

```

Gambar 19

Apabila dijalankan maka dilayar akan tampak seperti berikut



← → ↻ ⓘ localhost/php_pdo/unchaining_method.php

1 telah dihapus

Gambar 20

b. INSERT

Untuk perintah insert ada banyak cara, jika banyak data yang akan dikirim maka data disimpan dalam array dan array tersebut dijadikan parameter dalam fungsi execute(), perhatikan script berikut :

```

insert.php x
1  <?php
2  include 'koneksi.php';
3
4  $nim='15007';
5  $nama='Rifki Anggani';
6  $prodi='Akuntansi';
7  $jk='L';
8  $kelas='C';
9
10 $sql_insert=$pdo->prepare("INSERT INTO tbl_mahasiswa
11                             (no_induk,nama,program_studi,jenis_kelamin,kelas)
12                             VALUES (:nim,:nama,:prodi,:jk,:kelas)");
13

```

```

13
14     $data=array(
15         ':nim' => $nim,
16         ':nama' => $nama,
17         ':prodi' => $prodi,
18         ':jk' => $jk,
19         ':kelas' => $kelas
20     );
21
22     $sql_insert->execute($data);
23

```

Gambar 21

Atau dengan cara menyiapkan array dan melakukan bindParam saat melakukan looping anggota array, perhatikan cara kedua berikut :

```

insert2.php x
1  <?php
2  include 'koneksi.php';
3
4  $nim='15008';
5  $nama='Tika Ramdani';
6  $prodi='Sistem Informasi';
7  $jk='P';
8  $kelas='C';
9
10 $sql_insert=$pdo->prepare("INSERT INTO tbl_mahasiswa
11                             (no_induk,nama,program_studi,jenis_kelamin,kelas)
12                             VALUES (:nim,:nama,:prodi,:jk,:kelas)");
13
14     $data=array(
15         ':nim' => $nim,
16         ':nama' => $nama,
17         ':prodi' => $prodi,
18         ':jk' => $jk,
19         ':kelas' => $kelas
20     );
21
22     foreach ($data as $key => $val) {
23         $sql_insert->bindParam($key, $val);
24     }
25
26     $sql_insert->execute();

```

Gambar 22

Bandingkan pernyataan \$sql_insert->execute() pada script pertama dengan \$sql_insert->execute() pada script kedua ! apa perbedaanya ??

c. SELECT

Perintah select tidak akan banyak dibahas dibagian ini, tetapi pada setiap sub bagian akan menggunakan perintah SELECT sebagai contoh penggunaan pengambilan data. Pada dasarnya perintah SELECT diikuti dengan fetch untuk single row result dan fetchAll dengan multirow result.

d. UPDATE

Proses update hampir sama dengan proses insert, terdapat dua cara pengiriman data, baik menggunakan fungsi execute() dengan parameter data array atau looping binding parameter secara looping dan menjalankan fungsi execute() tanpa parameter array seperti pada kasus INSERT data.

Perhatikan contoh pertama yaitu update data dengan mengirimkan parameter data array kepada fungsi execute().



```
1 <?php
2 include 'koneksi.php';
3
4 $nim='15007';
5 $nama='M. Rifki Anggani';
6 $prodi='Manajemen';
7 $jk='L';
8 $kelas='A';
9
10 $sql_update=$pdo->prepare("UPDATE tbl_mahasiswa
11                               SET nama=:nama,
12                               program_studi=:prodi,
13                               jenis_kelamin=:jk,
14                               kelas =:kelas
15                               WHERE no_induk=:nim");
16
17 $data=array(
18     ':nim' => $nim, ':nama' => $nama,
19     ':prodi' => $prodi, ':jk' => $jk,
20     ':kelas' => $kelas
21 );
22 $sql_update->execute($data);
```

Gambar 23

```

1  <?php
2  include 'koneksi.php';
3
4  $nim='15007';
5  $nama='Rifki Anggani';
6  $prodi='Sistem Informasi';
7  $jk='L';
8  $kelas='B';
9
10 $sql_update=$pdo->prepare("UPDATE tbl_mahasiswa
11                               SET nama=:nama,
12                               program_studi=:prodi,
13                               jenis_kelamin=:jk,
14                               kelas =:kelas
15                               WHERE no_induk=:nim");
16
17 $data=array(
18     ':nim' => $nim, ':nama' => $nama,
19     ':prodi' => $prodi, ':jk' => $jk,
20     ':kelas' => $kelas
21 );
22
23 foreach ($data as $key => $val) {
24     $sql_update->bindParam($key, $val);
25 }
26
27 $sql_update->execute();

```

Gambar 24

K. Mengambil Data Dengan Fungsi *foreach()*

Karena pada dasarnya hasil select dari perintah query berupa array, maka pengambilan data baik untuk ditampilkan di layar maupun digunakan kembali dalam body program bisa menggunakan fungsi *foreach()*, untuk jelasnya perhatikan script berikut :

```

1  <?php
2  include('koneksi.php');
3
4  $jk='L';
5  $prodi='Akuntansi';
6
7  $sql = 'SELECT * FROM tbl_mahasiswa';
8
9  $perintah_query = $pdo->query($sql);
10
11 foreach($perintah_query as $baris_data){
12     echo $baris_data['no_induk'].' -
13     '.$baris_data['nama'].' -
14     '.$baris_data['jenis_kelamin'].' -
15     '.$baris_data['program_studi'].'<br/>';
16 }
17 ?>

```

Perhatikan script diatas (baris 9), jika proses pengambilan data tanpa menggunakan klausa where cukup menjalankan fungsi PDO query tanpa melalui prepare dan execute. Metode ini merupakan metode yang cukup ramah memory, karena tidak langsung menghasilkan semua baris data, namun baris per baris.

L. Mengambil Data Dengan Fungsi Fetch

Kita sudah menggunakan fungsi ini pada contoh-contoh sebelumnya, namun mari kita bahas lebih dalam tentang fungsi fetch ini. Fungsi ini mengambil satu baris data dari database (table) dan memindahkannya ke internal pointer didalam result set, jadi konsekwensinya jika menggunakan fungsi ini akan menghasilkan semua baris yang diambil satu persatu dan disimpan dalam result set. Kasarnya fungsi fetch ini bisa dianalogikan dengan mysql_fetch_array, namun cara kerjanya sedikit berbeda. Jika dalam mysql memiliki banyak fungsi seperti (mysql_fetch_assoc, mysql_fetch_row, mysql_fetch_array, dll) maka dengan fungsi fetch cukup satu fungsi tetapi perilakunya bisa diubah dengan mengubah-ubah parameternya. Terdapat banyak metode fetch didalam PDO, namun dalam hal ini akan dibahas empat metode yang sering digunakan saja, yaitu :

1. PDO::FETCH_NUM menghasilkan nilai balik berupa array numeric
2. PDO::FETCH_ASSOC menghasilkan nilai balik berupa array asosiatif
3. PDO::FETCH_BOTH - menghasilkan nilai balik berupa array numeric atau array asosiatif
4. PDO::FETCH_OBJ menghasilkan nilai balik berupa objek
5. PDO::FETCH_LAZY menghasilkan semua metedo diatas (numeric associative dan object) tanpa mengakibatkan memory overhead.

Dari nilai balik yang dihasilkan kelima metode diatas, terdapat dua kemungkinan kasus yang terjadi yaitu :

1. Nilai balik yang dihasilkan hanya menghasilkan satu baris data, sebagai contoh, mengambil detail mahasiswa dengan nomor induk 15002 :

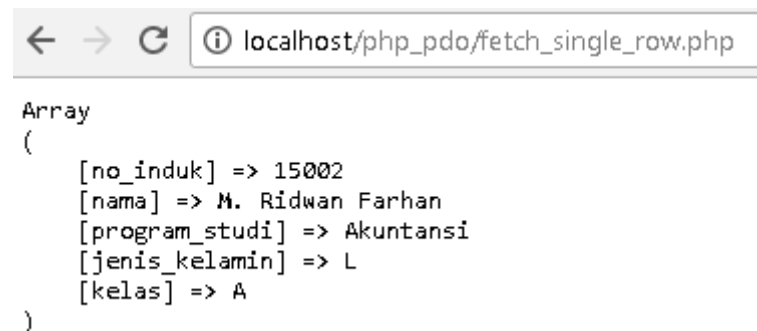
```

1  <?php
2  include('koneksi.php');
3
4  $no_induk='15002';
5
6  $sql='SELECT * FROM tbl_mahasiswa WHERE no_induk=:nomor_induk';
7
8  $sql_detail_mahasiswa=$pdo->prepare($sql);
9  $sql_detail_mahasiswa->bindParam(':nomor_induk',$no_induk);
10 $sql_detail_mahasiswa->execute();
11
12 $data_detail_mahasiswa=$sql_detail_mahasiswa->fetch(PDO::FETCH_ASSOC);
13
14 echo '<pre>';
15 print_r($data_detail_mahasiswa);
16 echo '</pre>';
17 ?>

```

Gambar 26

Dari script tersebut pasti akan menghasilkan satu baris data, maka untuk memanggilnya tidak perlu menggunakan looping foreach atau pun while, sehingga apabila dijalankan script tersebut akan menghasilkan data berupa array asosiatif (gambar 21).



```

Array
(
    [no_induk] => 15002
    [nama] => M. Ridwan Farhan
    [program_studi] => Akuntansi
    [jenis_kelamin] => L
    [kelas] => A
)

```

Gambar 27

Untuk membedakan hasilnya dengan PDO::FETCH_ASSOC, silahkan ganti metode PDO::FETCH_ASSOC pada baris 12 dengan PDO::FETCH_NUM, sehingga baris 12 menjadi seperti berikut :

```

11
12 $data_detail_mahasiswa=$sql_detail_mahasiswa->fetch(PDO::FETCH_NUM);
13

```

Gambar 28

Maka jika dijalankan hasilnya adalah data dengan tipe array numeric, seperti terlihat pada gambar diawah ini :



```
Array
(
    [0] => 15002
    [1] => M. Ridwan Farhan
    [2] => Akuntansi
    [3] => L
    [4] => A
)
```

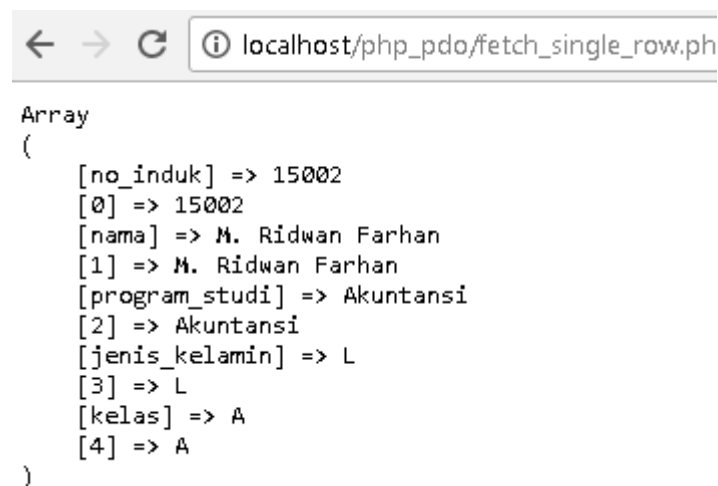
Gambar 29

Kemudian coba ganti kembali PDO::FETCH_NUM dengan PDO::FETCH_BOTH, sehingga baris 12 berubah menjadi :

```
11
12 $data_detail_mahasiswa=$sql_detail_mahasiswa->fetch(PDO::FETCH_BOTH);
13
```

Gambar 30

Dan perhatikan hasilnya jika dijalankan maka akan menghasilkan array dengan tipe numeric juga dengan tipe asosiatif.



```
Array
(
    [no_induk] => 15002
    [0] => 15002
    [nama] => M. Ridwan Farhan
    [1] => M. Ridwan Farhan
    [program_studi] => Akuntansi
    [2] => Akuntansi
    [jenis_kelamin] => L
    [3] => L
    [kelas] => A
    [4] => A
)
```

Gambar 31

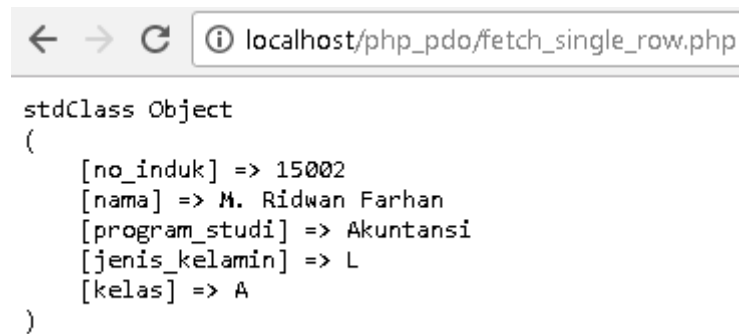
Berikut

ya ganti kembali PDO::FETCH_BITH dengan PDO::FETCH_OBJ, sehingga baris 12 berubah menjadi

```
11
12 $data_detail_mahasiswa=$sql_detail_mahasiswa->fetch(PDO::FETCH_OBJ);
13
```

Gambar 32

Dan data yang dihasilkan berupa objek tidak berupa array lagi.



```
localhost/php_pdo/fetch_single_row.php

stdClass Object
(
    [no_induk] => 15002
    [nama] => M. Ridwan Farhan
    [program_studi] => Akuntansi
    [jenis_kelamin] => L
    [kelas] => A
)
```

Gambar 33

Terakhir bagaimana jika diganti dengan PDO::FETCH_LAZY ?? silahkan anda coba sendiri ... ☺

2. Nilai balik yang dihasilkan hanya menghasilkan banyak baris data, hal ini perlu dilakukan looping untuk mengambil data seperti yang telah dicontohkan pada script-script di halaman sebelumnya.

Contoh diatas adalah membuktikan tipe output yang dihasilkan oleh keempat metode Fetch, berikutnya bagaimana penggunaan dalam script untuk mengambil datanya, berikut adalah contoh masing-masing metode dalam mengambil data.

1. PDO::FETCH_ASSOC

Karena nilai balik result set berupa array asosiatif maka pemanggilan field di table MySQL menggunakan pola pemanggilan array asosiatif dimana nama field di table MySQL menjadi array key (perhatikan baris 12 s.d. baris 16).



```
1 <?php
2 include('koneksi.php');
3
4
5
6 $sql='SELECT * FROM tbl_mahasiswa';
7
8 $sql_mahasiswa=$pdo->prepare($sql);
9 $sql_mahasiswa->execute();
10
11 while ($data_mahasiswa=$sql_mahasiswa->fetch(PDO::FETCH_ASSOC)) {
12     echo $data_mahasiswa['no_induk'].' -
13     '.$data_mahasiswa['nama'].' -
14     '.$data_mahasiswa['jenis_kelamin'].' -
15     '.$data_mahasiswa['program_studi'].'<br/>';
16 }
17 ?>
```

Gambar 34

2. PDO::FETCH_NUM

```

1  <?php
2  include('koneksi.php');
3
4
5
6  $sql='SELECT * FROM tbl_mahasiswa';
7
8  $sql_mahasiswa=$pdo->prepare($sql);
9  $sql_mahasiswa->execute();
10
11 while ($data_mahasiswa=$sql_mahasiswa->fetch(PDO::FETCH_NUM)) {
12     echo $data_mahasiswa[0].' -
13     '.$data_mahasiswa[1].' -
14     '.$data_mahasiswa[2].' -
15     '.$data_mahasiswa[3].'\n';
16 }
17 ?>

```

Gambar 35

Karena nilai balik result set berupa array numeric maka pemanggilan field di table MySQL menggunakan pola pemanggilan array numeric dimana nama field urutan pertama yang disebutkan dalam perintah select secara otomatis diberi nomor index 0 (perhatikan baris 12 s.d. baris 16), field no_induk berada pada urutan pertama dalam table di MySQL maka field tersebut memiliki index 0, dan field berikutnya naik satu tingkatan.

3. PDO::FETCH_BOTH

Untuk kasus PDO::FETCH_BOTH, maka nilai balik result set bisa berupa array numeric maupun array asosiatif, maka pemanggilannya bisa menggunakan kedua cara, perhatikan baris 12 dan 14, pemanggilan data menggunakan metode PDO::FETCH_NUM, sedangkan baris 13 dan 15 menggunakan metode PDO::FETCH_ASSOC.

```

1  <?php
2  include('koneksi.php');
3
4
5
6  $sql='SELECT * FROM tbl_mahasiswa';
7
8  $sql_mahasiswa=$pdo->prepare($sql);
9  $sql_mahasiswa->execute();
10
11 while ($data_mahasiswa=$sql_mahasiswa->fetch(PDO::FETCH_BOTH)) {
12     echo $data_mahasiswa[0].' -
13     '.$data_mahasiswa['nama'].' -
14     '.$data_mahasiswa[2].' -
15     '.$data_mahasiswa['kelas'].'\n';
16 }
17 ?>

```

Gambar 36

4. PDO::FETCH_OBJ

Berebda dengan ketiga metode diatas, result set yang dihasilkan tidak berupa array tapi berupa objek, untuk itu pemanggilan data / field di tabel MySQL menggunakan format pemanggilan objek dengan syntax :

\$objek->property, misal akan memanggil field no_induk dipanggil

dengan cara **\$data_mahasiswa->no_induk**, begitu pula field lainnya

Untuk jelasnya perhatikan baris program nomor 12 s.d. baris nomor 15 pada gambar 30 berikut :

```
1  <?php
2  include('koneksi.php');
3
4
5
6  $sql='SELECT * FROM tbl_mahasiswa';
7
8  $sql_mahasiswa=$pdo->prepare($sql);
9  $sql_mahasiswa->execute();
10
11 while ($data_mahasiswa=$sql_mahasiswa->fetch(PDO::FETCH_OBJ)) {
12     echo $data_mahasiswa->no_induk.' -
13         '.$data_mahasiswa->nama.' -
14         '.$data_mahasiswa->program_studi.' -
15         '.$data_mahasiswa->kelas.'  
';
16 }
17 ?>
```

Gambar 37

5. PDO::FETCH_LAZY

Untuk metode ini pada prinsipnya sama dengan metode PDO::FETCH_OBJ, karena pada dasarnya result set nya adalah objek.

```

1 <?php
2 include('koneksi.php');
3
4
5
6 $sql='SELECT * FROM tbl_mahasiswa';
7
8 $sql_mahasiswa=$pdo->prepare($sql);
9 $sql_mahasiswa->execute();
10
11 while ($data_mahasiswa=$sql_mahasiswa->fetch(PDO::FETCH_LAZY)){
12     echo $data_mahasiswa->no_induk.' -
13     '.$data_mahasiswa->nama.' -
14     '.$data_mahasiswa->program_studi.' -
15     '.$data_mahasiswa->kelas.'<br/>';
16 }
17 ?>

```

Gambar 38

Sebagai catatan, PHP secara standar menggunakan metode PDO:FETCH_BOTH, tetapi anda dapat merubahnya dengan menggunakan pernyataan PDO::ATTR_DEFAULT_FETCH_MODE yang dibuat ketika membuat koneksi dan disimpan dalam opsi koneksi , perhatikan kembali file koneksi berikut :

```

1 <?php
2 $host = 'localhost';
3 $db   = 'coba_db';
4 $user = 'root';
5 $pass = '';
6 $charset = 'utf8';
7
8
9 $dsn = "mysql:host=$host;dbname=$db;charset=$charset";
10
11 $opsi = [
12     PDO::ATTR_ERRMODE           => PDO::ERRMODE_EXCEPTION,
13     PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
14     PDO::ATTR_EMULATE_PREPARES  => true,
15 ];
16
17 $pdo = new PDO($dsn, $user, $pass,$opsi);
18 ?>

```

Gambar 39

Perhatikan baris 13 !, ketika anda menjalankan \$pdo->fetch maka secara otomatis akan menjalankan PDO::FETCH_ASSOC.

M. Mengambil Data Dengan Fungsi fetchColumn()

fetchColumn() merupakan fungsi yang mengembalikan field tunggal dari sebuah result set, sangat

membantu ketika hanya akan mengambil data dari satu field, sebagai contoh perhatikan script berikut :

```
1 <?php
2 include('koneksi.php');
3
4 $no_induk='15001';
5
6 $sql='SELECT * FROM tbl_mahasiswa WHERE no_induk=:nomor_induk';
7
8 $sql_mahasiswa=$pdo->prepare($sql);
9 $sql_mahasiswa->bindParam(':nomor_induk',$no_induk);
10 $sql_mahasiswa->execute();
11
12 $prodi_nya = $sql_mahasiswa->fetchColumn(2);
13
14
15 echo 'Mahasiswa dengan nomor induk '.$no_induk.'<br/>Program
16 studinya yaitu '.$prodi_nya;
17 ?>
```

Gambar 40

Perhatikan angka 2 pada fungsi fetchColumn, apa artinya angka tersebut ? angka tersebut mewakili index column, sehingga

0 adalah nilai index dari coloum no_induk di tbl_mahasiswa

1 adalah nilai index dari coloum nama di tbl_mahasiswa

2 adalah nilai index dari coloum program_studi di tbl_mahasiswa

3 adalah nilai index dari coloum jenis_kelamin di tbl_mahasiswa

4 adalah nilai index dari coloum kelas di tbl_mahasiswa

Maka untuk menampilkan program studinya saja maka fungsi fetchColumn() diisi dengan index 2.

N. Mengambil Data Dengan Format Lain Yang Berbeda, (fetchAll)

Ada banyak fungsi yang menarik, hal ini karena PDO, dengan fungsi PDO bisa mengotomatisasi banyak operasi, apabila tidak dengan PDO maka harus dilakukan secara manual.

PDOStatement :: fetchAll() mengembalikan sebuah array yang terdiri dari semua baris yang dikembalikan oleh kueri. Dari fakta ini kita memiliki dua kesimpulan yaitu :

1. Fungsi ini tidak boleh digunakan, jika banyak baris di-SELECT. Dalam kasus seperti konvensional, looping while harus digunakan, mengambil baris satu per satu dan bukannya memasukkan semuanya ke dalam array sekaligus.
2. Fungsi ini sebagian besar berguna dalam aplikasi web modern yang tidak langsung segera mengeluarkan data saat SELECT, namun meneruskannya ke template.

Anda akan dibuat kagum, dalam banyak format yang berbeda fungsi ini dapat mengembalikan data

(dan rata-rata programmer PHP sedikit mengetahuinya), semuanya dikendalikan oleh variabel PDO :: FETCH_*. Beberapa dari fungsi tersebut adalah :

a. Mendapatkan data berupa array

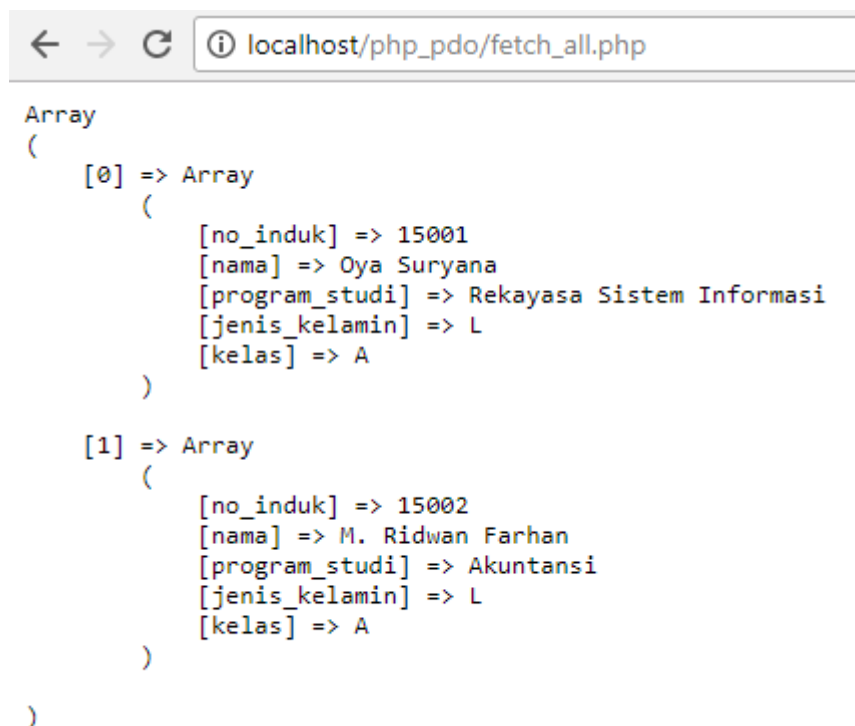
Secara default, fungsi ini hanya mengembalikan array sederhana. Konstanta format, seperti PDO::FETCH_NUM, PDO::FETCH_ASSOC, PDO::FETCH_OBJ dll dapat mengubah format ini menjadi array format lain. Perhatikan contoh berikut :



```
1 <?php
2 include('koneksi.php');
3
4
5 $sql='SELECT * FROM tbl_mahasiswa LIMIT 2';
6
7 $data_mahasiswa=$pdo->query($sql)->fetchAll();
8
9 echo '<pre>';
10 print_r($data_mahasiswa);
11 echo '</pre>';
12
13 ?>
```

Gambar 41

Jalankan script tersebut dan perhatikan hasilnya, variabel \$data_mahasiswa isinya berupa array sederhana.



```
localhost/php_pdo/fetch_all.php

Array
(
    [0] => Array
        (
            [no_induk] => 15001
            [nama] => Oya Suryana
            [program_studi] => Rekayasa Sistem Informasi
            [jenis_kelamin] => L
            [kelas] => A
        )

    [1] => Array
        (
            [no_induk] => 15002
            [nama] => M. Ridwan Farhan
            [program_studi] => Akuntansi
            [jenis_kelamin] => L
            [kelas] => A
        )

)
```


Gambar 42

b. Mendapatkan data berupa array satu dimensi

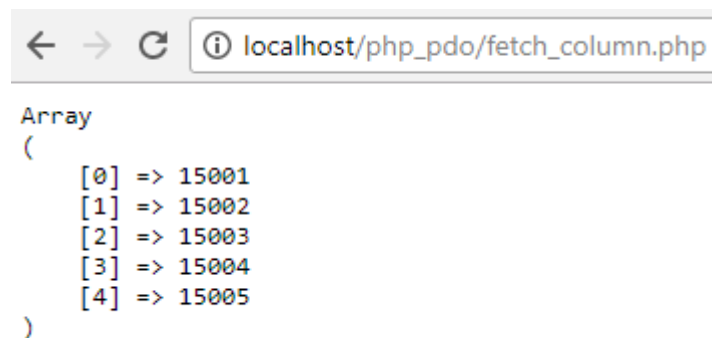
Untuk mengambil data berupa array satu dimensi (satu kolom), syarat menggunakan parameter ini adalah harus menyebutkan nama field, jika tidak maka field index ke-0 yang akan diambil datanya, perhatikan contoh berikut :



```
1 <?php
2 include('koneksi.php');
3
4
5 $sql='SELECT nama FROM tbl_mahasiswa ';
6
7 $data_mahasiswa=$pdo->query($sql)->fetchAll(PDO::FETCH_COLUMN);
8
9 echo '<pre>';
10 print_r($data_mahasiswa);
11 echo '</pre>';
12
13 ?>
```

Gambar 43

Perhatikan baris 7 pada gambar 36 diatas, fungsi `fetchAll()` kini memiliki parameter yaitu `PDO::FETCH_COLUMN` (bandingkan dengan gambar 35), hasil dari script diatas adalah :



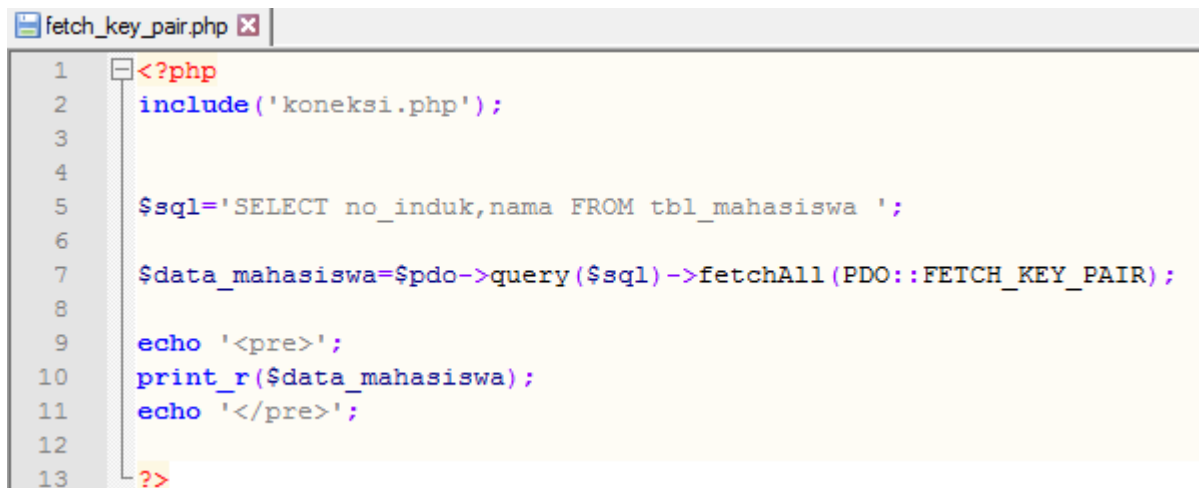
```
localhost/php_pdo/fetch_column.php

Array
(
    [0] => 15001
    [1] => 15002
    [2] => 15003
    [3] => 15004
    [4] => 15005
)
```

Gambar 44

c. Mendapatkan data berupa Pasangan Key=>Value

Jika kita ingin mendapatkan kolom yang sama, tapi tidak di indeks oleh nomor angka namun oleh field lain, maka gunakan parameter `PDO::FETCH_KEY_PAIR`, untuk menggunakan parameter ini hasil dari perintah query harus menyebutkan tepat dua kolom tidak boleh kurang atau lebih. Perhatikan contoh berikut :



```
1 <?php
2 include('koneksi.php');
3
4
5 $sql='SELECT no_induk,nama FROM tbl_mahasiswa ';
6
7 $data_mahasiswa=$pdo->query($sql)->fetchAll(PDO::FETCH_KEY_PAIR);
8
9 echo '<pre>';
10 print_r($data_mahasiswa);
11 echo '</pre>';
12
13 ?>
```

Gambar 45

Jalankan script tersebut dan perhatikan hasilnya pada gambar dibawah ini, perhatikan parameter PDO::FETCH_KEY_PAIRS membuat array dua dimensi dengan index nya yang tidak berupa numeric.




```
Array
(
    [15001] => Oya Suryana
    [15002] => M. Ridwan Farhan
    [15003] => Abiq Sabiqul Khoir
    [15004] => Zaki Nur Fatah
    [15005] => Rika Widiningsih
)
```

Gambar 46

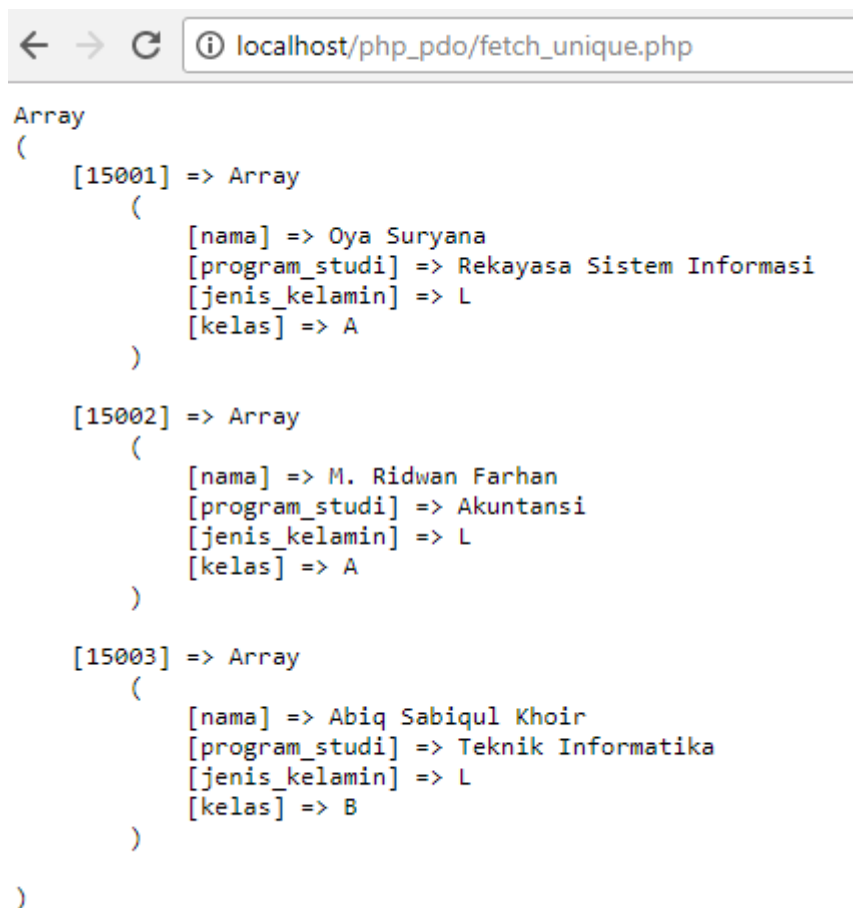
d. Mendapatkan baris data yang diindex dengan field unik

Sama seperti contoh-contoh diatas, namun pada parameter yang satu ini tidak untuk mengambil satu field namun semua field, tapi diindex oleh field yang sifatnya unik (dalam hal ini primary key), dimana Field yang unik akan berlaku sebagai array, jelasnya perhatikan script berikut :



```
1 <?php
2 include('koneksi.php');
3
4
5 $sql='SELECT * FROM tbl_mahasiswa LIMIT 2';
6
7 $data_mahasiswa=$pdo->query($sql)->fetchAll(PDO::FETCH_UNIQUE);
8
9 echo '<pre>';
10 print_r($data_mahasiswa);
11 echo '</pre>';
12
13 ?>
```

Gambar 47



```
localhost/php_pdo/fetch_unique.php

Array
(
    [15001] => Array
        (
            [nama] => Oya Suryana
            [program_studi] => Rekayasa Sistem Informasi
            [jenis_kelamin] => L
            [kelas] => A
        )
    [15002] => Array
        (
            [nama] => M. Ridwan Farhan
            [program_studi] => Akuntansi
            [jenis_kelamin] => L
            [kelas] => A
        )
    [15003] => Array
        (
            [nama] => Abiq Sabiqul Khoir
            [program_studi] => Teknik Informatika
            [jenis_kelamin] => L
            [kelas] => B
        )
)
```

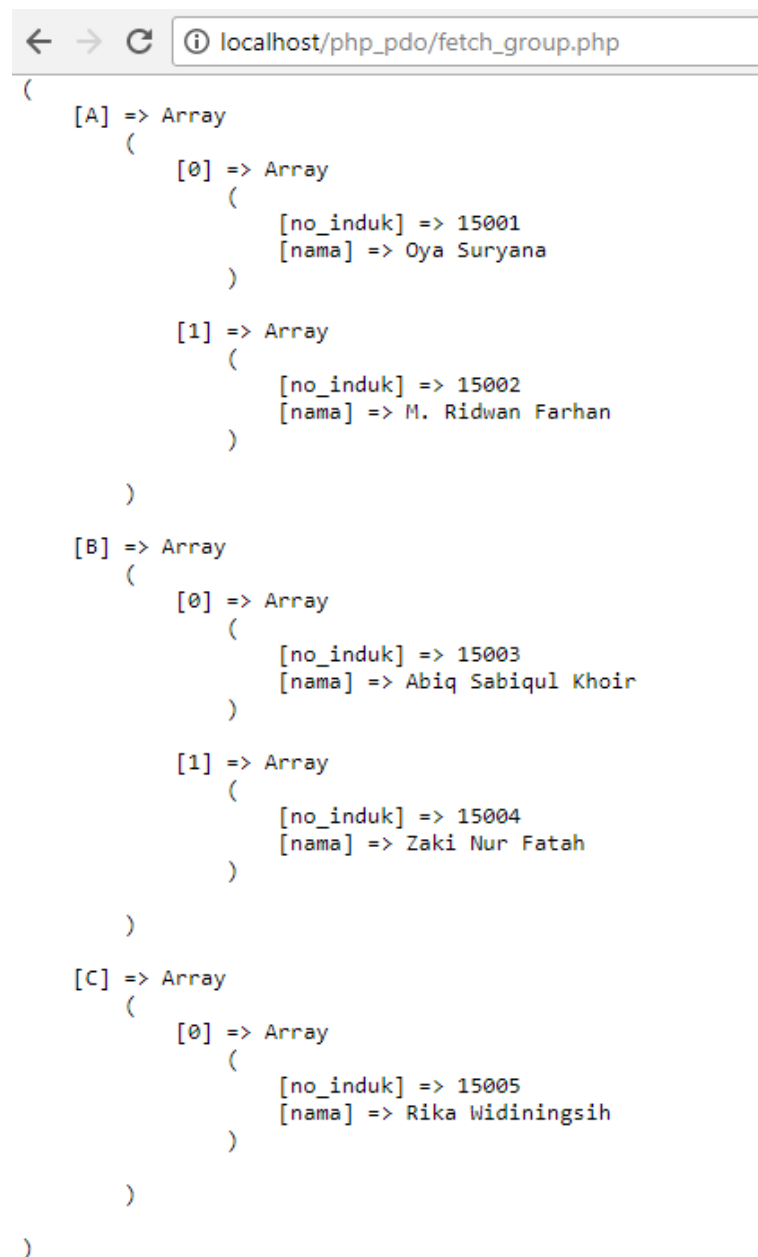
Gambar 48

Perhatikan, data yang menjadi primary key berubah menjadi array dua dimensi.

e. Mendapatkan baris data yang digrup berdasarkan field tertentu

Paramter berikut hamper mirip dengan PDO::FETCH_UNIQUE, namun paramterer ini akan menghasilkan array yang multidimensi(array bersarang / nested), yang dijadikan dasar peng-group-an adalah

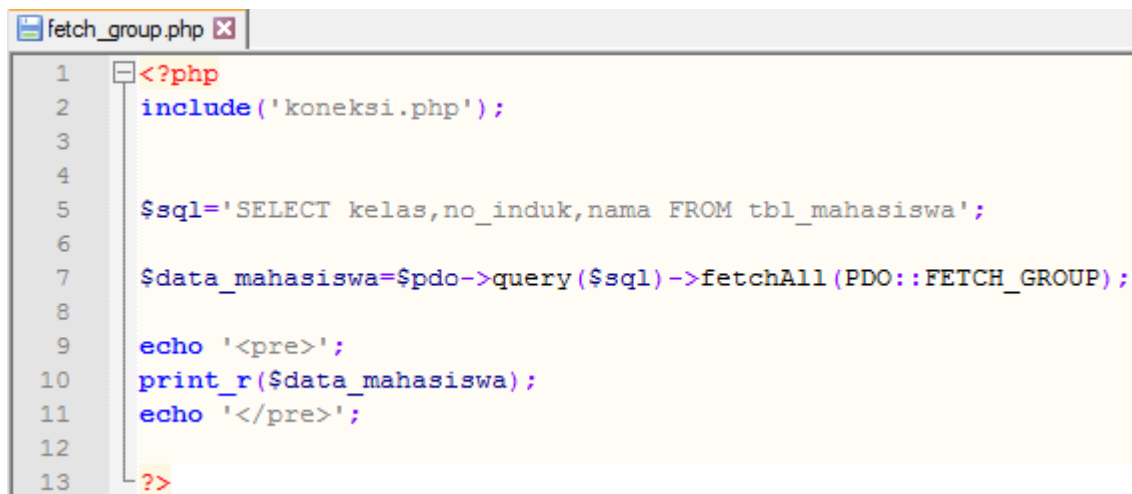
nama field yang pertama kali disebut dalam perintah SELECT, perhatikan contoh berikut, kita akan mengambil data mahasiswa dan di group berdasarkan kelas, perhatikan output berikut :



```
(
  [A] => Array
    (
      [0] => Array
        (
          [no_induk] => 15001
          [nama] => Oya Suryana
        )
      [1] => Array
        (
          [no_induk] => 15002
          [nama] => M. Ridwan Farhan
        )
    )
  [B] => Array
    (
      [0] => Array
        (
          [no_induk] => 15003
          [nama] => Abiq Sabiqul Khoir
        )
      [1] => Array
        (
          [no_induk] => 15004
          [nama] => Zaki Nur Fatah
        )
    )
  [C] => Array
    (
      [0] => Array
        (
          [no_induk] => 15005
          [nama] => Rika Widiningsih
        )
    )
)
```

Gambar 49

Untuk menghasilkan data yang digroup berdasarkan field kelas, maka parameter yang digunakan adalah PDO::FETCH_GROUP (perhatikan baris 7), dan kelas disebut pertama kali dalam query SELECT (baris 5) maka data akan digroup berdasarkan kelas.



```

1 <?php
2 include('koneksi.php');
3
4
5 $sql='SELECT kelas,no_induk,nama FROM tbl_mahasiswa';
6
7 $data_mahasiswa=$pdo->query($sql)->fetchAll(PDO::FETCH_GROUP);
8
9 echo '<pre>';
10 print_r($data_mahasiswa);
11 echo '</pre>';
12
13 ?>

```

Gambar 50

0. Mengambil Satu Record Data (Single Row Fetch)

Kadangkala kita hanya membutuhkan satu record data, misal untuk kasus edit data atau untuk menampilkan detail data, apabila menggunakan fetchAll hasilnya akan dianggap array dua dimensi, hal itu tidak efisien karena sudah jelas data yang diambil adalah satu record berarti hasilnya adalah array satu dimensi. Untuk jelasnya perhatikan contoh berikut :

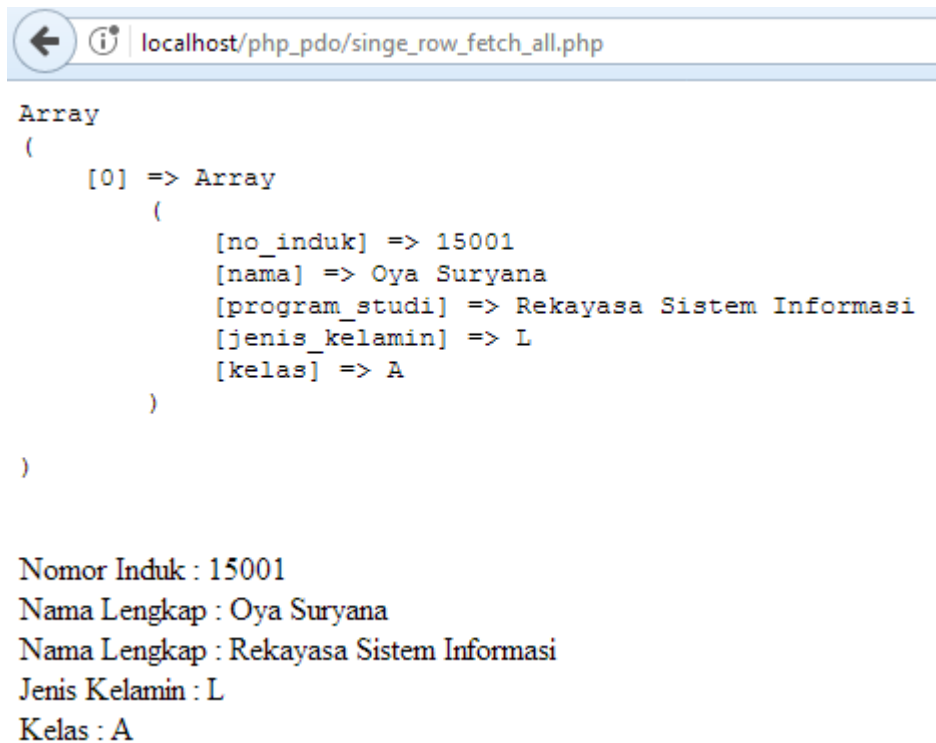


```

1 <?php
2 include 'koneksi.php';
3
4 $nim='15001';
5
6 $sql=$pdo->prepare("SELECT * FROM tbl_mahasiswa WHERE no_induk=:nim");
7 $sql->bindParam(':nim',$nim);
8 $sql->execute();
9
10 $detail_mahasiswa=$sql->fetchAll(PDO::FETCH_ASSOC);
11
12 echo '<pre>';
13 print_r($detail_mahasiswa);
14 echo '</pre>';
15 echo '<br/>';
16 echo 'Nomor Induk : '.$detail_mahasiswa[0]['no_induk'].'<br/>';
17 echo 'Nama Lengkap : '.$detail_mahasiswa[0]['nama'].'<br/>';
18 echo 'Nama Lengkap : '.$detail_mahasiswa[0]['program_studi'].'<br/>';
19 echo 'Jenis Kelamin : '.$detail_mahasiswa[0]['jenis_kelamin'].'<br/>';
20 echo 'Kelas : '.$detail_mahasiswa[0]['kelas'].'<br/>';

```

Gambar 51

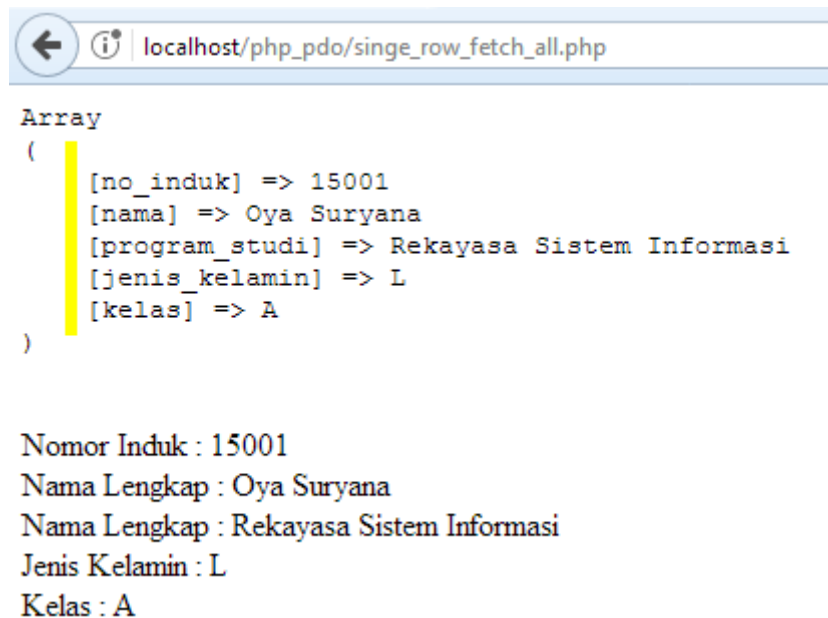


Gambar 52

Perhatikan script pada gambar 43 ketika menggunakan fetchAll maka array yang dihasilkan adalah array dua dimensi , bandingkan dengan script dibawah ini :



Gambar 53

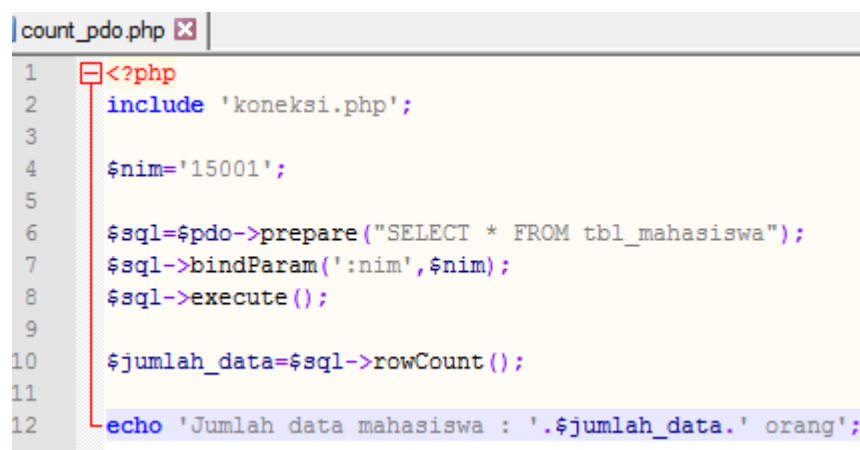


Gambar 54

Pada gambar 45 jelas sekali array yang dihasilkan array satu dimensi, dimana tidak ada index array (perhatikan yang diberi arsir warna).

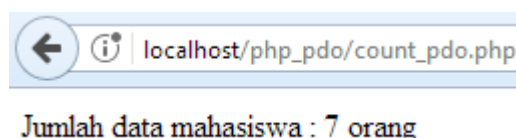
P. Menghitung Jumlah Data Dengan PDO

Untuk menghitung jumlah data untuk keperluan tertentu misal pengecekan login, maka kita gunakan fungsi rowCount, perhatikan script berikut :



Gambar 55

Jika dijalankan maka hasilnya akan tampak seperti berikut :



Gambar 56

Q. Affected Rows Dan Insert id

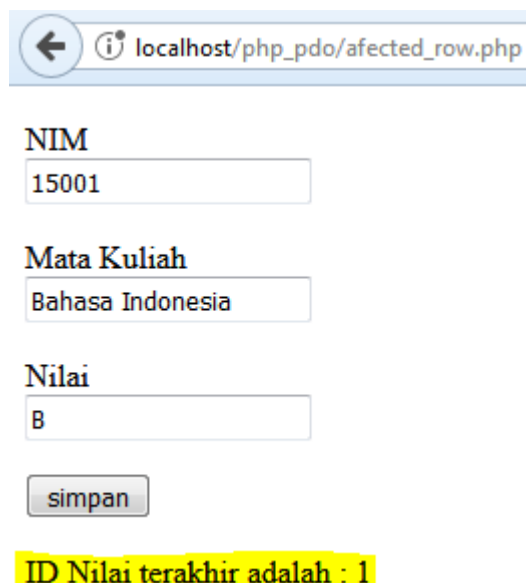
Adakalanya kita menggunakan tipe data INT dengan autoincrement sebagai primary key untuk tabel-tabel transaksi, misal dari kasus tbl_mahasiswa kita kembangkan dengan dibuat tabel baru yaitu tbl_nilai yang akan menampung semua nilai mahasiswa dari sini penulis akan membuat tbl_nilai dengan struktur sebagai berikut :

Nama	Jenis	Penyortiran	Atribut	Kosong	Bawaan	Ekstra
id_nilai	int(11)			Tidak	Tidak ada	AUTO_INCREMENT
nim	char(5)	latin1_swedish_ci		Tidak	Tidak ada	
mata_kuliah	varchar(25)	latin1_swedish_ci		Tidak	Tidak ada	
nilai	char(1)	latin1_swedish_ci		Tidak	Tidak ada	

Gambar 57

Sehingga terdapat relasi 1->N (satu ke banyak, satu mahasiswa memiliki banyak nilai), dari kasus diatas kadang kita membutuhkan id_nilai terakhir yang diinput, cara lama yang digunakan adalah dengan menggunakan fungsi rowCount(), namun hal itu akan menjadi masalah jika salah satu atau beberapa record ada yang dihapus. Informasi yang dihasilkan rowCount() menjadi tidak akurat.

Untuk mengatasi masalah ini, PDO telah menyediakan fungsi lastInsertId() yang berguna untuk mengambil ID terakhir dari field yang memiliki karakteristik auto_increment. Untuk jelasnya perhatikan form input nilai berikut.



← localhost/php_pdo/afected_row.php

NIM

Mata Kuliah

Nilai

ID Nilai terakhir adalah : 1

Gambar 58

Dari form diatas berguna untuk menginput nilai mahasiswa, setiap terjadi penambahan data

akan ditampilkan id_nilai yang terakhir diinput. Adapun script-nya adalah :

```

1 <html>
2 <form method="POST">
3 <p>NIM<br/><input type="text" name="nim"></p>
4 <p>Mata Kuliah<br/><input type="text" name="mata_kuliah"></p>
5 <p>Nilai<br/><input type="text" name="nilai"></p>
6 <p><input type="submit" name="simpan" value="simpan"></p>
7 </form>
8 <?php
9 if(isset($_POST['simpan'])) {
10 include 'koneksi.php';
11 $nim_nya=$_POST['nim'];
12 $mata_kuliah_nya=$_POST['mata_kuliah'];
13 $nilai_nya=$_POST['nilai'];
14
15 $sql_insert=$pdo->prepare("INSERT INTO tbl_nilai
16                             (nim,mata_kuliah,nilai)
17                             VALUES (:nim,:mata_kuliah,:nilai)");
18
19 $data=array(
20     ':nim' => $nim_nya,
21     ':mata_kuliah' => $mata_kuliah_nya,
22     ':nilai' => $nilai_nya
23 );
24
25 foreach ($data as $key => $val) {
26     $sql_insert->bindParam($key, $val);
27 }
28
29 $sql_insert->execute();
30 echo 'ID Nilai terakhir adalah : '.$pdo->lastInsertId();
31 }
32 <?>
33 </html>

```

Gambar 59

Lakukan pengecekan kedalam database dengan phpmyadmin, jika berhasil maka akan tampak sebagai berikut, lihat field id_nilai. Dengan lastInsertId() tersebut kita bisa tahu ID terakhir yang diinput kedalam table.

id_nilai	nim	mata_kuliah	nilai
1	15001	Bahasa Indonesia	B
2	15002	Pancasila	C

Gambar 60

R. Prepared Statement dalam Klausula Like

Meskipun secara keseluruhan PDO memberikan kemudahan bagi penggunaannya, namun ada

beberapa hal yang harus diperhatikan, dan saya akan menjelaskan salah satunya terkait hal penggunaan placeholder (tanda ?) didalam kalusa like. Untuk pertama kali ketika baru memahami tentang binding param menggunakan placeholder (tanda ?) maka orang akan berfikir menggunakannya seperti berikut :

```
$stmt = $pdo->prepare("SELECT * FROM table WHERE name LIKE '%%?'");
```

Tetapi setelah memahami betul tentang pemenggalan sql statement untuk proses binding, orang akan menyadari kesalahan yang terjadi, seperti yang telah penulis jelaskan diatas tentang pemenggalan pada proses binding paramter. Sebuah placeholder (tanda ?) harus merepresentasikan sebuah data, apakah berupa string atau integer jika melihat contoh diatas %?% tidak merepresentasikan tipe data baik string maupun integer. Untuk itu proses binding harus kita ubah dengan teknik sebagai berikut :

S. Prepared Statement dalam Klausa IN

Sama seperti yang telah dibahas di atas, tidak mungkin mengganti bagian query seenaknya dengan menggunakan placeholder tanda tanya (?). Jadi, untuk nilai yang dipisahkan koma, seperti untuk operator IN (),harus membuat satu set placeholder (?) secara manual dan memasukkannya ke dalam kueri.

Perhatikan contoh dibawah , contoh dibawah adalah query untuk menampilkan mahasiswa yang program studinya adlaah sistem informasi, akuntansi, dan teknik informatika dengan klausa IN. SQL native nya adalah :

```
SELECT * FROM tbl_mahasiswa WHERE program_studi IN ('Sistem Informasi', 'Akuntansi','Teknik informatika')
```

Berdasarkan query diatas maka klausa IN dapat dibuat dengan cara menyiapkannya terlebih dahulu dalam array, kemudian membuat placeholder (?) sebanyak jumlah anggota array. Jelasnya perhatikan script dibawah ini :

```

1 <?php
2 include 'koneksi.php';
3
4 $program_studi =array('Sistem Informasi','Akuntansi','Teknik Informatika');
5 $in_program_studi = str_repeat('?',', count($program_studi) - 1) . '?';
6
7 $in_program_studi = str_repeat('?',', count($program_studi) - 1) . '?';
8
9 $sql = "SELECT * FROM tbl_mahasiswa WHERE program_studi IN ($in_program_studi)";
10 $sql_mahasiswa = $pdo->prepare($sql);
11 $sql_mahasiswa->execute($program_studi);
12
13 echo '<table border="1">
14     <tr><th>No. Induk</th><th>Nama Mahasiswa</th>
15     <th>Program Studi</th><th>L/P</th><th>Kelas</th></tr>';
16 while ($data_mahasiswa=$sql_mahasiswa->fetch(PDO::FETCH_ASSOC)) {
17     echo
18     '<tr>
19     <td>'.$data_mahasiswa['no_induk'].'</td>
20     <td>'.$data_mahasiswa['nama'].'</td>
21     <td>'.$data_mahasiswa['program_studi'].'</td>
22     <td>'.$data_mahasiswa['jenis_kelamin'].'</td>
23     <td>'.$data_mahasiswa['kelas'].'</td>
24     </tr>';
25 }
26 echo '</table>';
27
28 ?>

```

Gambar 61

Jalankan script tersebut dan lihat hasilnya :

localhost/php_pdo/klausu_in.php				
No. Induk	Nama Mahasiswa	Program Studi	L/P	Kelas
15002	M. Ridwan Farhan	Akuntansi	L	A
15003	Abiq Sabiqul Khoir	Teknik Informatika	L	A
15004	Zaki Nur Fatah	Sistem Informasi	L	B
15007	Rifki Anggani	Sistem Informasi	L	A
15008	Tika Ramdani	Sistem Informasi	P	C

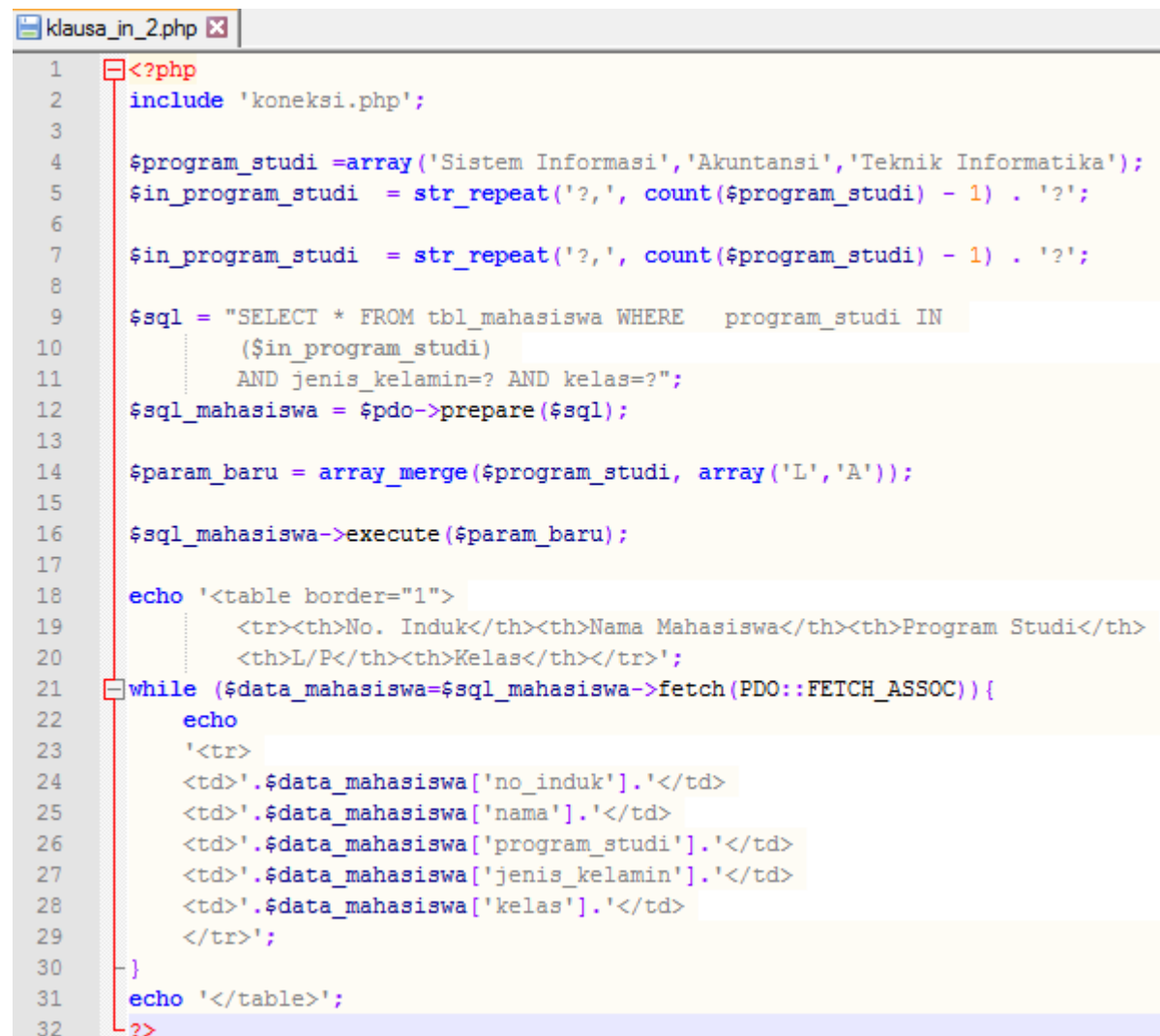
Gambar 62

Untuk kasus lain dimana klausa IN digabung dengan klausa yang lain, anda dapat menggunakan fungsi `array_merge()` untuk menggabungkan array yang menampung data untuk klausa IN, dan menggabungkan dengan single array yang lain, sebagai contoh dibawah adalah query untuk menampilkan mahasiswa yang program studinya adalah sistem informasi, akuntansi, dan teknik

informatika dengan klausa IN dengan jenis kelamin Laki-laki.

SQL native nya adalah

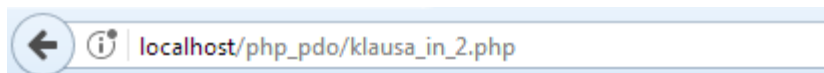
```
SELECT * FROM tbl_mahasiswa WHERE program_studi IN ('Sistem
Informasi', 'Akuntansi', 'Teknik informatika') AND
jenis_kelamin='L' AND kelas='A'
```



```
1 <?php
2 include 'koneksi.php';
3
4 $program_studi = array('Sistem Informasi', 'Akuntansi', 'Teknik Informatika');
5 $in_program_studi = str_repeat('?', count($program_studi) - 1) . '?';
6
7 $in_program_studi = str_repeat('?', count($program_studi) - 1) . '?';
8
9 $sql = "SELECT * FROM tbl_mahasiswa WHERE program_studi IN
10      ($in_program_studi)
11      AND jenis_kelamin=? AND kelas=?";
12 $sql_mahasiswa = $pdo->prepare($sql);
13
14 $param_baru = array_merge($program_studi, array('L', 'A'));
15
16 $sql_mahasiswa->execute($param_baru);
17
18 echo '<table border="1">
19      <tr><th>No. Induk</th><th>>Nama Mahasiswa</th><th>Program Studi</th>
20      <th>L/P</th><th>Kelas</th></tr>';
21 while ($data_mahasiswa=$sql_mahasiswa->fetch(PDO::FETCH_ASSOC)) {
22     echo
23     '<tr>
24     <td>'.$data_mahasiswa['no_induk'].'</td>
25     <td>'.$data_mahasiswa['nama'].'</td>
26     <td>'.$data_mahasiswa['program_studi'].'</td>
27     <td>'.$data_mahasiswa['jenis_kelamin'].'</td>
28     <td>'.$data_mahasiswa['kelas'].'</td>
29     </tr>';
30 }
31 echo '</table>';
32 ?>
```

Gambar 63

Jalankan script tersebut dan lihat hasilnya :



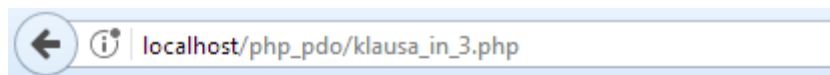
No. Induk	Nama Mahasiswa	Program Studi	L/P	Kelas
15002	M. Ridwan Farhan	Akuntansi	L	A
15003	Abiq Sabiqul Khoir	Teknik Informatika	L	A
15007	Rifki Anggani	Sistem Informasi	L	A

Gambar 64

Untuk kasus dimana Anda memberikan nama pada setiap placeholder (tanda ?), script akan tampak lebih kompleks dari kedua cara diatas, sebagai contoh cara pertama kita ubah dengan memberi nama pada setiap placeholder (tanda ?) dengan nama :prodi1, :prodi2, dan :prodi3, maka script akan berubah seperti tampak pada gambar 66.

Perhatikan script pada gambar 66, array \$program_studi di binding ke :prodi1, :prodi2, dan :prodi3 yang semula menggunakan placeholder (tanda ?), akibat cara binding params berubah maka proses penggabungan binding menggunakan looping foreach (baris 8 s.d baris 14).

Jalankan script tersebut dan hasilnya akan sama dengan cara yang pertama



No. Induk	Nama Mahasiswa	Program Studi	L/P	Kelas
15002	M. Ridwan Farhan	Akuntansi	L	A
15003	Abiq Sabiqul Khoir	Teknik Informatika	L	A
15004	Zaki Nur Fatah	Sistem Informasi	L	B
15007	Rifki Anggani	Sistem Informasi	L	A
15008	Tika Ramdani	Sistem Informasi	P	C

Gambar 65

```

1 <?php
2 include 'koneksi.php';
3
4 $program_studi =array(':prodi1'=>'Sistem Informasi',':prodi2'=>'Akuntansi',
5                       ':prodi3'=>'Teknik Informatika');
6 $data=json_encode($program_studi);
7
8 $params=null;
9 foreach($program_studi as $key=> $val)
10 {
11     $params=$key.','.$params;
12 }
13
14 $in_program_studi=rtrim($params,',');
15
16 $sql = "SELECT * FROM tbl_mahasiswa WHERE   program_studi IN
17         ($in_program_studi)";
18 $sql_mahasiswa = $pdo->prepare($sql);
19
20 $sql_mahasiswa->execute($program_studi);
21
22 echo '<table border="1">
23     <tr><th>No. Induk</th><th>Nama Mahasiswa</th><th>Program Studi</th>
24     <th>L/P</th><th>Kelas</th></tr>';
25 while ($data_mahasiswa=$sql_mahasiswa->fetch(PDO::FETCH_ASSOC)) {
26     echo
27     '<tr>
28     <td>'.$data_mahasiswa['no_induk'].'</td>
29     <td>'.$data_mahasiswa['nama'].'</td>
30     <td>'.$data_mahasiswa['program_studi'].'</td>
31     <td>'.$data_mahasiswa['jenis_kelamin'].'</td>
32     <td>'.$data_mahasiswa['kelas'].'</td>
33     </tr>';
34 }
35 echo '</table>';
36 ?>

```

Gambar 66

T. Permasalahan dengan Klausa LIMIT

Didalam PDO, klausa limit dapat menjadi masalah ketika menjalankan query dengan pernyataan prepare. Ketika mode emulation dalam keadaan ON (default), dan ketika melakukan binding parameter dengan metode 'Lazy', PDO akan menganggap semua data adalah string. Sehingga jika Anda memberikan perintah **LIMIT 10,10** dalam perintah query oleh PDO akan diubah menjadi **LIMIT '10','10'**, dan ini akan mengakibatkan kesalahan query yang pada akhirnya mengirim data kosong.

Untuk hal tersebut terdapat dua solusi, yaitu :

1. Ubah mode emulation kedalam mode off dengan men-set nya menjadi false, lakukan ini ketika membuat file koneksi (lihat gambar)

```

1  <?php
2  $host = 'localhost';
3  $db   = 'coba_db';
4  $user = 'root';
5  $pass = '';
6  $charset = 'utf8';
7
8
9  $dsn = "mysql:host=$host;dbname=$db;charset=$charset";
10
11  $opsi = [
12      PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
13      PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
14      PDO::ATTR_EMULATE_PREPARES   => false
15  ];
16
17  $pdo = new PDO($dsn, $user, $pass, $opsi);
18  ?>

```

Gambar 67

2. Cara kedua adalah merubah tipe data saat melakukan binding parameter, perhatikan contoh berikut, saya akan menunjukkan perintah query berikut :

```
SELECT * FROM tbl_mahasiswa LIMIT 1,2
```

Maka scriptnya adalah :

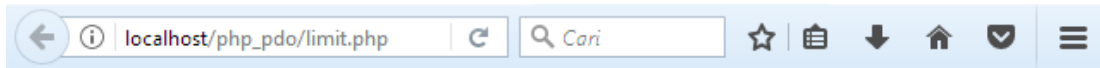
```

1  <?php
2  include('koneksi.php');
3
4  $offset=1;
5  $limit=2;
6
7  $sql = 'SELECT * FROM tbl_mahasiswa LIMIT :offset,:limit';
8
9  $perintah_query = $pdo->prepare($sql);
10
11  $perintah_query->bindParam(':offset',$offset);
12  $perintah_query->bindParam(':limit',$limit);
13  $perintah_query->execute();
14
15  while($data_mahasiswa=$perintah_query->fetch(PDO::FETCH_ASSOC)){
16      echo $data_mahasiswa['no_induk'].' -
17      '.$data_mahasiswa['nama'].' -
18      '.$data_mahasiswa['jenis_kelamin'].' -
19      '.$data_mahasiswa['program_studi'].'<br/>';
20  }
21  ?>

```

Gambar 68

Dan mari kita jalankan !



Fatal error: Uncaught exception 'PDOException' with message 'SQLSTATE[42000]: Syntax error or access violation: 1064 You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "1,2" at line 1' in C:\xampp\htdocs\php_pdo\limit.php:13 Stack trace: #0 C:\xampp\htdocs\php_pdo\limit.php(13): PDOStatement->execute() #1 {main} thrown in C:\xampp\htdocs\php_pdo\limit.php on line 13

Gambar 69

Perhatikan, Anda lihat ! hasilnya error, padahal script benar ini terjadi karena klausa LIMIT seharusnya dikirim berupa LIMIT 1,2 , namun karena PDO::ATTR_EMULATE_PREPARES bernilai true (di file koneksi.php, lihat gambar dibawah !), atau bahkan anda tidak membuatnya (maka default bernilai true), maka akan dikirim menjadi LIMIT '1','2' (ada kutipnya karena dianggap string)

```
$opsi = [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES => true
];
```

Gambar 70

Untuk memperbaiki error tersebut cara pertama adalah merubah nilai PDO::ATTR_EMULATE_PREPARES menjadi false, atau merubah tipe data pada saat melakukan binding, jika cara ini yang ditempuh, maka script diatas akan menjadi :

```
1 <?php
2 include('koneksi.php');
3
4 $offset=1;
5 $limit=2;
6
7 $sql = 'SELECT * FROM tbl_mahasiswa LIMIT :offset,:limit';
8
9 $perintah_query = $pdo->prepare($sql);
10
11 $perintah_query->bindParam(':offset',$offset,PDO::PARAM_INT);
12 $perintah_query->bindParam(':limit',$limit,PDO::PARAM_INT);
13 $perintah_query->execute();
14
15 while($data_mahasiswa=$perintah_query->fetch(PDO::FETCH_ASSOC)){
16 echo $data_mahasiswa['no_induk'].' -
17 '.$data_mahasiswa['nama'].' -
18 '.$data_mahasiswa['jenis_kelamin'].' -
19 '.$data_mahasiswa['program_studi'].'<br/>';
20 }
21 ?>
```

Gambar 66

Perhatikan proses binding menambah parameter baru yaitu PDO::PARAM_INT artinya nilai offset dan limit diubah menjadi integer / numeric.

U. Transactions

Untuk memastikan transaksi berjalan sukses, Anda harus memastikan error mode diatur ke exceptions, dan pelajari canonical methods :

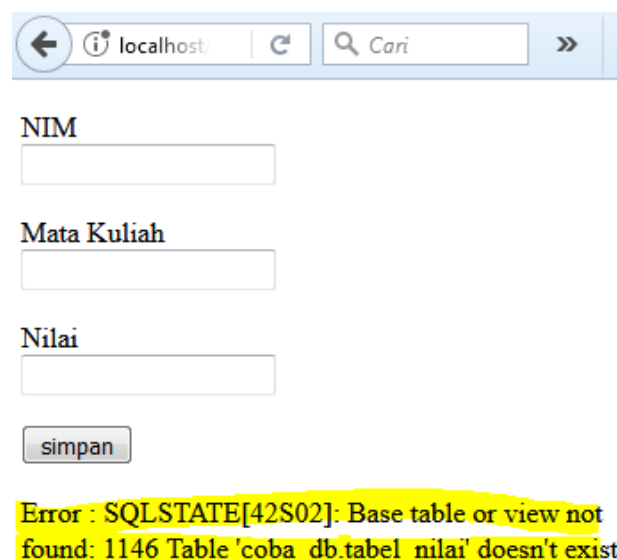
- *beginTransaction()* to memulai transaction
- *commit()* to menjalankan transaction
- *rollback()* to membatalkan semua perubahan yang dibuat sejak dimulainya transaction ketika terjadi nya error.

Exception sangat penting untuk transaksi karena bisa tertangkap. Jadi jika salah satu query gagal, eksekusi akan dihentikan dan langsung dipindahkan ke blok exception, di mana keseluruhan transaksi akan dikembalikan ke awal transaksi.

Berikut contoh penggunaan transaction untuk proses insert, pada contoh sebelumnya proses insert tanpa transaction sedangkan contoh sekarang menggunakan transaction, perhatikan gambar 68.

Untuk skenario gambar 68, coba anda ubah nama tabel tbl_nilai menjadi tabel_nilai seperti tampak pada baris 17, kemudian jalankan, maka akan tampak kesalahan ketika proses simpan, proses simpan digagalkan dan menangkap kejadian error dimana error disebutkan bahawa Table *coba_db.tabel_nilai' doesn't exist* (tabel_nilai tidak ada).

Penggunaan exception sebaiknya dilakukan pada setiap operasi query untuk menangkap terjadinya error,



The screenshot shows a web application interface with a light blue header bar containing navigation icons and a search box labeled 'Cari'. Below the header, there is a form with three input fields labeled 'NIM', 'Mata Kuliah', and 'Nilai'. A 'simpan' button is located below the 'Nilai' field. At the bottom of the form, an error message is displayed in a yellow highlighted box: 'Error : SQLSTATE[42S02]: Base table or view not found: 1146 Table 'coba_db.tabel_nilai' doesn't exist'.

Gambar 67

```

transaction.php
1 <html>
2 <form method="POST">
3 <p>NIM<br/><input type="text" name="nim"></p>
4 <p>Mata Kuliah<br/><input type="text" name="mata_kuliah"></p>
5 <p>Nilai<br/><input type="text" name="nilai"></p>
6 <p><input type="submit" name="simpan" value="simpan"></p>
7 </form>
8 <?php
9 if(isset($_POST['simpan'])) {
10 include 'koneksi.php';
11 $nim_nya=$_POST['nim'];
12 $mata_kuliah_nya=$_POST['mata_kuliah'];
13 $nilai_nya=$_POST['nilai'];
14
15 try {
16     $pdo->beginTransaction();
17     $sql_insert=$pdo->prepare("INSERT INTO tabel_nilai
18                               (nim,mata_kuliah,nilai)
19                               VALUES (:nim,:mata_kuliah,:nilai)");
20     $data=array(
21         ':nim' => $nim_nya, ':mata_kuliah' => $mata_kuliah_nya,
22         ':nilai' => $nilai_nya
23     );
24
25     foreach ($data as $key => $val) {
26         $sql_insert->bindParam($key, $val);
27     }
28
29     $sql_insert->execute();
30     echo 'ID Nilai terakhir adalah : '.$pdo->lastInsertId();
31     $pdo->commit();
32 } catch (PDOException $e) {
33     $pdo->rollback();
34     echo 'Error : '.$e->getMessage();
35 }
36 }
37 ?>
38 </html>

```

Gambar 68

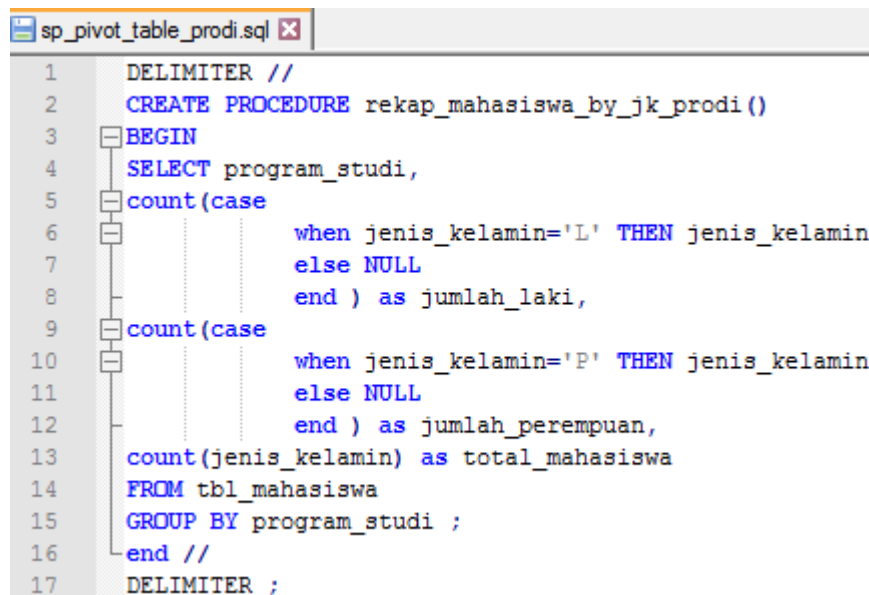
V. Menjalankan Stored Procedures di PDO

Ada satu batu sandungan bagi programmer yang menggunakan stored procedure, yaitu stored procedure selalu menghasilkan lebih dari satu result set. satu (atau banyak) result set dengan data aktual dan hanya kosongkan akan mengakibatkan terjadinya error **"Cannot execute queries while other unbuffered queries are active"**, jika hal itu terjadi maka Anda harus mengosongkan result set yang kosong yang terjadi ketika menjalankan stored procedure tadi. Hal tersebut dilakukan dengan cara menjalankan fungsi `PDOStatement::nextRowset()` setiap setelah menjalankan stored procedure.

Perhatikan kasus berikut, berdasarkan tabel mahasiswa yang kita miliki kita diminta rekapitulasi

data jumlah mahasiswa berdasarkan program studi dan jenis kelamin, maka tahapan pembuatan script-nya adalah

1. Buat stored procedure untuk menampilkan rekap data mahasiswa dengan nama rekap_mahasiswa_by_jk_prodi, adapun scriptnya



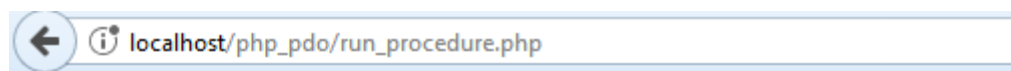
```

1  DELIMITER //
2  CREATE PROCEDURE rekap_mahasiswa_by_jk_prodi()
3  BEGIN
4  SELECT program_studi,
5  count(case
6  when jenis_kelamin='L' THEN jenis_kelamin
7  else NULL
8  end ) as jumlah_laki,
9  count(case
10 when jenis_kelamin='P' THEN jenis_kelamin
11 else NULL
12 end ) as jumlah_perempuan,
13 count(jenis_kelamin) as total_mahasiswa
14 FROM tbl_mahasiswa
15 GROUP BY program_studi ;
16 end //
17 DELIMITER ;

```

Gambar 69

2. Jalankan script tersebut dimenu SQL pada phpmyadmin
3. Buat script php untuk menjalankan stored procedure yang telah dibuat
4. Jalankan aplikasinya, jika scriptnya tidak ada yang salah maka tampilan rekap data akan tampak seperti berikut :



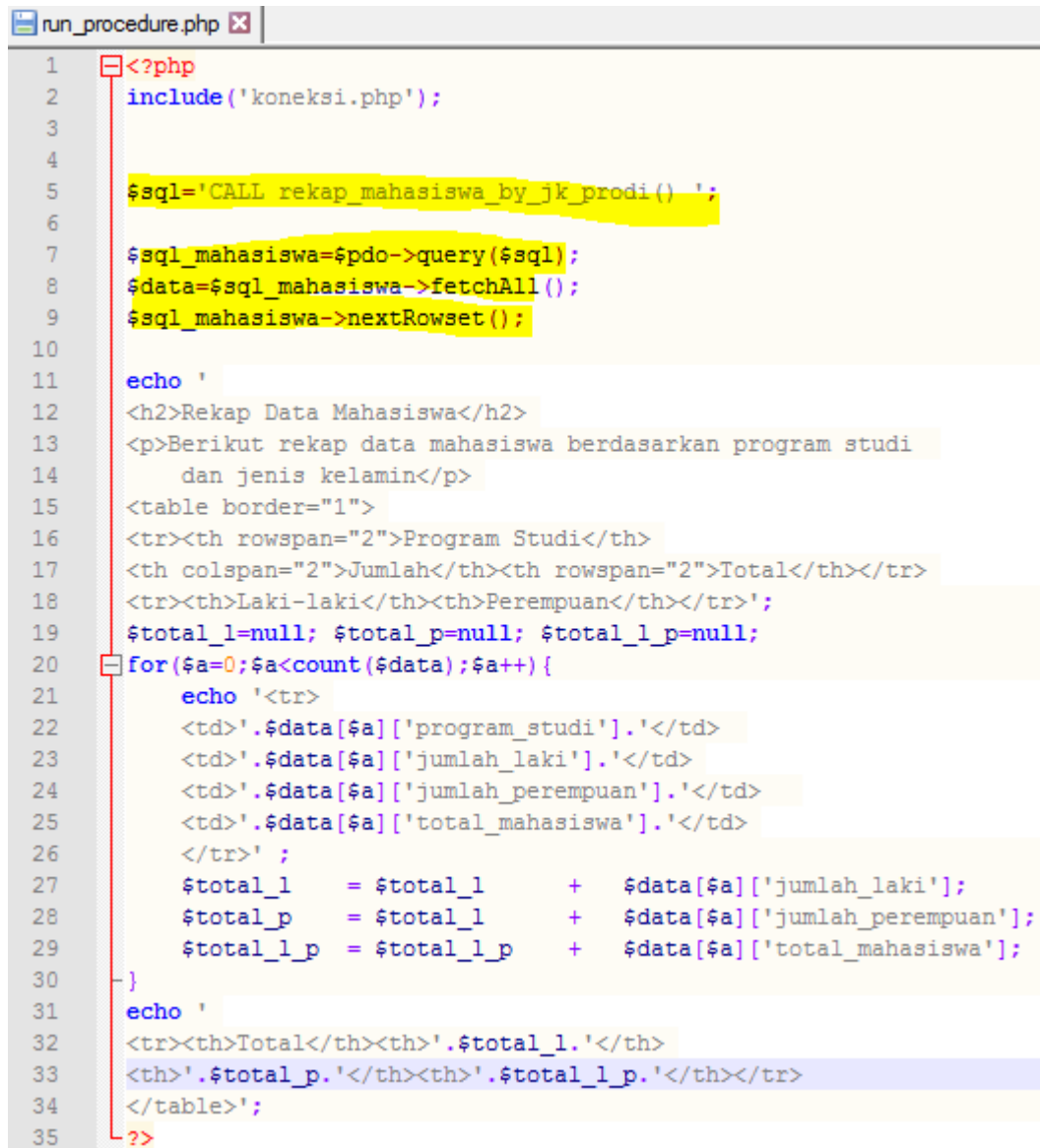
Rekap Data Mahasiswa

Berikut rekap data mahasiswa berdasarkan program studi dan jenis kelamin

Program Studi	Jumlah		Total
	Laki-laki	Perempuan	
Akuntansi	1	1	2
Pendidikan Pancasila	0	3	3
Rekayasa Sistem Informasi	1	0	1
Sistem Informasi	3	3	6
Teknik Informatika	1	0	1
Total	6	6	13

Gambar 70

Adapun script php nya adalah sebagai berikut :

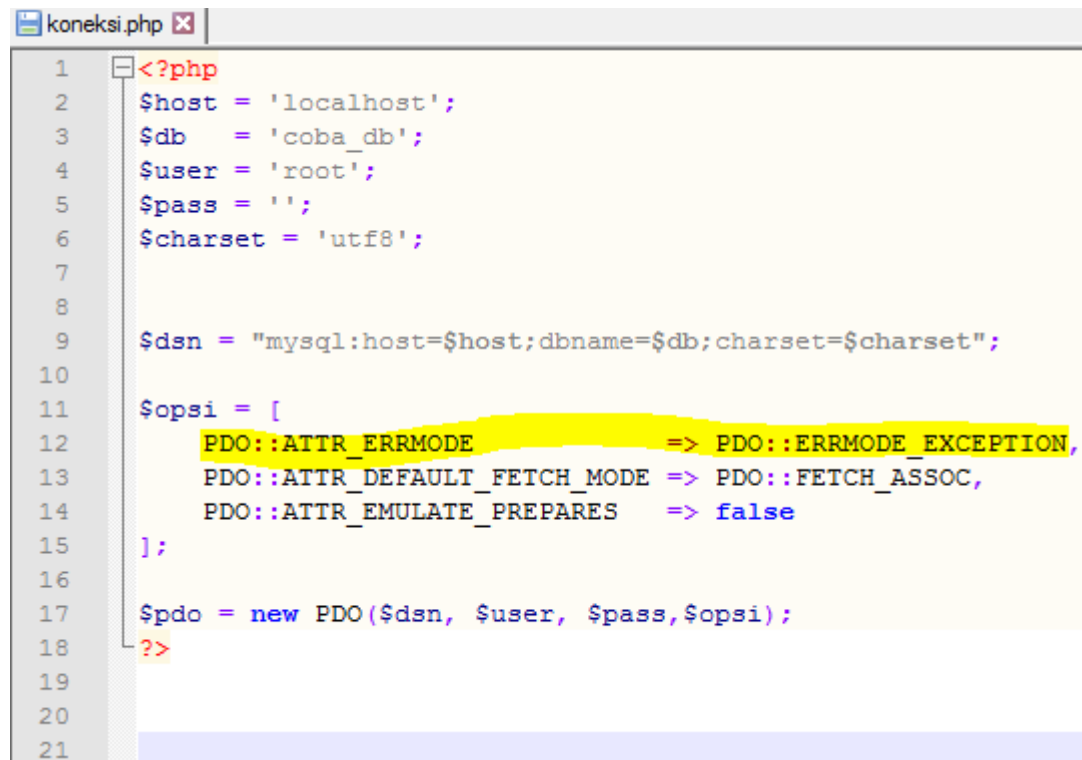


```
1 <?php
2 include('koneksi.php');
3
4
5 $sql='CALL rekap_mahasiswa_by_jk_prodi() ';
6
7 $sql_mahasiswa=$pdo->query($sql);
8 $data=$sql_mahasiswa->fetchAll();
9 $sql_mahasiswa->nextRowset();
10
11 echo '
12 <h2>Rekap Data Mahasiswa</h2>
13 <p>Berikut rekap data mahasiswa berdasarkan program studi
14 dan jenis kelamin</p>
15 <table border="1">
16 <tr><th rowspan="2">Program Studi</th>
17 <th colspan="2">Jumlah</th><th rowspan="2">Total</th></tr>
18 <tr><th>Laki-laki</th><th>Perempuan</th></tr>';
19 $total_l=null; $total_p=null; $total_l_p=null;
20 for ($a=0;$a<count($data);$a++) {
21     echo '<tr>
22         <td>'.$data[$a]['program_studi'].'</td>
23         <td>'.$data[$a]['jumlah_laki'].'</td>
24         <td>'.$data[$a]['jumlah_perempuan'].'</td>
25         <td>'.$data[$a]['total_mahasiswa'].'</td>
26     </tr>';
27     $total_l = $total_l + $data[$a]['jumlah_laki'];
28     $total_p = $total_l + $data[$a]['jumlah_perempuan'];
29     $total_l_p = $total_l_p + $data[$a]['total_mahasiswa'];
30 }
31 echo '
32 <tr><th>Total</th><th>'.$total_l.'</th>
33 <th>'.$total_p.'</th><th>'.$total_l_p.'</th></tr>
34 </table>';
35 ?>
```

Gambar 71

W. Penanganan Error dengan Exceptions

Ada banyak cara untuk penanganan error di PDO, namun satu-satunya yang tepat adalah PDO::ERRMODE_EXCEPTION. Jadi, anda harus selalu mengaturnya dengan menambahkan baris metode ini setelah pembuatan instance PDO, perhatikan kembali file koneksi.php yang telah kita buat diatas .



```
1 <?php
2 $host = 'localhost';
3 $db = 'coba_db';
4 $user = 'root';
5 $pass = '';
6 $charset = 'utf8';
7
8
9 $dsn = "mysql:host=$host;dbname=$db;charset=$charset";
10
11 $opsi = [
12     PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
13     PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
14     PDO::ATTR_EMULATE_PREPARES => false
15 ];
16
17 $pdo = new PDO($dsn, $user, $pass, $opsi);
18 ?>
19
20
21
```

Gambar 72

Daftar Pustaka

<https://phpdelusions.net/pdo>

<http://php.net/manual/en/book.pdo.php>

https://www.w3schools.com/Php/php_mysql_intro.asp

<http://ozs.web.id/?s=stored+procedure&searchsubmit=Search>

Penulis



Penulis saat ini aktif sebagai kepala bagian administrasi akademik di Universitas Kuningan, selain itu juga aktif sebagai tenaga pengajar tidak tetap di program keahlian Rekayasa Perangkat Lunak SMK Negeri 2 Kuningan.

Selain aktifitas diatas penulis juga mengasuh dua buah blog yaitu <http://ozs.web.id> dan <http://latihanexcel.web.id> , juga single fighter sebagai freelancer untuk beberpa project yang dikerjakan dikala waktu senggang. Artikel lepas milik penulis bisa diakses di halaman <https://uniku.academia.edu/TeuWawuh>.

Sampai saat ini konsetrasi penulis adalah web programming terutama di OPP dan Framework Codeigniter serta database administrator.